Online Companion for

"Scheduling with Fixed Delivery Dates"

Operations Research
Volume 49, Number 1, January-February 2001

Nicholas G. Hall
The Ohio State University
Columbus, Ohio

Maseka Lesaoana
Labour Information & Statistics
Department of Labour, Pretoria, South Africa

Chris N. Potts
University of Southampton, United Kingdom

Abstract

In most classical scheduling models, it is assumed that a job is dispatched to a customer immediately its processing completes. In many practical situations, however, a set of delivery dates may be fixed before any jobs are processed. This is particularly relevant where delivery is an expensive or complicated operation, for example as with heavy machinery. A similar situation arises where customers find deliveries disruptive, and thus require them to be made within a limited time interval that repeats periodically. A third possibility is that a periodic business function, for example the supplier's billing cycle, effectively defines a delivery date, and includes all jobs that have been completed since the previous billing cycle. These situations are not adequately represented by classical scheduling models. We consider a variety of deterministic scheduling problems in which a job is dispatched to a customer at the earliest fixed delivery date which is no earlier than the completion time of its processing. Problems where the number of delivery dates is constant, and others where it is specified as part of data input, are studied. For almost all problems considered, we provide either an efficient algorithm, or a proof that such an algorithm is unlikely to exist. By doing so, we permit comparisons between the solvability of these fixed delivery date problems and of the corresponding classical scheduling problems.

Key words and phrases: deterministic scheduling, fixed delivery dates, polynomial time algorithm, *NP*-complete.

A standard assumption used in most of the classical scheduling literature is that a job is dispatched to a customer at the instant it completes processing. However, several practical situations can be observed which are not adequately represented by this classical assumption about dispatch dates. In these situations, a job is dispatched at the earliest of a series of *fixed delivery dates* which falls at or after the completion of its processing. The purpose of this paper is to investigate the solvability of scheduling problems in which this alternative definition of dispatch dates applies.

Fixed delivery dates may occur in supply chain management, where either economies of scale in delivery cost, or the preferences of a downstream decision maker, determine a list of possible delivery dates before scheduling issues are considered.

In some industrial applications, such as heavy engineering, the delivery of a job may be an expensive or complicated operation which requires planning before decisions about the job processing sequence and timing have been made. Thus, delivery dates may be announced before the exact time at which any particular job completes processing is known. Even if delivery dates are planned after a schedule is fixed, it would be natural to include some slack time in case of processing delays. In either case, assuming that processing completes no later than the planned delivery date, the time at which the job is dispatched is independent of the time at which processing completes.

A similar situation occurs where customers find deliveries to be disruptive of their operations, and they therefore limit the times during which deliveries can be made. The standard way to achieve this is to permit deliveries only within a time interval that repeats periodically, for example every Monday from 8:00-10:00 a.m. In this example, if processing completes on Monday afternoon, delivery will not take place until the following Monday morning. Again, the dispatch time is independent of the time within that week at which processing completes.

A fourth possibility is that a periodic business function, for example the supplier's billing cycle, effectively defines a fixed delivery date. This billing cycle may be set by company policy in advance, without knowledge of the completion times of jobs. Here economies of scale, as discussed in the following paragraph, require the consolidation of bills for each period within a cyclic pattern. The jobs that are dispatched at each billing cycle will be those which

have completed processing and not previously been dispatched, i.e., exactly those which have completed processing since the previous billing cycle. Another example of such consolidation is the mail collection times provided by the U.S. Post Office.

Apparently, the earliest reference to fixed delivery dates in the literature is by Matsuo (1988), who considers a single machine environment. He points out that, in many manufacturing environments, the number of shipping times is far less than the number of jobs. He provides an example in which 200 jobs are shipped over a 40-day planning horizon with one shipping time per day. Consolidation of shipments in this way results in substantial cost savings in transportation and handling. He considers the objective of finding an overtime utilization level and a job sequence that jointly provide a good tradeoff between overtime cost and penalties for late delivery. The problem is shown to be intractable. A heuristic, based on a capacitated transshipment formulation, is described and tested computationally. Lesaoana (1991) also considers problems with fixed delivery dates, and provides algorithms and computational complexity results for several different scheduling environments and objectives.

In this paper, we consider a variety of the most practically and theoretically important nonpreemptive problems from the deterministic scheduling literature. In line with the above discussion, we assume that a job is dispatched to a customer at the earliest fixed delivery date that is no earlier than the completion time of its processing. We study problems where the number of delivery dates is constant, and also other problems where it is specified as part of the input for a particular instance. For almost all problems considered, we provide either an efficient algorithm or a proof that such an algorithm is unlikely to exist. This analysis permits comparisons between the solvability of problems with fixed delivery dates, and the solvability of the corresponding classical problems.

This paper is organized as follows. In Section 1, we introduce some basic definitions and notation, and provide an overview of our results for scheduling problems with fixed delivery dates. Section 2 gives some general results that apply across several different scheduling environments or objectives. In Section 3, we study single machine problems. Section 4 considers identical parallel machine problems. Flow shops, job shops and open shops are studied in Sections 5, 6 and 7, respectively. Finally, Section 8 contains a conclusion and some suggestions

for future research.

1 Preliminaries

1.1 Notation and classification

Let $N = \{1, \ldots, n\}$ denote the set of jobs to be processed nonpreemptively on m machines, $M_1, \ldots, M_m$. Let $p_{ij} \geq 1$ denote the processing time of job j on machine M_i, for $i = 1, \ldots, m$, $j = 1, \ldots, n$. In single and identical parallel machine problems, the first subscript is omitted. Where no ambiguity will arise, we may write $\sum_{j=1}^{n} p_{1j}$, $\sum_{j=1}^{n} p_{2j}$ and $\sum_{j=1}^{n} p_j$ as $\sum p_{1j}$, $\sum p_{2j}$ and $\sum p_j$, respectively. Let $D_1, \ldots, D_s$ denote the fixed delivery dates, where $0 < D_1 < \cdots < D_s$. In many applications, the time intervals between consecutive pairs of delivery dates are equal. However, we do not treat such situations separately, since we have not identified any cases where this property makes a problem easier to solve. We consider problems where s is a fixed constant, and also more general problems where s is arbitrary, i.e., part of the input. Other parameters of job j that occur in some problems include a due date d_j and a weight or value w_j. We assume throughout that all p_{ij}, d_j, w_j and D_k are non-negative integers.

We define the following variables in schedule σ.

$$
\begin{array}{lcl}
\hat{C}_j(\sigma) & = & \text{the time at which job } j \text{ is dispatched to its customer;} \\
\hat{L}_j(\sigma) & = & \hat{C}_j(\sigma) - d_j, \text{ the lateness of job } j; \\
\hat{U}_j(\sigma) & = & \begin{cases} 1 \text{ if job } j \text{ is late, i.e., } \hat{C}_j(\sigma) > d_j \\ 0 \text{ if job } j \text{ is dispatched to its customer by its due date, i.e., } \hat{C}_j(\sigma) \leq d_j; \end{cases} \\
\hat{T}_j(\sigma) & = & \max\{\hat{C}_j(\sigma) - d_j, 0\}, \text{ the tardiness of job } j.
\end{array}
$$

When there is no ambiguity, we simplify $\hat{C}_j(\sigma)$, $\hat{L}_j(\sigma)$, $\hat{U}_j(\sigma)$, and $\hat{T}_j(\sigma)$ to $\hat{C}_j$, $\hat{L}_j$, $\hat{U}_j$, and $\hat{T}_j$, respectively. We base performance measures on functions of $\hat{C}_j(\sigma)$, since these dispatch times represent completion times as perceived by the customer. When we discuss a similar classical scheduling problem, the equivalent variables are C_j, L_j, U_j, and T_j, respectively.

The standard classification scheme for scheduling problems (Graham *et al.*, 1979) is $\psi_1|\psi_2|\psi_3$, where ψ_1 indicates the scheduling environment, ψ_2 describes the job characteristics or restrictive requirements, and ψ_3 defines the objective function to be minimized.

We let $\psi_1 = \alpha m$, where $\alpha \in \{P, F, J, O\}$ denotes identical parallel machine, flow shop, job shop and open shop environments, respectively, with m machines. If m is not shown, then the

number of machines is arbitrary. If $m = 1$, then we consider a single machine environment and omit α. If $\psi_1 = Jm$, then we assume that each job has no more than one operation on any machine, for $m \geq 2$.

Under ψ_2, we may have:

$s = \bar{s}$: the number of delivery dates is a constant;
s: the number of delivery dates is arbitrary.

The objective functions that we consider under ψ_3 require the minimization of the following cost functions:

$\hat{C}_{\max}$ = $\max_{1 \leq j \leq n}\{\hat{C}_j\}$, the dispatch time of the last job, or the makespan;
$\sum(w_j)\hat{C}_j$ = the total (weighted) dispatch time of the jobs;
$\hat{L}_{\max}$ = $\max_{1 \leq j \leq n}\{\hat{L}_j\}$, the maximum lateness of the jobs;
$\sum(w_j)\hat{U}_j$ = the (weighted) number of jobs not dispatched by their due dates;
$\sum(w_j)\hat{T}_j$ = the total (weighted) tardiness of the jobs.

An example of the classification scheme is problem $1|s = \bar{s}|\hat{L}_{\max}$, which denotes the minimization of the maximum lateness on a single machine with a constant number of fixed delivery dates. In general, we use γ to denote an objective function for a classical problem, and $\hat{\gamma}$ to denote the corresponding objective function for a problem with fixed delivery dates. Since all the above objective functions are nondecreasing in $\hat{C}_j$, we restrict our attention to schedules which contain no inserted idle time.

It is useful to define the following categories of objective functions:

$\mathcal{A}$. $\sum \hat{f}_j$, where $\hat{f}_j \in \{\hat{C}_j, w_j\hat{C}_j\}$;

$\mathcal{B}$. $\hat{f}_{\max}$, where $\hat{f}_{\max} \in \{\hat{C}_{\max}, \hat{L}_{\max}\}$;

$\mathcal{C}$. $\sum \hat{f}_j$, where $\hat{f}_j \in \{\hat{U}_j, w_j\hat{U}_j\}$;

$\mathcal{D}$. $\sum \hat{f}_j$, where $\hat{f}_j \in \{\hat{T}_j, w_j\hat{T}_j\}$.

We use the convention that $f_j(D_k)$ denotes the cost of scheduling job j for delivery at time D_k, for any objective in $\mathcal{A}, \mathcal{B}, \mathcal{C}$ or $\mathcal{D}$, for $j \in N$.

For any time $t > 0$, we let $|\lceil t \rceil|$ denote the earliest delivery date no earlier than t, and $|\lfloor t \rfloor|$ denote the latest delivery date no later than t. The time, $C_j(\sigma)$, at which the last operation of

job j completes processing defines that job's dispatch time to be $\hat{C}_j(\sigma) = D_k$, where k is such that $D_k = |\lceil C_j(\sigma) \rceil|$, or equivalently $D_{k-1} < C_j(\sigma) \leq D_k$, where $D_0 = 0$. If $C_j(\sigma) > D_s$, then we define $\hat{C}_j(\sigma) = \infty$. For single machine scheduling problems and flow shops, if $\hat{C}_j(\sigma) = D_k$, then job j is said to be *within block* k. For parallel machine scheduling problems, job shops, and open shops, an operation that is processed on some machine M_i is *within block* k *on* M_i if the time t at which it completes processing is such that $D_k = |\lceil t \rceil|$.

1.2 Overview of the results

Table 1 provides a summary of complexity results for a variety of scheduling problems with both a constant and an arbitrary number of fixed delivery dates. Because the feasibility and cost of a schedule can be checked efficiently in all the problems considered here, the recognition versions of those problems belong to the class *NP*. We use *BNPC* (respectively, *UNPC*) to denote that the recognition version of a problem is binary (resp., unary) *NP*-complete. Related definitions can be found in Garey and Johnson (1979). Where a polynomial or pseudopolynomial time algorithm exists, we provide its time complexity. Included in the appropriate cell is a reference to where a proof of that result can be found.

In Table 1, some of the *BNPC* and *UNPC* entries for a problem $\alpha|s|\hat{\gamma}$ are deduced from the corresponding complexity results for problem $\alpha|s = \bar{s}|\hat{\gamma}$, since a problem with an arbitrary number of fixed delivery dates is more general than one with a constant number. Similarly, a property or algorithm for problem $\alpha|s|\hat{\gamma}$ also holds for problem $\alpha|s = \bar{s}|\hat{\gamma}$, which is a special case. In the following sections, such deductions are made without stating the result explicitly.

The relationship between classical and fixed delivery date problems is clarified by the following observation. Given an instance of a classical scheduling problem, we can construct an instance of the corresponding fixed delivery date problem in which each schedule has the same cost, by adding fixed delivery dates at every integer point in time. However, this observation cannot be used to imply complexity results, since the size of the encoding of these delivery dates is not a polynomial function of the size of the input for the classical problem. Nonetheless, when the length of each interval between each consecutive pair of due dates is equal to a constant, it would be possible to encode the delivery dates by specifying only D_1

and this constant, although doing so requires the use of an encoding scheme that is different from the one used throughout the paper.

2 General Results

In this section, we provide a number of results which apply to problems in several machine environments or with a variety of objectives.

2.1 Sequencing within blocks

The following two results identify problems for which the jobs assigned to a given machine and block may be sequenced using a simple rule.

Theorem 2.1 *For problems* $1|s|\hat{\gamma}$ *and* $Pm|s|\hat{\gamma}$*, where* $\hat{\gamma}$ *is any objective defined in Section 1.1, there exists an optimal schedule in which the jobs within each block on a given machine are in an arbitrary sequence.*

Proof. Consider an optimal schedule with the property that reassigning a job to an earlier block is impossible, unless some other job is reassigned to a later block. Any reordering within a block for such a schedule does not affect the dispatch time of any job. □

Theorem 2.2 *For problem* $F2|s|\hat{\gamma}$*, where* $\hat{\gamma}$ *is any objective defined in Section 1.1, there exists an optimal schedule in which the jobs within each block are sequenced on each machine according to the algorithm of Johnson (1954).*

Proof. Suppose that there is an optimal schedule σ in which the jobs within some block k are not sequenced according to Johnson's algorithm. Let σ' denote a schedule in which those jobs are sequenced according to Johnson's algorithm. Since Johnson's algorithm minimizes the makespan in a two-machine flowshop, even if the machines do not become available simultaneously, the completion time of processing of the last job within block k is no greater in σ' than in σ. Therefore, $\hat{C}_j(\sigma') \leq \hat{C}_j(\sigma)$ for $j = 1, \ldots, n$, which implies that σ' is also an optimal schedule. □

2.2 Relationships between objectives

The following two results describe relationships between the complexity of fixed delivery date scheduling problems with different objectives.

Theorem 2.3 *If the recognition version of problem* $\alpha|s = \bar{s}|\hat{C}_{\max}$ *is binary (respectively, unary)* NP*-complete, then the recognition version of problem* $\alpha|s = \bar{s}|\hat{\gamma}$ *is also binary (resp., unary)* NP*-complete, where* $\hat{\gamma}$ *is any other objective defined in Section 1.1.*

Proof. Given an arbitrary instance of the recognition version of problem $\alpha|s = \bar{s}|\hat{C}_{\max}$ with a threshold cost C, we note that the corresponding instance with a threshold cost $||\lfloor C \rfloor||$ is equivalent. We construct an instance of problem $\alpha|s = \bar{s}|\hat{\gamma}$ with the same processing times, $\bar{s} = 1$, $D_1 = ||\lfloor C \rfloor||$, and (if applicable) $w_j = 1$ and $d_j = D_1$ for $j = 1, \ldots, n$. The recognition version of this problem asks whether there exists a schedule with $\hat{\gamma} \leq C'$, where

$$C' = \begin{cases} nD_1 \text{ for } \hat{\gamma} = \sum (w_j)\hat{C}_j \\ 0 \text{ otherwise.} \end{cases}$$

If there exists a schedule with $\hat{C}_{\max} \leq C \Rightarrow \hat{C}_{\max} \leq ||\lfloor C \rfloor||$ for problem $\alpha|s = \bar{s}|\hat{C}_{\max}$, then $\hat{C}_j \leq ||\lfloor C \rfloor||$ for $j = 1, \ldots, n$ in that schedule. Using the same schedule in $\alpha|s = \bar{s}|\hat{\gamma}$ gives $\hat{C}_j \leq D_1$ for $j = 1, \ldots, n$, and thus $\hat{\gamma} \leq C'$.

