Fixed interval scheduling with third-party machines

Ilia Fridman, Mikhail Kovalyov, Erwin Pesch, Andrew Ryzhikov

Received: 2 November 2019 | Revised: 21 April 2020 | Accepted: 23 June 2020 | Published on: 8 August 2020

DOI: 10.1002/net.21973

RESEARCH ARTICLE

WILEY

Fixed interval scheduling with third-party machines

Ilia Fridman[1,2] | **Mikhail Y. Kovalyov**[2] | **Erwin Pesch**[1,3] | **Andrew Ryzhikov**[2,4]

[1]Center for Advanced Studies in Management, HHL Leipzig Graduate School of Management, Leipzig, Germany
[2]United Institute of Informatics Problems, National Academy of Sciences of Belarus, Minsk, Belarus
[3]Institute of Information Systems, Faculty III, University of Siegen, Siegen, Germany
[4]LIGM, Université Paris-Est, Marne-la-Vallée, France

Correspondence
Erwin Pesch, Institute of Information Systems, Faculty III, University of Siegen, 57068 Siegen, Germany.
Email: erwin.pesch@uni-siegen.de

Funding information
This research was supported by the Friede Springer foundation, Grant/Award Number: 6000108.

Abstract

We study a problem of scheduling n jobs on machines of two types: in-house machines and third-party machines. Scheduling on in-house machines incurs no additional costs, while using third-party machines implies costs depending on their number and the time of usage. Each job has a fixed time interval for being processed which can be divided and allocated among several machines, as long as there is only one machine processing the job at any time. No machine can process more than one job at a time. Jobs can be rejected, and they are of different importance that is reflected in the weight of each job. The objective is to find a subset of the jobs and the number of third-party machines for any period of time so that the accepted jobs can be feasibly scheduled, the total weight of the accepted jobs is maximized, and the total machine usage costs does not exceed a given upper bound. We also study a similar problem in which the objective is to maximize the total time at which at least one job is processed. Both problems are encountered in situations in which certain activities with given start and completion times have to be serviced by human operators. Examples are air traffic control and the monitoring safe vehicle unloading. Other examples are the employment of subcontractors in agriculture, construction or transportation. We will present NP-hardness proofs, polynomial and pseudo-polynomial optimal algorithms and an approximation algorithm for these problems and their special cases. These problems admit graph-theoretical interpretations associated with finding independent sets and a proper vertex coloring in interval graphs.

KEYWORDS

fixed interval scheduling, outsourcing, parallel machines, subcontracting, interval graphs, air traffic control

1 | INTRODUCTION

Small and medium-sized companies often reach the point where internal resources—both technological and human—are inadequate. Therefore, collaboration within the business network is becoming increasingly important. Among other things, subcontractors are employed in the event of a bottleneck in the main contractor's human resources or due to time constraints when the main contractor is overloaded. A subcontractor executes orders or parts of orders for the entrepreneur (i.e., the main contractor). The main contractor is the one who accepts the order from the client. The main entrepreneur is virtually intermediary. Subcontracting is therefore particularly suitable for entrepreneurs to compensate for weak customer acquisition. Subcontractors are primarily active in industries like construction, IT, travel and transportation, or agriculture. One of the most common and oldest examples is house building. Subcontractors are often commissioned here, for example, because there are not enough employees available for the construction contract. An entrepreneur receives the order to build the house, but has no capacity to accept another order. But instead of rejecting it, the entrepreneur hires a subcontractor. The costs for the entrepreneur consist

mainly of two components: a fixed cost share that only depends on the duration of the subcontractor's use of resources (e.g., machinery) and a variable cost share for the amount and duration of the resources (e.g., number of workers on each day) used. The entrepreneur is interested in keeping the number of external workers and their duration of employment low and will agree with the subcontractor on a cost limit that cannot be exceeded.

