POLITECNICO DI TORINO

Master's Degree in Mathematical Engineering

Politecnico
di Torino

Master's Degree Thesis

Review of Batch Process Scheduling

Supervisors

Prof. PAOLO BRANDIMARTE

Prof. EDOARDO FADDA

Candidate

LEONARDO TURRISI

03/2024

“Alla mia famiglia,il mio tutto”

Table of Contents

Chapter 1

Introduction

1.1 Abstract

Extensive research has been conducted on models that combine scheduling with decisions related to batching. Tasks may be grouped together if they utilize the same setup on a machine. Another scenario for batching arises when a machine has the capability to handle multiple tasks concurrently. This document examines the existing body of literature concerning scheduling with batching, providing in-depth information about fundamental algorithms and citing other noteworthy findings.In the paper, we pinpoint areas where techniques developed independently should be assessed and compared.As a multitude of articles have emerged in a short span, there are instances where various researchers have independently addressed the same issue, occasionally employing similar techniques, such as the genetic algorithm or simulated annealing algorithm or dynamic programming. This paper categorize the literature according to shop environments, including single machine, parallel machines, flow shop, no-wait flow shop, flexible flow shop, job shop, open shop, and others. In the past ten years, there has been a notable interest in scheduling challenges that incorporate a batching element. In this context, the motivation behind batching jobs lies in achieving increased efficiency, as it might be more cost-effective or quicker to process jobs collectively rather than individually. A scenario where advantages of batching become apparent is when machines require setups to process jobs with distinct characteristics. Setups may involve changing tools or cleaning the machine. In a family scheduling model, jobs are categorized into families based on their similarities. Consequently, no setup is necessary for a job if it belongs to the same family as the previously processed one. However, a setup time is needed at the beginning of the schedule and whenever the machine transitions from processing jobs in one family to those in another. In this model, a batch represents the maximum set of jobs scheduled consecutively on a machine,

sharing a common setup. Larger batches offer the benefit of enhanced machine utilization due to fewer setup requirements. However, processing a large batch might lead to delays in handling a crucial job from a different family. The family scheduling model has two variations depending on when the jobs become available, either for dispatch to the customer or processing on the next machine. In batch availability, a job becomes available only when the entire batch it belongs to has been processed. An alternative assumption is job availability, where a job becomes available immediately after its processing is completed. Unless stated otherwise, the assumption of job availability is adopted.Another scenario where batching can enhance efficiency is when a batching machine can process multiple jobs simultaneously. For instance, in the manufacturing of circuit boards, 'burn in' operations are conducted in ovens capable of accommodating several jobs. Similar applications occur in chemical processes performed in tanks or kilns. In these cases, a batching machine processes a batch of jobs simultaneously, with occasional upper limits on batch size. There are existing reviews of models that integrate scheduling with batching by Potts and Van Wassenhove [63] and Webster and Baker [79]. In this paper, an updated review is provided, encompassing many recent results in this field. The review categorizes problems as polynomially or pseudopolynomially solvable, binary or unary NP-hard (NP-hard in the ordinary or strong sense), or open. Additionally, enumerative algorithms, heuristic procedures, and local search methods are described, indicating their efficiency and effectiveness.

1.2 Problem specification

We are examining issues related to the arrangement of tasks for a single machine, m parallel machines, and m-machine flow shops, job shops, and open shops. In the case of parallel machines, the machines can be either identical, uniform, or unrelated. This implies that the time it takes to process a job on a machine depends solely on the job, depends solely on the job and the speed of the assigned machine, or depends on both the job and the assigned machine, respectively. In the context of flow shops, each job consists of m operations. The initial operation requires processing on machine 1, the second on machine 2, and so forth, with the final operation requiring processing on machine m. In job shops, each job follows a predetermined sequence of operations on the machines, causing different jobs to traverse the machines in varying orders. In open shops, every job involves an operation on each of the m machines, but the order in which these operations are processed can be selected freely.In a scheduling scenario, machines can be classified as classical (handling one job at a time) or batching (processing multiple jobs simultaneously). Batching machines determine the processing time of a batch as the maximum processing time among the jobs or operations within the batch. All

jobs or operations in a batch share the same completion time. The set of jobs to be processed is denoted as $1; ...; n$. For a single-machine problem, the processing time of job j is represented by p_j ($j = 1; ...; n$). Additional parameters for job j may 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, jobs are grouped into F families, each containing a certain number of jobs denoted as n_f for $f = 1; ...; F$. No setup is needed between jobs of the same family. However, when a job of family g immediately follows a job of a different family f on machine i, a family setup time of s_{ifg} (or s_{i0g} if there is no preceding job) is incurred. If the setup times are consistent across families such that $s_{ifg} = s_{i0g} = s_{ig}$ for all $f \neq g$, then the setup times on machine i are sequence independent; otherwise, they are sequence dependent. Additionally, if for each machine i, the setup times are the same across all families f and g, including the case $f = 0$ (denoted as $s_{ifg} = s_{fg}$), then the setup times are machine independent; otherwise, they are machine dependent. In the context of a single machine, setup times are inherently considered machine independent. Additionally, we adopt the reasonable assumption that the triangle inequality is valid for each machine i, signifying that $s_{ifh} \leq s_{ifg} + s_{igh}$, for all distinct families f, g, and h, including the case where $f \neq 0$. Unless specified otherwise, the setups are presumed to be anticipatory, implying that a setup on a machine doesn't necessitate the presence of any job. When release dates are present and in-shop problems, there is occasional allowance for non-anticipatory setups. This means that the setup preceding the processing of a batch cannot commence on the current machine until all jobs of that batch are released and have completed processing on any previous machine. In scenarios involving sequence-independent family setup times and batch availability, there might be two or more successive batches of the same family. In such instances, a family setup time is required before processing a batch, even if it belongs to the same family as the previous one. If there exists a maximum batch size for any machine i, it is denoted as b_i. In the case of a single machine or when restrictions on batch sizes are uniform across all machines, we use b to represent any imposed maximum batch size. We establish the following variables for each job j in a schedule:

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

The standard classification scheme for scheduling problems, as outlined by Graham et al. [38], is represented as $\psi_1|\psi_2|\psi_3$, where ψ_1 denotes the scheduling environment, ψ_2 describes the job and family characteristics along with any restrictive

requirements, and ψ_3 defines the objective function aimed at minimization. In this scheme, we define ψ_1 as am, where m is the number of machines, and a belongs to the set $\{P, Q, R, F, J, O\}$ for classical machines in identical, uniform, and unrelated parallel-machine, flow shop, job shop, and open shop environments. For batching machines in the same environments, we use a from the set $\{\tilde{P}, \tilde{Q}, \tilde{R}, \tilde{F}, \tilde{J}, \tilde{O}\}$. If m is not specified, the number of machines is considered arbitrary. In the case of a single machine, we omit a, and $m = 1$ signifies a classical machine, while $m = 1$ signifies a batching machine.

- r_j: each job j has a release date r_j;
- d_j: each job j has a deadline d_j;
- $pmtn$: preemption of jobs is allowed;
- $p_j \coloneqq p$: all jobs have a common processing time p;
- $\bar{d_j} \coloneqq \bar{d}$: all jobs have a common deadline $\bar{d}$;
- $d_j \coloneqq d$: 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;
- $F \coloneqq k$: the number of families is a constant k;
- $n_f \coloneqq \frac{n}{F}$: all families have the same number of jobs $\frac{n}{F}$;
- 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;
- $s_f \coloneqq s$: all families have a common (machine- and sequence-independent) setup time s.

The goals that fall under the category ψ_3 involve discovering a viable schedule in which the minimization of one of the following cost functions is necessary.

- $*$: the constant cost function, which is minimized by any feasible schedule;
- $C_{\max} = \max_{j=1\ldots n}\{C_j\}$: the completion time of the last job, or the makespan;
- $\sum w_j C_j$: the total (weighted) completion time of the jobs;

- $L_{\max} = \max_{j=1....n}\{L_j\}$: 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 illustration of the classification scheme is the instance denoted as problem $\tilde{F}2|b_1 := 1, b_2 := 2|C_{\max}$, representing the minimization of makespan in a two-machine flow shop. In this scenario, the first machine is a classical machine, while the second is a batching machine capable of processing up to two jobs simultaneously. Another case is represented by problem $1|s_{fg}|\sum_j C_j$, signifying the minimization of the total completion time on a single (classical) machine. In this context, there are job families and sequence-dependent family setup times. Throughout the paper, it is assumed that all data defining a problem instance are integers.

Chapter 2

Family scheduling models

2.1 Single Machine

Monma and Potts investigated scheduling problems related to job families on a single machine, considering various objective functions. Some of their algorithms have undergone enhancements, and our review presents the most efficient algorithm currently known. In describing these algorithms, it is helpful to represent the j-th job of each family f (where f ranges from 1 to F) as the pair $\langle j, f \rangle$, where j ranges from 1 to n_f.

2.1.1 Total weighted completion time

Monma and Potts [**59**] demonstrate that, for the problem $1|s_{fg}| \sum_j w_j \cdot C_j$, there exists an optimal schedule in which the Smith's Weighted Processing Time (SWPT) rule [**66**] is applicable within each family f. This implies that jobs are sequenced in non-decreasing order of $\frac{p_{j,f}}{w_{j,f}}$. Given that jobs within each family f are arranged in SWPT order, a reindexing of the jobs is performed such that $\frac{p_{1,f}}{w_{1,f}} \leq \ldots \leq \frac{p_{n_f,f}}{w_{n_f,f}}$. Ghosh [**35**] introduces a backward dynamic programming algorithm with job insertion for the problem $1|s_{fg}| \sum_j w_j \cdot C_j$ denote the minimum total weighted completion time for schedules containing jobs $\{(q_f, f), \ldots, (n_f, f)\}$ for $f = 1, \ldots, F$, where job (q_g, g) is the first job to be processed, and its processing starts at time zero. It's important to 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 $f = 1, \ldots, F$ is defined as $G\{n_1 + 1, \ldots, n_F + 1; f\} = 0$. The recursive formula for $q_f = n_f + 1, n_f, \ldots, 1$ and $f = 1, \ldots, F$, and $g = 1, \ldots, F$, where $q_g \neq n_g + 1$, is expressed as

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

$$\left.\left. - s_{g,g'}w_{(q_g,g)}\right\}\right.$$

where $q'_f = q_f$ for $f \neq g$, $q'_g = q_g{+}1$, and $s_{g,g\prime} = 0$ if $g = g'$. This recursive process selects a previous schedule into which job $\{q_g, g\}$ is inserted at the beginning. If the first job of the previous schedule is from family g, then this schedule experiences a delay of $p_{(q_g,g)}$. Conversely, if the first job of the previous schedule is from family g_0, where $g_0 \neq g$, then the delay is $p_{(q_g,g)} + s_{g,g'}$, as the first job in the previous schedule starts a batch, and the associated setup is also included. The optimal solution value is determined by

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

where the final term in the minimization accounts for the delay resulting from inserting the relevant setup at the beginning of the schedule.The time complexity of the algorithm is $O(n^F)$, indicating a polynomial relationship with a fixed value of F. For the problem $1|s_{fg}|\ \sum_j C_j$, Ahn and Hyun [2] introduce a forward dynamic programming algorithm that involves appending jobs and also operates within $O(n^F)$ time. Rinnooy Kan [67] demonstrates that the problem $1|s_{fg}|\ \sum_j C_j$ is unary NP-hard for any arbitrary value of F. Nevertheless, the computational complexity status remains undetermined for $1|s_f|\ \sum_j C_j$ and $1|s_f|\ \sum_j w_j \cdot C_j$ when considering arbitrary values of F. Mason and Anderson [57], along with Crauwels et al. [24], propose branch and bound algorithms for the problem $1|s_{fg}|\ \sum_j w_j \cdot C_j$. Mason and Anderson employ a forward branching rule and utilize dominance rules extensively to limit the size of the branch and bound search tree.The lower bound is determined through the technique of objective splitting. The overall weighted completion time is divided into components related to processing times and setup times, each optimized independently. Crauwels et al. establish a lower bound by employing Lagrangean relaxation on the machine capacity constraints within a time-indexed formulation of the problem. Multiplier values are obtained using either a constructive 'multiplier adjustment' method or subgradient optimization. In their initial algorithm, they apply a forward branching rule incorporating various dominance rules and the multiplier adjustment method to derive lower bounds. Their second algorithm employs a binary branching rule that determines whether adjacent jobs (based on SWPT ordering) within the same family should be in the same or different batches, using subgradient optimization for lower bound computation. Computational results demonstrate

that the algorithm utilizing a forward branching rule and the multiplier adjustment method effectively solves instances with up to 70 jobs and is more efficient than both Mason and Anderson's algorithm and the algorithm employing binary branching and subgradient optimization.For addressing problem $1|s_f|\ \sum_j C_j$, several heuristics have been proposed by Gupta [39], Williams and Wirth [80], and Ahn and Hyun [2]. Gupta's heuristic constructs schedules by iteratively adding jobs, with each job chosen to minimize completion time. Williams and Wirth adopt a similar strategy but incorporate the dominance rules of Mason and Anderson [57] to exclude certain candidates. They also permit specific batch interchanges and backward insertions of jobs into previous batches. Ahn and Hyun present a descent heuristic where sub-batches are repositioned within the sequence of batches. Computational results indicate that the heuristics by Williams and Wirth, as well as Ahn and Hyun, produce higher-quality solutions compared to Gupta's heuristic. An alternative heuristic approach, inspired by Liao and Liao [55] for a more general problem, involves gradually introducing complete families into the current schedule using dynamic programming.Two investigations have formulated local search heuristics for problem $1|s_f|\ \sum_j w_j \cdot C_j$. Mason [56] develops a genetic algorithm based on the insight that knowledge of the first job in each batch allows constructing a solution by ordering the batches using an extension of the SWPT rule. A binary representation is employed, where each element denotes whether the corresponding job initiates a batch. Standard genetic operators are applied to this representation. Crauwels et al. [27] introduce various neighborhood search heuristics (descent, simulated annealing, threshold accepting, and tabu search). The neighborhood involves selecting a sub-batch at the beginning (end) of a batch and moving it to an earlier (later) position in the sequence. Simulated annealing follows a periodic temperature pattern, and a descent algorithm precedes each temperature change. Threshold accepting is applied similarly to simulated annealing. In their tabu search, sub-batches are restricted to a single job, and a limited batch reordering, according to a SWPT rule, is implemented. Computational tests for different problem sizes and family counts demonstrate that all local search methods produce solutions close to the optimum. The best results are achieved with a multi-start version of tabu search for a small number of families and Mason's genetic algorithm for a large number of families. Herrmann and Lee [42] propose a genetic algorithm for problem $1|s_{fg}, \bar{d}_j|\ \sum_j C_j$. They utilize a binary encoding of perturbations to the original deadlines. To obtain a solution, a backward scheduling heuristic minimizes setup time and processes longer jobs as late as possible. Since the resulting schedule may not be feasible with respect to the original (or perturbed) deadlines, a penalty function approach is employed to guide the solution towards feasibility.