Conversely, if there exists a schedule for problem $\alpha|s = \bar{s}|\hat{\gamma}$ with $\hat{\gamma} \leq C'$, then $\hat{C}_j \leq D_1 = ||\lfloor C \rfloor||$ for $j = 1, \ldots, n$. Using the same schedule in $\alpha|s = \bar{s}|\hat{C}_{\max}$ gives $\hat{C}_{\max} \leq ||\lfloor C \rfloor||$. □

Theorem 2.3 provides complexity results for several problems with fixed delivery dates which are binary or unary NP-complete from Theorem 2.6.

The following theorem shows how complexity results for problems with the maximum lateness objective can be used to derive similar results for problems with other objectives.

Theorem 2.4 *If the recognition version of problem* $\alpha|s = \bar{s}|\hat{L}_{\max}$ *(or* $\alpha|s|\hat{L}_{\max}$*) is binary (respectively, unary)* NP*-complete, then the recognition version of problem* $\alpha|s = \bar{s}|\hat{\gamma}$ *(or* $\alpha|s|\hat{\gamma}$*) is also binary (resp., unary)* NP*-complete, where* $\hat{\gamma} \in \{\sum(w_j)\hat{U}_j, \sum(w_j)\hat{T}_j\}$*. Furthermore, if the recognition version of problem* $\alpha|s = \bar{s}|\sum(w_j)\hat{C}_j$ *(or* $\alpha|s|\sum(w_j)\hat{C}_j$*) is binary (resp., unary)* NP*-complete, then the recognition version of problem* $\alpha|s = \bar{s}|\sum(w_j)\hat{T}_j$ *(or* $\alpha|s|\sum(w_j)\hat{T}_j$*) is also binary (resp., unary)* NP*-complete.*

Proof. Given an arbitrary instance of the recognition version of problem $\alpha|s = \bar{s}|\hat{L}_{\max}$ (or $\alpha|s|\hat{L}_{\max}$) with due dates $d_1, \ldots, d_n$ and a threshold cost C, we construct an instance of problem $\alpha|s = \bar{s}|\hat{\gamma}$ (or $\alpha|s|\hat{\gamma}$), where $\hat{\gamma} \in \{\sum(w_j)\hat{U}_j, \sum(w_j)\hat{T}_j\}$, with due dates $d_1 + C, \ldots, d_n + C$, the same processing times and fixed delivery dates, and a threshold cost of zero. The recognition versions of these two problems are clearly equivalent. Also, problem $\alpha|s = \bar{s}|\sum(w_j)\hat{C}_j$ (or $\alpha|s|\sum(w_j)\hat{C}_j$) is a special case of problem $\alpha|s = \bar{s}|\sum(w_j)\hat{T}_j$ (or $\alpha|s|\sum(w_j)\hat{T}_j$) in which $d_j = 0$ for $j = 1, \ldots, n$. □

Theorem 2.4 provides complexity results for several problems with fixed delivery dates, including the recognition versions of: problem $1|s = \bar{s}|\sum w_j\hat{T}_j$, which is binary *NP*-complete from Theorem 3.3; problems $1|s|\sum w_j\hat{T}_j$ and $Pm|s|\sum w_j\hat{T}_j$, which are unary *NP*-complete from Theorem 3.4; problems $\alpha 2|s = \bar{s}|\sum(w_j)\hat{U}_j$ and $\alpha 2|s = \bar{s}|\sum(w_j)\hat{T}_j$, where if $\alpha \in \{F, J, O\}$, the problems are binary *NP*-complete from Theorem 2.8; and problems $\alpha 2|s|\sum(w_j)\hat{U}_j$ and $\alpha|s|\sum(w_j)\hat{T}_j$, where if $\alpha \in \{F, J, O\}$, the problems are unary *NP*-complete from Theorem 2.8.

2.3 Makespan

The first result shows that any approach for solving a classical makespan problem can also be used to solve the corresponding makespan problem with fixed delivery dates.

Theorem 2.5 *An algorithm that finds an optimal schedule for problem $\alpha||C_{\max}$, also does so for problem $\alpha|s|\hat{C}_{\max}$ with the same time complexity.*

Proof. Any feasible schedule for problem $\alpha||C_{\max}$ with makespan C is a feasible schedule for $\alpha|s|\hat{C}_{\max}$ with makespan $|\lceil C \rceil|$. There are no schedules for problem $\alpha|s|\hat{C}_{\max}$ that are not feasible for $\alpha||C_{\max}$. Thus, any optimal schedule for $\alpha||C_{\max}$ is also an optimal schedule for $\alpha|s|\hat{C}_{\max}$. □

From Theorem 2.5, the existence of a polynomial time algorithm for a classical makespan problem $\alpha||C_{\max}$ implies that the same algorithm will solve the corresponding fixed delivery date problem in polynomial time. For example, an arbitrary sequence of the jobs solves problem $1|s|\hat{C}_{\max}$ in $O(n)$ time; an algorithm of Johnson (1954) solves problem $F2|s|\hat{C}_{\max}$ in

$O(n \log n)$ time; an algorithm of Jackson (1956) solves problem $J2|s|\hat{C}_{\max}$ in $O(n \log n)$ time; and an algorithm of Gonzalez and Sahni (1976) solves problem $O2|s|\hat{C}_{\max}$ in $O(n)$ time.

The close relationship between classical makespan problems and the corresponding problems with fixed delivery dates also provides the following result.

Theorem 2.6 *If the recognition version of any classical scheduling problem $\alpha||C_{\max}$ is binary (respectively, unary)* NP*-complete, then the recognition version of problem $\alpha|s = \bar{s}|\hat{C}_{\max}$ is binary (resp., unary)* NP*-complete, even if $\bar{s} = 1$.*

Proof. Assume that we are given an arbitrary instance of the recognition version of problem $\alpha||C_{\max}$ with n' jobs and a threshold cost of C'. We construct an instance of the recognition version of problem $\alpha|s = \bar{s}|\hat{C}_{\max}$ with n jobs, where $n = n'$, $s = 1$, $D_1 = C'$, the processing times are the same as in problem $\alpha||C_{\max}$ and the threshold cost is C'. The recognition versions of the classical and the fixed delivery date problems are clearly equivalent. □

Theorem 2.6 provides complexity results for several makespan problems with fixed delivery dates, including the recognition versions of: problem $P2|s = \bar{s}|\hat{C}_{\max}$, which is binary *NP*-complete (Karp, 1972); problem $P|s = \bar{s}|\hat{C}_{\max}$, which is unary *NP*-complete (Garey and Johnson, 1978); problems $F3|s = \bar{s}|\hat{C}_{\max}$ and $J3|s = \bar{s}|\hat{C}_{\max}$, which are unary *NP*-complete (Garey, Johnson and Sethi, 1976); problem $O3|s = \bar{s}|\hat{C}_{\max}$, which is binary *NP*-complete (Gonzalez and Sahni, 1976); and problem $O|s = \bar{s}|\hat{C}_{\max}$, which is unary *NP*-complete (Williamson *et al.*, 1997). Using Theorem 2.3, corresponding results are deduced for the other objective functions defined in Section 1.1.

2.4 Maximum lateness

We study the problem of minimizing the maximum lateness with an arbitrary number of fixed delivery dates, or problem $\alpha|s|\hat{L}_{\max}$. The following result shows how an algorithm for the equivalent classical problem can be used here.

Theorem 2.7 *An algorithm that finds an optimal schedule in polynomial or pseudopolynomial time for problem $\alpha||L_{\max}$ also does so for problem $\alpha|s|\hat{L}_{\max}$.*

Proof. Assume that there exists a polynomial or pseudopolynomial time algorithm for answering the recognition version of problem $\alpha||L_{\max}$ with a threshold cost of zero. Given an instance of problem $\alpha|s|\hat{L}_{\max}$, we set $d'_j = D_s$ for $j = 1, \ldots, n$ and apply the algorithm to the instance of $\alpha||L_{\max}$ with due dates $d'_1, \ldots, d'_n$ and a threshold cost of zero. If no solution with $L_{\max} \leq 0$ is found, then none exists and we terminate. Otherwise, D_s becomes an upper bound on $\hat{L}_{\max}$ for binary search. Also, a trivial lower bound is $\min_{j \in N}\{-d_j\}$. Between these bounds, we perform binary search on the optimal value of $\hat{L}_{\max}$. At any given value of $\hat{L}_{\max} = L$, we set $d'_j = ||\lfloor d_j + L \rfloor||$ for $j = 1, \ldots, n$, and apply the polynomial or pseudopolynomial time algorithm for solving the recognition version of problem $\alpha||L_{\max}$ with due dates $d'_1, \ldots, d'_n$ and a threshold cost of zero. If there is a solution with $L_{\max} \leq 0$, then L is a new upper bound on the maximum lateness; otherwise, L is a new lower bound. Binary search finds the minimum value of L such that $L_{\max} \leq 0$, and the associated schedule is optimal for problem $\alpha|s|\hat{L}_{\max}$. Since the number of binary search iterations is polynomial, the overall algorithm runs in polynomial or pseudopolynomial time according to the complexity of the algorithm for problem $\alpha||L_{\max}$. $\square$

Although Theorem 2.7 establishes that problem $1|s|\hat{L}_{\max}$ is solvable in polynomial time, we provide a more efficient algorithm in Theorem 3.5. The following result also provides a connection between classical problems and those with fixed delivery dates.

Theorem 2.8 *If the recognition version of a classical scheduling problem $\alpha||L_{\max}$ is binary (respectively, unary)* NP*-complete, then the recognition version of the corresponding problem $\alpha|s|\hat{L}_{\max}$ is binary (resp., unary)* NP*-complete. Furthermore, if the recognition version of a special case of a classical problem $\alpha||L_{\max}$ with a constant number of due dates is binary (resp., unary)* NP*-complete, then the recognition version of the corresponding problem $\alpha|s = \bar{s}|\hat{L}_{\max}$ is binary (resp., unary)* NP*-complete.*

Proof. Assume that we are given an arbitrary instance of the recognition version of problem $\alpha||L_{\max}$ with n' jobs, where we assume without loss of generality that the threshold cost is zero. We construct an instance of the recognition version of problem $\alpha|s|\hat{L}_{\max}$ (or $\alpha|s = \bar{s}|\hat{L}_{\max}$) with n jobs, where $n = n'$, the fixed delivery dates are equal to the s distinct values of the

due dates, the processing times and due dates are the same as in problem $\alpha||L_{\max}$, and the threshold cost is zero. The recognition versions of these two problems are clearly equivalent. □

Theorem 2.8 provides complexity results for several maximum lateness problems with fixed delivery dates, including the recognition versions of: problem $\alpha 2|s = \bar{s}|\hat{L}_{\max}$, which is binary *NP*-complete for $\alpha \in \{F, J\}$ (Lenstra, 1977) and $\alpha = O$ (by a straightforward adaptation of the reduction of Lawler, Lenstra and Rinnooy Kan, 1981, 1982); and problem $\alpha 2|s|\hat{L}_{\max}$, which is unary *NP*-complete for $\alpha \in \{F, J\}$ (Garey, Johnson and Sethi, 1976), and for $\alpha = O$ (Lawler, Lenstra and Rinnooy Kan, 1981, 1982). Using Theorems 2.4 and 2.8, corresponding computational complexity results for the $\sum(w_j)\hat{U}_j$ and $\sum(w_j)\hat{T}_j$ objectives can be deduced.

2.5 Weighted number of late jobs

We study the problem of minimizing the weighted number of late jobs with an arbitrary number of fixed delivery dates, or problem $\alpha|s|\sum w_j\hat{U}_j$. The following result shows that there is an equivalent instance of the classical problem $\alpha||\sum U_j$. Due dates in this instance are obtained from the observation that if some job j is on time in $\alpha|s|\sum w_j\hat{U}_j$, then it must complete processing no later than time $||\lfloor d_j \rfloor||$.

Theorem 2.9 *Problems $\alpha|s|\sum w_j\hat{U}_j$ and $\alpha||\sum w_jU_j$ are equivalent, where an instance of the latter problem is constructed from the corresponding instance of the former by setting due dates $d'_j = ||\lfloor d_j \rfloor||$ for $j = 1, \ldots, n$.*

Proof. In a given schedule, consider a job j with $D_{k-1} < C_j \leq D_k$, which implies that the dispatch time of job j in problem $\alpha|s|\sum w_j\hat{U}_j$ is $\hat{C}_j = D_k$. First, suppose that $d_j \geq D_k$, in which case job j is on time. But, in problem $\alpha||\sum w_jU_j$, we have $d'_j = ||\lfloor d_j \rfloor|| \geq D_k \geq C_j$, so job j is also on time there. Alternatively, suppose that $d_j < D_k$, so that job j is late in problem $\alpha|s|\sum w_j\hat{U}_j$. By definition, the corresponding due date of job j in problem $\alpha|s|\sum w_jU_j$ is $d'_j = ||\lfloor d_j \rfloor|| \leq D_{k-1} < C_j$ and thus job j is also late in that problem. It follows that the two problems are equivalent. □

Corollary 2.1 *An algorithm that finds an optimal schedule for problem* $\alpha||\sum(w_j)U_j$ *also does so for problem* $\alpha|s|\sum(w_j)\hat{U}_j$ *with the same time complexity.*

Proof. Given an instance of problem $\alpha|s|\sum w_j\hat{U}_j$, we invoke Theorem 2.9 to convert the problem into an instance of problem $\alpha||\sum w_jU_j$. Defining the new due dates requires $O(n)$ time, which does not exceed the complexity of any algorithm for problem $\alpha||\sum w_jU_j$, since both input and output require that much time. This algorithm is then applied to the constructed instance of problem $\alpha||\sum w_jU_j$. □

Corollary 2.1 implies that the algorithm of Moore (1968) solves problem $1|s|\sum\hat{U}_j$ in $O(n\log n)$ time. Similarly, the algorithm of Lawler and Moore (1969) solves problem $1|s|\sum w_j\hat{U}_j$ in $O(n\sum p_j)$ time.

Corollary 2.2 *If the recognition version of a classical scheduling problem* $\alpha||\sum(w_j)U_j$ *is binary (respectively, unary)* NP*-complete, then the recognition version of the corresponding problem* $\alpha|s|\sum(w_j)\hat{U}_j$ *is binary (resp., unary)* NP*-complete. Furthermore, if the recognition version of a special case of a classical problem* $\alpha||\sum(w_j)U_j$ *with a constant number of due dates is binary (resp., unary)* NP*-complete, then the recognition version of the corresponding problem* $\alpha|s=\bar{s}|\sum(w_j)\hat{U}_j$ *is binary (resp., unary)* NP*-complete.*

Proof. Given an arbitrary instance of the recognition version of problem $\alpha||\sum(w_j)U_j$ with n' jobs and a threshold cost of C', we construct an instance of the recognition version of problem $\alpha|s|\sum(w_j)\hat{U}_j$ (or $\alpha|\bar{s}|\sum(w_j)\hat{U}_j$) with n jobs, where $n=n'$, the fixed delivery dates are equal to the s distinct values of the due dates, the processing times and due dates (and, if applicable, the weights) are the same as in problem $\alpha||\sum(w_j)U_j$, and the threshold cost is C'. From Lemma 2.9, the recognition versions of these two problems are equivalent. □

The work of Karp (1972) and Corollary 2.2 imply that the recognition version of problem $1|s=\bar{s}|\sum w_j\hat{U}_j$ is binary *NP*-complete. We have now established the computational complexity of problems $1|s=\bar{s}|\sum(w_j)\hat{U}_j$ and $1|s|\sum(w_j)\hat{U}_j$.

3 Single Machine

In this section, we consider a number of the most widely studied single machine scheduling problems. Each subsection considers a different objective.

3.1 Total and maximum cost

We now describe a pseudopolynomial time dynamic programming algorithm, BLOCK, for problem $1|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective in category $\mathcal{A}$, $\mathcal{B}$ or $\mathcal{D}$. It is useful to define the operator $\oplus$, where $\oplus$ represents the "max" operator for objectives in $\mathcal{B}$, and the "+" operator elsewhere. Recall from Theorem 2.1 that any sequence of jobs within a block can be used. Thus, Algorithm BLOCK considers jobs in an arbitrary order, and evaluates the possible assignment of any job to each block.

Algorithm BLOCK

State variables

In addition to a variable j which indicates that jobs $1, \ldots, j$ have been scheduled, we also use state variables $(t_1, \ldots, t_{\bar{s}})$, where t_k is the total processing time of jobs assigned to block k, for $k = 1, \ldots, \bar{s}$. When jobs $1, \ldots, j$ have been scheduled, we restrict the choice of state variables to satisfy $(t_1, \ldots, t_{\bar{s}}) \in \mathcal{T}_j$, where $\mathcal{T}_j = \{(t_1, \ldots, t_{\bar{s}}) | \sum_{h=1}^{k} t_h \leq D_k \text{ for } k = 1, \ldots, \bar{s}, \sum_{k=1}^{\bar{s}} t_k = \sum_{l=1}^{j} p_l\}$.