Problems studied in this paper generally arise in planning human operator services, where both the service quality and ergonomics of the operator's work substantially drop as the load of the operator with simultaneously serviced jobs exceeds a certain threshold. The operator's work costs are fixed if this threshold is not exceeded, but if it is exceeded then extra costs incur which depend on the number of the additional jobs and the time of their servicing. The problems can be formulated as follows.

From the jobs of a set $N = \{1, \dots, n\}$, some can be accepted for processing on machines of two types: *m in-house* machines and an unlimited number of *third-party* machines. No machine can process more than one job at a time. Each job j has a fixed processing time interval $(a_j, b_j]$ of length $d_j = b_j - a_j > 0$, and it can either be accepted and processed entirely and without interruption in this interval by the in-house and third-party machines or it can be rejected and not processed at all. Processing of a job can be distributed among several machines, but by a single machine at any time instant reflecting the fact that an operator may immediately continue the service of a colleague. Using in-house machines incurs no additional costs, while using x third-party machines over a time interval of length δ implies a costs $\mathrm{sgn}(x\delta)(\alpha + \beta x)\delta$, where $\mathrm{sgn}(\cdot)$ is the signum function and α and β are given numbers. Thus, if no third-party machine is used in any time interval, there are no costs, and if $x \geq 1$ third-party machines are used in a time interval of non-zero length, the corresponding costs are a linear function of x in each unit-time sub-interval of the latter interval. Each job j has a weight w_j. The objective is to find a set $X \subseteq N$ of accepted jobs, the number of third-party machines and an assignment of in-house and third-party machines to the accepted jobs over time so that the accepted jobs are feasibly scheduled, the total weight of the accepted jobs is maximized, and the total machine costs do not exceed a given upper bound U. All numerical data and function values are assumed to be non-negative integer numbers. We denote this problem as MAXWEIGHT and the value (maximum weight) of an optimal solution as W^*.

Furthermore, we study a problem which differs from MAXWEIGHT in that the goal is to maximize the total amount of time at which jobs are processed. We denote this problem as MAXCOVER and the value of an optimal solution as C^*. There is an $O(n\log n)$ time algorithm of Gupta et al. [16] for the interval graph coloring problem which can be used to find the minimum number of machines required to feasibly schedule all the jobs with fixed processing intervals. If this minimum number does not exceed m, then the algorithm in Gupta et al. [16] is optimal for both problems MAXWEIGHT and MAXCOVER. In what follows we assume that it is not the case.

Another example of MAXWEIGHT is the work planning of a flight dispatcher, who can qualitatively and ergonomically service at most m flights at the same time in his responsibility area. There are n flights to be serviced, and flight j is permanently present in the responsibility area in a given time interval $(a_j, b_j]$. If there are more than m flights at the same time in the responsibility area, then the ergonomics of the dispatcher's work drops, and he is paid a compensation for each unit-time interval of such a service, which is a linear function of the number of extra flights to be serviced simultaneously. Thus, the compensation includes a fixed part α and a variable part βx which is proportional to the number x of the extra flights. Both parts are paid for each time unit of the reduced ergonomic work. The compensation payment can be used to motivate the dispatcher and to evaluate the dispatching service quality. Each flight is associated with a value. The problem is to select a subset of flights to be serviced by the dispatcher so that the total value of the selected flights is maximized and the total compensation paid does not exceed a given upper bound. The upper bound bases on past experience as to balance service quality and the compensating payment.

An example of the problem MAXCOVER is similar. It is the work planning of an operator who monitors the safe unloading of vehicles (vessels, trains, buses) with the aid of an electronic video surveillance system. There are n vehicles to be monitored, and vehicle j is unloaded in a given time interval $(a_j, b_j]$. The operator can qualitatively and ergonomically monitor the unloading of m vehicles at the same time at most. If more than m vehicles are unloaded at the same time, the ergonomics of the monitoring process drops and the operator is paid a compensation for each unit-time interval of such service, which is a linear function of the number of extra vehicles monitored simultaneously. The problem is to select a subset of vehicles to be monitored by the operator so that the total duration of the intervals in which unloading of at least one vehicle is monitored, that is, the operator's productive time, is maximized and the total payment compensating low ergonomic service does not exceed a given upper bound.