2.1.2 Maximum lateness

A dynamic programming approach can be employed to tackle the challenge presented by problem $1|s_{fg}|\ L_{max}$, utilizing similar reasoning as applied to problem $1|s_{fg}|\ \sum_j w_j \cdot C_j$. The fundamental observation, as elucidated by Monma and Potts [59], establishes the presence of an optimal schedule where the EDD rule of Jackson [47] is applicable within each family f (jobs are ordered in non-decreasing fashion based on $d_{(j,f)}$). Since jobs within each family f are sequenced according to the EDD rule, a reindexing of the jobs ensures $d_{(1,f)} \leq \ldots \leq d_{(n_f,f)}$. Ghosh and Gupta [36] propose a backward dynamic programming algorithm with job insertion for addressing problem $1|s_{fg}|\ L_{max}$. Here, $G(q_1, \ldots, q_F, g)$ represents the minimum value of the maximum lateness for schedules involving jobs $(q_f, f) \ldots, (n_f, f)$ for $f = 1, \ldots, F$. In this context, job (q_g, g) takes precedence, commencing processing at time zero, and the setup for the batch of jobs in family g at the schedule's outset is not presently taken into account.The initialization for $f = 1, \ldots, F$ is given by $G(n_1 + 1, \ldots, n_F + 1, \ldots, f) = -\infty$ and the recursion for q_f is defined as $n_f + 1, n_f, \ldots, 1$, where f ranges from 1 to F, and similarly, g ranges from 1 to F, with the condition $q_g \neq n_g + 1$, is

$$G\left(q_1, \ldots, q_F, g\right) \coloneqq \min_{g'=1\ldots F} \left\{ \max \left\{ G\left(q'_1, \ldots, q'_F, g'\right) + p_{(q_g,g)} + s_{g,g'}, p_{(q_g,g)} - d_{(q_g,g)} \right\} \right\}$$

where $q\prime_f = q_f$ for $f \neq g$, $q\prime_g = q_g + 1$, and $s_{g,g'} = 0$ if $g = g'$. The optimal solution value is then equal to $\min_{g=1\ldots F}\{G(1, \ldots, 1, g) + s_{0,g}\}$. Bruno and Downey [8] have demonstrated that, for any given value of F, the problem $1|s_f|\ L_{max}$ is binary NP-hard, even when there are two jobs in each family or all setup times are unitary, and there are three jobs in each family. Currently, the problem remains open regarding unary NP-hardness. Hariri and Potts [40] propose a branch and bound algorithm for the problem $1|s_f|\ L_{max}$. They derive an initial lower bound by neglecting setups, except for those associated with the first job in each family, and solving the resulting problem using the EDD (Earliest Due Date) rule. This lower bound is enhanced by a procedure that assesses whether specific families are split into two or more batches. A binary branching rule determines whether adjacent jobs in the same family, based on the EDD ordering, should be assigned to the same batch or different batches. Computational outcomes indicate the algorithm's success in resolving scenarios involving up to approximately 60 jobs. Schutten et al.[65] introduce a branch and bound algorithm for the problem $1|s_f, r_j|\ L_{max}$. When release dates are present, there is no information available regarding the job order within a family. An essential aspect of their algorithm involves using dummy jobs to represent setup configurations. Rapidly computed lower bounds are derived by relaxing setups and solving the corresponding preemptive problem, employing a forward branching rule. Computational results demonstrate the algorithm's effectiveness in solving instances with up to about 40 jobs. Hariri and

Potts[40], Zdrzařka [83], and Woeginger [81] have devised approximation algorithms for problem $1|s_f|\ L_{max}$ assuming that due dates are non-positive to guarantee a positive objective function. In Hariri and Potts' algorithm, two schedules are considered: the first assigns all jobs of a family to a single batch, and the second splits each family into at most two batches based on due dates. The algorithm has a time complexity of $O(n \log n)$ and produces a schedule with maximum lateness no more than 5/3 times the optimal value. Zdrzařka's algorithm provides a better performance guarantee at the cost of additional computation. The algorithm starts with a schedule where each batch contains all jobs from a family and allows each family to be split into at most two batches. In each iteration, a job is moved from the first batch of its family to the second batch. The algorithm has a time complexity of $O(n^2)$ and generates a schedule with maximum lateness no more than 3/2 times the optimal value. These performance bounds of 5/3 and 3/2 are proven to be tight. Zdrzařka notes that his algorithm can be adapted to problem $1|s_f, r_j|\ L_{max}$ to produce a schedule with maximum lateness no more than 5/2 times the optimal value. Woeginger develops a polynomial-time approximation scheme, which is a family of approximation algorithms denoted as A_ϵ with the property that for any $\epsilon > 0$, algorithm A_ϵ generates a schedule with maximum lateness no more than $1 + \epsilon$ times the optimal value. The time requirement of the scheme is polynomial in the size of the input but may be exponential in $1/\epsilon$.

2.1.3 Weighted number of late jobs

Monma and Potts [59] have devised a forward dynamic programming algorithm with job appending for problem $1|s_{fg}|\ \sum_j w_j \cdot U_j$. This algorithm leverages the property that there exists an optimal schedule where the early jobs within each family are sequenced in Earliest Due Date (EDD) order. Assuming the jobs within each family f are reindexed so that $d_{(1,f)} \leq \ldots \leq d_{(n_f,f)}$, the algorithm aims to minimize the makespan for schedules containing early jobs selected from $(1, f) \ldots, (q_f, f)$, for each family f (where q_f is the number of early jobs selected from the family), and u denotes the number of late jobs. Additionally, the last (early) job in the schedule belongs to family g, and it's worth noting that $q_g \geq 1$. Problem $1|s_{fg}|\ \sum_j w_j \cdot U_j$ is considered quasi-polynomial solvable for a fixed value of F. This conclusion stems from the observation that the dynamic programming algorithm mentioned earlier generalizes to minimize the weighted number of late jobs in $O(n^F W)$ time, where $W = \sum_{j=1}^{n} w_j$. Another dynamic programming algorithm offers an alternative approach, employing the completion time of the schedule of early jobs as a state variable, and expressing the weighted number of late jobs as a function value. This alternative algorithm has a pseudopolynomial time complexity of $O(n^F \min\{d_{\max}, P\})$, where $d_{\max} = \max_{j=1\ldots n} d_j$ and $P = \sum_{j=1}^{n} p_j + \sum_{g=1}^{F} n_g \max_{f=0\ldots F} s_{f,g}$. The NP-hardness in the binary sense for the problem

$1|s_f| \sum_j U_j$ with an arbitrary F can be inferred from the analogous result established for $1|s_f| L_{max}$. Nevertheless, Rote and Woeginger [64] demonstrate that when all jobs within the same family share a common due date, the problem $1|s_f| \sum_j U_j$ becomes solvable within a time complexity of $O(n^2)$ using dynamic programming. The problems $1|s_f| \sum_j U_j$ and $1|s_f| \sum_j w_j \cdot Uj$ remain open concerning unary NP-hardness. Crauwels et al.[25] introduce branch and bound algorithms designed for addressing problem $1|s_f| \sum_j U_j$. They present two strategies for establishing lower bounds. In the first scheme, setup times are relaxed, except for a single setup time for each family, which is distributed among the job processing times. The resulting problem is then solved using Moore's algorithm [61]. The second lower bounding scheme involves due date relaxation. By setting each due date to be equal to $d_{\max}$, the largest among all original due dates, the resulting problem is tackled using dynamic programming. These lower bounds are integrated into branch and bound algorithms that either employ a standard forward branching rule or opt for ternary branching. In the ternary branching rule, each job is categorized as either set to be late, set to be early and initiate a batch, or set to be early and not start a batch. While forward branching allows the application of numerous dominance rules, ternary branching offers more flexibility. Computational results, considering various combinations of lower bounding schemes and branching rules, indicate that the most efficient algorithm incorporates a lower bound based on setup time relaxation and employs forward branching. Furthermore, this algorithm proves successful in solving instances with up to approximately 50 jobs. Crauwels et al.[26] propose enhanced versions of descent, simulated annealing, tabu search, and a genetic algorithm as multi-start approaches for addressing problem $1|s_f| \sum_j U_j$. These neighborhood search algorithms employ 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 while maintaining the early status of all jobs. The batch neighborhood is similar, but it involves removing an entire batch, and then attempts are made to insert late jobs. The genetic algorithm adopts a representation where two binary elements are associated with each job, indicating whether the job is early or late, and whether it ends a batch. To derive a corresponding schedule of early jobs, due dates are associated with each batch of early jobs, and the batches are sequenced in Earliest Due Date (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 involving 30, 40, and 50 jobs, and 4, 6, 8, and 10 families, all heuristics demonstrate good performance. The genetic algorithm, particularly a version that includes a procedure aiming to enhance each solution in the final population, yields the highest-quality solutions.

2.2 Parallel machines

According to observations made by Monma and Potts [59], the orderings of jobs within each family, utilized in the development of dynamic programming algorithms for problems such as $1|s_{fg}|\ \sum_j w_j \cdot C_j$, $1|s_{fg}|\ L_{max}$, and $1|s_{fg}|\ \sum_j (w_j) \cdot U_j$, are also applicable to each machine for identical parallel-machine scheduling. By incorporating state variables to store the total setup plus processing time on each machine, forward dynamic programming algorithms with job appending can be formulated for problems $P|s_{fg}|\ \sum_j w_j \cdot C_j$, $P|s_{fg}|\ L_{max}$, and $P|s_{fg}|\ \sum_j w_j \cdot U_j$, all of which require pseudopolynomial time for a fixed number of machines (m) and families (F). Most NP-hardness results for scheduling an arbitrary number of families of jobs on parallel machines are derived from the corresponding results for classical parallel-machine scheduling. An exception is found in the case of problem $P|s_f|\ \sum_j C_j$, which Webster[77] demonstrates to be unary NP-hard. However, the complexity of the corresponding problem with a fixed number of machines remains an open question. Monma and Potts [59] demonstrate that the problem $P2|s_f, pmtn|\ C_{max}$ is binary NP-hard. In their subsequent work [60], they propose an approximation algorithm for the problem $P|s_f, pmtn|\ C_{max}$, reminiscent of McNaughton's algorithm [58] for the classical scheduling problem $P|pmtn|\ C_{max}$. This algorithm has a time complexity of $O(n)$ and produces a schedule with a makespan no more than $2 - (1/(\lfloor m/2 \rfloor) + 1)$ times the optimal makespan. For a specific category of instances where the setup plus total processing time for each single family doesn't exceed the optimal makespan, Monma and Potts [60] and Chen [10] demonstrate that this performance can be enhanced using an approximation algorithm. This algorithm initially employs list scheduling for complete families and subsequently splits families between selected pairs of machines. Specifically, Chen's algorithm has a time complexity of $O(n \log n)$ and generates a schedule with makespan no more than $\max\{3m/(2m+1), (3m-4)/(2m-2)\}$ times the optimal makespan.

2.3 Shop problems

The classical two-machine flow shop problem, denoted as $F2||Cmax$, exhibits an optimal permutation schedule where the job sequence is the same on both machines. However, it remains an open question whether this characteristic extends to the problem $F2|s_{ifg}|\ C_{max}$. Potts and Van Wassenhove [63] point out that, in the case of anticipatory setups, the search for an optimal permutation schedule can be constrained to schedules where jobs within each family follow the sequencing algorithm proposed by Johnson [48]. In such scenarios, similar arguments as those presented in Sections 4.1.1 and 4.1.2 can be employed to develop a backward

dynamic programming algorithm with job insertion. This algorithm can find an optimal permutation schedule in $O(n^F)$ time. Kleinau [50] demonstrates that problem $F2|s_{if}|\ C_{max}$, characterized by machine-dependent and sequence-independent setup times, is binary NP-hard for both anticipatory and non-anticipatory setups. However, the status of the problem remains open if setup times are machine independent. Kleinau also establishes the binary NP-hardness of problem $O2|s_f|\ C_{max}$, even when there are only two families. NP-hardness results for scheduling an arbitrary number of families with various objective functions are derived from the corresponding outcomes in classical shop scheduling. Chen et al. [11] propose two approximation algorithms for problem $F2|s_{if}|\ C_{max}$, each with a time complexity of $O(n \log n)$. The first algorithm assigns all jobs of a family to a single batch and then schedules the batches, resulting in a makespan no more than 3/2 times the optimal makespan. The second algorithm utilizes properties of the schedule created by the first algorithm to generate another schedule by splitting each family into at most two batches. The superior of the two schedules has a makespan no more than 4/3 times the optimal makespan. Notably, these performance bounds of 3/2 and 4/3 are proven to be tight. Vakharia and Chang [74] and Skorin-Kapov and Vakharia [71] present simulated annealing and tabu search techniques, respectively, for addressing the permutation flow shop problem $F|s_{if}|\ C_{max}$. However, their focus is specifically on schedules where each family is processed as a single batch. Sotskov et al. [72] conduct a comparative analysis of heuristics applied to both the permutation flow shop problems $F|s_{if}|\ C_{max}$ and $F|s_{if}|\ C_j$. Their approach involves restricting the search to schedules that share the same batches and maintain an identical processing order of batches on each machine. They introduce constructive heuristics based on the successive insertion of jobs into the sequence, along with local search methods such as simulated annealing, threshold accepting, and tabu search. The neighborhood structure employed in their methods is similar to the one utilized by Crauwels et al. [27] (refer to Section 4.1.1). In computational experiments involving 40, 60, and 80 jobs, as well as 5 and 10 machines, the most promising results are achieved using simulated annealing and tabu search, with the former yielding slightly superior-quality solutions.