Optimal value function

Let $g_j(t_1, \ldots, t_{\bar{s}})$ denote the minimum cost of scheduling jobs $1, \ldots, j$ such that the state $(t_1, \ldots, t_{\bar{s}})$ is achieved.

Recurrence relation

$g_j(t) = \min_{k \in \{1, \ldots, \bar{s}\}} \{g_{j-1}(\ldots, t_k - p_j, \ldots) \oplus f_j(D_k)\}$,

where $t = (t_1, \ldots, t_{\bar{s}})$, and $(\ldots, t_k - p_j, \ldots)$ is an abbreviation for $(t_1, \ldots, t_{k-1}, t_k - p_j, t_{k+1}, \ldots, t_{\bar{s}})$.

Boundary condition

$g_0(0, \ldots, 0) = 0$, and all other values are set to ∞, for $\hat{\gamma} \neq \hat{L}_{\max}$.

$g_0(0, \ldots, 0) = -\infty$, and all other values are set to ∞, for $\hat{\gamma} = \hat{L}_{\max}$.

Optimal solution

$\min_{(t_1,\ldots,t_{\bar{s}})\in\mathcal{T}_n}\{g_n(t_1,\ldots,t_{\bar{s}})\}$.

Theorem 3.1 *Algorithm BLOCK finds an optimal schedule for problem* $1|s=\bar{s}|\sum\hat{\gamma}$ *with time complexity* $O(n(\sum p_j)^{\bar{s}-1})$ *for objectives in* $\mathcal{A},\mathcal{B}$ *and* $\mathcal{D}$.

Proof. Theorem 2.1 shows that jobs within each block may be sequenced arbitrarily. Since the recurrence relation computes the cost of all state transitions that are obtained by considering all possible assignments of jobs to blocks, it finds an optimal schedule. Note that $t_k \le \sum_{j=1}^{n} p_j$ for $k=1,\ldots,\bar{s}$. We only compute $g_j(t_1,\ldots,t_{\bar{s}})$ for values $t_1,\ldots,t_{\bar{s}}$ such that $\sum_{k=1}^{\bar{s}} t_k = \sum_{h=1}^{j} p_h$. Thus, the number of state variables is effectively $\bar{s}-1$. The recurrence relation is applied for n values of j. Since $\bar{s}$ is a constant, each use of the recurrence relation requires constant time. Thus, the time complexity of BLOCK is $O(n(\sum p_j)^{\bar{s}-1})$. □

We note that Algorithm BLOCK can be adapted for objectives in $\mathcal{C}$. However, Corollary 2.1 provides more efficient procedures.

3.2 Sum of dispatch times

The classical problem of minimizing the sum of job completion times on a single machine, or $1||\sum C_j$, is optimized by the shortest processing time (SPT) rule which requires $O(n\log n)$ time. A similar approach can be used for the problem with fixed delivery dates. However, since jobs within any block have a common dispatch time, they can be processed in any sequence, which may permit a reduction in the time complexity.

Theorem 3.2 *An optimal schedule for problem* $1|s|\sum\hat{C}_j$ *can be found in* $O(\min\{n\log n, ns\})$ *time, and an optimal schedule for problem* $1|s=\bar{s}|\sum\hat{C}_j$ *can be found in* $O(n)$ *time.*

Proof. Let $C_{[j]}(\sigma)$ denote the completion time of the jth job in any schedule σ. Its dispatch time is $|\lceil C_{[j]}(\sigma)\rceil|$. Let σ_{SPT} denote an SPT schedule. Since in schedule σ we have $C_{[j]}(\sigma_{\mathrm{SPT}}) \le C_{[j]}(\sigma)$ for $j=1,\ldots,n$, it follows that $\sum_{j=1}^{n}|\lceil C_{[j]}(\sigma_{\mathrm{SPT}})\rceil| \le \sum_{j=1}^{n}|\lceil C_{[j]}(\sigma)\rceil|$. Thus, an SPT schedule gives an optimal solution for problems $1|s=\bar{s}|\sum\hat{C}_j$ and $1|s|\sum\hat{C}_j$. The dominant step is ordering the jobs, which requires $O(n\log n)$ time.

A schedule with the same total dispatch time as an SPT schedule can be found in $O(ns)$ time without ordering the jobs. We use the observation that the jobs within any block have a

common dispatch time. We consider blocks $1, \ldots, s$ in turn. For each block k, we find the set of jobs E_k that are scheduled within this block by using the following procedure, where initially $E_k = \emptyset$. Let $E = E_1 \cup \ldots \cup E_{k-1}$ be the set of jobs already assigned to blocks $1, \ldots, k-1$, and let E' be the set of jobs collectively assigned to blocks $k+1, \ldots, s$, where initially $E' = \emptyset$. Let $S = N \setminus (E \cup E')$ be the set of jobs not currently assigned to blocks. To simplify the exposition, we assume that processing times are distinct, which can be achieved if necessary by perturbation, and that $|S|$ is a power of 2, which can be achieved if necessary by temporarily adding dummy jobs with zero processing times. We find the median processing time P of the jobs in S and also sets $S_1 = \{i | i \in S, p_i \leq P\}$ and $S_2 = S \setminus S_1$. If $\sum_{i \in E \cup E_k \cup S_1} p_i \leq D_k$, then we set $E_k = E_k \cup S_1$; otherwise, we set $E' = E' \cup S_2$. This reduces the cardinality of the set of unassigned jobs to $|S|/2$. This procedure is repeated for block k until all jobs of S are assigned to either E_k or E', where the successive iterations contain $|S|, |S|/2, |S|/4, \ldots, 1$ unassigned jobs. Then blocks $k+1, \ldots, n$ are considered similarly. By using a linear time median-finding procedure (Blum *et al.*, 1973, Schonhäge, Patterson and Pippenger, 1976), the computations required at block k can be implemented in $O(n)$ time. Since the procedure is applied for s blocks, the overall time complexity is $O(ns)$. However, since s is a constant in problem $1|s = \bar{s}| \sum \hat{C}_j$, the time complexity reduces to $O(n)$. $\square$

3.3 Sum of weighted dispatch times

It is well known (Smith, 1956) that the shortest weighted processing time (SWPT) rule gives an optimal solution to problem $1||\sum w_j C_j$. We now provide a numerical example to show that the SWPT rule fails to optimize the corresponding fixed delivery date problem, $1|s = \bar{s}| \sum w_j \hat{C}_j$.

Example 1. Let $n = 3, \bar{s} = 2$: $p_1 = 2$, $p_2 = p_3 = 4$; $w_1 = 3$, $w_2 = 5$, $w_3 = 4$; and $D_1 = 5$, $D_2 = 10$.

The unique SWPT sequence $S_1 = (1, 2, 3)$ has value 105, but the optimal solution value of 95 is given by non-SWPT sequences $S_2 = (2, 1, 3)$ and $S_3 = (2, 3, 1)$.

Theorem 3.3 *The recognition version of problem* $1|s = \bar{s}| \sum w_j \hat{C}_j$ *is binary* NP*-complete.*

Proof. By reduction from the following problem which is known to be binary *NP*-complete.

Partition (Garey and Johnson, 1979): given t elements with positive integer sizes $a_1, \ldots, a_t$ where $\sum_{i=1}^{t} a_i = A$, does there exist a partition S, S' of the index set $\{1, \ldots, t\}$ such that $\sum_{i \in S} a_i = \sum_{i \in S'} a_i = A/2$?

Consider the following instance of problem $1|s = \bar{s}|\sum w_j \hat{C}_j$.

$$\begin{array}{lcll} n & = & t & \\ \bar{s} & = & 2 & \\ D_1 & = & A/2 & \\ D_2 & = & A & \\ p_j & = & a_j, & j = 1, \ldots, t \\ w_j & = & a_j, & j = 1, \ldots, t \\ C & = & 3A^2/4. & \end{array}$$

Let S_k denote the set of jobs within block k, for $k = 1, 2$, where $S_1 \cup S_2 = \{1, \ldots, n\}$. We prove that there exists a schedule with $\sum w_j \hat{C}_j \leq C$ if and only if there exists a solution to Partition.

($\Rightarrow$) If $\sum_{j \in S_1} p_j = A/2$, then since $w_j = p_j$ for $j = 1, \ldots, n$, we have $\sum w_j \hat{C}_j = D_1 \sum_{j \in S_1} w_j + D_2 \sum_{j \in S_2} w_j = A(D_1 + D_2)/2 = 3A^2/4 = C$.

($\Leftarrow$) Since $w_j = p_j$ for $j = 1, \ldots, n$, we have $\sum_{j \in S_1} w_j \leq A/2$. Suppose that $\sum_{j \in S_1} w_j = A/2 - \delta$ for some $\delta > 0$. Then we have $\sum w_j \hat{C}_j = D_1 \sum_{j \in S_1} w_j + D_2 \sum_{j \in S_2} w_j = (A/2 - \delta)D_1 + (A/2 + \delta)D_2 = C + \delta(D_2 - D_1) > C$. Thus, in a schedule with $\sum w_j \hat{C}_j \leq C$, we have $\sum_{j \in S_1} w_j = \sum_{j \in S_1} a_j = A/2$, which implies that S_1, S_2 is a solution to Partition. $\Box$

Algorithm BLOCK, described in Section 3.1, solves problem $1|s = \bar{s}|\sum w_j \hat{C}_j$ in pseudopolynomial time where we set $f_j(D_k) = w_j D_k$ for $j = 1, \ldots, n$, $k = 1, \ldots, \bar{s}$. However, the following result shows that the existence of a pseudopolynomial time algorithm for the more general problem with an arbitrary number of delivery dates is unlikely.

Theorem 3.4 *The recognition version of problem* $1|s|\sum w_j \hat{C}_j$ *is unary* NP*-complete.*

Proof. By reduction from the following problem, which is known to be unary *NP*-complete.

3-Partition (Garey and Johnson, 1979): given $3t$ elements with positive integer sizes $a_1, \ldots, a_{3t}$, where $\sum_{i=1}^{3t} a_i = ty$ and $y/4 < a_i < y/2$ for $i = 1, \ldots, 3t$, does there exist a partition $Q_1, \ldots, Q_t$ of the index set $\{1, \ldots, 3t\}$ such that $|Q_j| = 3$ and $\sum_{i \in Q_j} a_i = y$, for $j = 1, \ldots, t$?

We assume without loss of generality that, if there exists a solution to 3-Partition, then the elements are numbered such that $a_{3i-2} + a_{3i-1} + a_{3i} = y$ for $i = 1, \ldots, t$.

Consider the following instance of problem $1|s|\sum w_j\hat{C}_j$.

$$\begin{array}{lcll} n & = & 3t & \\ s & = & t & \\ D_k & = & ky, & k = 1,\ldots,t \\ p_j & = & a_j, & j = 1,\ldots,3t \\ w_j & = & a_j, & j = 1,\ldots,3t \\ C & = & t(t+1)y^2/2. & \end{array}$$

We prove that there exists a schedule with $\sum w_j\hat{C}_j \leq C$ if and only if there exists a solution to 3-Partition.

($\Rightarrow$) Consider the schedule $(1,2,3,\ldots,3t-2,3t-1,3t)$. From the definition of 3-Partition, three jobs with total processing time y complete processing in each interval $(0,y],(y,2y],\ldots,$ $(ty-y,ty]$. Since $w_j = p_j$ for $j = 1,\ldots,n$, we have $\sum w_j\hat{C}_j = y(D_1+\ldots+D_s) = C$.

($\Leftarrow$) Since $w_j = p_j$ for $j = 1,\ldots,n$, the maximum total weight of jobs that can complete processing by time D_k in any schedule is ky, for $k = 1,\ldots,s$. Thus, the total weight of jobs within block k is of the form $y+\delta_{k-1}-\delta_k$, for $k = 1,\ldots,s$, where $\delta_k \geq 0$ and $\delta_0 = \delta_s = 0$. This yields $\sum w_j\hat{C}_j = C + \sum_{k=1}^{s-1}\delta_k(D_{k+1}-D_k)$. Therefore, in a schedule with $\sum w_j\hat{C}_j \leq C$, we have $\delta_k = 0$ for $k = 1,\ldots,s-1$, and the total weight of jobs within each block is equal to y. From the definition of 3-Partition, each block contains exactly three jobs. Thus, the jobs scheduled within the blocks provide a solution to 3-Partition. $\Box$

3.4 Maximum lateness

We consider problem $1|s|\hat{L}_{\max}$. Recall that for the corresponding classical problem, $1||L_{\max}$, the earliest due date (EDD) rule, in which jobs are sequenced in nondecreasing due date order, gives an optimal schedule (Jackson, 1955). We show how to use an EDD ordering to solve problem $1|s|\hat{L}_{\max}$.

Theorem 3.5 *An optimal schedule for problem $1|s|\hat{L}_{\max}$ can be found in $O(\min\{n\log n, ns\})$ time, and an optimal schedule for problem $1|s=\bar{s}|\hat{L}_{\max}$ can be found in $O(n)$ time.*

Proof. We show first that an EDD schedule gives an optimal solution. Let σ be a schedule in which jobs are not ordered according to nondecreasing due dates. Then, in σ there is a job i processed immediately before a job h, where $d_i > d_h$. Let σ' be a schedule obtained

from σ by interchanging the processing order of jobs h and i, while the rest of the sequence remains unaltered. Let $\hat{L}(\sigma)$ and $\hat{L}(\sigma')$ denote the maximum lateness for the jobs of $N \setminus \{h, i\}$ for schedules σ and σ', respectively. Note that, since h and i are consecutive jobs in both σ and σ', we have $\hat{L}(\sigma) = \hat{L}(\sigma')$. Also, job h completes processing earlier in σ' than in σ, and thus $\hat{L}_h(\sigma') \leq \hat{L}_h(\sigma)$. Further, job i completes processing in σ' at the same time as job h does in σ, but $d_i > d_h$, and thus $\hat{L}_i(\sigma') < \hat{L}_h(\sigma)$. Combining these three results, we obtain $\hat{L}_{\max}(\sigma') = \max\{\hat{L}(\sigma'), \hat{L}_i(\sigma'), \hat{L}_h(\sigma')\} \leq \max\{\hat{L}(\sigma), \hat{L}_h(\sigma)\} = \max\{\hat{L}(\sigma), \hat{L}_i(\sigma), \hat{L}_h(\sigma)\} = \hat{L}_{\max}(\sigma)$. Thus, an EDD sequence minimizes $\hat{L}_{\max}$.

The time requirement for finding an EDD schedule is $O(n \log n)$. Since the ordering of jobs within a block does not affect lateness, a schedule with the same maximum lateness as an EDD schedule can be found without ordering the jobs. Using a similar procedure to that in Theorem 3.2, in which the median due date of a subset of jobs is found in linear time, we can construct an optimal assignment of jobs to blocks $1, \ldots, s$ in $O(ns)$ time. For problem $1|\bar{s}|\hat{L}_{\max}$, the time complexity is $O(n)$. $\square$

3.5 Total (weighted) tardiness

Theorem 3.6 *The recognition version of problem* $1|s = \bar{s}|\sum \hat{T}_j$ *is binary* NP*-complete.*

Proof. By reduction from the following problem, which is known to be binary *NP*-complete.
Equal Cardinality Partition (Garey and Johnson, 1979, Garey, Tarjan and Wilfong, 1988): given $2t$ elements with positive integer sizes $a_1, \ldots, a_{2t}$, where $\sum_{i=1}^{2t} a_i = 2A$, does there exist a partition S, S' of the index set $\{1, \ldots, 2t\}$ such that $|S| = |S'| = t$, and $\sum_{i \in S} a_i = \sum_{i \in S'} a_i = A$?

Consider the following instance of problem $1|s = \bar{s}|\sum \hat{T}_j$.

$$
\begin{array}{lcll}
n & = & 2t & \\
s & = & 2 & \\
D_1 & = & A(t+1) & \\
D_2 & = & 2A(t+1) & \\
p_j & = & A + a_j, & j = 1, \ldots, 2t \\
d_j & = & 2tA + A - a_j, & j = 1, \ldots, 2t \\
C & = & A(t+1). &
\end{array}
$$

Let S_k denote the set of jobs within block k, for $k = 1, 2$, where $S_1 \cup S_2 = \{1, \ldots, n\}$. We prove that there exists a schedule with $\sum \hat{T}_j \leq C$ if and only if there exists a solution to Equal

Cardinality Partition.

($\Rightarrow$) If $\sum_{j\in S_1} a_j = A$ and $|S_1| = n/2$, then since the jobs of S_1 are on time, we have $\sum \hat{T}_j = \sum_{j\in S_2}(D_2 - d_j) = \sum_{j\in S_2}(A + a_j) = tA + A = C$.