Problems MAXWEIGHT and MAXCOVER belong to the category of *fixed interval scheduling* problems, see the different surveys and further results in Kolen et al. [19], Kovalyov et al. [20], Krumke et al. [21], Angelelli et al. [1], van Bevern et al. [4] and Bentert et al. [3]. These problems admit graph-theoretical interpretations, associated with finding *independent sets* and *a proper vertex coloring* in *interval graphs*. The set of vertices of the corresponding interval graph is the set of jobs, and an edge between two vertices exists if and only if the corresponding intervals intersect. MAXWEIGHT and MAXCOVER differ from fixed interval scheduling problems in that third-party machines exist and using them causes costs depending on their number and the length of their operating period. If $U = 0$, no third-party machine can be used, and problem MAXWEIGHT can be solved in $O(mn\log n)$ time by the algorithms of Bouzina and Emmons [6] and Carlisle and Lloyd [7], and problem MAXCOVER can be solved in $O(n\log n)$ time by the algorithm of Gupta et al. [16].

TABLE 1 Results

Problem	Results	Reference
MAXWEIGHT	NP-hard for $\alpha+\beta>0$	Section 2.1
	$O\left(n^{m+2}\max\left\{W, \frac{(m+1)^2}{n}\right\}\right)$	Section 2.2
	$O\left(n^{m+2}\max\left\{U, \frac{(m+1)^2}{n}\right\}\right)$	Section 2.3
MAXCOVER ($m\in\{0,1\}$)	NP-hard for $\alpha+\beta>0$	Section 3.1
	$O(n^{m+2}L)$	Section 3.2
	$O(n^{m+2}U)$	Section 3.3
	1/2-approximation	Section 3.4
MAXCOVER($m\geq 2$)	$O(n\log n)$	Section 3.5

The time window concept is a generalization of the fixed interval concept, according to which there is a flexibility in the time of performing a job in its time window. Studies of scheduling and vehicle routing problems with time windows were initiated by Schrage [30] and Bodin et al. [5]. Recent publications include Sarasola and Doerner [29], Gnegel and Fügenschuh [15], Mohammadi et al. [25], Chen et al. [8] and Lera-Romero et al. [23], among others.

Problems MAXWEIGHT, MAXCOVER and scheduling problems with *order acceptance* and *job rejection* are related. The difference is that, with reference to the latter two problems, a job can be either accepted (entirely processed on in-house machines) or rejected (entirely processed on third-party machines). Problems MAXWEIGHT and MAXCOVER allow a job can be rejected and not at all processed. If a job is accepted, then any part of this job can be processed on in-house and third-party machines. Besides, in the order acceptance and job rejection settings the jobs are not restricted to being processed in fixed time intervals. Recent results on scheduling with job rejection can be found in Bartal et al. [2], Engels et al. [12], Dosa and He [11], Shabtay et al. [31], Mnich and Wiese [24] and Hermelin et al. [17], and those on scheduling with order acceptance in Slotnick [32] and Zhong et al. [37].

A large part of the relevant literature deals with scheduling with sub-contracting. It differs from scheduling with order acceptance or job rejection in two ways. In addition to the rejection option, a job can be processed at a sub-contractor facility, and conditions of its processing such as job delivery time, or job sequence, or sub-contractor selection are considered. Again, problems of this kind do not include fixed time intervals for jobs and the possibility of processing the same job by in-house and third-party machines. Publications on scheduling with sub-contracting include Chen and Li [9], Qi [27], Mokhtari and Abadi [26], Zhong and Huo [36], Vairaktarakis [33], Wang et al. [35], Hezarkhani and Kubiak [18], Vairaktarakis and Aydinliyim [34], Ren et al. [28], and the references therein.