2.4 Identical jobs in each family

In this section, we consider scenarios where setups are assumed to be independent of both sequence and machine. Additionally, each family, denoted as f, comprises n_f identical jobs, each having a processing time p_f, and relevant parameters such as deadline $\bar{d}$, due date d_f, and weight w_f for f ranging from 1 to F. For problems $1|s_f|\ \sum_j w_j \cdot C_j$ and $1|s_f|\ L_{max}$, Dobson et al. [29] and Santos [68], respectively, demonstrate the existence of an optimal schedule where no family is

divided. Moreover, the batches formed from complete families of jobs are arranged based on a generalization of the Shortest Processing Time (SWPT) rule and the Earliest Due Date (EDD) rule, respectively. Consequently, both of these problems can be solved within a time complexity of $O(F \log F)$. Various special cases of problem $1|s_f| \sum_j w_j \cdot U_j$ have been investigated, leading to the development of polynomial time algorithms and NP-hardness proofs. Specifically, the problem is established as binary NP-hard in instances where $1|s_f, p_j = 1, d_j = d| \sum_j U_j$ and $1|s_f = s, d_j = d| \sum_j U_j$ [52]. Another NP-hardness result is demonstrated for the case $1|s_f = s, p_j = 1, d_j = d| \sum_j w_j \cdot C_j$ [51]. Furthermore, algorithms with a time complexity of $O(F \log F)$, inspired by Moore's algorithm [61], are applicable to certain scenarios. These include cases such as $1|s_f = s, d_j = d| \sum_j U_j$ [52], $1|s_f, n_f = n/F, d_j = d| \sum_j U_j$, and $1|s_f = s, n_f = n/F, p_j = p| \sum_j U_j$ [16]. Additionally, the problem $1|s_f = s, n_f = n/F, p_j = p, d_j = d| \sum_j w_j \cdot U_j$ is proven to be solvable in $O(n \log n)$ time. However, the complexity of special cases with general due dates remains an open question. Kovalyov et al.[52] and Cheng and Kovalyov [16] have proposed pseudopolynomial dynamic programming algorithms for addressing problems $1|s_f| \sum_j U_j$ and $1|s_f| \sum_j w_j \cdot U_j$, respectively. These algorithms introduce rounding techniques that enable the derivation of a fully polynomial approximation scheme. This scheme comprises a family of approximation algorithms denoted as A_ε. The defining characteristic of such algorithms is that, for any $\varepsilon > 0$, the algorithm A_ε produces a schedule with a weighted number of late jobs no more than $1 + \varepsilon$ times the optimal value. Moreover, the time requirement of these algorithms is polynomial in both n and $1/\varepsilon$. Building upon the framework established by Kovalyov et al.[52] and enhanced by an idea introduced by Hassin [41], algorithm A_ε, as part of the approximation scheme, demonstrates a time complexity of $O(n^3/\varepsilon + n^3 \log \log n)$. As a notable by-product of their analysis, Kovalyov et al.[52] also derive an $O(n \log n)$ algorithm for minimizing the maximum number of late jobs within each family. Cheng and Chen [12] and Webster [77] have independently demonstrated that problems involving machines with varying setup times and processing times, specifically denoted as $P2|s_f, p_j = p| \sum_j C_j$, and their corresponding simplified version $P|s_f| \sum_j C_j$, are binary and unary NP-hard, respectively. Notably, the binary NP-hardness result for $P2|s_f, p_j = p| \sum_j C_j$ does not directly imply NP-hardness for the analogous problem without the constraint of identical jobs[78], given the differences in input size. As discussed in Section 4.2, the computational complexity of $P2|s_f| \sum_j C_j$ remains an unresolved question. Brucker et al. [7] investigate the computational complexity of various special cases within the unrelated parallel-machine problem $R|s_j, \bar{d}_j|$, where the objective is to find a schedule that satisfies the given deadlines. Specifically, they establish that problem $P2|s_f = 1, p_j = 1, \bar{d}_j = \bar{d}|$ is binary NP-hard. Additionally, 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 proven to be unary NP-hard. The latter problem is

shown to be solvable in $O(m \cdot m!)$ time through a modification of McNaughton's wrap-around rule [58], making it polynomially solvable for a fixed number of machines (m). Furthermore, this algorithm's polynomial solvability extends 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 situations involving identical machines, an iterative algorithm is developed with a time complexity of $O(\log m)$. In each iteration of this algorithm, the maximum possible batch size is calculated, and batches of this size are assigned to all machines. The study by Kovalyov and Shafransky [53] establishes that, in the absence of a common deadline, these problems with an arbitrary number of machines become unary NP-hard, even for the specific case of scheduling families with one machine, unit processing time, and a deadline for each job. It is noteworthy that, when scheduling families of identical jobs on parallel machines, the NP-hardness of a special case implies the NP-hardness of the corresponding general problem, whether binary or unary. Consequently, NP-hardness results for problems with identical parallel machines and a zero cost function extend to analogous results for unrelated parallel-machine problems and other introduced objective functions discussed in Section 2, respectively. The outstanding questions pertain to the complexities of specific problems when family sizes are fixed ($s_f = s$) and processing times are uniform ($p_j = p$), with the condition that s is not a multiple of p. For the problem $R|s_f, \bar{d}_j|$, Brucker et al. [7] introduce a set of approximation algorithms denoted as DP_ϵ. For any $\epsilon > 0$, the algorithm DP_ϵ generates a schedule where $C_j \leq (1 + \epsilon)d_j$ for $j = 1; \ldots; n$, provided that a feasible schedule exists. The time complexity of algorithm DP_ϵ is $O(F^{2m+1}/\epsilon^{2m})$ $O(F^{2m+1}/\epsilon^{2m})$

Chapter 3

Batch availability

Batch availability, whether on a single machine or parallel machines, entails the simultaneous completion of all jobs within the same batch. This occurs when the last job in the batch finishes its processing. In the context of shop problems, batch availability extends to the point where all jobs within a batch are ready for processing on the next machine or are deemed complete if no further operations are required, aligning with the batch's processing completion. In both scenarios, jobs within a batch undergo sequential processing, resulting in the batch's processing time being the sum of the individual processing times of its constituent jobs or operations. The discussion of single-machine, parallel-machine, and shop problems is presented in distinct subsections. It is worth noting that the majority of complexity results pertain to scenarios involving a single family.

3.1 Single machine

Firstly, we examine the problem $1|s_f = s, F = 1|\ \sum_j C_j$. Coffoman et al. [23] have demonstrated that there exists an optimal schedule in which jobs are arranged in Shortest Processing Time (SPT) order. To facilitate further discussion, we redefine the job indices such that $p_1 \leq \ldots \leq p_n$. Coffman et al. [23] have introduced a backward dynamic programming algorithm with batch insertion specifically designed for addressing problem $1|s_f = s, F = 1|\ \sum_j C_j$. Let G_j represent the minimum total completion time for schedules that include jobs from j to n, where the schedule initiates at time zero with a setup time, followed by the processing of a batch containing job j. The initialization is

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

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

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

The minimization process involves selecting a batch $\{j, \ldots, k-1\}$ for insertion at the beginning of the preceding schedule that contains jobs from k to n. The batch $\{j, \ldots, k-1\}$ completes at time $s + \sum_{h=j}^{k-1} p_h$, and the processing of the batches containing jobs from k to n is delayed by $s + \sum_{h=j}^{k-1} p_h$ due to the insertion. The optimal solution value is then determined by G_1. In the most straightforward implementation, the algorithm has a time complexity of $O(n^2)$. Coffman et al. [23] offer an implementation that solves the aforementioned recursion in $O(n)$ time, given that the jobs have been reindexed. Firstly, we examine the problem $1|s_f = s, F = 1| \sum_j C_j$. Coffman et al. [23] have demonstrated that there exists an optimal schedule in which jobs are arranged in Shortest Processing Time (SPT) order. To facilitate further discussion, we redefine the job indices such that $p_1 \leq \ldots \leq p_n$. Coffman et al. [23] have introduced a backward dynamic programming algorithm with batch insertion specifically designed for addressing problem $1|s_f = s, F = 1| \sum_j C_j$. Let G_j represent the minimum total completion time for schedules that include jobs from j to n, where the schedule initiates at time zero with a setup time, followed by the processing of a batch containing job j. Hence, the problem $1|s_f = s, F = 1| \sum_j C_j$ can be solved within a time complexity of $O(n \log n)$. In this more efficient approach, a queue is employed to store candidate jobs, denoted as k, that can initiate the second batch. Here, the first batch begins with job j, where j ranges from n to 1. Through the derivation of various properties of the dynamic program, Coffman et al. demonstrate that the queue can be initialized and maintained in $O(n)$ time. Alternatively, an implementation utilizing the geometric techniques proposed by van Hoesel et al. [75] achieves the same time complexity. Albers and Brucker [3] establish the unary NP-hardness of the problem $1|s_f = s, F = 1| \sum_j w_j \cdot C_j$. They further demonstrate that, given a job sequence, a dynamic programming approach, as described above, can solve the problem $1|s_f = s, F = 1| \sum_j w_j \cdot C_j$ in $O(n)$ time. Additionally, Albers and Brucker prove that when the problem $1|s_f = s, F = 1, p_j = p| \sum_j w_j \cdot C_j$ involves an optimal schedule where jobs are sequenced in non-increasing order of weights, the problem is solvable in $O(n \log n)$ time. Cheng et al.[14] address the problem $1|s_f| \sum_j C_j$ with an arbitrary number of families. They establish the existence of an optimal schedule in which the jobs within each family are sequenced in Shortest Processing Time (SPT) order. Drawing on concepts from Monma and Potts [59] and Coffman et al.[23], Cheng et al.[14] propose a backward dynamic programming algorithm with batch insertion, which has a time complexity of $O(n^F)$ and is polynomial for a fixed F. For the problem $1|s_f = s, F = 1, p_j = p| \sum_j C_j$

where all jobs are identical, a solution can be achieved in $O(n)$ time using the dynamic programming algorithm mentioned earlier. However, it is important to note that this is not a polynomial algorithm since the input is solely comprised of n, s, and p. Santos[68] and Santos and Magazine [69] offer a crucial contribution to the development of a polynomial algorithm by analyzing a continuous relaxation where batch sizes are not restricted to integers. In particular, they demonstrate that the optimal number of batches is given by $B = \lceil \sqrt{\frac{1}{4} + \frac{2np}{s}} - \frac{1}{2} \rceil$, and optimal batch sizes are $n/B + s(B+1)/(2p) - ks/\ p$ for $k = 1, \ldots, B$. Shallcross [70] presents an $O(\log p \log(np))$ algorithm for the discrete problem. However there are similarities between the solutions for integer and continuous cases, the algorithm is intricate. Hurink [46] conducts a comparison of tabu search algorithms applied to problem $1|s_f|\ \sum_j w_j \cdot C_j$. In his approach, solutions are represented as sequences, and he employs the $O(n)$ dynamic programming algorithm to partition the sequence into batches. Preliminary findings suggest that this representation outperforms an alternative representation, where solutions are depicted as a set of batches sequenced using a generalized version of the Shortest Weighted Processing Time (SWPT) rule. The study involves evaluating the transpose neighborhood against a restricted version of the insert or shift neighborhood. In the transpose neighborhood, two adjacent jobs are exchanged, and Hurink develops an $O(n)$ algorithm to identify the best transpose neighbor. For the restricted insert neighborhood, a job is extracted from its current position and reinserted into a new position, ensuring that the absolute difference in positions is at most 3, 5, or 10. Computational results, based on instances with up to 200 jobs, indicate that performance differences between the two methods are minimal. Furthermore, the most superior solutions are obtained through a hybrid approach that alternates between the two neighborhoods.Now, let's explore the objective function related to the maximum lateness. In the context of problem $1|s_f = s, F = 1|\ L_{max}$, Webster and Baker [79] establish the existence of an optimal schedule in which jobs are arranged in Earliest Due Date (EDD) order. To facilitate further discussion, we redefine the job indices so that $d_1 \leq \ldots \leq d_n$. Webster and Baker [79] introduce a backward dynamic programming algorithm with batch insertion tailored for addressing problem $1|s_f = s, F = 1|\ L_{max}$. In this algorithm, G_j represents the minimum value of the maximum lateness for schedules that include jobs from j to n, with the schedule commencing at time zero, involving a setup time followed by the processing of a batch containing job j. The initialization is $G_{n+1} = -\infty$ and the recursion for $j = n, n-1, \ldots, 1$ is

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

The optimal value of the solution is given by G_1. The algorithm operates with a time complexity of $O(n^2)$. For the problem $1|s_f = s, F = 1, \bar{d}_j|$, where the

objective is to find a feasible schedule respecting specified deadlines, Hochbaum and Landy [43] propose a more efficient algorithm. The jobs are assumed to be indexed in Earliest Due Date (EDD) order based on the deadlines. The algorithm works as follows: The first batch comprises jobs 1 through j, where j is the maximum number of jobs that can fit in the first batch without surpassing the deadline $\bar{d}_1$. Similarly, each subsequent batch includes the maximum number of jobs with the smallest indices (excluding those in the preceding batches) that can be scheduled to complete by time $\bar{d}_{j+1}$. This process repeats until job n is included in some batch, resulting in a feasible schedule, or until a job cannot be included in a batch without violating the deadline for that batch, indicating that no feasible schedule exists. The algorithm involves $O(n \log n)$ time for reindexing the jobs and $O(n)$ time for constructing the batches. We will now explore the objective function related to the (weighted) number of late jobs. In the context of problem $1|s_f = s, F = 1| \sum_j w_j \cdot U_j$, Hochbaum and Landy [43] demonstrate the existence of an optimal schedule where early jobs are arranged in Earliest Due Date (EDD) order, followed by any late jobs. For clarity, we redefine the job indices so that $d_1 \leq \ldots \leq d_n$. Brucker and Kovalyov [6] introduce a forward dynamic programming algorithm with job appending specifically designed for addressing problem $1|s_f = s, F = 1| \sum_j U_j$ denote the minimum makespan for schedules containing early jobs selected from jobs 1 to 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 $j = 1, \ldots, n$, $u = 0, 1, \ldots, j$, and $h = 0, 1, \ldots, j$ is

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

The three components in the minimization correspond to scenarios where job j is late, job j is early but doesn't initiate a batch, and job j is early and starts a batch, respectively. The minimum number of late jobs is determined by finding the smallest u for which $\min_{h=0,1,\ldots,n} G_n(u,h) < \infty$.

The algorithm's time complexity is $O(n^3)$. For the problem $1|s_f = s, F = 1| \sum_j w_j \cdot U_j$, it is pseudopolynomially solvable in $O(n^2 W)$ time, where $W = \sum_{j=1}^{n} w_j$, through a straightforward extension of the aforementioned algorithm. An alternative backward dynamic programming algorithm with job insertion is proposed by Hochbaum and Landy [43], requiring $O(n^2 \min\{d_{\max}, P\})$ time, where $d_{\max} = \max_{j=1,\ldots,n} d_j$ and $P = ns + \sum_{j=1}^{n} p_j$. For the problem $1|s_f = s, F = 1, p_j = p| \sum_j w_j \cdot U_j$, this algorithm can be implemented in $O(n^4)$ time. Employing standard rounding techniques, the dynamic programming algorithm by Brucker

and Kovalyov [6] results in a fully polynomial approximation scheme: the time complexity to generate a schedule with the weighted number of late jobs, no more than $1+\epsilon$ times the optimal value, is $O(n^3/\epsilon + n^3 \log\log n)$.

3.2 Parallel machines

Two notable investigations focus on parallel-machine scheduling when batches are available. Cheng et al. [13] developed a backward dynamic programming algorithm with batch insertion for the problem $P|s_f = s, F = 1|\ \sum_j C_j$. This algorithm is built upon the concepts introduced by Co*m*an et al. [23], as discussed in the preceding subsection. It leverages the characteristics of an optimal schedule, where jobs on each machine are sequenced according to Shortest Processing Time (SPT), and each batch consists of adjacent jobs in the SPT order. The algorithm's time complexity is polynomial, specifically $O(mn^{m+1})$, making it efficient for a fixed number of machines (m). Cheng et al. also demonstrate that when there is a common processing time, the problem $P|s_f = s, F = 1, p_j = p|\ \sum_j C_j$ reduces to a single-machine case. As a result, it becomes solvable in polynomial time using Shallcross's algorithm [70]. Another problem, $R|s_f = s, F = 1, \bar{d}_j|$, is investigated by Cheng and Kovalyov [18]. They introduce a dynamic programming algorithm along with a family of approximation algorithms denoted by A_ϵ. For any $\epsilon > 0$, algorithm A_ϵ constructs a schedule in which the completion time of each job is at most $(1+\epsilon)$ times the value of its deadline, provided there exists a feasible schedule respecting the deadlines. The time complexity of A_ϵ is $O(n^{2m+1}/\epsilon^m)$. Cheng and Kovalyov also explore special cases, most of which are NP-hard, especially when the corresponding classical parallel-machine problems with $s = 0$ are NP-hard. Additionally, problem $P|s_f = s, F = 1, p_j = p, \bar{d}_j|$ is proven to be unary NP-complete, in contrast to its classical counterpart with $s = 0$, which is polynomially solvable. They present a dynamic programming algorithm for this problem with a time complexity of $O(m^2 n^{2m+1})$. The exceptional 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.