($\Leftarrow$) Since $(t+1)\min_{j\in\{1,\ldots,2t\}}\{p_j\} \geq (t+1)(A+1) > D_1$, we have $|S_1| \leq t$. If $|S_1| < t$, then $\sum \hat{T}_j = \sum_{j\in S_2}(D_2 - d_j) \geq (t+1)(A+1) > C$. Therefore, in any schedule with $\sum \hat{T}_j \leq C$, we have $|S_1| = t$. Since $\sum_{j\in S_1} p_j \leq D_1$ by definition of S_1, we deduce that $\sum_{j\in S_1} a_j \leq A$. If $\sum_{j\in S_1} a_j < A$, then $\sum \hat{T}_j = \sum_{j\in S_2}(A + a_j) > tA + A = C$. Thus, in any schedule with $\sum \hat{T}_j \leq C$, we have $\sum_{j\in S_1} a_j = A$, which implies that S_1, S_2 is a solution to Equal Cardinality Partition. $\square$

However, we are able to provide two pseudopolynomial time algorithms for total tardiness problems.

Theorem 3.7 *An optimal schedule for problem* $1|s|\sum \hat{T}_j$ *can be found in* $O(n^4 \sum p_j)$ *time, and an optimal schedule for problem* $1|s = \bar{s}|\sum \hat{T}_j$ *can be found in* $O(\min\{n(\sum p_j)^{\bar{s}-1}, n^4 \sum p_j\})$ *time.*

Proof. The algorithm of Lawler (1977), also described by Pinedo (1995), can be adapted to both these problems. In the algorithm and supporting proofs, we replace the completion time C_j of job j by $\hat{C}_j = |\lceil C_j \rceil|$, for $j = 1, \ldots, n$. The time complexity of the algorithm is unchanged. For problem $1|s = \bar{s}|\sum \hat{T}_j$, Theorem 3.1 shows that Algorithm BLOCK finds an optimal schedule where we set $f_j(D_k) = \max\{D_k - d_j, 0\}$ for $j = 1, \ldots, n$, $k = 1, \ldots, \bar{s}$. $\square$

4 Parallel Machines

In this section, we develop two pseudopolynomial time dynamic programming algorithms for solving a variety of identical parallel machine problems.

The first algorithm considers the jobs in an appropriate sequence and assigns each in turn to a machine, or in the case of (weighted) numbers of late jobs problems may alternatively discard jobs. Let $\mathcal{A}'$ denote the restriction of $\mathcal{A}$ to $\sum \hat{f}_j = \sum \hat{C}_j$. The algorithm solves problem $Pm|s|\hat{\gamma}$, where $\hat{\gamma}$ is any objective in $\mathcal{A}'$, $\mathcal{B}$ or $\mathcal{C}$. For the objective in $\mathcal{A}'$, the jobs are indexed in SPT order. For the objectives in $\mathcal{B}$, the jobs are indexed in arbitrary order for $\hat{C}_{\max}$, and in EDD order for $\hat{L}_{\max}$. For the objectives in $\mathcal{C}$, the jobs are indexed in EDD order.

Algorithm PARALLEL

State variables

In addition to a variable j which indicates that jobs $1,\ldots,j$ have been scheduled, we also use state variables $(t_1,\ldots,t_m)$, where t_i is the total processing time of jobs scheduled on machine M_i, for $i=1,\ldots,m$. When jobs $1,\ldots,j$ have been scheduled, we restrict the choice of state variables to satisfy $(t_1,\ldots,t_m)\in\mathcal{T}_j$, where $\mathcal{T}_j=\{(t_1,\ldots,t_m)|\sum_{i=1}^{m}t_i=\sum_{l=1}^{j}p_l\}$ for objectives in $\mathcal{A}'$ and $\mathcal{B}$, and where $\mathcal{T}_j=\{(t_1,\ldots,t_m)|\sum_{i=1}^{m}t_i\le\sum_{l=1}^{j}p_l,\ t_i\le||\lfloor d_j\rfloor||$ for $i=1,\ldots,m\}$ for objectives in $\mathcal{C}$.

Optimal value function

Let $g_j(t_1,\ldots,t_m)$ denote the minimum cost of scheduling jobs $1,\ldots,j$ such that the state $(t_1,\ldots,t_m)$ is achieved.

Recurrence relation

$\mathcal{A}',\mathcal{B}.\ g_j(t)=\min\limits_{i\in\{1,\ldots,m\}}\{g_{j-1}(\ldots,t_i-p_j,\ldots)\oplus f_j(|\lceil t_i\rceil|)\}$;

$\mathcal{C}.\ g_j(t)=\min\{\min\limits_{i\in\{1,\ldots,m\}}\{g_{j-1}(\ldots,t_i-p_j,\ldots)\},g_{j-1}(t)+w_j\}$,

where $t=(t_1,\ldots,t_m)$.

Boundary condition

$g_0(0,\ldots,0)=0$, and all other values are set to ∞, for $\hat{\gamma}\neq\hat{L}_{\max}$.

$g_0(0,\ldots,0)=-\infty$, and all other values are set to ∞, for $\hat{\gamma}=\hat{L}_{\max}$.

Optimal solution

$\min\limits_{(t_1,\ldots,t_m)\in\mathcal{T}_n}\{g_n(t_1,\ldots,t_m)\}$.

Theorem 4.1 *Algorithm PARALLEL finds an optimal schedule for problem $Pm|s|\hat{\gamma}$, where $\hat{\gamma}$ is any objective in $\mathcal{A}'$ or $\mathcal{B}$ with time complexity $O(n(\sum p_j)^{m-1})$, and where $\hat{\gamma}$ is any objective in $\mathcal{C}$ with time complexity $O(n(\sum p_j)^m)$.*

Proof. For the $C_{\max}$ objective in $\mathcal{B}$, the jobs can be sequenced arbitrarily on each machine. We may assume that the sequences used on each machine should be SPT from the arguments used in Theorem 3.2 for the objective in $\mathcal{A}'$, and EDD from the arguments used in Theorem 3.5 for the $L_{\max}$ objective in $\mathcal{B}$. The arguments used in Theorem 3.5 also show that the on-time jobs should be sequenced in EDD order for the objectives in $\mathcal{C}$. The recurrence relation

computes the cost of all possible state transitions that generate such sequences of jobs on the machines, and therefore finds an optimal schedule. Note that $t_i \leq \sum_{j=1}^{n} p_j$ for $i = 1, \ldots, m$. For objectives in $\mathcal{A}'$ and $\mathcal{B}$, $g_j(t_1, \ldots, t_m)$ is only computed for values $t_1, \ldots, t_m$ where $\sum_{i=1}^{m} t_i = \sum_{h=1}^{j} p_h$. Thus, the number of state variables is effectively $m - 1$. For objectives in $\mathcal{C}$, the number of state variables is m. The recurrence relation is applied for n values of j. Since m is a constant, each use of the recurrence relation requires constant time. Thus, the time complexity of PARALLEL is $O(n(\sum p_j)^{m-1})$ for objectives in $\mathcal{A}'$ and $\mathcal{B}$, and $O(n(\sum p_j)^m)$ for objectives in $\mathcal{C}$. $\square$

The next algorithm to be described, PBLOCK, is an extension of Algorithm BLOCK in Section 3.1 to a parallel machine environment. In view of Theorem 2.1, we consider the jobs in arbitrary order. The algorithm solves problem $Pm|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective in $\mathcal{A}$, $\mathcal{B}$ or $\mathcal{D}$.

Algorithm PBLOCK

State variables

In addition to a variable j which indicates that jobs $1, \ldots, j$ have been scheduled, we also use state variables $(t_{11}, \ldots, t_{1\bar{s}}, \ldots, t_{m1}, \ldots, t_{m\bar{s}})$ where t_{ik} is the total processing time of jobs assigned to block k on machine M_i for $i = 1, \ldots, m$, $k = 1, \ldots, \bar{s}$. When jobs $1, \ldots, j$ have been scheduled, we restrict the choice of state variables to satisfy $(t_{11}, \ldots, t_{1\bar{s}}, \ldots, t_{m1}, \ldots, t_{m\bar{s}}) \in \mathcal{T}_j$, where $\mathcal{T}_j = \{(t_{11}, \ldots, t_{1\bar{s}}, \ldots, t_{m1}, \ldots, t_{m\bar{s}}) | \sum_{h=1}^{k} t_{ih} \leq D_k$ for $i = 1, \ldots, m$, $k = 1, \ldots, \bar{s}$, $\sum_{i=1}^{m} \sum_{k=1}^{\bar{s}} t_{ik} = \sum_{h=1}^{j} p_h\}$.

Optimal value function

Let $g_j(t_{11}, \ldots, t_{1\bar{s}}, \ldots, t_{m1}, \ldots, t_{m\bar{s}})$ denote the minimum cost of scheduling jobs $1, \ldots, j$ such that the state $(t_{11}, \ldots, t_{1\bar{s}}, \ldots, t_{m1}, \ldots, t_{m\bar{s}})$ is achieved.

Recurrence relation

$g_j(t) = \min_{i \in \{1,\ldots,m\}, k \in \{1,\ldots,\bar{s}\}} \{g_{j-1}(\ldots, t_{ik} - p_j, \ldots) \oplus f_j(D_k)\}$,
where $t = (t_{11}, \ldots, t_{1\bar{s}}, \ldots, t_{m1}, \ldots, t_{m\bar{s}})$.

Boundary condition

$g_0(0, \ldots, 0) = 0$, and all other values are set to ∞, for $\hat{\gamma} \neq \hat{L}_{\max}$.

$g_0(0,\ldots,0) = -\infty$, and all other values are set to ∞, for $\hat{\gamma} = \hat{L}_{\max}$.

Optimal solution

$$\min_{(t_{11},\ldots,t_{1\bar{s}},\ldots,t_{m1},\ldots,t_{m\bar{s}})\in\mathcal{T}_n} \{g_n(t_{11},\ldots,t_{1s},\ldots,t_{m1},\ldots,t_{m\bar{s}})\}.$$

Theorem 4.2 *Algorithm PBLOCK finds an optimal schedule for problem $Pm|s = \bar{s}|\hat{\gamma}$ with time complexity $O(n(\sum p_j)^{m\bar{s}-1})$ for objectives in $\mathcal{A}, \mathcal{B}$ and $\mathcal{D}$.*

Proof. Since Theorem 2.1 shows that the jobs within each block may be sequenced arbitrarily, the recurrence relation computes the cost of all possible state transitions and therefore finds an optimal schedule. Note that $t_{ik} \leq \sum_{j=1}^{n} p_j$ for $i = 1,\ldots,m$, $k = 1,\ldots,\bar{s}$. We only compute $g_j(t_{11},\ldots,t_{1\bar{s}},\ldots,t_{m1},\ldots,t_{m\bar{s}})$ for values of the state variables such that $\sum_{i=1}^{m}\sum_{k=1}^{\bar{s}} t_{ik} = \sum_{h=1}^{j} p_h$. Thus, there are effectively $m\bar{s}-1$ state variables. The recurrence relation is applied for n values of j. Since m and $\bar{s}$ are constant, each use of the recurrence relation requires constant time. Thus, the time complexity of PBLOCK is $O(n(\sum p_j)^{m\bar{s}-1})$. □

It is possible to adapt Algorithm PBLOCK for objectives in $\mathcal{C}$. However, Algorithm PARALLEL is more efficient.

A comment about problem $Pm|s|\sum \hat{T}_j$ appears in Section 8.

5 Flow Shops

Recall that the recognition version of the classical problem $F2||\sum C_j$ is unary *NP*-complete (Garey, Johnson and Sethi, 1976). However, we now demonstrate that the corresponding fixed delivery date problem is only binary *NP*-complete.

Theorem 5.1 *The recognition version of problem $F2|s = \bar{s}|\sum \hat{C}_j$ is binary* NP*-complete.*

Proof. By reduction from Equal Cardinality Partition, as defined in the proof of Theorem 3.6. Consider the following instance of problem $F2|s = \bar{s}|\sum \hat{C}_j$.

$$\begin{array}{lcll} n & = & 2t+1 & \\ s & = & 2 & \\ D_1 & = & A(t+1)+1 & \\ D_2 & = & 2A(t+1)+1 & \\ p_{1j} & = & 0, & j = 1,\ldots,2t \\ p_{1j} & = & A(t+1), & j = 2t+1 \\ p_{2j} & = & A + a_j, & j = 1,\ldots,2t \\ p_{2j} & = & 1, & j = 2t+1 \\ C & = & (t+1)D_1 + tD_2. & \end{array}$$

We prove that there exists a schedule with $\sum \hat{C}_j \leq C$ if and only if there exists a solution to Equal Cardinality Partition.

($\Rightarrow$) If Equal Cardinality Partition has a solution S_1, S_2, then consider the permutation schedule defined by the sequence $(S_1, 2t+1, S_2)$. The last job of S_1 completes processing at time $tA + \sum_{j\in S_1} a_j = A(t+1) = D_1 - 1$, and job $2t+1$ completes processing at time D_1. Further, the last job of S_2 completes processing at time $D_1 + A(t+1) = D_2$. Thus, $\sum \hat{C}_j = (t+1)D_1 + tD_2 = C$.

($\Leftarrow$) Job $2t+1$ has an earliest possible dispatch time of D_1. The earliest time by which $t+1$ jobs of $\{1,\ldots,2t\}$ can complete processing is $(t+1)(A+1) > A(t+1)+1 = D_1$, which implies that at most t jobs of $\{1,\ldots,2t\}$ can be dispatched at time D_1. Thus, we obtain a lower bound on the total dispatch time for all jobs of $(t+1)D_1 + tD_2 = C$, and this lower bound can only be attained if job $2t+1$ together with the jobs of some set S_1, where $S_1 \subset \{1,\ldots,2t\}$ and $|S_1| = t$, are dispatched at time D_1, and the jobs of $S_2 = \{1,\ldots,2t\} \setminus S_1$ are dispatched at time D_2.

Since $\sum_{j=1}^{2t+1} p_{2j} = D_2$, there is no idle time on machine M_2 in any schedule with $\sum \hat{C}_j \leq C$. To achieve its dispatch time of D_1, job $2t+1$ is processed on machine M_2 in the interval $[A(t+1), A(t+1)+1]$. Therefore, the interval $[0, A(t+1)]$ on machine M_2 is exactly filled with the processing of the jobs of S_1. Thus, we obtain $\sum_{j\in S_1} p_{2j} = A(t+1)$, or equivalently $\sum_{j\in S_1} a_j = A$, which implies that S_1, S_2 is a solution to Equal Cardinality Partition. □

We now describe a dynamic programming algorithm, FBLOCK, which finds an optimal schedule for problem $F2|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective in $\mathcal{A}, \mathcal{B}, \mathcal{C}$ or $\mathcal{D}$.

In view of Theorem 2.2, we assume that the jobs are indexed according to the algorithm of Johnson (1954). Our dynamic programming algorithm, FBLOCK, assigns jobs to blocks, and sequences the jobs according to Johnson's algorithm. A complete schedule is obtained by concatenating the individual schedules for the blocks.

Algorithm FBLOCK

State variables

In addition to a variable j which indicates that jobs $1,\ldots,j$ have been scheduled, we also use state variables $(t_{11},t_{21},I_1,\ldots,t_{1\bar{s}},t_{2\bar{s}},I_{\bar{s}})$, where t_{ik} denotes the minimum time required to complete all jobs within block k on machine M_i, for $i=1,2$, and I_k denotes the idle time on machine M_2 in block k, for $k=1,\ldots,\bar{s}$. We associate with each state the value T_{ik}, which denotes the time that the last job of block k completes processing on machine M_i, for $i=1,2$, $k=1,\ldots,\bar{s}$. The values T_{ik} are related to the state variables through the equations $T_{11}=t_{11}$, $T_{21}=t_{21}$, $T_{1k}=T_{1,k-1}+t_{1k}$ and $T_{2k}=\max\{T_{1,k-1}+t_{2k},T_{2,k-1}+t_{2k}-I_k\}$, for $k=2,\ldots,\bar{s}$. When jobs $1,\ldots,j$ have been scheduled, we restrict the choice of state variables to satisfy $(t_{11},t_{21},I_1,\ldots,t_{1\bar{s}},t_{2\bar{s}},I_{\bar{s}})\in\mathcal{T}_j$, where $\mathcal{T}_j=\{(t_{11},t_{21},I_1,\ldots,t_{1\bar{s}},t_{2\bar{s}},I_{\bar{s}})|T_{2k}\le D_k,\ \sum_{k=1}^{\bar{s}}t_{1k}=\sum_{h=1}^{j}p_{1h},\ I_k\le t_{1k},\ t_{2k}\le\sum_{h=1}^{j}p_{2h}+I_k,\ \text{for } k=1,\ldots,\bar{s}\}$ for objectives in $\mathcal{A},\mathcal{B}$ and $\mathcal{D}$, and where $\mathcal{T}_j=\{(t_{11},t_{21},I_1,\ldots,t_{1\bar{s}},t_{2\bar{s}},I_{\bar{s}})|T_{2k}\le D_k,\ \sum_{k=1}^{\bar{s}}t_{1k}\le\sum_{h=1}^{j}p_{1h},\ I_k\le t_{1k},\ t_{2k}\le\sum_{h=1}^{j}p_{2h}+I_k,\ \text{for } k=1,\ldots,\bar{s}\}$ for objectives in $\mathcal{C}$.