Fukunaga et al. [13] studied a *"rent-or-buy" scheduling and cost coloring problem*, in which machines belong to different classes and the machine usage costs depend on the class of the machine and on the set of jobs assigned to it. Two classes of in-house and third-party machines can be identified for problems MAXWEIGHT and MAXCOVER. However, the problem considered in Fukunaga et al. [13] cannot model the situation in which in-house and third-party machines work on the same set of jobs and the costs depend on the working duration of the third-party machines.

Our results for the problems MAXWEIGHT and MAXCOVER are described in Sections 2 and 3, respectively. Each problem is proved to be NP-hard in the ordinary sense and two dynamic programming algorithms of incomparable computational complexities are presented for each problem. It is convenient to introduce the following notation.

$$W=\sum_{i=1}^{n} w_i.$$

L is the number of integer points in the union of the intervals $(a_j, b_j], j\in N$.

Our results are summarized in Table 1.

The paper concludes with a short summary of the results and suggestions for future research. In this article, we will use the names "job" and "interval" interchangeably.

2 | PROBLEM MAXWEIGHT

We start with an NP-hardness proof in Section 2.1 and then proceed with two dynamic programming algorithms, denoted as **DPW1** and **DPW2**. The running time of **DPW1** is $O\left(n^{m+2}\max\left\{W, \frac{(m+1)^2}{n}\right\}\right)$ and it is described in Section 2.2. The running time of **DPW2** is $O\left(n^{m+2}\max\left\{U, \frac{(m+1)^2}{n}\right\}\right)$ and it is described in Section 2.3.

2.1 | Ordinary NP-hardness

Theorem 1. MAXWEIGHT *is NP-hard even if the number of in-house machines m is any non-negative constant, α and β are any constants satisfying α + β > 0, and* $w_j = b_j - a_j, j = 1, \ldots, n$.

Proof. We use a reduction from the following NP-complete problem Partition (Garey and Johnson [14]): Given $k+1$ positive integer numbers $p_1, \ldots, p_k$ and P such that $\sum_{j=1}^{k} p_j = 2P$, is there a subset $X \subseteq K = \{1, \ldots, k\}$ such that $\sum_{j \in X} p_j = P$?

Given an instance of Partition, construct an instance of MAXWEIGHT, in which the number of in-house machines m is a given non-negative integer number, α and β are given numbers satisfying $\alpha + \beta > 0$, and the number of jobs is $n = (m+1)k$. The jobs are partitioned into k *groups* numbered $1, \ldots, k$. Each group j contains $m+1$ identical jobs associated with the same interval of length p_j and the same weight $w_j = p_j, j = 1, \ldots, k$. It is required that intervals of jobs from any two different groups do not intersect. We show that the instance of Partition has a solution if and only if there exists a solution of the constructed instance of MAXWEIGHT with the total weight of at least $(2m+1)P$ and the total machine costs of at most $(\alpha + \beta)P$.

Let X be a solution of the instance of Partition. Construct a solution of the instance of MAXWEIGHT, in which the subset of the selected jobs includes all $m+1$ jobs of each group $j, j \in X$, and it includes m jobs of each group $j, j \in K \backslash X$. The total weight of the selected jobs is $(m+1)\sum_{j \in X} p_j + m\sum_{j \in K \backslash X} p_j = 2mP + \sum_{j \in X} p_j = (2m+1)P$. Since the processing intervals of jobs from any two different groups do not intersect, no more than one third-party machine is needed at any time, and therefore, the total machine costs of the selected jobs are equal to $(\alpha + \beta)\sum_{j \in X} p_j = (\alpha + \beta)P$. Hence, the selected jobs are a solution of the instance of MAXWEIGHT.