3.3 Shop problems

Glass, Potts, and Strusevich [37] present complexity results and approximation algorithms for problems involving two parallel machines. Specifically, the problem $F2|s_{if}, F = 1|\ C_{max}$ is proven to be unary NP-hard. In this context, an optimal solution exists with identical batches on each machine, ensuring consistency. The authors also introduce an approximation algorithm for problem $F2|s_{if}, F = 1|\ C_{max}$, which produces a schedule with a makespan that is at most $\frac{4}{3}$ times the optimal value. It's noteworthy that this bound is considered tight. The problem $O2|s_{if}, F =$

$1|\ C_{max}$ is proven to be NP-hard in binary and has an optimal solution featuring one, two, or three consistent batches on each machine. By establishing a close connection between problem $O2|s_{if}, F = 1|\ C_{max}$ and the task of minimizing makespan on two and three identical parallel machines, the authors demonstrate that $O2|s_{if}, F = 1|\ C_{max}$ is solvable in pseudopolynomial time. Furthermore, algorithms designed to approximate the scheduling of two and three identical parallel machines, aiming to minimize the makespan, offer corresponding performance assurances for the problem $O2|s_{if}, F = 1|\ C_{max}$. Cheng and Wang [21] investigate a specific instance of the problem $F2|s_{if}, F = 1|\ C_{max}$, where the setup time on the first machine is zero, resulting in a batch for each job's initial operation. This problem is established as being NP-hard for both anticipatory and non-anticipatory setups, with some polynomially solvable special cases being outlined. Cheng et al. [20] investigate the problem $F2|s_{if}, F = 1|\ C_{max}$ under the condition of non-anticipatory setups, with the additional constraint that batches must be consistent. The study establishes the NP-hardness of the problem, even when each job's processing time on the first machine does not exceed that on the second machine (or vice versa). The problem becomes solvable in $O(n^2)$ time if all jobs share a common processing time on the first machine or the second machine, as demonstrated by algorithms proposed by Cheng et al. [20]. Additionally, for cases where all operations have a common processing time, an algorithm by Tanaev et al. [73] achieves a solution in $O\left(\sqrt{n}\right)$ time. However, it's worth noting that the latter time complexity is not polynomial, leaving the problem open when considering a common processing time for all operations. Tanaev et al. [73] extend these results to the scenario where setups are anticipatory. The methods developed by Sotskov et al. [72], as explained in Section 4.3 for the permutation flow shop problem $F|s_{if}|\ \sum_j C_j$, are employed with slight modifications in the analogous problems involving batch availability and mixed availability (where some machines follow job availability, and others follow batch availability). The outcomes are comparable to those observed in the case of job availability, with a slight preference for tabu search over simulated annealing specifically in scenarios involving batch availability.

3.4 Batch delivery scheduling

In this section, we consider a scenario where there is a single group of jobs, and a setup time is necessary before processing each batch. All jobs within a batch are released when the last job in that batch finishes processing. For any given schedule where job j is part of batch k, we define $H_j = D_k - C_j$, where D_k represents the completion time of batch k. This definition characterizes the holding time, representing the duration during which job j awaits delivery. Cheng et al. [15] demonstrate that problems involving single- and parallel-machine scheduling with

criteria based on job holding times Hj for $j = 1, \ldots, n$ are equivalent to traditional parallel-machine scheduling problems. Consequently, existing algorithms designed for parallel-machine scheduling can be appropriately adapted for these scenarios. Additionally, Cheng et al. [19] establish that the single-machine problem, aiming to minimize $\sum_{j=1}^{n} w_j H_j + \frac{1}{B} \sum_{k=1}^{B} D_k$, where B is the unknown number of batches, is unary NP-hard. However, it becomes solvable in $O(n^2)$ time when all jobs share a common weight or a common processing time. Yang [82] examines single-machine problems where batch delivery dates, denoted as $D_1 \leq \ldots \leq D_B$, are specified, with $D_B = \sum_{j=1}^{n} p_j$. The objective is to allocate jobs to B batches in a way that ensures batch k is completed no later than time D_k for $k = 1, \ldots, B$. The optimization goal is either to minimize the total weighted holding time $\sum_{j=1}^{n} w_j H_j$ or the maximum weighted holding time $\max_{j=1,\ldots,n}\{w_j H_j\}$. Yang proves that these problems are unary NP-hard and become binary NP-hard when $B = 2$ and all weights are equal. However, in the case of a common processing time, he demonstrates that the problems can be solved in $O(n \log n)$ time.

3.5 Multi-operations job

This section explores the model of a single-machine, multi-operation job, where each job (denoted as j) comprises a single operation belonging to a family (indexed as f) within the range of f from 1 to F. Each operation has a processing time denoted as p_{jf}, although some operations within this context may be absent. The completion time of a job is defined as the moment when the processing of its final operation concludes. As highlighted by Julien and Magazine [49], this model finds application when a job includes multiple orders that require simultaneous delivery to the customer. Additionally, Gerodimos et al. [32] describe other scenarios in which distinct components are manufactured for subsequent assembly into a final product or when ingredients are produced for subsequent blending into a final product. In this scenario, we assume that the time required for the final assembly or blending stage is negligible. Consequently, the problem is simplified to scheduling a single machine. To accommodate multi-operation jobs, we modify our problem classification scheme, as outlined in Section 2, by introducing the entry "multi-op" in w_2 to signify the presence of jobs with multiple operations. Gerodimos et al. [34] present complexity results for various single-machine multi-operation job problems. They highlight an equivalence between the problem $1|s_f, multi - op|\ L_{max}$ and the single-operation problem $1|s_f|\ L_{max}$. In the latter, operations are treated as jobs, and each operation is assigned the due date of the corresponding job. Using the outcomes discussed in Section 4.1.2, this correspondence demonstrates that the problem $1|s_f, multi - op|\ L_{max}$ is proven to be binary NP-hard for any arbitrary F, as established by Bruno and Downey [8]. However, it can be solved

in $O(n^F)$ time utilizing the dynamic programming algorithm developed by Ghosh and Gupta [36]. Additionally, Gerodimos et al. illustrate that the problem $1|s_f = s, multi - op| \sum_j U_j$ is binary NP-hard, even when there are only two families involved. Furthermore, the problem $1|s_f = 1, multi - op| \sum_j U_j$ is unary NP-hard, even when all operations (except those that are missing) have a processing time of one unit. However, for a fixed value of F, a backward dynamic programming algorithm efficiently solves problem $1|s_f = 1, multi - op| \sum_j w_j \cdot U_j$ time, where $d_{\max}$ is the largest due date. The crucial observation lies in scheduling the final operation of a job, after which the remaining operations are grouped in a sub-batch awaiting scheduling. It suffices to store the processing time of such sub-batches as state variables. In a specific scenario of problem $1|s_f, multi - op| \sum_j C_j$, where the processing times of operations between families align harmoniously (there exists a job indexing such that$p_{1,f} < \ldots < p_{n,f}$ for $f = 1, \ldots, F$), a similar dynamic programming algorithm yields a solution in $O(n^F)$ time.However, the computational complexity of the general instance of the problem $1|s_f, multi - op| \sum_j C_j$ remains unknown. Coffman et al. [22] demonstrate that the two-operation problem $1|s_f = s, multi - op| \sum_j C_j$ can be solved in $O(n^p)$ time, where p is a constant, given that all operations in the first family share a common processing time, and all operations in the second family share a common processing time. It's important to note that this time complexity is not polynomial. Gerodimos [31] establishes that the previously mentioned NP-hardness results also apply when considering batch availability. Additionally, the dynamic programming algorithms designed for problems $1|s_f, multi - op| \ L_{max}$ and $1|s_f, multi - op| \sum_j w_j \cdot U_j$ can be modified to accommodate batch availability, albeit with a slight increase in time complexity. Several complexity findings pertain to a distinct scenario within the two-operation job model introduced by Baker [4]. In this particular instance, each job comprises both a standard and a specific operation. The standard operations are grouped in the same family, while each specific operation has its unique family. Gerodimos et al. [33] unveil several inherent structural characteristics that lend themselves to the development of dynamic programming algorithms. Specifically, for a special case of problem $1|s_f, multi - op| \sum_j C_j$ where the processing times of the two operations align harmoniously, they propose a backward dynamic programming algorithm with a time complexity of $O(n^2)$—an improvement over Vickson et al.'s [76] $O(n^3)$ algorithm. A similar algorithm efficiently addresses problem $1|s_f, multi - op| \ L_{max}$ within $O(n^2)$ time. Additionally, Gerodimos et al. [33] demonstrate that problem $1|s_f, multi - op| \sum_j U_j$ is NP-hard in a binary sense but can be solved by a forward dynamic programming algorithm within $O(n^2 d_{\max})$ time. Moreover, the specific instance of problem $1|s_f, multi - op| \sum_j U_j$, where each standard operation shares a common processing time, is solvable in polynomial time, and problem $1|s_f, multi - op| \sum_j w_j \cdot U_j$ can be solved in pseudo-polynomial time. In the presence of batch availability, Gerodimos et al.[32] demonstrate that an optimal schedule

exists where the standard operation of any job precedes its specific operation, a condition not applicable under job availability. For problem $1|s_f, multi-op| \sum_j C_j$ under batch availability, assuming the mentioned agreeability, an algorithm by Coffman et al.[23] with a time complexity of $O(n \log n)$ surpasses a previous $O(n^2)$ algorithm by Baker. The findings of Gerodimos et al.[32] indicate that the complexity results mentioned earlier for problems $1|s_f, multi-op| L_{max}$ and $1|s_f, multi-op| \sum_j w_j \cdot U_j$ under job availability also extend to batch availability, although the algorithms and proofs differ.

Chapter 4

Batching machine models

A batching machine, also known as a batch processing machine, is a device designed to process multiple jobs simultaneously, with a key scheduling criterion being that the completion times of jobs should not decrease. Jobs that are processed together are grouped into a batch. More specifically, the focus is on the burn-in model, where the processing time of a batch is determined by the maximum processing time among the jobs assigned to it. All jobs within the same batch commence and conclude simultaneously, as the completion time of an individual job aligns with the completion time of the batch to which it belongs. The motivation behind this model stems from the challenge of scheduling burn-in operations in the manufacturing of large-scale integrated circuits.

Webster and Baker [85] provide an overview of algorithms and complexity results associated with scheduling batch processing machines. They categorize three types of models: the burn-in model, a model where the processing time of a batch is the sum of the processing times of its constituent jobs , and a model where the processing time of a batch remains constant, independent of the jobs it contains .

The analysis delves into two variations of the burn-in model: the unbounded model, where b is greater than or equal to n, allowing for an effectively unlimited number of jobs to be processed in the same batch; and the bounded model, where b is a constant smaller than n, imposing a restrictive upper limit. The unbounded model is applicable, for instance, in scenarios where compositions need to undergo hardening in kilns, and the kiln is sufficiently large to accommodate varying batch sizes without constraints. It is assumed throughout that both jobs and the machine are available from time zero onwards, or equivalently, jobs have equal release dates. Notably, when b equals 1, it corresponds to the classical single-machine scheduling model, where the machine can handle only one job at a time. Consequently, the bounded model introduces challenges that are at least as difficult as their traditional counterparts. For the unbounded model, a classification of optimal schedules is provided, leading to a versatile dynamic

programming algorithm aimed at minimizing any regular cost function $\sum_{j=1}^{n} f_j$. This algorithm has a time complexity of $O(n^2P)$ and a space complexity of $O(nP)$, where P represents the sum of job processing times. The identified classification serves as the foundation for polynomial dynamic programming algorithms tailored to specific cost functions. Specifically, an algorithm with a time complexity of $O(n^3)$ is presented for minimizing the number of tardy jobs $\sum_{j=1}^{n} U_j$, an algorithm with $O(n \log n)$ time complexity for minimizing the total weighted completion time $\sum_{j=1}^{n} w_j C_j$, and an algorithm with $O(n^2)$ time complexity for minimizing the maximum lateness $L_{\max}$. The latter algorithm can be utilized to formulate a polynomial algorithm for minimizing the maximum cost $f_{\max}$. Additionally, it is demonstrated that minimizing the weighted number of tardy jobs $\sum_{j=1}^{n} w_j U_j$ and the total weighted tardiness $\sum_{j=1}^{n} w_j T_j$ are NP-hard problems.

As for the bounded model, Lee et al.[86] and Uzsoy [87] present polynomial algorithms to minimize the number of tardy jobs $\sum_{j=1}^{n} U_j$ and the maximum lateness $L_{\max}$ under specific assumptions regarding the relationship between processing times, job release dates, and due dates. Minimizing the total completion time $\sum_{j=1}^{n} C_j$ is acknowledged as the most challenging problem in the bounded setting. Chandru et al.[88,89] propose heuristics, a branch-and-bound algorithm, and an $O(m^3 b^{m+1})$ time dynamic programming algorithm for the case of m different job processing times ($m \leq n$). Hochbaum and Landy[90] introduce a more efficient algorithm with a time complexity of $O(m^{2^3 m})$.

It is proven that minimizing the total completion time $\sum_{j=1}^{n} C_j$ is solvable in polynomial time for a fixed b, where $b > 1$, by devising an $O(nb^{(b-1)})$ time dynamic programming algorithm for its solution. The special case where $b = 1$ corresponds to a classical scheduling problem solvable in $O(n \log n)$ time. For the scenario with m different processing times, a more efficient dynamic programming algorithm than that of Hochbaum and Landy is presented, requiring $O(b^2 m^2 2^m)$ time. Moreover, we demonstrate that alternative criteria lead to problems that are NP-hard. However, there are two noteworthy exceptions: the minimization of makespan $C_{\max}$ can be solved in time $\min\{O(n \log n), O\left(\frac{n^2}{b}\right)\}$, while minimizing the total completion time $\sum_{j=1}^{n} C_j$ for arbitrary b and minimizing the total weighted completion time $\sum_{j=1}^{n} w_j C_j$ for both fixed and arbitrary b remain open problems. In conclusion, we present a polynomial algorithm for minimizing any regular cost function, specifically designed for the case where the number of batches is fixed. Table 4.1 provides a summary of our key findings.

Table 4.1: Overview of time complexities for problems with equal release dates

Obj. Function	Unbounded ($b \geq n$)	Bounded ($b = 1$)	Bounded ($b \geq 2$)
$f_{\max}$	Polynomial	$O(n^2)$	Unary NP-hard
$C_{\max}$	$O(n)$	$O(n)$	$\min\{O(n \log n), O\left(\frac{n^2}{b}\right)\}$
$L_{\max}$	$O(n^2)$	$O(n \log n)$	Unary NP-hard
$\sum_{j=1}^{n} f_j$	$O(n^2 P)$	Unary NP-hard	Unary NP-hard
$\sum_{j=1}^{n} C_j$	$O(n \log n)$	$O(n \log n)$	$O(n^{b(b-1)})$
$\sum_{j=1}^{n} w_j C_j$	$O(n^3)$	$O(n \log n)$	Open
$\sum_{j=1}^{n} U_j$	$O(n^3)$	$O(n \log n)$	Unary NP-hard
$\sum_{j=1}^{n} w_j U_j$	O($n^2 P$)	O(nP)	Unary NP-hard
$\sum_{j=1}^{n} T_j$	$O(n^2 P)$	O($n^4 P$)	Unary NP-hard
$\sum_{j=1}^{n} w_j T_j$	$O(n^2 P)$	Unary NP-hard	Unary NP-hard

4.1 The unbounded model

In this section, we make the assumption that the batching machine has a capacity greater than or equal to the total number of jobs, denoted by $b \geq n$. This implies that the batching machine is capable of simultaneously processing any number of jobs. It's important to note that minimizing the makespan becomes a straightforward task in this scenario, as it can be achieved by placing all jobs in a single batch (B_1). The minimum makespan, denoted as $C_{\max}$, is then determined by the maximum processing time among the jobs, represented as $p(B_1) = \max_{1 \leq j \leq n}\{p_j\}$.