Optimal value function

Let $g_j(t_{11},t_{21},I_1,\ldots,t_{1\bar{s}},t_{2\bar{s}},I_{\bar{s}})$ denote the minimum cost of scheduling jobs $1,\ldots,j$ such that the state $(t_{11},t_{21},I_1,\ldots,t_{1\bar{s}},t_{2\bar{s}},I_{\bar{s}})$ is achieved.

Recurrence relation

$\mathcal{A},\mathcal{B},\mathcal{D}$. $g_j(t)=\min\Big\{\min_{k\in K_1}\{g_{j-1}(\ldots,t_{1k}-p_{1j},t_{2k}-p_{2j},I_k,\ldots)\oplus f_j(D_k)\},$
$\min_{k\in K_2,\tau\in\{t_{1k}-p_{1j},\ldots,t_{1k}\}}\{g_{j-1}(\ldots,t_{1k}-p_{1j},\tau,I_k-(t_{1k}-\tau),\ldots)\oplus f_j(D_k)\}\Big\};$

$\mathcal{C}$. $g_j(t)=\min\Big\{\min_{k\in K_3}\{g_{j-1}(\ldots,t_{1k}-p_{1j},t_{2k}-p_{2j},I_k,\ldots)\oplus f_j(D_k)\},$
$\min_{k\in K_4,\tau\in\{t_{1k}-p_{1j},\ldots,t_{1k}\}}\{g_{j-1}(\ldots,t_{1k}-p_{1j},\tau,I_k-(t_{1k}-\tau),\ldots)\oplus f_j(D_k)\},g_{j-1}(t)+w_j\Big\},$
where $t=(t_{11},t_{21},I_1,\ldots,t_{1\bar{s}},t_{2\bar{s}},I_{\bar{s}})$, $K_1=\{k|k\in\{1,\ldots,\bar{s}\},t_{1k}+p_{2j}<t_{2k}\}$, $K_2=\{k|k\in\{1,\ldots,\bar{s}\},t_{1k}+p_{2j}=t_{2k}\}$, $K_3=\{k|k\in K_1,D_k\le|\lfloor d_j\rfloor|\}$, and $K_4=\{k|k\in K_2,D_k\le|\lfloor d_j\rfloor|\}$.

Boundary condition

$g_0(0,\ldots,0) = 0$, and all other values are set to ∞, for $\hat{\gamma} \neq \hat{L}_{\max}$.

$g_0(0,\ldots,0) = -\infty$, and all other values are set to ∞, for $\hat{\gamma} = \hat{L}_{\max}$.

Optimal solution

$$\min_{(t_{11},t_{21},I_1,\ldots,t_{1\bar{s}},t_{2\bar{s}},I_{\bar{s}})\in\mathcal{T}_n} \{g_n(t_{11},t_{21},I_1,\ldots,t_{1\bar{s}},t_{2\bar{s}},I_{\bar{s}})\}.$$

In the recurrence relation, K_1 and K_3 correspond to a situation where the time t_{1k} at which job j completes processing on machine M_1 is earlier than the time $t_{2k} - p_{2j}$ at which it starts processing on machine M_2. In this case, job j does not generate any additional idle time. On the other hand, K_2 and K_4 correspond to a situation where job j is processed in no-wait mode. In this case, $t_{1k} = t_{2k} - p_{2j}$, and the previous schedule for block k completes processing on machine M_1 at time $t_{1k} - p_{1j}$ and on machine M_2 at some time τ, where $t_{1k} - p_{1j} \leq \tau \leq t_{2k} - p_{2j}$. During the interval $[\tau, t_{1k}]$, machine M_2 is idle.

Theorem 5.2 *Algorithm FBLOCK finds an optimal schedule for problem $F2|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective in $\mathcal{A}, \mathcal{B}$ or $\mathcal{D}$ with time complexity $O(n(\max\{\sum p_{1j}, \sum p_{2j}\})^{3\bar{s}})$, and where $\hat{\gamma}$ is any objective in $\mathcal{C}$ with time complexity $O(n(\max\{\sum p_{1j}, \sum p_{2j}\})^{3\bar{s}+1})$.*

Proof. Since Theorem 2.2 shows that the jobs within each block may be sequenced according to Johnson's algorithm, the recurrence relation computes the cost of all possible state transitions and therefore finds an optimal schedule. Note that $I_k \leq \sum p_{1j}$ for $k = 1,\ldots,\bar{s}$. Also, $t_{1k} \leq \sum p_{1j}$ and $t_{2k} \leq \sum p_{2j} + I_k \leq 2\max\{\sum p_{1j}, \sum p_{2j}\}$, for $k = 1,\ldots,\bar{s}$. For objectives in $\mathcal{A}, \mathcal{B}$ and $\mathcal{D}$, $g_j(t_{11}, t_{21}, I_1, \ldots, t_{1\bar{s}}, t_{2\bar{s}}, I_{\bar{s}})$ is only computed for values $(t_{11}, t_{21}, I_1, \ldots, t_{1\bar{s}}, t_{2\bar{s}}, I_{\bar{s}})$ where $\sum_{k=1}^{\bar{s}} t_{1k} = \sum_{h=1}^{j} p_{1h}$. Thus, there are $O(\sum p_{1j})^{\bar{s}-1}$ states of type t_{1k}, and a total of $O((\max\{\sum p_{1j}, \sum p_{2j}\})^{3\bar{s}-1})$ states. For objectives in $\mathcal{C}$, there are $O((\max\{\sum p_{1j}, \sum p_{2j}\})^{3\bar{s}})$ states. The recurrence relation is applied for n values of j. Since $\bar{s}$ is a constant, each use of the recurrence relation for $g_j(t)$ requires $O(p_{1j})$ time. Thus, the time complexity of Algorithm FBLOCK is $O(n(\max\{\sum p_{1j}, \sum p_{2j}\})^{3\bar{s}})$ for objectives in $\mathcal{A}, \mathcal{B}$ and $\mathcal{D}$, and $O(n(\max\{\sum p_{1j}, \sum p_{2j}\})^{3\bar{s}+1})$ for objectives in $\mathcal{C}$. □

However, the following *NP*-completeness result shows that the corresponding problem with an arbitrary number of fixed delivery dates is less tractable.

Theorem 5.3 *The recognition version of problem* $F2|s|\sum \hat{C}_j$ *is unary* NP*-complete.*

Proof. By reduction from 3-Partition, as defined in the proof of Theorem 3.4. Consider the following instance of problem $F2|s|\sum \hat{C}_j$.

$$
\begin{array}{lcll}
n & = & 4t+1 & \\
s & = & t & \\
D_k & = & kty(3t+1)+k+1, & k=1,\ldots,t \\
p_{1j} & = & 0, & j=1,\ldots,3t \\
p_{1j} & = & ty(3t+1)+1, & j=3t+1,\ldots,4t \\
p_{1j} & = & 0 & j=4t+1 \\
p_{2j} & = & t(ty+a_j), & j=1,\ldots,3t \\
p_{2j} & = & 1, & j=3t+1,\ldots,4t+1 \\
C & = & D_1+4\sum_{k=1}^{s} D_k. &
\end{array}
$$

We prove that there exists a schedule with $\sum \hat{C}_j \leq C$ if and only if there exists a solution to 3-Partition.

($\Rightarrow$) Consider a permutation schedule in which both machines process jobs in the sequence $(4t+1, 1, 2, 3, 3t+1, 4, 5, 6, 3t+2, \ldots, 3t-2, 3t-1, 3t, 4t)$. Jobs 1, 2, 3, $3t+1$ and $4t+1$ are dispatched at time D_1, jobs 4, 5, 6 and $3t+2$ are dispatched at time D_2, and so on. Thus, $\sum \hat{C}_j = 5D_1 + 4(D_2 + \cdots + D_s) = C$.

($\Leftarrow$) Since $\sum_{j=3t+1}^{4t} p_{1j} + \min_{j\in\{3t+1,\ldots,4t\}}\{p_{2j}\} = t^2y(3t+1)+t+1 = D_s$, there is no idle time on machine M_1 in the interval $[0, t^2y(3t+1)+t]$ in any schedule with $\sum \hat{C}_j \leq C$. Jobs $3t+1,\ldots,4t$ are identical. Thus, we assume without loss of generality that machine M_1 processes job $3t+k$ in the interval $[(k-1)ty(3t+1)+k-1, kty(3t+1)+k]$, for $k=1,\ldots,s$.

The earliest possible dispatch time of job $3t+k$ is D_k, for $k=1,\ldots,s$, and the earliest possible dispatch time of job $4t+1$ is D_1. Therefore, a lower bound on the total dispatch time for jobs $3t+1,\ldots,4t+1$ is $D_1 + (D_1 + \cdots + D_s)$. The earliest time by which $3k+1$ jobs of $\{1,\ldots,3t\}$ can complete processing is $(3k+1)t(ty+1) > 3kt^2y + kty + k + 1 = D_k$, for $k=1,\ldots,s-1$. Thus, no more than $3k$ jobs of $\{1,\ldots,3t\}$ can by dispatched by time D_k, for $k=1,\ldots,s$, which gives a lower bound on the total dispatch time for jobs $1,\ldots,3t$ of $3(D_1+\cdots+D_s)$. In a schedule with $\sum \hat{C}_j \leq C$, both of these lower bounds on the total dispatch time must be attained. Thus, job $4t+1$ is dispatched at time D_1, and job $3t+k$ and three jobs of $\{1,\ldots,3t\}$ are dispatched at time D_k, for $k=1,\ldots,s$.

Since $\sum_{j=1}^{4t+1} p_{2j} = D_s$, there is no idle time on machine M_2 in a schedule with $\sum \hat{C}_j \leq C$. Further, since job $3t+k$ is dispatched at time D_k, it must be processed on machine M_2 in the interval $[kty(3t+1)+k, kty(3t+1)+k+1]$, for $k = 1, \ldots, s$. Without loss of generality, we assume that job $4t+1$ is processed in the interval $[0,1]$ on machine M_2. Let S_k denote the three jobs of $\{1, \ldots, 3t\}$ that are dispatched at time D_k, for $k = 1, \ldots, s$. Since the processing of jobs of S_k on machine M_2 exactly fills the interval $[(k-1)ty(3t+1)+k, kty(3t+1)+k]$, we have $\sum_{j \in S_k} t(ty + a_j) = ty(3t+1)$ or equivalently $\sum_{j \in S_k} a_j = y$. Thus, $S_1, \ldots, S_s$ is a solution to 3-Partition. $\square$

6 Job Shops

In this section, we describe a dynamic programming algorithm, JBLOCK, for several job shop problems with a constant number of fixed delivery dates. There are four types of jobs in problem $J2|s|\hat{\gamma}$, where $\hat{\gamma}$ is any objective in $\mathcal{A}, \mathcal{B}, \mathcal{C}$ or $\mathcal{D}$. We let J_i denote the set of jobs that require processing on machine M_i only, for $i = 1, 2$, and we let J_{12} (respectively, J_{21}) denote the set of jobs that require processing on machine M_1 followed by machine M_2 (resp., machine M_2 followed by machine M_1).

Block k of a given schedule, for $k = 1, \ldots, \bar{s}$, is partitioned into *groups* of jobs as follows. Let J_{12k}^1 (respectively, J_{21k}^1) denote the jobs of J_{12} (resp., J_{21}) that have their first operation within block k and their second operation within a later block, let J_{12k}^2 (respectively, J_{21k}^2) denote the jobs of J_{12} (resp., J_{21}) that have their second operation within block k and their first operation within an earlier block, let J_{12k}^0 (respectively, J_{21k}^0) denote the jobs of J_{12} (resp., J_{21}) that have both operations within block k, and let J_{1k} (respectively, J_{2k}) denote the jobs of J_1 (resp., J_2) that have their single operation within block k.

Lemma 6.1 *For problem $J2|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective defined in Section 1.1, suppose that all operations have been assigned to groups within blocks. Further suppose that within each group a sequence of operations has been determined. Then there exists an optimal schedule in which the sequence of groups of block k on machine M_1 is $(J_{12k}^0, J_{12k}^1, J_{1k}, J_{21k}^2, J_{21k}^0)$, and the sequence of groups of block k on machine M_2 is $(J_{21k}^0, J_{21k}^1, J_{2k}, J_{12k}^2, J_{12k}^0)$, for $k = 1, \ldots, \bar{s}$.*

Proof. An interchange argument shows that group J^0_{12k} precedes J^1_{12k} on machine M_1, since the second operation of jobs in the former group precedes that of jobs in the latter group. Similarly, J^2_{21k} precedes group J^0_{21k} on machine M_1. It follows from Jackson (1956) that groups J^0_{12k} and J^1_{12k} both precede group J_{1k} which in turn precedes both groups J^2_{21k} and J^0_{21k} on machine M_1. Similar arguments apply to the sequence of groups on machine M_2. □

In accordance with Lemma 6.1, we refer to $J^0_{12k}, J^1_{12k}, J_{1k}, J^2_{21k}, J^0_{21k}$ as groups $1, 2, 3, 4, 5$, respectively, within block k on machine M_1. Similarly, we refer to $J^0_{21k}, J^1_{21k}, J_{2k}, J^2_{12k}, J^0_{12k}$ as groups $1, 2, 3, 4, 5$, respectively, within block k on machine M_2. Also, we let v_{ik} denote the first job of group 4 of block k on machine M_i, for $i = 1, 2$, $k = 1, \ldots, \bar{s}$, where $v_{ik} = 0$ if the group is empty.

Lemma 6.2 *For problem $J2|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective defined in Section 1.1, there exists an optimal schedule in which for each block k:*

(a) the jobs of groups 1 and 5 are sequenced according to the algorithm of Jackson (1956);

(b) the jobs of group 2 in $\{v_{11}, \ldots, v_{2\bar{s}}\}$ are sequenced consistently with $(v_{11}, \ldots, v_{2\bar{s}})$ before other jobs within that group, and these other jobs are then sequenced arbitrarily;

(c) the jobs of group 3 are sequenced arbitrarily;

(d) the jobs of group 4, with the exception of the first job, are sequenced arbitrarily.

Proof. If there exists a feasible schedule for a given assignment of operations to groups, then a schedule that minimizes the completion time of each block k on both machines, given the schedule of earlier blocks, is feasible. We prove each part of the lemma in turn.

(a) Jackson (1956) shows that the algorithm he describes minimizes the completion time of the group 1 and group 5 jobs that have both operations within the same block, on both machines.

(b) Within group 2, at most one job starts its second operation before time D_k. An interchange argument shows that the sequence described in part (b) of the lemma minimizes any delay in starting the processing of the second operation of such a job.

(c) Since the group 3 jobs have only one operation, they may be sequenced arbitrarily.

(d) Within group 4, except for the first job, all processing of second operations starts after

time D_{k-1}, and therefore cannot be delayed by first operations that complete processing no later than time D_{k-1}. $\Box$

Lemma 6.1 suggests the following selection of state variables in our dynamic programming algorithm. We let t_{ikl} denote the total processing time of operations on machine M_i within group l of block k, where $l = 1,2,3,4,5$ denotes $J_{12}^0, J_{12}^1, J_1, J_{21}^2, J_{21}^0$ if $i = 1$, and $J_{21}^0, J_{21}^1, J_2, J_{12}^2, J_{12}^0$ if $i = 2$, respectively, for $k = 1, \ldots, \bar{s}$. Similarly, we let u_{ikl} denote the start time of the first operation on machine M_i within group l of block k, for $i = 1,2$, $k = 1, \ldots, \bar{s}$ and $l = 1,2,4,5$. Since the jobs of J_1 and J_2 have only one operation, we do not need state variables u_{ik3} to prevent the simultaneous processing of two operations of the same job. Finally, we let v_{ik} denote the first job of group 4 of block k on machine M_i, for $i = 1,2$, $k = 1, \ldots, \bar{s}$, where $v_{ik} = 0$ if the group is empty. We do not need t_{i14}, u_{i14} and v_{i1} for $i = 1,2$, since for block 1 no earlier block exists. Similarly, $t_{i\bar{s}2}$ and $u_{i\bar{s}2}$ can be eliminated from block $\bar{s}$ for $i = 1,2$. Without loss of generality, the u_{ikl} variables can be set so that there is no idle time within any group.