Let Y be the set of the selected jobs in a solution of the instance of MAXWEIGHT with the total weight of at least $(2m+1)P$ and the total machine costs of at most $(\alpha + \beta)P$. Observe that if r, $0 \le r \le m-1$, jobs of group j are present in Y for any j, then, since processing intervals of the jobs from any two different groups do not intersect, and the use of in-house machines has zero costs, $m - r$ such jobs can be added to Y with no decrease of the total weight and no increase of the total costs. Therefore, assume without loss of generality that if a job of group j is in Y, then either $m+1$ or m such jobs are in Y. Denote by $X(Y)$ the set of indices $j \in K$ for which Y includes exactly $m+1$ jobs of group j. For the total weight of the jobs in $X(Y)$, we must have $(m+1)\sum_{j \in X(Y)} p_j + m\sum_{j \in K \backslash X(Y)} p_j = 2mP + \sum_{j \in X(Y)} p_j \ge (2m+1)P$. Hence, $\sum_{j \in X(Y)} p_j \ge P$. Furthermore, for the total machine costs, we must have $(\alpha + \beta)\sum_{j \in X(Y)} p_j \le (\alpha + \beta)P$, which implies $\sum_{j \in X(Y)} p_j \le P$. We deduce $\sum_{j \in X(Y)} p_j = p$, which means that $X(Y)$ is a solution of the instance of PARTITION. ■

We now describe two dynamic programming algorithms of incomparable computational complexities for the problem MAXWEIGHT. In the algorithms it is assumed that the jobs are re-numbered in a non-decreasing order of the left endpoints of their intervals such that $a_1 \le \cdots \le a_n$, and the following notations are used.

- $M = \{1, \ldots, m+1\}$.
- $B_j = \{0, b_1, \ldots, b_j\}, j \in N$.
- $e^{(0)}$ is vector with $m+1$ components such that $e^{(0)} = (0, \ldots, 0)$.
- $e^{(1)}$ is vector with $m+1$ components such that $e^{(1)} = (b_1, 0, \ldots, 0)$.
- $c^{(1)} = \max\{0, (1-m)(\alpha + \beta)d_1\}$.

2.2 | Algorithm DPW1

In the algorithm **DPW1**, partial solutions are constructed by considering the jobs in the order $1, \ldots, n$ and deciding for each job j (interval $(a_j, b_j]$) whether to accept or reject it. Once the decision has been made, it will not be changed later. Each partial solution is characterized by a *state* (j, w, r) and a *cost function* $F_j(w, r)$, where j is the number of jobs considered so far, w is the total weight of the accepted jobs, $r = (r_1, \ldots, r_{m+1})$, and r_i is the latest endpoint of the accepted intervals, which belongs to at least i accepted intervals, $i \in M$. If an endpoint satisfying this condition does not exist, then $r_i := 0$ by the definition, $i \in M$. If $r_i = r_{i+1}$, $1 \le i \le m$, then we assume that r_i corresponds to a job accepted later, and r_{i+1} corresponds to a job accepted earlier. For example, if $m = 2$ and intervals $(a_1, b_1] = (0, 3]$, $(a_2, b_2] = (1, 2]$ and $(a_3, b_3] = (2, 3]$ were accepted in this order, then $r_1 = b_3 = 3$, $r_2 = b_1 = 3$ and $r_3 = 0$. The value of $F_j(w, r)$ is equal to the minimum total costs of the partial solutions in the same state (j, w, r).

Consider a partial solution in the state (j, w, r) and assume that it is obtained from a partial solution in the state $(j-1, w^0, r^0)$ by making a decision about job j (interval $(a_j, b_j]$). Observe that.

$$r_1^0 \ge \cdots \ge r_{m+1}^0 \quad \text{and} \quad r_1 \ge \cdots \ge r_{m+1} \tag{1}$$

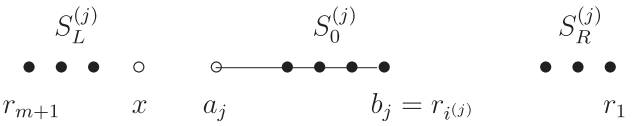

FIGURE 1 The case $b_j \geq r_{m+1}$. Sets $S_L^{(j)}$, $S_0^{(j)}$ and $S_R^{(j)}$

by the definition of the state variables. Denote by l_i^0 the left endpoint of the interval whose right endpoint is r_i^0. Due to the consideration of jobs in the order $1, \ldots, n$,

$$a_j \geq \max_{i \in M}\{l_i^0\} \tag{2}$$

at any state of the dynamic programming.