For the remainder of this section, we consistently assume that the jobs have been re-ordered based on the shortest processing time (SPT) rule, resulting in a sequence where $p_1 \leq \ldots \leq p_n$.

4.1.1 Minimizing a regular minsum function

In this section, we formalize the previously outlined generic forward dynamic programming algorithm with batch enumeration. This algorithm is designed for the task of minimizing any arbitrary regular minsum objective function

$$\sum_{j=1}^{n} f_j.$$

We demonstrate that the solution to this problem can be achieved in $O(n^2 P)$ time and $O(nP)$ space, where $P = \sum_{j=1}^{n} p_j$. Let $F_j(t)$ represent the minimum objective value for SPT-batch schedules that include jobs $J_1, \ldots, J_j$, given the condition that the last batch completes at time t.Considering $F_j(t)$ and any corresponding

SPT-batch schedule, batch $\{J_{i+1}, \ldots, J_j\}$ (where $0 \leq i \leq j$) appears in the last position.

Now, we introduce our dynamic programming recursion. The initialization is defined as follows:

$$F_0(t) = \begin{cases} 0 & \text{if } t = 0 \\ \infty & \text{otherwise} \end{cases}$$

For $j = 1, \ldots, n$ and $t = p_j, \ldots, \sum_{k=1}^{j} p_k$, the recursion is given by:

$$F_j(t) = \min_{0 \leq i \leq j-1} \left\{ F_i(t - p_j) + \sum_{k=i+1}^{j} f_k(t) \right\}$$

The optimal solution value is equal to $\min_{p_n \leq t \leq P}\{F_n(t)\}$, and the corresponding optimal schedule can be determined through backtracking. To enhance the algorithm's efficiency, the partial sums $\sum_{k=1}^{j} f_k(t)$ are precomputed and stored for $j = 1, \ldots, n$ and $t = p_j, \ldots, \sum_{k=1}^{j} p_k$ during a preprocessing step that requires $O(nP)$ time. Subsequently, each application of the recursion equation necessitates $O(n)$ time. Therefore, the dynamic algorithm requires $O(n^2P)$ time and $O(nP)$ space.

Woeginger [91] establishes that dynamic programming algorithms with a specific structure inherently lead to a fully polynomial approximation scheme. It is worth noting that our dynamic programming algorithm can be easily transformed into a form that possesses this structure.

4.1.2 Minimizing the number of tardy jobs

In this section, we introduce a dynamic programming algorithm with a time complexity of $O(n^3)$ for the task of minimizing the number of tardy jobs. This algorithm departs from the generic pseudopolynomial procedure in two significant aspects. Firstly, the objective value serves as a state variable, and the makespan represents the value of a state. This substitution alone leads to an algorithm with a time complexity of $O(n^4)$. Secondly, to achieve an $O(n^3)$ time complexity, we construct the schedule by adding individual jobs instead of entire batches, and we fix the last job to be scheduled in the current batch.

We define a schedule for jobs $J_1, \ldots, J_j$ to be in a state (j, u, k), where $u \leq j \leq k$. This state indicates that the schedule contains exactly u tardy jobs, and the last batch is to be expanded by including jobs $J_{j+1}, \ldots, J_k$, but no others. Thus, the goal is to create a schedule in which jobs $J_j, \ldots, J_k$ are contained in the same batch, and this batch has a processing time p_k. Let $F_j(u, k)$ be the minimum makespan for SPT-batch schedules in the state (j, u, k).

A schedule in state (j, u, k) with value $F_j(u, k)$ is created by making one of the following decisions in a previous state:

1. Add job J_j in a way that it does not initiate the last batch. The last batch, to which J_j is incorporated, includes job J_{j-1} and has a processing time p_k. This processing time p_k contributes to the makespan of the preceding state, which is $F_{j-1}(u,k)$ or $F_{j-1}(u-1,k)$ based on whether J_j is timely or tardy. If $F_{j-1}(u,k) \leq d_j$, then we consider $(j-1,u,k)$ as a prior state with J_j scheduled to be timely; if $F_{j-1}(u-1,k) > d_j$, then we regard $(j-1,u-1,k)$ as a previous state with J_j scheduled to be tardy.

2. Incorporate job J_j in a way that initiates the last batch. The preceding batch concludes with job J_{j-1}, and the processing time of the new batch is p_k. After factoring in the contribution from the previous state, the makespan is determined as $F_{j-1}(u,j-1)+p_k$ or $F_{j-1}(u-1,j-1)+p_k$, contingent on whether J_j is timely or tardy. If $F_{j-1}(u,j-1)+p_k \leq d_j$, then we regard $(j-1,u,j-1)$ as a prior state with J_j scheduled to be on time; if $F_{j-1}(u-1;j-1)+p_k > d_j$, then we consider $(j-1,u-1,j-1)$ as a previous state with J_j scheduled to be tardy.

We are now prepared to present the dynamic programming recursion. The initial conditions are defined as follows:

$$F_0(u,k) = \begin{cases} 0 & \text{if } u = 0 \text{ and } k = 0 \\ \infty & \text{otherwise} \end{cases}$$

The recursion is applied for $j = 1,\ldots,n$, $u = 0,\ldots,j$, and $k = j,\ldots,n$ as follows:

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

The minimum number of tardy jobs is then determined by the smallest value of u for which $F_n(u,n) < \infty$, and the corresponding optimal schedule can be obtained through backtracking. It is important to note that the algorithm has a time complexity of $O(n^3)$ and requires $O(n^3)$ space.

4.1.3 Minimizing total weighted completion time

In this section, we introduce a dynamic programming algorithm with a time complexity of $O(n \log n)$ for minimizing the total weighted completion time $\sum_{j=1}^{n} w_j C_j$.

Given the additive minsum objective function, we leverage the generic backward dynamic programming algorithm.

Let F_j represent the minimum total weighted completion time for SPT-batch schedules that include the last $n - j + 1$ jobs $J_j, \ldots, J_n$. The processing of the first batch in the schedule commences at time zero. Additionally, when a new batch is added to the beginning of the schedule, there is a corresponding delay in the processing of all batches. Suppose a batch $\{J_j, \ldots, J_{k-1}\}$, with a processing time of p_{k-1}, is inserted at the start of a schedule for jobs $J_k, \ldots, J_n$. The total weighted completion time of jobs $J_k, \ldots, J_n$ increases by $p_{k-1} \sum_{i=j}^{n} w_i$, while the total weighted completion time for jobs $J_j, \ldots, J_{k-1}$ is $p_{k-1} \sum_{i=j}^{k-1} w_i$. Thus, the overall increase in total weighted completion time is $p_{k-1} \sum_{i=j}^{n} w_i$.

The dynamic programming recursion is initiated with $F_{n+1} = 0$, and for $j = n, n-1, \ldots, 1$, the recursion is defined as:

$$F_j = \min_{j \le k \le n+1} \left(F_k + p_{k-1} \sum_{i=j}^{n} w_i \right)$$

The optimal solution value is then equal to F_1, and the corresponding optimal schedule is determined through backtracking. In the standard implementation, the algorithm requires $O(n^2)$ time and $O(n)$ space, provided we compute and store the values $\sum_{i=j}^{n} w_i$ for $j = 1, \ldots, n$ in a preprocessing step. However, leveraging the structural characteristics of our dynamic program, which permit the application of geometric techniques, the time complexity can be reduced to $O(n \log n)$.

4.1.4 Minimizing maximum lateness and maximum cost

In this section, we introduce a dynamic programming algorithm with a time complexity of $O(n^2)$ to minimize the maximum lateness $\mathrm{L}_{\max}$. This algorithm functions as a subroutine for a polynomial algorithm aimed at minimizing the maximum cost $f_{\max}$. Since $\mathrm{L}_{\max}$ is characterized as an incremental minmax objective function, we utilize a backward recursion of the type .

Let F_j denote the minimum value of the maximum lateness for SPT-batch schedules that include the last $n - j + 1$ jobs $J_j, \ldots, J_n$, with processing starting at time zero. If a batch $\{J_j, \ldots, J_{k-1}\}$ with processing time p_{k-1} is inserted at the beginning of a schedule for jobs $J_k, \ldots, J_n$, the maximum lateness of jobs $J_k, \ldots, J_n$ increases by p_{k-1}, while the maximum lateness for jobs $J_j, \ldots, J_{k-1}$ is $\max\_{j<i\le k-1}(p\_k - 1 - d\_i)$.We are prepared to present the dynamic programming recursion. We initialize with $F_{n+1} = -\infty$, and the recursion for $j = n, n-1, \ldots, 1$ is defined as:

$$F_j = \min_{j \le k \le n+1} \max\{F_k + p_{k-1}, \max_{j < i \le k-1} (p_{k-1} - d_i)\}$$

The optimal solution value is then F_1, and the corresponding optimal schedule is determined through backtracking. It's important to note that the algorithm has a time complexity of $O(n^2)$ and requires $O(n)$ space.

We will now demonstrate how to devise a polynomial algorithm for minimizing $f_{\max}$ using the $O(n^2)$ algorithm for minimizing $L_{\max}$ as a subroutine. The problem of minimizing $f_{\max}$ can be seen as a finite series of decision problems of the form 'is $f_{\max} \leq k$?', where k is iteratively adjusted using binary search over an appropriate interval. Thus, if the decision problem is solvable in polynomial time, then minimizing $f_{\max}$ is solvable in polynomial time, provided the optimal solution value is an integer with a logarithm polynomially bounded in the input size. We make the assumption that this is the case. This assumption is not overly restrictive, as it holds when f_j has the form $f_j = (\beta_j)^{\alpha_j}$, where $\beta_j \in \{w_jC_j, w_jT_j, w_jU_j\}$, and α_j is a polynomial in n. The question 'is $f_{\max} \leq k$?' can be answered in polynomial time as follows. Notice that the given upper bound, denoted as k, establishes a deadline $\overline{d}_j$ on the completion time of each job J_j, where j takes values from 1 to n. Each deadline can be determined efficiently in $O(\log P)$ time through binary search over the $P + 1$ potential completion times. Once the deadlines are established, the algorithm for minimizing $L_{\max}$ can be applied to ascertain whether there exists a solution in which each job is completed before its respective deadline. In this context, the deadlines are treated as due dates: if $L_{\max} \leq 0$, then a schedule exists where no deadlines are violated; otherwise, there is no such schedule. Consequently, the question 'is $f_{\max} \leq k$?' can be answered in $O(n^2 + n \log P)$ time, which is polynomial. Therefore, the problem of minimizing $f_{\max}$ can be resolved within polynomial time.

4.2 The bounded model

In this section, we explore problems where the machine constraint, denoted as b, is significantly small, specifically assuming $1 \leq b \leq n$. These constrained problems are deemed at least as challenging as their conventional counterparts, as for the particular case $b = 1$, the machine's capacity is limited to handling only one job at a time. Moreover, these constrained problems inherently pose greater difficulty compared to their unconstrained counterparts. The primary reason for this increased complexity is that the search for an optimal schedule can no longer be confined to SPT-batch schedules, with one exception being the case of minimizing makespan, where an optimal SPT-batch schedule still exists.

For the bounded problem of minimizing makespan, we assume that n is an integer multiple of b, denoted as $n = br$. This assumption is justified by the insight that introducing dummy jobs with zero processing time does not affect the minimum makespan. The approach to solving the problem involves assigning the b

jobs with the smallest processing times to B_1, the next b jobs with the next smallest processing times to B_2, and so forth, until the b jobs with the largest processing times are assigned to B_r. Consequently, the problem can be solved in $O(n \log n)$ time if the jobs are initially ordered using the SPT rule. Alternatively, the problem can be solved in $O(rn)$ time by leveraging linear-time median-finding techniques. Specifically, $B_1 \cup \ldots \cup B_r$ can be determined by identifying some job J_j in $O(n)$ time such that $|\{i \mid p_i \leq p_j\}| \leq b$ and $|\{i \mid p_i > p_j\}| \leq b(r-l)$, and subsequently introducing a subset of $\{i \mid p_i = p_j\}$ into the first set in such a way that its cardinality is exactly b. With this approach, the problem is resolved in $O\left(\frac{n^2}{b}\right)$ time. In Subsection 5.2.1, we explore the task of minimizing the total completion time. We illustrate that the problem can be efficiently solved using dynamic programming in $O(n^{b(b-1)})$ time, where $b \geq 1$. Furthermore, for scenarios involving m distinct processing times, we introduce an algorithm with a time complexity of $O(b^2m^2 2^m)$. Lastly, Subsection 5.2.2 focuses on the special scenario where the number of batches to be utilized is fixed. We reveal that bounded problems of this nature can be resolved in $O(n^{r+3})$ time, where r represents the given number of batches.

4.2.1 Minimizing total completion time

Chandru et al. [88] introduce the constrained problem of minimizing total completion time, providing a branch-and-bound algorithm along with some heuristics. It can be demonstrated that the general problem can be solved within $O(n^{b(b-1)})$ time and $O(n^{b(b-1)})$ space, considering the case where b is greater than or equal to 1. For scenarios involving m distinct job types, with m less than or equal to n, Chandru et al. [89] propose a dynamic programming algorithm with a time complexity of $O(m^3 b^{m+1})$. An even more efficient algorithm is presented by Hochbaum and Landy [90], requiring $O(m^2 3^m)$ time. Throughout this section, we assume a re-indexing of jobs based on the SPT rule, such that p1pn. We now present two results from Chandru et al. [7]. The first result establishes the existence of an optimal schedule where each batch contains jobs with consecutive indices.

Lemma 2, as stated by Chandru et al.[88], asserts the existence of an optimal schedule $(B_1, \ldots, B_r)$ under the SPT indexing, where each batch B_l is represented as $\{J_{il}, \ldots, J_{jl}\}$ for indices $1 \leq i_l \leq j_l \leq n$ and l ranging from 1 to r.

Chandru et al.[88] provide a second result that extends the classical Shortest Processing Time (SWPT) rule, traditionally used for sequencing jobs on a single machine, to the sequencing of batches.

Lemma 3 (Chandru et al.[88]): For given batches B_1,...,B_r, an optimal sequence is (B_1,....B_r) if and only if

$$p(B_1)/|B_1| \leq \ldots \leq p(B_r)/|B_r| \quad (1)$$

Here, a batch is defined as "full" if it contains exactly b jobs, and otherwise, it is termed "non-full." Additionally, a batch B_l is considered deferred concerning another batch B_q if B_l is scheduled after B_q, and the processing time of B_l is less than the processing time of B_q. A further result presented by Hochbaum and Landy[9] addresses deferred batches.

Lemma 4 (Hochbaum and Landy [90]): In any optimal schedule, there is no batch that is deferred with respect to a non-full bat

4.2.2 Restricted number of batches

In this section, we explore the constrained scenario of minimizing any regular objective function, where the schedule is limited to include at most r batches. We demonstrate that addressing this problem is achievable in $O(n^{r+3})$ time, constituting a polynomial time complexity when r is held constant.