For problem $J2|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective defined in Section 1.1, we may restrict our attention to schedules in which both machines are not simultaneously idle, and consequently all processing is completed by time T, where $T = \sum(p_{1j} + p_{2j})$. We assume that $D_{s-1} < T$; if $D_{k-1} < T \leq D_k$ for some $k \in \{1, \ldots, s\}$, then we can set $s = k$ without affecting the cost of a schedule. The following algorithm solves this problem.

Algorithm JBLOCK

The following optimal value function and recurrence relation are defined for the values of u_{ikl} and v_{ik} described above, where $u_{ikl} \in \{0, \ldots, \min\{D_k, T\}\}$, $u_{i,k-1,5} \leq u_{ik1} \leq \cdots \leq u_{ik5}$ (with $u_{i05} = 0$), $u_{i11} = 0$, $v_{ik} \in \{0, \ldots, n\}$, and all nonzero v_{ik} values are distinct. The recurrence relation is evaluated for all such values of u_{ikl} and v_{ik}. Let $V = \{v_{11}, v_{21}, \ldots, v_{1\bar{s}}, v_{2\bar{s}}\}$. The jobs are indexed so that each job in V receives a smaller index than any job that is not in V. The jobs in V are indexed according to the sequence $(v_{11}, v_{21}, \ldots, v_{1\bar{s}}, v_{2\bar{s}})$ and the jobs not in V are indexed according to the sequence used on machine M_1 in the algorithm of Jackson (1956). The optimal schedule is the minimum cost schedule found over all the values of u_{ikl}

and v_{ik} specified above.

State variables

In addition to a variable j which indicates that jobs $1,\ldots,j$ have been scheduled, we also use state variables $(t_{111},\ldots,t_{113},t_{115},t_{211},\ldots,t_{213},t_{215},\ t_{121},\ldots,t_{125},t_{221},\ldots,t_{225},\ldots,t_{1,\bar{s}-1,1},\ldots,$ $t_{1,\bar{s}-1,5},\ t_{2,\bar{s}-1,1},\ldots,t_{2,\bar{s}-1,5},\ t_{1\bar{s}1},t_{1\bar{s}3},\ldots,t_{1\bar{s}5},t_{2\bar{s}1},t_{2\bar{s}3},\ldots,t_{2\bar{s}5})$, which we abbreviate to $(t_{111},$ $\ldots,t_{2\bar{s}5})$, where t_{ikl} is the total processing time of the operations within group l of block k on machine M_i, for $i=1,2,\ k=1,\ldots,\bar{s},\ l=1,\ldots,5$. When jobs $1,\ldots,j$ have been scheduled, we restrict the choice of state variables to satisfy $(t_{111},\ldots,t_{2\bar{s}5})\in\mathcal{T}_j$, where $\mathcal{T}_j=\{(t_{111},\ldots,t_{2\bar{s}5})|u_{ikl}+t_{ikl}\le u_{i,k,l+1},\ l=1,4,\ u_{ik2}+t_{ik2}+t_{ik3}\le u_{ik4},\ u_{ik5}+t_{ik5}\le u_{i,k+1,1}(k\ne \bar{s}),\ t_{ik4}=0 \text{ if } v_{ik}=0,\ u_{ik5}+t_{ik5}\le D_k,\ \text{for } k=1,\ldots,\bar{s};\ \sum_{k=1}^{\bar{s}}\sum_{l=1}^{5}t_{ikl}=\sum_{h=1}^{j}p_{ih}+\sum_{h\in V_j}p_{ih},\ \text{for } i=1,2\}$, where $V_j=\{j+1,\ldots,n\}\cap V$, for objectives in $\mathcal{A},\mathcal{B}$ and $\mathcal{D}$. For objectives in $\mathcal{C}$, the definition of $\mathcal{T}_j$ is similar, except that the last condition is replaced by $\sum_{k=1}^{\bar{s}}\sum_{l=1}^{5}t_{ikl}\le\sum_{h=1}^{j}p_{ih}+\sum_{h\in V_j}p_{ih}$, for $i=1,2$.

Optimal value function

Let $g_j(t_{111},\ldots,t_{2\bar{s}5})$ denote the minimum cost of scheduling jobs $1,\ldots,j$ such that the state $(t_{111},\ldots,t_{2\bar{s}5})$ is achieved.

Recurrence relation

$\mathcal{A},\mathcal{B},\mathcal{D}$. $g_j(t)=\min\Big\{\min_{k\in K_1}\{g_{j-1}(\ldots,t_{1k1}-p_{1j},\ldots,t_{2k5}-p_{2j},\ldots)\oplus f_j(D_k)\},$
$\min_{k_1,k_2\in\{1,\ldots,\bar{s}\},k_1<k_2}\{g_{j-1}(\ldots,t_{1k_12}-p_{1j},\ldots,t_{2k_24}-p_{2j},\ldots)\oplus f_j(D_{k_2})\}\Big\}$, for $j\in J_{12}\setminus V$;
$g_j(t)=\min_{k_1\in K_2}\{g_{j-1}(\ldots,t_{1k_12}-p_{1j},\ldots)\oplus f_j(D_{k_2})\}$, for $j\in J_{12}$ and $j=v_{2k_2}$;
$g_j(t)=\min\Big\{\min_{k\in K_3}\{g_{j-1}(\ldots,t_{1k5}-p_{1j},t_{2k1}-p_{2j},\ldots)\oplus f_j(D_k)\},$
$\min_{k_1,k_2\in\{1,\ldots,\bar{s}\},k_1<k_2}\{g_{j-1}(\ldots,t_{2k_12}-p_{2j}\ldots,t_{1k_24}-p_{1j},\ldots)\oplus f_j(D_{k_2})\}\Big\}$, for $j\in J_{21}\setminus V$;
$g_j(t)=\min_{k_1\in K_4}\{g_{j-1}(\ldots,t_{2k_12}-p_{2j},\ldots)\oplus f_j(D_{k_2})\}$, for $j\in J_{21}$ and $j=v_{1k_2}$;
$g_j(t)=\min_{k\in\{1,\ldots,\bar{s}\}}\{g_{j-1}(\ldots,t_{1k3}-p_{1j},\ldots)\oplus f_j(D_k)\}$, for $j\in J_1$;
$g_j(t)=\min_{k\in\{1,\ldots,\bar{s}\}}\{g_{j-1}(\ldots,t_{2k3}-p_{2j},\ldots)\oplus f_j(D_k)\}$, for $j\in J_2$,
where $t=(t_{111},\ldots,t_{2\bar{s}5})$, $K_1=\{k|k\in\{1,\ldots,\bar{s}\},u_{1k1}+t_{1k1}\le u_{2k5}+t_{2k5}-p_{2j}\}$, $K_2=\{k|k\in\{1,\ldots,k_2-1\},u_{1k2}+t_{1k2}\le u_{2k_24}\}$, $K_3=\{k|k\in\{1,\ldots,\bar{s}\},u_{2k1}+t_{2k1}\le u_{1k5}+t_{1k5}-p_{1j}\}$, and $K_4=\{k|k\in\{1,\ldots,k_2-1\},u_{2k2}+t_{2k2}\le u_{1k_24}\}$.

$\mathcal{C}$. Similar to $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{D}$, except that the $\oplus f_j(\cdot)$ term is removed, and we include the following

extra term in the minimization for each case above: $g_{j-1}(t) + w_j$.

Boundary condition

$g_0(t_{111}, \ldots, t_{2\bar{s}5}) = 0$ for $t_{ik1} = t_{ik2} = t_{ik3} = t_{ik5} = 0$ and $t_{ik4} = p_{iv_{ik}}$, where $p_{i0} = 0$, for $i = 1, 2$, $k = 1, \ldots, \bar{s}$, and all other values are set to ∞, for $\hat{\gamma} \neq \hat{L}_{\max}$.

$g_0(t_{111}, \ldots, t_{2\bar{s}5}) = -\infty$ for $t_{ik1} = t_{ik2} = t_{ik3} = t_{ik5} = 0$ and $t_{ik4} = p_{iv_{ik}}$, where $p_{i0} = 0$, for $i = 1, 2$, $k = 1, \ldots, \bar{s}$, and all other values are set to ∞, for $\hat{\gamma} = \hat{L}_{\max}$.

Optimal solution

$\min_{(t_{111}, \ldots, t_{2\bar{s}5}) \in \mathcal{T}_n} \{g_n(t_{111}, \ldots, t_{2\bar{s}5})\}$.

In the recurrence relation, K_1 defines those blocks for which job j, where $j \in J_{12}$, can occupy the last positions within group 1 on machine M_1 and within group 5 on machine M_2 without creating idle time on M_2. Similarly, K_3 defines such blocks for jobs in J_{21}. Finally, K_2 and K_4 define those blocks for which the scheduling of the first operation of job j does not create idle time before the second operation which has already been fixed, for $j \in J_{12}$ and $j \in J_{21}$, respectively.

Theorem 6.1 *Algorithm JBLOCK finds an optimal schedule for problem* $J2|s = \bar{s}|\hat{\gamma}$, *where* $\hat{\gamma}$ *is any objective in* $\mathcal{A}, \mathcal{B}$ *or* $\mathcal{D}$ *with time complexity* $O(n^{2\bar{s}-1}(\max\{\sum p_{1j}, \sum p_{2j}\})^{18\bar{s}-12})$, *and where* $\hat{\gamma}$ *is any objective in* $\mathcal{C}$ *with time complexity* $O(n^{2\bar{s}-1}(\max\{\sum p_{1j}, \sum p_{2j}\})^{18\bar{s}-10})$.

Proof. The recurrence relation computes the cost of all possible state transitions that are consistent with Lemmas 6.1 and 6.2, and therefore finds an optimal schedule.

Note that $t_{ikl} \leq \sum_{j=1}^{n} p_{ij}$ for $i = 1, 2$, $k = 1, \ldots, \bar{s}$, $l = 1, \ldots, 5$. There are $10\bar{s} - 4$ state variables of type t_{ikl}. Also, $u_{ikl} \leq T \leq 2\max\{\sum p_{1j}, \sum p_{2j}\}$ for $i = 1, 2$, $k = 1, \ldots, \bar{s}$, $l = 1, \ldots, 5$. Since $u_{i11} = 0$ for $i = 1, 2$, there are $8\bar{s} - 6$ state variables of type u_{ikl}. Also, for objectives in $\mathcal{A}, \mathcal{B}$ and $\mathcal{D}$, the recurrence relation is only computed for values of t satisfying $\sum_{k=1}^{\bar{s}} \sum_{l=1}^{5} t_{ikl} = \sum_{h=1}^{j} p_{ih} + \sum_{h \in V_j} p_{ih}$, for $i = 1, 2$. Thus, there are effectively $10\bar{s} - 6$ state variables of type t_{ikl} and a total of $18\bar{s} - 12$ state variables of types t_{ikl} and u_{ikl}. For objectives in $\mathcal{C}$, there are $18\bar{s} - 10$ state variables of these types. Each of the $2\bar{s} - 2$ states v_{ik} has n possible values. The recurrence relation is applied for n values of j. Since $\bar{s}$ is a constant, each use of the recurrence relation requires constant time. Thus, the time complexity of Algorithm

JBLOCK is $O(n^{2\bar{s}-1}(\max\{\sum p_{1j}, \sum p_{2j}\})^{18\bar{s}-12})$ for objectives in $\mathcal{A}, \mathcal{B}$ and $\mathcal{D}$. Similarly, the time complexity of Algorithm JBLOCK is $O(n^{2\bar{s}-1}(\max\{\sum p_{1j}, \sum p_{2j}\})^{18\bar{s}-10})$ for objectives in $\mathcal{C}$. $\Box$

Corollary 6.1 *An optimal schedule for problem $J2|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective defined in Section 1.1, and where the maximum number of operations of any job is a constant, can be found in pseudopolynomial time, by expanding the state space used in JBLOCK to account for each operation of a job.*

7 Open Shops

Recall that the recognition version of the classical problem $O2||\sum C_j$ is unary *NP*-complete (Achugbue and Chin, 1982). However, we now demonstrate that the corresponding fixed delivery date problem is only binary *NP*-complete.

Theorem 7.1 *The recognition version of problem $O2|s = \bar{s}|\sum \hat{C}_j$ is binary* NP*-complete.*

Proof. By reduction from Equal Cardinality Partition, as defined in the proof of Theorem 3.6. Consider the following instance of problem $O2|s = \bar{s}|\sum \hat{C}_j$.

$$
\begin{array}{lcll}
n & = & 2t+2 & \\
s & = & 2 & \\
D_1 & = & A(t+1)+2 & \\
D_2 & = & 2A(t+1)+2 & \\
p_{1j} & = & 0, & j = 1, \ldots, 2t \\
p_{1j} & = & A(t+1)+1, & j = 2t+1, 2t+2 \\
p_{2j} & = & A + a_j, & j = 1, \ldots, 2t \\
p_{2j} & = & 1, & j = 2t+1, 2t+2 \\
C & = & (t+1)D_1 + (t+1)D_2, &
\end{array}
$$

where we assume that $t > 1$. We prove that there exists a schedule with $\sum \hat{C}_j \leq C$ if and only if there exists a solution to Equal Cardinality Partition.

($\Rightarrow$) Let S_1, S_2 denote a solution to Equal Cardinality Partition. We schedule the jobs in the sequence $(S_1, S_2, 2t+1, 2t+2)$ on machine M_1, and in the sequence $(S_1, 2t+2, 2t+1, S_2)$ on machine M_2. All the t jobs of S_1 and job $2t+1$ complete by time D_1, and the t jobs of S_2 and job $2t+2$ complete by D_2. Therefore, $\sum \hat{C}_j = (t+1)D_1 + (t+1)D_2 = C$.

($\Leftarrow$) In a schedule with $\sum \hat{C}_j \leq C$, since $p_{1,2t+1} + p_{1,2t+2} = D_2$, there is no idle time on machine M_1. Jobs $2t+1$ and $2t+2$ are identical, so we assume without loss of generality that these jobs are processed on machine M_1 in the intervals $[0, (t+1)A+1]$ and $[(t+1)A+1, 2(t+1)A+2]$, respectively. Further, job $2t+2$ is processed for one unit of time on machine M_2 within the interval $[0, (t+1)A+1]$.

Jobs $2t+1$ and $2t+2$ have earliest possible dispatch times of D_1 and D_2, respectively. The earliest time by which $t+1$ jobs of $\{1, \ldots, 2t\}$ can complete processing is $(t+1)(A+1) > D_1$, which implies that at most t jobs of $\{1, \ldots, 2t\}$ can be dispatched at time D_1. Thus, we obtain a lower bound on the total dispatch time of all jobs of $(t+1)D_1 + (t+1)D_2$. This lower bound can only be attained if job $2t+1$ together with the jobs of some set S_1, where $S_1 \subset \{1, \ldots, 2t\}$ and $|S_1| = t$, are dispatched at time D_1, and job $2t+2$ together with the jobs of $S_2 = \{1, \ldots, 2t\} \setminus S_1$ are dispatched at time D_2.

Since $\sum_{j=1}^{2t+1} p_{2j} = D_2$, there is no idle time on machine M_2 in any schedule with $\sum \hat{C}_j \leq C$. To achieve its dispatch time of D_1, job $2t+1$ is processed on machine M_2 in the interval $[A(t+1)+1, A(t+1)+2]$. Therefore, the interval $[0, A(t+1)+1]$ on machine M_2 is exactly filled with the processing of the jobs of S_1 and job $2t+2$. Thus, we obtain $\sum_{j \in S_1} p_{2j} = A(t+1)$, or equivalently $\sum_{j \in S_1} a_j = A$, which implies that S_1, S_2 is a solution to Equal Cardinality Partition. $\Box$

We now describe a dynamic programming algorithm, OBLOCK, for several open shop problems with a constant number of fixed delivery dates. Each block k of a given schedule for problem $O2|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective in $\mathcal{A}, \mathcal{B}, \mathcal{C}$ or $\mathcal{D}$, is partitioned into *groups* of jobs as follows. Let O^1_{12k} (respectively, O^1_{21k}) denote the set of jobs that have their first operation processed by machine M_1 (respectively, M_2) within block k and their second operation within a later block. Also, we let O^2_{12k} (respectively, O^2_{21k}) denote the set of jobs that have their second operation processed by machine M_2 (resp., M_1) within block k and their first operation within an earlier block. Finally, let O^0_{12k} (respectively, O^0_{21k}) denote the set of jobs that are processed first by machine M_1 (resp., M_2) and have both operations within block k.