It's important to note that when the longest job within each batch is specified along with the sequencing order of the r batches, the computation of processing times $p(B_1), \ldots, p(B_r)$ and completion times $C(B_1), \ldots, C(B_r)$ becomes straightforward. The problem then transforms into assigning the remaining jobs to the r batches, ensuring that no batch exceeds b jobs, and each job is assigned to a batch without a designated longest job of smaller size. If a job J_j has a deadline $\bar{d}_j$, it is imperative to guarantee $C_j \leq \bar{d}_j$.

Given these considerations, the cost c_{ij} associated with assigning any of the $n-r$ remaining jobs J_j $(j = 1, \ldots, n)$ to batch B_i $(i = 1, \ldots, r)$ is defined as follows:

$$c_{ij} = \begin{cases} \infty & \text{if } p(B_i) \leq p_j \text{ or } C(B_i) \geq \bar{d}_j \\ f_j(C(B_i)) & \text{otherwise} \end{cases}$$

Consequently, the task of minimizing a minsum cost function for given batch processing times and a predetermined processing order of the r batches reduces to a bipartite weighted matching problem, a challenge solvable in $O(n^3)$ time [92].

4.3 Single batching machine

Brucker et al. [5] investigate the intricacies of scheduling a single batching machine, exploring scenarios with unrestricted batch sizes as well as situations where batches are limited to containing at most b jobs. In the context of a batch B, defined as a set of jobs, its processing time is determined by the maximum processing time among the jobs within the batch. The completion time for each job in B corresponds to the moment when the processing of batch B is completed. Assuming jobs are arranged according to the SPT (Shortest Processing Time) rule, denoted as $p_1 \leq \ldots \leq p_n$, an SPT-batch schedule allows for grouping adjacent jobs in the sequence $1, \ldots, n$

into batches. As an illustration, consider a possible batch schedule for a 10-job problem represented by the sequence ({1,2,3},{4},{5,6,7,8},{9,10}). For problems $\tilde{1} \parallel \psi_3$, where no restriction is imposed on the batch size and ψ_3 is any objective function introduced in Section 2, Brucker et al. [5] employ a straightforward argument to demonstrate the existence of an optimal solution that conforms to an SPT-batch schedule. Dynamic programming algorithms proposed by Brucker et al. [5] are designed for various problems without imposing restrictions on the batch size. These algorithms, along with other pertinent findings, are discussed in the subsequent sections.

4.3.1 Total weighted completion time

For the problem $\tilde{1} \mid\mid \sum_j w_j \cdot C_j$, Brucker et al. [5] introduced a backward dynamic programming algorithm that incorporates batch insertion. The variable Gj represents the minimum total weighted completion time for SPT-batch schedules, including jobs from j to n, with the processing of the first batch in the schedule commencing at time zero. The initialization is $G_{n+1} = 0$ and the recursion for $j = n, n-1, \ldots, 1$ is $G_j := \min_{k=j+1,\ldots,n+1} \left(G_k + p_{k-1} \sum_{h=j}^{n} w_h \right)$.The minimization process involves selecting a batch $j, \ldots, k-1$, with a processing time of p_{k-1}, to be inserted at the beginning of a pre-existing schedule consisting of jobs $k, \ldots, n$. The optimal solution value is then represented by G_1. Although the algorithm typically requires $O(n^2)$ time under a standard implementation, its inherent structure allows the application of geometric techniques introduced by van Hoesel et al. [75], leading to a reduced time complexity of $O(n \log n)$. In scenarios where batch sizes are constrained by a limit b, Brucker et al.[5] present a dynamic programming algorithm with a time complexity of $O(n^{b(b-1)})$. However, for arbitrary b, the computational complexity of $\tilde{1}|b| \sum_j C_j$ remains an open question, as does the complexity of $\tilde{1}|b| \sum_j w_j \cdot C_j$ for both fixed and arbitrary b. When there are q distinct processing times for problem $\tilde{1}|b| \sum_j C_j$, dynamic programming algorithms have been proposed by Chandru et al.[9], Hochbaum and Landy[44], and Brucker et al.[5]. Among these, the most efficient is the algorithm by Brucker et al., with a time complexity of $O(b^2 q^2 2^q)$.

4.3.2 Maximum lateness

For the problem $\tilde{1} \mid\mid L_{max}$, Brucker et al. [5] introduce a backward dynamic programming algorithm with batch insertion, akin to the one discussed in the preceding subsection and the algorithm designed for the batch availability problem $1|s_f, F = 1| L_{max}$, as explained in Section 4.5.1. The variable G_j denotes the minimum value of the maximum lateness for SPT-batch schedules, encompassing jobs from j to n, with processing commencing at time zero. The initialization is

$G_{n+1} \coloneqq -\infty$ and the recursion for $j = n, n-1, \ldots, 1$ is

$$G_j \coloneqq \min_{k=j+1,\ldots,n+1} \{\max\{G_k + p_{k-1}, \max_{h=j,\ldots,k-1}(p_{k-1} - d_h)\}.\}$$

The optimal solution value is represented by G_1, and the algorithm has a time complexity of $O(n^2)$. For a limited batch size b, Brucker et al. [5] demonstrate the unary NP-hardness of the problem $\tilde{1}|b|L_{max}$. Lee et al. [54] investigate the complexity of various specific instances of the problem $\tilde{1}|b, r_j|L_{max}$, introducing assumptions about release dates, processing times, and due dates.

4.3.3 Weighted number of late jobs

For the problem $\tilde{1}\,||\ \sum_j U_j$, Brucker et al. [5] introduce a forward dynamic programming algorithm that involves appending jobs. The variable $G_j(u,k)$ represents the minimum makespan for SPT-batch schedules that include jobs from 1 to j. Here, u denotes the number of late jobs in the schedule, and the last batch is allocated a processing time p_k. This is because it is designated to accommodate jobs from $j+1$ to k, excluding job k itself, where $j < k$. The initialization for $k \coloneqq 0, 1, \ldots, n$ is $G_0(0,k) = \begin{cases} 0 & \text{if } k = 0 \\ \infty & \text{otherwise} \end{cases}$ and the recursion for $j \in 1, \ldots, n$, $u \in 0, 1, \ldots, j$, and $k \in j, \ldots, n$ 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 smallest number of late jobs is determined by finding the minimum value of u for which $G_n(u,n) < \infty$. The algorithm's time complexity is $O(n^3)$. When applied to the problem $\tilde{1}\,||\ \sum_j w_j \cdot U_j$, the dynamic programming algorithm mentioned above can be adapted to minimize the weighted number of late jobs within $O(n^2 W)$ time, where $W = \sum_{j=1}^{n} w_j$. Another pseudopolynomial dynamic programming algorithm, discussed in the subsequent subsection, provides a solution in $O(n^2 P)$ time, where $P = \sum_{j=1}^{n} p_j$. Brucker et al. [5] demonstrate that this problem is binary NP-hard. The problem $\tilde{1}|b|\sum_j U_j$ is demonstrated to be unary NP-hard, drawing from the analogous outcome established for problem $\tilde{1}|b|L_{max}$. Lee et al. [54] extend their analysis to explore specific instances of problem $\tilde{1}|b, r_j|\sum_j U_j$ under diverse assumptions regarding release dates, processing times, and other factors.

4.3.4 Total weighted tardiness

For the problem $\tilde{1} \parallel \sum_j w_j \cdot T_j$, Brucker et al. [5] introduce a forward dynamic programming algorithm that involves adding batches. The variable $G_j(t)$ represents the minimum total weighted tardiness for SPT-batch schedules that include jobs from 1 to 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 $j = 1, \ldots, n$ and $t = p_j, \ldots, \sum_{k=1}^{j} p_k$ is

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

The minimum value of the optimal solution is determined by considering the set of completion times t ranging from $\sum_{j=1}^{n} p_j$ to the total processing time $P = \sum_{j=1}^{n} p_j$. The algorithm's time complexity is $O(n^2 P)$. Brucker et al. [5] establish that the problem $\tilde{1}||\sum_j w_j \cdot T_j$ is binary NP-hard. Additionally, the problem $\tilde{1}|b|\sum_j T_j$ is shown to be unary NP-hard, building on the analogous outcome for problem $\tilde{1}|b|L_{max}$.

4.4 Parallel batching machines

In scenarios with m identical parallel batching machines and an arbitrary regular objective function, where there are no limitations on the batch size for any machine, Brucker et al. [5] note that an optimal solution can be formed by employing an SPT-batch schedule on each machine. Consequently, for a fixed number of identical parallel machines, the dynamic programming algorithms developed for single-machine problems extend to provide pseudopolynomial algorithms.

4.4.1 Maximum lateness

In this portion, we introduce two algorithms designed to discover optimal schedules for the $P|batch|\ L_{max}$ and $P|batch|\ \sum_j U_j$ problems. Hochbaum and Landy (1994) examined the single-machine problem to ascertain the existence of a viable sequence within deadline constraints. The subsequent characteristic was outlined in their publication.

Lemma 1 For the $P|batch|\ L_{max}$ (as well as $P|batch|\ \sum_j U_j$) problem, there is an optimal schedule in which the jobs are arranged in EDD order on each machine.

Proof The validity readily follows from job interchange arguments.

In the subsequent discussion, we initially formulate a dynamic program to construct an optimal schedule for the maximum lateness problem. Leveraging the property articulated in Lemma 1, an initial sequence is established by arranging jobs in non-decreasing order based on their due dates. Consequently, job re-indexing is performed to satisfy the condition $i \leq j$ if and only if $d_i \leq d_j$. The function $F(j, t_1, t_2, .., t_m)$ is then defined as the optimal maximum lateness given that the first j jobs have been scheduled, and the completion time of the last batch on machine l is t_l, $1 \leq l \leq m$. Assuming the first $j-1$ jobs have been properly scheduled, the focus shifts to the processing of job j. This involves the choice between creating a new batch exclusively for job j or appending job j to the last batch on some machine. In the latter scenario, the lateness of each job in the batch to which job j is appended might be affected. Consequently, a reevaluation of the maximum lateness for the first $j-1$ jobs is necessary. With these considerations, the function F can be defined in a backward recursive form, as elucidated below.

Algorithm Maximum Lateness

Initial conditions:

$$F(j, t_1, t_2, ..., t_m) = \begin{cases} 0 & \text{if } j = t_1 = t_2 = \ldots = t_m = 0, \\ \infty & \text{otherwise.} \end{cases}$$

Recursive formula: For $0 \leq j \leq n$, $0 \leq t_l \leq ns + \sum_i p_i$ for $1 \leq l \leq m$,

$$F(j, t_1, t_2, ..., t_m) == \min_{l=1,....,m} \{min \begin{cases} max\{F(j-1, t_1, t_2, ..., t_l - s - p_j, ..., t_m), \\ t_l - d_j\} \\ max\{t_l - d_l(j-1, t_1, t_2, ..., t_l - p_j, ..., t_m), \\ F(j-1, t_1, t_2, ..., t_l - p_j, ...t_m)\} \end{cases}$$

In this context, the term $d_l(j-1, t_1, ..., t_l - p_j, ..., t_m)$ represents a variable that holds the due date of the initial job within the last batch on machine l, as calculated by the function $F(j-1, t_1, ..., t_l - p_j, ..., t_m)$. Once the value of $F(j, t_1, t_2, ..., t_m)$ is established, the due date $d_l(j, t_1, t_2, ..., t_m)$ for each machine l is subsequently determined based on the specific outcomes of the solution scenarios.
Goal:

Minimize $\{F(n, t_1, t_2, ..., t_m) | 0 <= t_l <= ns + \sum_{i=1}^{n} p_i, 1 <= l <= m\}$

The recursive formula above describes two scenarios: one where job j independently initiates a new batch on machine l, and another where job j is added to the last batch on machine l. The function F encompasses $O(n(ns + \sum_i p_i)^m)$ possible states, each requiring $O(m)$ time for optimal decision-making. Consequently, the overall computation time of this algorithm is dominated by $O(mn(ns + \sum_i p_i)^m)$. This computational complexity is inherently exponential in relation to the input length and becomes pseudo-polynomial when the number of machines (m) is fixed. In an optimal schedule, the completion time of the last job on any machine cannot exceed $L_{\max} + d_{\max}$. Utilizing this observation, the recursive formula's inequality restricting the range of t_l can be refined to $0 \leq t_l \leq u_1$, where u_1 is determined by $\min(ns + \sum_i p_i, L_{\max} + d_{\max})$. Although the exact value of the objective $L_{\max}$ is unknown initially, heuristic procedures can be designed and invoked to approximate values for this refinement. If L_H represents an approximate solution obtained through some heuristic, then $L_{\max}$ is approximated as L_H. Consequently, the time complexity of Algorithm Maximum Lateness is adjusted to $O(mn(\min\{ns + \sum_i p_i, L_H + d_{\max}\})^m)$ without affecting the existence of optimal solutions.

4.4.2 Number of tardy jobs

In this section, we move on to examining the objective related to the count of tardy jobs. For clarity in presentation, we will instead focus on maximizing the number of non-tardy jobs. A job is considered non-tardy if the batch it belongs to is fully completed no later than its due date. We define the function $G(j, t_1, t_2, ..., t_m)$ as the maximum number of non-tardy jobs, given that the first j jobs have been considered, and the completion time of machine l is $t_l, 1 \leq l \leq m$. When considering job j, three possible situations may arise: job j is discarded and treated as tardy, job j is scheduled early or on time and independently forms a batch on some machine, or job j is added to an existing batch without making other jobs in the same batch tardy. With these observations, we can formulate the following recursive program for constructing a schedule with the maximum number of non-tardy jobs.

Algorithm Number of Tardy Jobs

Initial conditions:

$$G(j, t_1, t_2, ..., t_m) = \begin{cases} 0 & \text{if } j = t_1 = t_2 = \ldots = t_m = 0; \\ -\infty & \text{otherwise.} \end{cases}$$

Recursive formula: For $0 \leq j \leq n$, $0 \leq t_l \leq ns + \sum_i p_i$ for $1 \leq l \leq m$,

$$G(j,t_1,t_2,..,t_m) = \max \begin{cases} G(j-1,t_1,t_2,..,t_m) \\ max_{l=1,2,..,m}\{G(j-1,t_1,t_2,..,t_l-s-p_j,..,t_m)+1\}, \\ max_{l=1,2,..m}\{G'_l(j,t_1,t_2,..,t_m)\} \end{cases}$$

The function $G'_l(j,t_1,t_2,...,t_m)$ is defined as the count of non-tardy jobs when the first j jobs are under consideration, and job j belongs to the last batch on machine l. Its value is determined by the following rule: if $d_l(j-1,t_1,...,t_l-p_j,...,t_m)$, the due date of the first job of the last batch on machine l in the solution $G(j-1,t_1,y,t_l-p_j,...,t_m)$, is no greater than t_l, then $G'_l(j,t_1,t_2,....,t_m) = G(j-1,t_1,....,t_l-p_j,y,t_m)+1$; otherwise, $G'_l(j,t_1,t_2,...,t_m) = N$. Once $G(j,t_1,t_2,...,t_m)$ is determined, the variable $d_l(j,t_1,t_2,...,t_m)$ for each machine l will be defined accordingly based on the scenarios of the solution.

Goal:: Maximize $\{G(n,t_1,t_2,...,t_m).: 0 \le t_l \le ns + \sum_{i=1}^{n} p_i, 1 \le l \le m\}$.