Lemma 7.1 *For problem $O2|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective defined in Section 1.1, suppose*

that all operations have been feasibly assigned to groups within blocks. Further suppose that within each group a sequence of operations has been determined. Then there exists an optimal schedule in which the sequence of groups of block k on machine M_1 is $(O^0_{12k}, O^1_{12k}, O^2_{21k}, O^0_{21k})$, and the sequence of groups of block k on machine M_2 is $(O^0_{21k}, O^1_{21k}, O^2_{12k}, O^0_{12k})$, for $k = 1, \ldots, \bar{s}$.

Proof. Once the order of the two operations of all jobs has been determined, the resulting problem is a job shop. The result then follows from Lemma 6.1. □

In accordance with Lemma 7.1, we refer to $O^0_{12k}, O^1_{12k}, O^2_{21k}, O^0_{21k}$ as groups $1, 2, 3, 4$, respectively, within block k on machine M_1. Similarly, we refer to $O^0_{21k}, O^1_{21k}, O^2_{12k}, O^0_{12k}$ as groups $1, 2, 3, 4$, respectively, within block k on machine M_2. Also, we let v_{ik} denote the first job of group 3 of block k on machine M_i, for $i = 1, 2$, $k = 1, \ldots, \bar{s}$, where $v_{ik} = 0$ if the group is empty.

Lemma 7.2 *For problem $O2|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective defined in Section 1.1, there exists an optimal schedule in which for each block k:*

(a) the jobs of groups 1 and 4 are sequenced according to the algorithm of Jackson (1956);

(b) the jobs of group 2 in $\{v_{11}, \ldots, v_{2\bar{s}}\}$ are sequenced consistently with $(v_{11}, \ldots, v_{2\bar{s}})$ before other jobs within that group, and these other jobs are then sequenced arbitrarily;

(c) the jobs of group 3, with the exception of the first job, are sequenced arbitrarily.

Proof. The result follows from Lemma 6.2. □

For the job shop, all jobs with two operations are partitioned into sets J_{12} and J_{21} as part of the problem instance, whereas a partition of jobs into sets O_{12} and O_{21} for the open shop is not known until a solution is specified. To account for this extra generality in the open shop, we first index jobs according to the algorithm of Johnson (1954), which defines the processing order for groups O^0_{12k} on machine M_1 and O^0_{12k} on machine M_2 for jobs assigned to O_{12}. For jobs assigned to O_{21}, the order is reversed, which we implement by using a backward scheduling procedure.

Lemma 7.1 suggests the following selection of state variables in our dynamic programming algorithm. We let t_{ikl} denote the total processing time of operations on machine M_i within

group l of block k, where $l = 1,2,3,4$ denotes $O_{12}^0, O_{12}^1, O_{21}^2, O_{21}^0$ if $i = 1$, and $O_{21}^0, O_{21}^1, O_{12}^2, O_{12}^0$ if $i = 2$, respectively, for $k = 1, \ldots, \bar{s}$. Similarly, we let u_{ikl} denote the start (respectively, finish) time of the first (resp., last) operation on machine M_i within group l of block k, for $i = 1, l = 1,2$ and $i = 2$, $l = 3,4$ (resp., $i = 1$, $l = 3,4$ and $i = 2$, $l = 1,2$). Finally, we let v_{ik} denote the first job of group 3 of block k on machine M_i, for $i = 1,2$, $k = 1, \ldots, \bar{s}$, where $v_{ik} = 0$ if the group is empty. Since we can set the start time of group 1 of block $k+1$ to be equal to the finish time of group 4 of block k on machine M_1 for $k = 1, \ldots, \bar{s}-1$, we can eliminate u_{1k1} for $k = 2, \ldots, \bar{s}$, and we can set $u_{111} = 0$. We do not need t_{i13}, u_{i13} and v_{i1} for $i = 1,2$, since for block 1 no earlier block exists. Similarly, $t_{i\bar{s}2}$ and $u_{i\bar{s}2}$ can be eliminated from block $\bar{s}$ for $i = 1,2$. Without loss of generality, the u_{ikl} variables can be set so that there is no idle time within any group.

We assume, as for problem $J2|s = \bar{s}|\hat{\gamma}$, that $D_{s-1} < T$, where $T = \sum(p_{1j} + p_{2j})$. The following algorithm solves problem $O2|s = \bar{s}|\hat{\gamma}$, where $\hat{\gamma}$ is any objective defined in Section 1.1.

Algorithm OBLOCK

The following optimal value function and recurrence relation are defined for the values of u_{ikl} and v_{ik} described above, where $u_{ikl} \in \{0, \ldots, \min\{D_k, T\}\}$, $u_{1,k-1,4} \leq u_{1k2} \leq u_{1k3} \leq u_{1k4}$ (with $u_{104} = 0$), $u_{2,k-1,4} \leq u_{2k1} \leq \cdots \leq u_{2k4}$ (with $u_{204} = 0$), $u_{111} = 0$, $u_{1\bar{s}4} = T$, $v_{ik} \in \{0, \ldots, n\}$, and all nonzero v_{ik} values are distinct. The recurrence relation is evaluated for all such values of u_{ikl} and v_{ik}. Let $V = \{v_{11}, \ldots, v_{1\bar{s}}\}$ and $V' = \{v_{21}, \ldots, v_{2\bar{s}}\}$. The jobs are indexed so that each job in V' receives a smaller index than any job that is not in V', and each job in V receives a larger index than any job that is not in V. The jobs in $V \cup V'$ are indexed according to the sequence $(v_{21}, \ldots, v_{2\bar{s}}, v_{1\bar{s}}, \ldots, v_{11})$, and the jobs not in $V \cup V'$ are indexed according to the sequence obtained by the algorithm of Johnson (1954). The optimal schedule is the minimum cost schedule found over all the values of u_{ikl} and v_{ik} specified above.

State variables

In addition to a variable j which indicates that jobs $1, \ldots, j$ have been scheduled, we also use state variables $(t_{111}, t_{112}, t_{114}, t_{211}, t_{212}, t_{214}, t_{121}, \ldots, t_{124}, t_{221}, \ldots, t_{224}, \ldots, t_{1,\bar{s}-1,1}, \ldots, t_{1,\bar{s}-1,4}, t_{2,\bar{s}-1,1}, \ldots, t_{2,\bar{s}-1,4}, t_{1\bar{s}1}, t_{1\bar{s}3}, t_{1\bar{s}4}, t_{2\bar{s}1}, t_{2\bar{s}3}, t_{2\bar{s}4})$, which we abbreviate to $(t_{111}, \ldots, t_{2\bar{s}4})$, where

t_{ikl} is the total processing time of the operations within group l of block k on machine M_i, for $i = 1, 2$, $k = 1, \ldots, \bar{s}$, $l = 1, \ldots, 4$. When jobs $1, \ldots, j$ have been scheduled, we restrict the choice of state variables to satisfy $(t_{111}, \ldots, t_{2\bar{s}4}) \in \mathcal{T}_j$, where $\mathcal{T}_j = \{t_{111}, \ldots, t_{2\bar{s}4}) | u_{1,k-1,4} + t_{1k1} \leq u_{1k2}$ (with $u_{104} = 0$), $u_{1k2} + t_{1k2} + t_{1k3} \leq u_{1k3}$, $u_{1k3} + t_{1k4} \leq u_{1k4}$ $(\leq D_k)$, $t_{1k3} = 0$ if $v_{1k} = 0$, $u_{2,k-1,4} + t_{2,k-1,4} + t_{2k1} \leq u_{2k1}$ (with $u_{204} = t_{204} = 0$), $u_{2k1} + t_{2k2} \leq u_{2k2}$, $u_{2k3} + t_{2k3} \leq u_{2k4}$, $u_{2k4} + t_{2k4} \leq D_k$, $t_{2k3} = 0$ if $v_{2k} = 0$, for $k = 1, \ldots, \bar{s}$; $\sum_{k=1}^{\bar{s}} \sum_{l=1}^{4} t_{ikl} = \sum_{h=1}^{j} p_{ih} + \sum_{h \in V_j} p_{ih}$, for $i = 1, 2\}$, where $V_j = \{j+1, \ldots, n\} \cap (V \cup V')$ for objectives in $\mathcal{A}, \mathcal{B}$ and $\mathcal{D}$. For objectives in $\mathcal{C}$, the definition of $\mathcal{T}_j$ is similar, except that the last condition is replaced by $\sum_{k=1}^{\bar{s}} \sum_{l=1}^{4} t_{ikl} \leq \sum_{h=1}^{j} p_{ih} + \sum_{h \in V_j} p_{ih}$, for $i = 1, 2$.

Optimal value function

Let $g_j(t_{111}, \ldots, t_{2\bar{s}4})$ denote the minimum cost of scheduling jobs $1, \ldots, j$ such that the state $(t_{111}, \ldots, t_{2\bar{s}4})$ is achieved.

Recurrence relation

$\mathcal{A}, \mathcal{B}, \mathcal{D}$. $g_j(t) = \min\Big\{ \min_{k \in K_1} \{g_{j-1}(\ldots, t_{1k1} - p_{1j}, \ldots, t_{2k4} - p_{2j}, \ldots) \oplus f_j(D_k)\}$,

$\min_{k_1, k_2 \in \{1, \ldots, \bar{s}\}, k_1 < k_2} \{g_{j-1}(\ldots, t_{1k_12} - p_{1j}, \ldots, t_{2k_23} - p_{2j}, \ldots) \oplus f_j(D_{k_2})\}$,

$\min_{k \in K_2} \{g_{j-1}(\ldots, t_{1k4} - p_{1j}, t_{2k1} - p_{2j}, \ldots,) \oplus f_j(D_k)\}$,

$\min_{k_1, k_2 \in \{1, \ldots, \bar{s}\}, k_1 < k_2} \{g_{j-1}(\ldots, t_{2k_12} - p_{2j}, \ldots, t_{1k_23} - p_{1j}, \ldots) \oplus f_j(D_{k_2})\}\Big\}$, for $j \notin V \cup V'$;

$g_j(t) = \min_{k_1 \in K_3} \{g_{j-1}(\ldots, t_{1k_12} - p_{1j}, \ldots) \oplus f_j(D_{k_2})\}$, for $j = v_{2k_2}$;

$g_j(t) = \min_{k_1 \in K_4} \{g_{j-1}(\ldots, t_{2k_12} - p_{2j}, \ldots) \oplus f_j(D_{k_2})\}$, for $j = v_{1k_2}$,

where $t = (t_{111}, \ldots, t_{2\bar{s}4})$, $K_1 = \{k | k \in \{1, \ldots, \bar{s}\}, u_{1,k-1,4} + t_{1k1} \leq u_{2k4} + t_{2k4} - p_{2j}\}$, $K_2 = \{k | k \in \{1, \ldots, \bar{s}\}, u_{2k1} - t_{2k1} + p_{2j} \leq u_{1k4} - t_{1k4}\}$, $K_3 = \{k | k \in \{1, \ldots, k_2 - 1\}, u_{1k2} + t_{1k2} \leq u_{2k_22}\}$, and $K_4 = \{k | k \in \{1, \ldots, k_2 - 1\}, u_{2k2} - t_{2k2} + p_{2j} \leq u_{1k_23} - t_{1k_23}\}$.

$\mathcal{C}$. Similar to $\mathcal{A}, \mathcal{B}$ and $\mathcal{D}$, except that the $\oplus f_j(\cdot)$ term is removed, and we include the following extra term in the minimization for each case above: $g_{j-1}(t) + w_j$.

Boundary condition

$g_0(t_{111}, \ldots, t_{2\bar{s}4}) = 0$ for $t_{ik1} = t_{ik2} = t_{ik4} = 0$ and $t_{ik3} = p_{iv_{ik}}$, where $p_{i0} = 0$, for $i = 1, 2$, $k = 1, \ldots, \bar{s}$, and all other values are set to ∞, for $\hat{\gamma} \neq \hat{L}_{\max}$.

$g_0(t_{111}, \ldots, t_{2\bar{s}4}) = -\infty$ for $t_{ik1} = t_{ik2} = t_{ik4} = 0$ and $t_{ik3} = p_{iv_{ik}}$, where $p_{i0} = 0$, for $i = 1, 2$, $k = 1, \ldots, \bar{s}$, and all other values are set to ∞, for $\hat{\gamma} = \hat{L}_{\max}$.

Optimal solution

$$\min_{(t_{111},\ldots,t_{2\bar{s}4})\in\mathcal{T}_n}\{g_n(t_{111},\ldots,t_{2\bar{s}4})\}.$$

For any group, by scheduling those jobs that are processed on machine M_1 before machine M_2 in increasing index order, the resulting sequence is consistent with Lemma 7.2. The other jobs are scheduled backwards, so that they are sequenced in reverse index order, which is again consistent with Lemma 7.2. In the recurrence relation, K_1 defines those blocks for which job j can occupy the last positions within group 1 on machine M_1 and within group 4 on machine M_2 without creating idle time on M_2. Similarly, K_2 defines those blocks for which the scheduling of job j in the first positions within group 1 on machine M_2 and within group 4 on machine M_1 does not create idle time on M_1. Finally, K_3 (respectively, K_4) defines those blocks for which the scheduling of the first operation of job j does not create idle time before the second operation which has already been fixed, for jobs with their first (resp., second) operation on machine M_1.

Theorem 7.2 *Algorithm OBLOCK finds an optimal schedule for problem* $O2|s=\bar{s}|\hat{\gamma}$, *where* $\hat{\gamma}$ *is any objective in* $\mathcal{A},\mathcal{B}$ *or* $\mathcal{D}$ *with time complexity* $O(n^{2\bar{s}-1}(\max\{\sum p_{1j},\sum p_{2j}\})^{15\bar{s}-11})$, *and where* $\hat{\gamma}$ *is any objective in* $\mathcal{C}$ *with time complexity* $O(n^{2\bar{s}-1}(\max\{\sum p_{1j},\sum p_{2j}\})^{15\bar{s}-9})$.

Proof. The recurrence relation computes the cost of all possible state transitions that are consistent with Lemmas 7.1 and 7.2, and therefore finds an optimal schedule.

Note that $t_{ikl}\leq\sum_{j=1}^{n}p_{ij}$ for $i=1,2$, $k=1,\ldots,\bar{s}$, $l=1,\ldots,4$. There are $8\bar{s}-4$ state variables of type t_{ikl}. Also, $u_{ikl}\leq T\leq 2\max\{\sum p_{1j},\sum p_{2j}\}$ for $i=1,2$, $k=1,\ldots,\bar{s}$, $l=1,\ldots,4$. Since $u_{111}=0$ and $u_{1\bar{s}4}=T$, there are $7\bar{s}-5$ state variables of type u_{ikl}. The remainder of the analysis follows that of Algorithm JBLOCK in the proof of Theorem 6.1. Thus, the time complexity of Algorithm OBLOCK is $O(n^{2\bar{s}-1}(\max\{\sum p_{1j},\sum p_{2j}\})^{15\bar{s}-11})$ for objectives in $\mathcal{A},\mathcal{B}$ and $\mathcal{D}$. Similarly, the time complexity of Algorithm OBLOCK is $O(n^{2\bar{s}-1}(\max\{\sum p_{1j},\sum p_{2j}\})^{15\bar{s}-9})$ for objectives in $\mathcal{C}$. □

However, the following *NP*-completeness result shows that the corresponding problem with an arbitrary number of fixed delivery dates is less tractable.

Theorem 7.3 *The recognition version of problem* $O2|s|\sum\hat{C}_j$ *is unary* NP*-complete.*

Proof. By reduction from 3-Partition, as defined in the proof of Theorem 3.4. Consider the following instance of problem $O2|s|\sum \hat{C}_j$.

$$
\begin{array}{lcll}
n & = & 4t+3 & \\
s & = & t+1 & \\
D_k & = & kty(3t+1)+k+1, & k=1,\ldots,t \\
D_k & = & 25t^4y+t^2y+t+1, & k=t+1 \\
p_{1j} & = & 0, & j=1,\ldots,3t \\
p_{1j} & = & ty(3t+1)+1, & j=3t+1,\ldots,4t \\
p_{1j} & = & 0, & j=4t+1,4t+2 \\
p_{1j} & = & 25t^4y-3t^3y+1, & j=4t+3 \\
p_{2j} & = & t(ty+a_j), & j=1,\ldots,3t \\
p_{2j} & = & 1, & j=3t+1,\ldots,4t+1 \\
p_{2j} & = & 25t^4y-3t^3y, & j=4t+2 \\
p_{2j} & = & 0, & j=4t+3 \\
C & = & D_1+4\sum_{k=1}^{s-1} D_k+2D_s. &
\end{array}
$$

Note that this instance is the same as that in the proof of Theorem 5.3, except that there are two additional jobs and one additional delivery date. We prove that there exists a schedule with $\sum \hat{C}_j \leq C$ if and only if there exists a solution to 3-Partition.

($\Rightarrow$) The calculations in the proof of Theorem 5.3 show that, if we process jobs on both machines in the sequence $(4t+1,1,2,3,3t+1,4,5,6,3t+2,\ldots,3t-2,3t-1,3t,4t,4t+2,4t+3)$, then jobs $1,\ldots,4t+1$ have a total dispatch time of $D_1+4\sum_{k=1}^{t} D_k$. Since jobs $4t+2$ and $4t+3$ both have dispatch times of D_s, we obtain $\sum \hat{C}_j = C$.

($\Leftarrow$) Since $\sum_{j=1}^{4t+3} p_{1j} = \sum_{j=1}^{4t+3} p_{2j} = D_s$, there is no idle time on either machine in any schedule with $\sum \hat{C}_j \leq C$. Suppose that some job of $\{1,\ldots,4t+1\}$ is processed after job $4t+3$ on machine M_1 or after job $4t+2$ on machine M_2. Since $p_{1,4t+3} > D_{s-1}$ and $p_{2,4t+2} > D_{s-1}$, we obtain $\sum \hat{C}_j > 3D_s > C$. Thus, in any schedule with $\sum \hat{C}_j \leq C$, job $4t+3$ is processed on machine M_1 in the interval $[t^2y(3t+1)+t, D_s]$ and job $4t+2$ is processed on machine M_2 in the interval $[t^2y(3t+1)+t+1, D_s]$. Since jobs $3t+1,\ldots,4t$ are identical, we assume without loss of generality that machine M_1 processes job $3t+k$ in the interval $[(k-1)ty(3t+1)+k-1, kty(3t+1)+k]$, for $k=1,\ldots,s-1$.

As in the proof of Theorem 5.3, we may assume that any schedule with $\sum \hat{C}_j \leq C$ has the following properties. Job $4t+1$ is processed on machine M_2 in the interval $[0,1]$ and is dispatched at time D_1. Job $3t+k$ and the jobs of some set S_k, where $S_k \subset \{1,\ldots,3t\}$, are

dispatched at time D_k, for $k = 1, \ldots, s-1$.

Since job $3t+1$ is dispatched at time D_1, it is processed on machine M_2 in the interval $[ty(3t+1)+1, ty(3t+1)+2]$. Thus, the interval $[1, ty(3t+1)+1]$ on machine M_2 is exactly filled with the processing of jobs of $S_1 \cup \{3t+2, \ldots, 4t\}$. Since this interval is of length $ty(3t+1)$ and the time to process the jobs of S_1 is a multiple of t, any part of this interval which is filled by jobs of $\{3t+2, \ldots, 4t\}$ also has length which is a multiple of t. However, since $p_{2j} = 1$ for $j \in \{3t+2, \ldots, 4t\}$, no jobs of this set can be processed in this interval. Therefore, we obtain $|S_1| = 3$ and $\sum_{j \in S_1} a_j = y$. Repetition of this argument for dispatch times $D_2, \ldots, D_{s-1}$ shows that $|S_k| = 3$ and $\sum_{j \in S_k} a_j = y$, for $k = 1, \ldots, s-1$. Thus, $S_1, \ldots, S_{s-1}$ is a solution to 3-Partition. $\square$

Some comments about problems $O3|s = \bar{s}|\hat{C}_{\max}$ and $O3|s|\hat{C}_{\max}$ appear in Section 8.

8 Concluding Remarks

This paper examines the solvability of scheduling problems in which jobs are only dispatched to customers at times that are fixed before the exact details of the schedule are determined. These problems are important in a variety of industrial applications that arise because of constraints on the resources needed for delivery. Moreover, there are several examples of a problem which has very different solvability when delivery dates are fixed, compared to the equivalent classical problem. These results, along with several practical considerations mentioned in the introduction, may lead to recommendations either for or against the use of a fixed delivery date schedule.

As shown in Table 1, we provide an almost complete "map" of the fixed delivery date problems that are defined by all the standard scheduling objectives and machine environments. Among the few open questions are the pseudopolynomial time solvability of problems $O3|s = \bar{s}|\hat{C}_{\max}$ and $O3|s|\hat{C}_{\max}$. However, the discovery of a pseudopolynomial time algorithm for either of those problems, and (unless *P=NP*) the contrapositive of Theorem 2.6, would imply a similar result for the classical problem $O3||C_{\max}$. Also, a proof that the recognition version of either of those fixed delivery date problems is unary *NP*-complete, and (unless *P=NP*) the contrapositive of Theorem 2.5, would imply a similar result for problem $O3||C_{\max}$. Thus,

problems $O3|s = \bar{s}|\hat{C}_{\max}$ and $O3|s|\hat{C}_{\max}$ are exactly equivalent in complexity to $O3||C_{\max}$, which is a well known open problem (Lawler *et al.*, 1993, Pinedo, 1995). The only other open question is the pseudopolynomial time solvability of problem $Pm|s|\sum \hat{T}_j$.

In view of the limited amount of earlier work on problems of this type, we believe that this paper opens up an important topic to future researchers in scheduling and in supply chain management. In particular, we believe that our demonstration in Section 2 of several techniques for proving general results in this area should be useful to other researchers. We now identify some of the most important research issues for future study. First, a number of the problems discussed are intractable. This motivates the design of enumerative algorithms, including the search for good bounding procedures. The design of simple but effective heuristics, and their computational and worst-case analyses, are also suggested. Work is needed to improve the running time of several pseudopolynomial time algorithms that we have described, to make them more practical. Also, an analysis similar to the one given here could be performed for preemptive scheduling problems. In addition, on-line scheduling problems with fixed delivery dates are worthy of investigation. We hope that our work will stimulate research in these directions.

Acknowledgments

This work was supported in part by NATO Collaborative Research Grant CRG 950773 and by the Summer Fellowship Program, Fisher College of Business, The Ohio State University. An Associate Editor and two anonymous referees provided helpful comments on an earlier draft of this paper.

References

Achugbue, J.O. and F.Y. Chin. 1982. Scheduling the Open Shop to Minimize Mean Flow Time. *SIAM Journal on Computing*, **11**, 709–720.

Blum, N., R.W. Floyd, V. Pratt, R.L. Rivest and R.E. Tarjan. 1973. Time Bounds for Selection. *Journal of Computer and Systems Sciences*, **7**, 448–461.

Garey, M.R. and D.S. Johnson. 1978. Strong *NP*-Completeness Results: Motivation, Examples and Implications. *Journal of the Association for Computing Machinery*, **25**, 499–508.

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

Garey, M.R., D.S. Johnson and R. Sethi. 1976. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, **1**, 117–129.

Garey, M.R., R.E. Tarjan and G.T. Wilfong. 1988. One-Processor Scheduling with Symmetric Earliness and Tardiness Penalties. *Mathematics of Operations Research*, **13**, 330–348.

Gonzalez, T. and S. Sahni. 1976. Open Shop Scheduling to Minimize Finish Time. *Journal of the Association for Computing Machinery*, **23**, 665–679.

Graham, R.L., E.L. Lawler, J.K. Lenstra and A.H.G. Rinnooy Kan. 1979. Optimization and Approximation in Deterministic Machine Scheduling: A Survey. *Annals of Discrete Mathematics*, **5**, 287–326.

Jackson, J.R. 1955. Scheduling a Production Line to Minimize Maximum Tardiness. Research Report 43, Management Science Research Project, University of California, Los Angeles.

Jackson, J.R. 1956. An Extension of Johnson's Results on Job Lot Scheduling. *Naval Research Logistics*, **3**, 201–203.

Johnson, S.M. 1954. Optimal Two- and Three-Stage Production Schedules With Setup Times Included. *Naval Research Logistics*, **1**, 61–67.

Karp, R.M. 1972. Reducibility Among Combinatorial Problems. In *Complexity of Computer Computations*, R.E. Miller and J.W. Thatcher (eds.), Plenum Press, New York, 85–103.

Lawler, E.L. 1977. A "Pseudopolynomial" Algorithm for Sequencing Jobs to Minimize Total Tardiness. *Annals of Discrete Mathematics*, **1**, 331–342.

Lawler, E.L., J.K. Lenstra and A.H.G. Rinnooy Kan. 1981. Minimizing Maximum Lateness in a Two-machine Open-shop. *Mathematics of Operations Research*, **6**, 153–158; Erratum, *Mathematics of Operations Research*, **7**, 1982, 635.

Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys. 1993. Sequencing and Scheduling: Algorithms and Complexity. In *Handbooks in Operations Research and Management Science, Volume 4: Logistics of Production and Inventory*, S.C. Graves, A.H.G. Rinnooy Kan and P. Zipkin (eds.), North-Holland, New York, 445–522.

Lawler, E.L. and J.M. Moore. 1969. A Functional Equation and its Application to Resource Allocation and Sequencing Problems. *Management Science*, **16**, 77–84.

Lenstra, J.K. 1977. *Sequencing by Enumerative Methods*, Mathematical Centre Tract 69, Centre for Mathematics and Computer Science, Amsterdam.

Lesaoana, M. 1991. *Scheduling with Fixed Delivery Dates.* Ph.D. Thesis, University of Southampton, U.K.

Matsuo, H. 1988. The Weighted Total Tardiness Problem with Fixed Shipping Times and Overtime Utilization. *Operations Research*, **36**, 293–307.

Moore, J.M. 1968. An n Job, one Machine Sequencing Algorithm for Minimizing the Number of Late Jobs. *Management Science*, **15**, 102–109.

Pinedo, M. 1995. *Scheduling: Theory, Algorithms and Systems.* Prentice-Hall, Englewood Cliffs, N.J.

Schonhäge, A., M. Patterson and N. Pippenger. 1976. Finding the Median. *Journal of Computer and Systems Sciences*, **13**, 184–199.

Smith, W.E. 1956. Various Optimizers for Single Stage Production. *Naval Research Logistics Quarterly*, **3**, 59–66.

Williamson, D.P., L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevast'janov and D.B. Shmoys. 1997. Short Shop Schedules. *Operations Research*, **45**, 288–294.

Problem $\alpha \mid\mid \hat{\gamma}$	Complexity of $\alpha \mid s = \bar{s} \mid \hat{\gamma}$	Complexity of $\alpha \mid s \mid \hat{\gamma}$
$1 \mid\mid \hat{C}_{\max}$	$O(n)$ Thm 2.5	$O(n)$ Thm 2.5
$1 \mid\mid \sum \hat{C}_j$	$O(n)$ Thm 3.2	$O(\min\{n \log n, ns\})$ Thm 3.2
$1 \mid\mid \sum w_j \hat{C}_j$	*BNPC*, $O(n(\sum p_j)^{\bar{s}-1})$ Thm 3.3, 3.1	*UNPC* Thm 3.4
$1 \mid\mid \hat{L}_{\max}$	$O(n)$ Thm 3.5	$O(\min\{n \log n, ns\})$ Thm 3.5
$1 \mid\mid \sum \hat{U}_j$	$O(n \log n)$ Cor 2.1	$O(n \log n)$ Cor 2.1
$1 \mid\mid \sum w_j \hat{U}_j$	*BNPC*, $O(n \sum p_j)$ Cor 2.2, 2.1	*BNPC*, $O(n \sum p_j)$ Cor 2.2, 2.1
$1 \mid\mid \sum \hat{T}_j$	*BNPC*, $O(\min\{n(\sum p_j)^{\bar{s}-1}, n^4 \sum p_j\})$ Thm 3.6, 3.7	*BNPC*, $O(n^4 \sum p_j)$ Thm 3.6, 3.7
$1 \mid\mid \sum w_j \hat{T}_j$	*BNPC*, $O(n(\sum p_j)^{\bar{s}-1})$ Thm 3.3, 2.4, 3.1	*UNPC* Thm 3.4, 2.4
$Pm \mid\mid \hat{\gamma}$, $\hat{\gamma} \in \{\hat{C}_{\max}, \hat{L}_{\max}, \sum \hat{C}_j\}$	*BNPC*, $O(n(\sum p_j)^{m-1})$ Thm 2.6, 2.3, 4.1	*BNPC*, $O(n(\sum p_j)^{m-1})$ Thm 2.6, 2.3, 4.1
$Pm \mid\mid \hat{\gamma}$, $\hat{\gamma} \in \{\sum \hat{U}_j, \sum w_j \hat{U}_j\}$	*BNPC*, $O(n(\sum p_j)^{m})$ Thm 2.6, 2.3, 4.1	*BNPC*, $O(n(\sum p_j)^{m})$ Thm 2.6, 2.3, 4.1
$Pm \mid\mid \sum \hat{T}_j$	*BNPC*, $O(n(\sum p_j)^{m\bar{s}-1})$ Thm 2.6, 2.3, 4.2	*BNPC*, Open Thm 2.6, 2.3
$Pm \mid\mid \sum \hat{\gamma}$, $\hat{\gamma} \in \{w_j \hat{C}_j, w_j \hat{T}_j\}$	*BNPC*, $O(n(\sum p_j)^{m\bar{s}-1})$ Thm 2.6, 2.3, 4.2	*UNPC* Thm 3.4, 2.4
$P \mid\mid \hat{\gamma}$	*UNPC* Thm 2.6, 2.3	*UNPC* Thm 2.6, 2.3
$F2 \mid\mid \hat{C}_{\max}$	$O(n \log n)$ Thm 2.5	$O(n \log n)$ Thm 2.5
$F2 \mid\mid \hat{\gamma}$, $\hat{\gamma} \in \{\sum \hat{C}_j, \sum w_j \hat{C}_j, \hat{L}_{\max}, \sum \hat{T}_j, \sum w_j \hat{T}_j\}$	*BNPC*, $O(n(\max\{\sum p_{1j}, \sum p_{2j}\})^{3\bar{s}})$ Thm 2.8, 2.4, 5.1, 5.2	*UNPC* Thm 2.8, 2.4, 5.3
$F2 \mid\mid \hat{\gamma}$, $\hat{\gamma} \in \{\sum \hat{U}_j, \sum w_j \hat{U}_j\}$	*BNPC*, $O(n(\max\{\sum p_{1j}, \sum p_{2j}\})^{3\bar{s}+1})$ Thm 2.8, 2.4, 5.2	*UNPC* Thm 2.8, 2.4
$F3 \mid\mid \hat{\gamma}$	*UNPC* Thm 2.6, 2.3	*UNPC* Thm 2.6, 2.3
$J2 \mid\mid \hat{C}_{\max}$	$O(n \log n)$ Thm 2.5	$O(n \log n)$ Thm 2.5
$J2 \mid\mid \hat{\gamma}$, $\hat{\gamma} \in \{\sum \hat{C}_j, \sum w_j \hat{C}_j, \hat{L}_{\max}, \sum \hat{T}_j, \sum w_j \hat{T}_j\}$	*BNPC*, $O(n^{2\bar{s}-1}(\max\{\sum p_{1j}, \sum p_{2j}\})^{18\bar{s}-12})$ Thm 2.8, 2.4, 5.1, 6.1	*UNPC* Thm 2.8, 2.4, 5.3
$J2 \mid\mid \hat{\gamma}$, $\hat{\gamma} \in \{\sum \hat{U}_j, \sum w_j \hat{U}_j\}$	*BNPC*, $O(n^{2\bar{s}-1}(\max\{\sum p_{1j}, \sum p_{2j}\})^{18\bar{s}-10})$ Thm 2.8, 2.4, 6.1	*UNPC* Thm 2.8, 2.4
$J3 \mid\mid \hat{\gamma}$	*UNPC* Thm 2.6, 2.3	*UNPC* Thm 2.6, 2.3
$O2 \mid\mid \hat{C}_{\max}$	$O(n)$ Thm 2.5	$O(n)$ Thm 2.5
$O2 \mid\mid \hat{\gamma}$, $\hat{\gamma} \in \{\sum \hat{C}_j, \sum w_j \hat{C}_j, \hat{L}_{\max}, \sum \hat{T}_j, \sum w_j \hat{T}_j\}$	*BNPC*, $O(n^{2\bar{s}-1}(\max\{\sum p_{1j}, \sum p_{2j}\})^{15\bar{s}-11})$ Thm 2.8, 2.4, 7.1, 7.2	*UNPC* Thm 2.8, 2.4, 7.3
$O2 \mid\mid \hat{\gamma}$, $\hat{\gamma} \in \{\sum \hat{U}_j, \sum w_j \hat{U}_j\}$	*BNPC*, $O(n^{2\bar{s}-1}(\max\{\sum p_{1j}, \sum p_{2j}\})^{15\bar{s}-9})$ Thm 2.8, 2.4, 7.2	*UNPC* Thm 2.8, 2.4
$O3 \mid\mid \hat{C}_{\max}$	*BNPC*, Open Thm 2.6	*BNPC*, Open Thm 2.6
$O \mid\mid \hat{\gamma}$	*UNPC* Thm 2.6, 2.3	*UNPC* Thm 2.6, 2.3

Table 1: Complexity of Scheduling Problems with Fixed Delivery Dates.