With an analysis similar to that employed in Algorithm Maximum Lateness, we can readily conclude that the Algorithm Number of Tardy Jobs requires $O(mn(ns+\sum_{i=1}^{n} p_i)^m)$ time to identify an optimal set of non-tardy jobs. This is because the completion time of any non-tardy job does not exceed $d_{\max}$. To further streamline the function G, certain states can be eliminated by imposing an upper bound on t_l, $1 \le l \le m$, defined as $\min\{ns + \sum_{i=1}^{n} p_i, d_{\max}\}$. Consequently, the time complexity of the dynamic program is updated to $O(mn(\min\{ns + \sum_{i=1}^{n} p_i, d_{\max}\})^m)$.

4.4.3 Heuristic algorithms

The previous section introduced two dynamic programming algorithms designed to generate optimal schedules for the two given problems. However, these algorithms are more theoretically intriguing than practically significant due to their substantial memory and computation requirements. Additionally, addressing the variable dimensions of the recursive functions might necessitate advanced programming skills. Another commonly employed approach in combinatorial optimization involves implicit enumeration, particularly through the development of branch-and-bound methods. Given that the problems $P|batch|\ L_{max}$ and $P|batch|\ \sum_j U_j$ involve simultaneous considerations of dispatching and grouping decisions, the solution space experiences explosive growth with increasing problem sizes. Consequently, formulating a successful branch-and-bound algorithm for a batch scheduling problem alone poses a considerable challenge. In this section, we opt to develop heuristics as an alternative, aiming to produce approximate solutions to the $P|batch|\ L_{max}$ and

$P|batch|\ \sum_j U_j$ problems within a reasonable time frame. It is noteworthy that the single-machine cases $P|batch|\ L_{max}$ and $P|batch|\ \sum_j U_j$ can be optimally solved in polynomial time using dynamic programming algorithms developed by Brucker and Kovalyov (1996) and Webster and Baker (1995), respectively. The algorithms presented in this section will leverage these known algorithms for independently composing batches of jobs on each machine. Initially, our focus is on minimizing the maximum lateness, and considering the problem structures, our proposed algorithms involve three key phases: sequencing (establishing a job sequence), dispatching (assigning jobs to machines), and batching (grouping jobs on each machine into batches). In the *"sequencing"* phase, we determine a suitable job sequence based on the problem at hand. Moving on to the dispatching phase, we explore various methods for assigning jobs to machines. Under the *"Smallest Completion Time First"* (SCF) approach, unscheduled jobs are dispatched in ascending order of their index to the machine with the smallest completion time, continuing until all jobs are scheduled. Alternatively, the *"Smallest Lateness First"* (SLF) method involves dispatching the first unscheduled job based on the lateness it will introduce to the schedule. The details are outlined as follows:

Dispatching rule: SLF

Step 1: Set the completion time of each machine as zero.
Step 2: Let job i be the first unscheduled job.

2.1. For $k = 1$ to m:

- LC_k = the maximum lateness on machine k after assigning job i into the last batch of machine k;
- LN_k = the maximum lateness on machine k after creating a new batch for job i on machine k;

2.2. If $\min_{1\leq k\leq m} LC_k \leq 0$ or $\min_{1\leq k\leq m} LC_k \leq \min_{1\leq k\leq m} LN_k$, then dispatch job i into the last batch of the machine with $\min_{1\leq k\leq m} LC_k$; else dispatch job i to the machine with $\min_{1\leq k\leq m} LN_k$ and form a new batch on this machine.

Step 3: Update the completion time according to the outcome of Step 2; go to Step 2 until all jobs are scheduled.

Initially, our attention is focused on minimizing the maximum lateness, and given the problem structures, our proposed algorithms involve three main phases: sequencing (determining the job sequence), dispatching (assigning jobs to machines), and batching (grouping jobs on each machine into batches). In the sequencing

phase, Lemma 1 suggests that the Earliest Due Date (EDD) sequence of jobs is a promising starting point. Another consideration for job priority is based on slack times, where jobs with tight temporal cushions receive higher priority. Thus, we adopt EDD and Smallest Slack Time (SST) as our sequencing rules.

EDD rule: Re-index the jobs in non-decreasing order of their due dates.

SST rule: Re-index the jobs in non-decreasing order of slack time, $d_i - p_i$.

Moving on to the dispatching phase, we explore two different ways of assigning jobs to machines. Under the "Smallest Completion Time First" (SCF) approach, unscheduled jobs are dispatched to the machine with the smallest completion time. Alternatively, the "Smallest Lateness First" (SLF) method dispatches the first unscheduled job based on the lateness it will introduce. After the sequencing and dispatching phases, we enter the batching phase, where a subset of jobs remains on each machine. For jobs dispatched using the SLF rules, they have been grouped into batches. Leveraging an algorithm by Webster and Baker (1995) designed for the single-machine case $1|\text{batch}|L_{\max}$, we compose optimal solutions for jobs on each machine, enhancing the overall solution quality. It's important to note that this algorithm assumes jobs are arranged in EDD order. Consequently, if the SST rule is applied in the sequencing phase, the jobs on each machine must be rearranged using the EDD rule. Arranging jobs in a specific order takes $O(n \log n)$ time, dispatching jobs onto machines using a min-heap takes $O(nm \log m)$ time, and finding optimal batch compositions for all machines takes $O(mn^2)$ time. Therefore, the time complexities of the heuristic algorithms are dominated by $O(mn \max(n, \log m))$. Regarding the minimization of the number of tardy jobs, Ho and Chang (1995) proposed two heuristic approaches: job-focused and machine-focused. The job-focused heuristic involves dispatching and machine selection, similar to our sequencing and dispatching phases. Additionally, a rejection rule is used to remove the early-scheduled job with the longest processing time. In the machine-focused approach, Moore's algorithm is applied to each machine one by one, treating a tardy job on the current machine as an unscheduled candidate for the next machine. Brucker and Kovalyov (1996) developed an $O(n^3)$ dynamic programming algorithm for the $1|batch|\ \sum_j U_j$ problem. Three heuristic algorithms for $P|batch|\ \sum_j U_j$ are then derived from the aforementioned algorithms.

- **Heuristic H1:** Utilizes the machine-focused approach followed by the activation of Baker and Kovalyov's dynamic programming algorithm for the jobs arranged on each machine.

- **Heuristic H2:** Similar to H1 but adopts a job-focused approach for dispatching jobs onto machines.

- **Heuristic H3:** Modifies the job-focused approach to simultaneously address dispatching and grouping issues, given that applying the dynamic programming algorithm to all machines may be time-consuming for composing approximate solutions.

Heuristic H3
Step 1: Remove all jobs i with $d_i \leq p_i$.
Step 2: Re-index the jobs in Earliest Due Date (EDD) order.
Step 3: For $i = 1$ to n do the following steps:

3.1 Identify the machine, denoted as k, which possesses the greatest completion time while ensuring that incorporating job i into the last batch on this machine does not breach the due date restriction of the initial job in this batch. If no such machine is found, proceed to Step 3.2; otherwise, supplement the existing batch on machine k with job i.

3.2 Identify the machine, denoted as k, which has the largest completion time, making sure that job i can be finished by its due date if a new batch is created for it on this machine. If there is no such machine meeting this condition, proceed to Step 3.3; otherwise, create a new batch for job i on machine k.

3.3 Identify the job with the longest processing time among all the jobs that have been scheduled early, specifically job j on machine k. If the processing time of job i (p_i) is greater than that of job j (p_j), then exclude job i; otherwise, exclude job j and arrange job i as the last job on machine k. Subsequently, apply Hochbaum and Landy's algorithm (Hochbaum and Landy, 1994) to reorganize the batch formation on machine k.

Step 4: Output the schedule.

For each machine, the application of Moore's algorithm takes place sequentially. In this process, a tardy job on the current machine is considered as a candidate job that is unscheduled for the next machine. This iterative procedure continues until all machines or all jobs have been scrutinized. Brucker and Kovalyov (1996) introduced an $O(n^3)$ dynamic programming algorithm to address the $P|batch|\ \sum_j U_j$ problem. Subsequently, three heuristic algorithms for $P|batch|\ \sum_j U_j$ are predominantly derived from the previously mentioned algorithms. In Step 3.3, there is a possibility of replacing job j with job i. To create a new batch configuration, we employ Hochbaum and Landy's algorithm, which, in compliance with the deadline constraints, can rearrange the jobs into batches. Concerning the time complexities of heuristics H1 and H2, it is evident that the $O(n^3)$ dynamic

program developed by Brucker and Kovalyov (1996) plays a dominant role. As a result, the overall time complexity becomes $O(n^4)$. Since Hochbaum and Landy's algorithm requires $O(n)$ time and Steps 3.1 and 3.2 may take $O(m)$ time, and the time complexity of H3 is $O(nmax\{n, m\})$. Another heuristic is used in this research as a dispatching rule to resolve the parallel machine scheduling problem: it is ATC heuristic proposed by Vepsalainen and Morton. At each time point t when a machine becomes available, a single batch is selected from each family, and one batch is chosen and scheduled on the machine from the considered batches. A time window $(t, t + \delta t)$ is defined, and the set of unscheduled jobs from family j with arrival times less than the upper boundary of the time window interval is represented as $M(j, t, \delta t) := \{ij \mid r_{ij} \leq t + \delta t\}$. Then, the set $\tilde{M}$(j,t,δt, thres) := $\{ij \mid ij \in M(j, t, \delta t)$ and pos(ij) $\leq$ thres$\}$ is derived, where "thres" is the maximum number of jobs in $\tilde{M}$(j,t,δt, thres).

The criterion for evaluating jobs (I_{ij}) and the position of job ij with respect to I_{ij} (pos(ij)) are defined, based on the ATC (Batched Apparent Tardiness Cost) index. The ATC index $I_{ij,ATC}$ for job i belonging to family j and calculated at time t is given by:

$$I_{ij,\mathrm{ATC}}(t) = \frac{w_{ij}}{p_j} \exp - \frac{((d_{ij} - p_j + (r_{ij} - t)^+))}{k\bar{p}}$$

In $I_{ij,ATC}$, "k" is a look-ahead parameter, and $p\acute{r}$ is the average processing time of the remaining unscheduled jobs. The jobs in the set $M(j, t, \delta t)$ are sorted in non-increasing order according to $I_{ij,ATC}$, and the first "thres" of them are selected to form the set M $(j; t; Dt;$ thres).

For the evaluation of a particular batch combination, the index $I_{bj}(t)$ is considered, given by:

$$I_{bj}(t) = \sum_{i=1}^{n_{bj}} \frac{w_{ij}}{p_j} \exp - \frac{((d_{ij} - p_j - t + (r_{ij} - t)^+))}{k\bar{p}} \frac{n_{bj}}{B}$$

Here, "n_{bj}" is the number of jobs in the batch, and $r_{bj} := \max_{i \in Bbj}(r_{ij})$ represents the maximum ready time of the jobs in the batch. This rule aims to increase the fullness of the batches by adding $(r_{bj} - t)$ to the slack when jobs for a batch are not available. The BATC-based (Batched Apparent Tardiness Cost) dispatching heuristic follows these steps. It is noted that this rule outperforms other rules suggested in related studies (Mason et al., 2002; Mönch). The BATC-based (Batched Apparent Tardiness Cost) dispatching heuristic can be outlined through the following steps:

1. At time t, determine a window size δt and select at most $thres$ jobs from the set $M(j, t, \delta t)$. Consequently, compute the corresponding I_{ij} indexes for each

family with unscheduled jobs and arrange them in descending order. This process yields sets of jobs $\tilde{M}(j, t, D, thres)$.

2. Choose a machine m with a ready time less than or equal to t.

3. Calculate the BATC index for all batch combinations in each family and then choose the batch b with the highest BATC index.

4. Schedule the selected batch b on the machine m.

5. Update the ready time of machine m to $\max(t, r_{bj}) + p_b$, where p_b is the processing time of the family to which batch b belongs.

6. If all jobs are scheduled, proceed to step 7; otherwise, return to step 2.

7. Terminate the process.

The choice of the look-ahead parameter depends on the closeness of due dates and ready times, with the range of these factors also influencing the decision. The degree of closeness and the overall range should be linked to the makespan of the schedule. To estimate the makespan $C_{\max}$, the batch-machine factor $\mu := \frac{n}{mb_{eff}B}$ is introduced. The batch efficiency factor beff is employed to simulate situations where not all batches are complete, with $b_{eff} = 0.75$ used in our experiments. It's important to note that m represents the average number of batches on a single machine. The average processing time of the considered jobs is denoted by $\bar{p}$, and $\hat{C}_{\max} := \bar{p}\mu$ serves as a rough estimator of $C_{\max}$. The tightness of due dates is defined as $T := 1 - \frac{\bar{d}}{\bar{p}\mu}$, where $\bar{d}$ denotes the mean due date. A due date tightness close to zero indicates that most due dates are clustered around the makespan. If T approaches one, a large portion of due dates is clustered at time $t = 0$. To measure the range of due dates, the quantity $R := \frac{1}{\bar{d}}\sqrt{\frac{12}{(n-1)}\sum_{j=1}^{n}(dj - \bar{d})^2}$ is introduced. R is essentially an estimator of the variance of due dates relative to $\bar{d}$. Utilizing the expression $\bar{d} = (1 - T)\hat{C}_{\max}$, $\bar{d}$ linearly depends on $C_{\max}$.

In the case of uniformly distributed due dates for large sample sizes, the approximation $\frac{1}{n-1}\sum_{j=1}^{n}(dj - \bar{d})^2 \approx \frac{(d_{\max} - d_{\min})^2}{12}$ is valid, where $d_{\max}$ is the maximum due date among the samples, and $d_{\min}$ is the minimum. Therefore, an expression for R is obtained that closely resembles the quantity used in Lee and Pinedo (1997).

For characterizing ready times, a similar approach is adopted. The quantity $\tilde{T} := 1 - \frac{\bar{r}}{\bar{p}\mu}$ is defined as the tightness of ready times, where $\bar{r}$ represents the average ready time. The range of ready times can be estimated by $\tilde{R} := 1 - \frac{1}{\bar{r}}\sqrt{\frac{12}{(n-1)}\sum_{j=1}^{n}(r_j - \bar{r})^2}$.

4.4.4 Machine learning techniques applied to parameter setting in scheduling heuristics

This section covers the exploration of neural networks and inductive decision trees as approaches for parameter estimation. Aytuk et al. (1994) conducted a survey of various machine learning techniques employed in addressing scheduling problems. Jain and Meeran (1998) specifically delve into the application of neural networks in scheduling and explore related literature.

Neural networks

Neural networks are composed of interconnected nodes, connected by directed links, as explained by Mitchell (1997). The transmission of activation from one node to another occurs if there is a direct link between them. Each link is assigned a weight, denoted as $w_{j,i}$, determining the strength and direction of the connection. To compute the output a_i of a node i, it first calculates the weighted sum of its inputs. Subsequently, an activation function g is applied to this sum. The feed-forward networks lack internal states beyond the weights, functioning as a direct transformation of their inputs. Learning in neural networks involves adjusting the weights to minimize error on the training set. Feed-forward networks are often organized into layers, where each layer's nodes receive input exclusively from the preceding layer. In addition to input and output layers, hidden layers are commonly incorporated, expanding the representational capacity of the network. Networks with hidden layers are termed Multi-layer Perceptrons (MLP). Learning in MLPs involves back-propagating errors from the output layer to the input layer. The backpropagation (BP) algorithm, a gradient descent technique, is widely utilized for training MLPs (Mitchell, 1997).

Inductive decision trees

The text discusses inductive decision trees, characterized as straightforward and potent learning algorithms according to Utgoff (1998). These trees receive a situation described by a set of attributes as input and, based on this input, make a decision while providing output values. The structure of a decision tree comprises leaves, representing classes, and decision nodes that outline tests on the attributes of the test case, guiding the branching into sub-trees. The learned decision tree's structure consists of disjoint hypercubes created by partitioning attribute domains into intervals. Commonly, algorithms employed for learning inductive decision trees involve variations of a top-down greedy search algorithm, with the ID3 and C4.5 algorithms being widely used in tree learning (Mitchell1997, Quinlan1993).

Two fundamental approaches exist for determining input attributes. The first method divides the complete dynamic range of inputs and outputs into a predefined

number of equal intervals. The second approach involves dividing only the output into equal intervals, with machine learning determining optimal attribute borders for the inputs. Functional dependencies between input and output variables are described through rules. Fuzzy design principles enhance model performance by creating fuzzy sets for selected ranges and applying fuzzy methods for data processing. Triangular membership functions are initially assigned to input intervals, intersecting at interval borders with a parameter denoted as m equal to 0.5. The edge intervals receive one-sided open membership functions, allowing input values to belong to two intervals with different membership values. Subsequently, input values are transformed into linguistic attributes based on the fixed intervals, resulting in a description of static and/or dynamic process behavior in the form of linguistic expressions.

The ID3 algorithm (Quinlan1993) generates an optimal decision tree from linguistic examples. To further enhance fuzzy models, variable membership functions are expected to be more effective than homogeneous ones. An optimization of fuzzy sets is performed by minimizing the mean square error between the representation given by the decision tree and the measured output. This optimization involves describing the membership functions of input variables using four pivot places, and a standard method for solving the resulting constrained non-linear optimization problem is employed (Otto2002).

4.4.5 Application of the two techniques for setting the look ahead parameter in the BATC-rule

The utilization of two machine learning techniques aims to determine effective look-ahead values k, and the proposed methodology involves two distinct phases.

During Phase I, both the inductive decision tree and the neural network undergo training. The near-optimal value for k is computed for each factor combination. This involves setting k in the form $k = 0.1j$ for $j = 1, \ldots, 30$ and $k = 0.5j + 3.0$ for $j = 1, \ldots, 7$. The scheduling problem is solved for a fixed k using the dispatch scheme from Section 5.2.2. The schedule yielding the minimum total weighted tardiness determines the optimal k value. The mean k-value of the 10 independent test instances for the same parameter combination is calculated, ensuring statistically significant results. Additionally, the median of the 10 k-values is considered as an alternative.

Moving to Phase II, the quantities R, T, m, $\tilde{R}$, and $\tilde{T}$ are computed for the given job set. Subsequently, either the inductive decision tree or the neural network, as trained in Phase I, is employed to determine the parameter value k. This k value is then used to apply the BATC rule, resulting in the generation of a feasible schedule.

For activation purposes in the neural network, a Multi-Layer Perceptron (MLP)

with a sigmoid function is utilized. The hidden layer is configured with three, five, or seven nodes, and the network weights are learned through repeated examination of training cases. Input nodes R, T, m, $\tilde{R}$, and $\tilde{T}$ are incorporated, while the output layer of the neural network provides the k values. The employed neural network is depicted in Figure 4.1. The NeuralWorks Professional II/Plus tool from NeuralWare is employed to conduct the neural network experiments.

The ID3 algorithm is utilized to derive tests, where a rule essentially corresponds to traversing the decision tree. The construction of the trees is facilitated by custom-developed software (Otto)[84]. Specifically, the partitioning of the interval $(0.0, 6.5)$ of the look-ahead parameter into 15, 20, and 25 different classes is considered, resulting in the presentation of an inductive decision tree for the current situation, as illustrated in Figure 4.2.

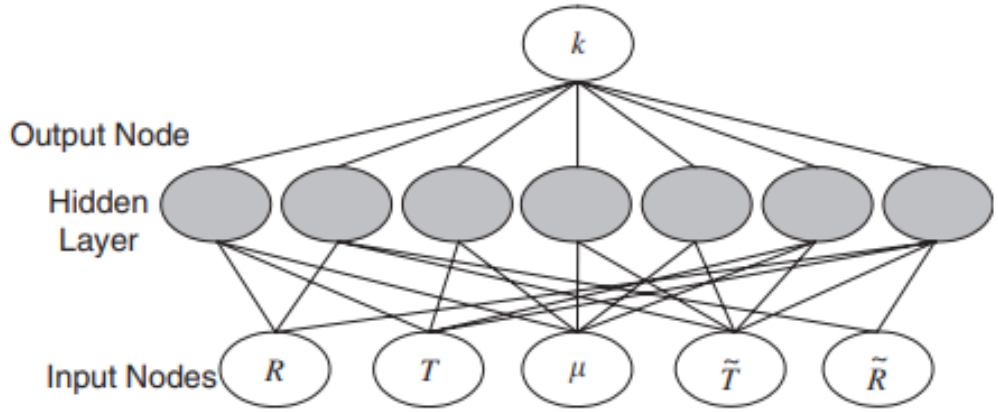

Figure 4.1: Neural network with seven hidden nodes for choosing k

4.5 Shop problems with batching machines

Potts et al. [62] investigate the complexity associated with minimizing the makespan in open shops, job shops, and flow shops equipped with two batching machines. In the case of open shops, they demonstrate that the problem $\tilde{O}2||C_{max}$ can be solved in $O(n)$ time. This solution involves creating two batches based on whether a job's processing time is shorter on the first machine than on the second. Subsequently, these batches are scheduled on the respective machines. On the contrary, the problem $\tilde{O}2|b_1|C_{max}$ is proven to be binary NP-hard for a fixed value of b_1, where $b_1 \geq 1$. Despite this, Potts et al. [62] establish that the problem has an optimal solution that involves a maximum of three batches on the second machine. They introduce a pseudopolynomial dynamic programming algorithm to address this. Additionally, they propose an algorithm with a time complexity of $O(n^{b2(b2-1)})$ for

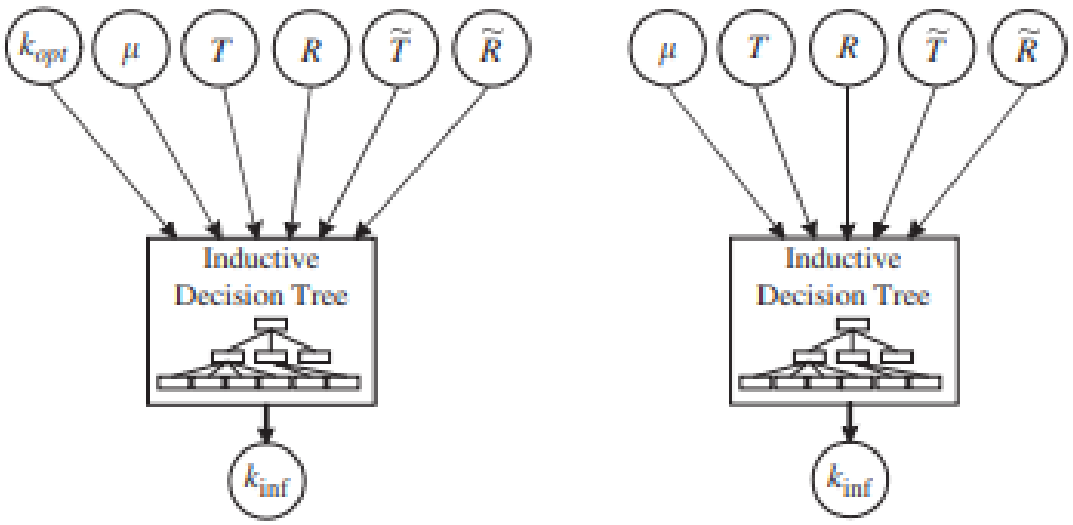

Figure 4.2: Inductive decision tree

the problem $\tilde{O}2|b_1 = 1, b_2|C_{max}$, which can be executed in $O(n \log n)$ time when $b_2 = 2$. For both the flow shop and job shop scenarios, they demonstrate that the problems $\tilde{F}2||C_{max}$ and $\tilde{J}2||C_{max}$, where there are at most two operations per job, can be solved in $O(n \log n)$ time by creating schedules with at most two batches and at most three batches on each machine, respectively. The problems $\tilde{F}2|b_1, b_2|C_{max}$ and $\tilde{J}2|b_1, b_2|C_{max}$ are proven to be binary NP-hard for fixed values of $b1$ and $b2$, where the maximum of $\{b1, b2\}$ is at least 2, and for fixed $b1$ when $b1$ is at least 2 and $b2$ is less than or equal to n. The problem $\tilde{F}2|b_1 = 1|C_{max}$ can be transformed into a single-machine problem $\tilde{1}||L_{max}$, and Brucker et al.'s $O(n^2)$ algorithm, as discussed in Section 5.1.2, can be employed. Similarly, the corresponding job shop problem $\tilde{J}2|b_1 = 1|C_{max}$, with at most two operations per job, is also polynomially solvable, albeit with a more intricate dynamic programming algorithm. It's important to note that, due to symmetry, the complexity outcomes for shop problems with the makespan objective remain consistent when interchanging $b1$ and $b2$. Ahmadi et al. [1] address the scheduling problem in a two-stage flow shop, where either or both stages may involve a batching machine. In their model, the processing time of a batch on a batching machine is fixed and not influenced by the jobs it contains. They conduct a comprehensive complexity classification for the $C_{\max}$ and $\sum_j C_j$ objectives. For situations involving a batching machine in the first stage and the $\sum_j C_j$ objective, Hoogeveen and van de Velde [45] propose a lower bounding scheme based on Lagrangean relaxation of constraints linking completion times of operations in the two stages. They also develop an approximation algorithm that produces a schedule with a total completion time no more than $\frac{5}{3}$ times the

optimal value. Dannenberg et al. [28]examine permutation flow shop problems denoted as $\tilde{F}|s_{if}, b|C_{max}$ and $\tilde{F}|s_{if}, b|\sum_j w_j \cdot C_j$. In these problems, jobs belonging to the same family, up to a specified limit b, can be processed together in a batch. The investigation focuses on schedules with identical batches and a consistent processing order on each machine. Following a methodology akin to Sotskov et al. [72], the authors assess various heuristics, including insertion heuristics and traditional local search techniques such as multi-start descent, simulated annealing, and tabu search. Additionally, they explore a 'multi-level' simulated annealing approach, where a neighbor is derived by executing a move in a relatively large neighborhood, followed by descent in a smaller neighborhood. Computational experiments conducted on problem instances featuring 40, 60, 80, 100, and 120 jobs, along with 5 and 10 machines, reveal that insertion heuristics are effective. Moreover, the results indicate a preference for the multi-level approach over other local search methods in terms of performance.

Chapter 5

Conclusions

This review examines research related to two types of scheduling models that necessitate the formation of batches. In the family scheduling model, similar tasks can be grouped into batches if they require the same setup. Conversely, the batching machine model creates batches of tasks processed simultaneously. The analysis of these models reveals that, in certain scenarios, the sequencing and batching of tasks can be separated. Once the sequence of tasks (within a family, in the family scheduling model) is determined, dynamic programming proves to be a valuable technique for addressing the batching dilemma. Many of the issues addressed in this article are known to be either solvable in polynomial time or NP-hard. Of the few remaining open problems, several involve the total (weighted) completion time objective function. For the NP-hard problems, there is a limited number of studies on the development of branch and bound and approximation algorithms. Given the interest in scheduling with batching, investing research efforts into devising algorithms for NP-hard problems is deemed worthwhile. Other models exist that incorporate both batching and scheduling, but they are not discussed in this review. For example, consider a scenario where jobs are delivered to customers in batches. In this model, consolidating larger batches helps reduce delivery costs, but it also leads to longer job completion times (assuming batches are readily available). Another instance where batching decisions are essential is highlighted by Fazle Baki and Vickson [30]. They examine multi-machine problems with a single operator, where each operation requires the operator's presence. As the operator moves between machines, a time delay, which can be considered a setup, is incurred.

Bibliography

[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] K.R. Baker, Scheduling the production of components at a common facility, IIE Transactions 20 (1988) 32-35.

[13] T.C.E. Cheng, Z.-L. Chen, Parallel machine scheduling with batch setup times, Operations Research 42 (1994) 1171-1174.

[14] 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

[15] 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.

[16] 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.

[17] 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.

[18] T.C.E. Cheng, M.Y. Kovalyov, Single machine batch scheduling with sequential job processing, in submission.

[19] T.C.E. Cheng, M.Y. Kovalyov, Algorithms for parallel machine batch scheduling with deadlines, in submission.

[20] 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.

[21] T.C.E. Cheng, A. Toker, B.M.T. Lin, Makespan minimization in the two-machine ¯ow-shop batch scheduling problem, Working paper 04/95-6, The Hong Kong Polytechnic University, Faculty of Business and Information Systems, 1995.

[22] T.C.E. Cheng, G. Wang, Batching and scheduling to minimize the makespan in the two-machine flowshop, IIE Transactions 30 (1998) 447-453.

[23] 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.

[24] 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.

[25] 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.

[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 ¯ow shop scheduling problems with setup times and limited batch size, Preprint No. 52, Fakultat fur Mathematik, Otto-von-Guericke-Universitat

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 fur 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 Nijho, 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 fur Mathematik, TU Graz, Austria, 1997.

[82] X. Yang, Scheduling with generalized batch delivery dates and earliness penalties, IIE Transactions, in press.

[83] S. Zdrzaøka, 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.

[84] Mo¨nch, L., Otto, P., 2002. Scheduling jobs on parallel batch processing machines using dispatching rules and machine learning techniques. In: Al-Akaidi, M (Ed.), Proceedings of the Fourth Middle East Simulation Symposium, Sharjah, UAE, (MESM 2002), pp. 192–196

[85] . S. Webster and K. R. Baker, 'Scheduling groups of jobs on a single machine', Oper. Res., 43, 692–703 (1995)

[86] . C.-Y. Lee, R. Uzsoy and L. A. Martin-Vega, 'Efficient algorithms for scheduling semiconductor burn-in operations', Oper. Res., 40, 764–775 (1992).

[87] . R. Uzsoy, 'Scheduling batch processing machines with incompatible job families', Int. J. Prod. Res., 33, 2685–2708 (1995)

[88] . V. Chandru, C.-Y. Lee and R. Uzsoy, 'Minimizing total completion time on batch processing machines', Int. J. Prod. Res., 31, 2097–2122 (1993).

[89] . V. Chandru, C.-Y. Lee and R. Uzsoy, 'Minimizing total completion time on a batch processing machine', Oper. Res. Lett., 13, 61–65 (1993).

[90] . D. S. Hochbaum and D. Landy, 'Scheduling semiconductor burn-in operations to minimize total owtime', Oper. Res., 45, 874–885 (1997).

[91] . G. J. Woeginger, 'When does a dynamic programming formulation guarantee the existence of an FPTAS?', Technical Report Woe-27, TU-Graz, Austria, 1998

[92] .. E. L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, New York, 1976.