If job j has been rejected in state (j, w, r), then the previous state is $(j-1, w, r)$, and $F_j(w, r) = F_{j-1}(w, r)$. Assume that job j has been accepted. The previous state is $(j-1, w-w_j, r^0)$. Consider the current state (j, w, r). It is obvious that if $b_j < r_{m+1}$, then $r^0 = r$ and $F_j(w, r) = F_{j-1}(w-w_j, r) + \beta d_j$.

Consider the case $b_j \geq r_{m+1}$. Define the following sets of indices after job j has been accepted.

$$S_L^{(j)} = \{i | r_i < a_j, i \in M\}, S_0^{(j)} = \{i \mid r_i \in (a_j, b_j], i \in M\}, \quad S_R^{(j)} = \{i | b_j < r_i, i \in M\}. \tag{3}$$

These definitions are illustrated in Figure 1. Numbers r_i with indices from $S_L^{(j)}$, $S_0^{(j)}$ and $S_R^{(j)}$ are represented by the filled circles.

Denote $i^{(j)} := \min\{i | i \in S_0^{(j)}\}$. Since job j is accepted last, we have $r_{i^{(j)}} = b_j$. For a set of indices $S \in M$, define $r(S)$ as a sequence of right endpoints r_i, $i \in S$, in the non-increasing order. If $S = \emptyset$, then define $r(S) = \emptyset$. It follows from Equations (1-3) that, if job j is accepted, then the previous state vector r^0 is.

$$r^0 = (r(S_R^{(j)}), r(S_0^{(j)} \backslash \{j\}), x, r(S_L^{(j)})), \tag{4}$$

where x is the $| S_R^{(j)} \cup S_0^{(j)} |$-th component of vector r^0 such that $\max\{r_i | i \in S_L^{(j)}\} \leq x$. Note that $x = 0$ if $\max\{r_i | i \in S_L^{(j)}\} = 0$.

Below we show that, after job j has been accepted, the change of the total costs when passing from state $(j-1, w-w_j, r^0)$ to state (j, w, r) depends only on j, r_m^0 and r_{m+1}^0. Denote this change as $\Delta_j(r_m^0, r_{m+1}^0) = F_j(w, r) - F_{j-1}(w-w_j, r^0)$. Taking into account (1) and (2), we have

$$\Delta_j(r_m^0, r_{m+1}^0) = \begin{cases} 0, & \text{if } r_m^0 \leq a_j, \\ (\alpha+\beta)(\min\{r_m^0, b_j\} - a_j), & \text{if } r_{m+1}^0 \leq a_j < r_m^0, \\ (\alpha+\beta)(\min\{r_m^0, b_j\} - r_{m+1}^0) + \beta(\min\{r_{m+1}^0, b_j\} - a_j), & \text{if } a_j < r_{m+1}^0. \end{cases}$$

Note that if $r_m^0 \leq a_j$, then job j is fully processed on an in-house machine. If $r_{m+1}^0 \leq a_j < r_m^0$, then job j is processed in the time interval $(r_m^0, \max\{b_j, r_m^0\})$ on an in-house machine and it is processed in the interval $(a_j, \min\{r_m^0, b_j\})$ on a third-party machine. The fixed cost α is accounted for the latter interval because no third-party machine was used in this interval before job j was assigned. If $a_j < r_{m+1}^0$, then job j is processed in the time interval $(r_m^0, \max\{b_j, r_m^0\})$ on an in-house machine and it is processed in the interval $(a_j, \min\{r_m^0, b_j\})$ on a third-party machine. The fixed cost α is accounted for the interval $(r_{m+1}^0, \min\{r_m^0, b_j\})$ and it is not accounted for the interval $(a_j, \min\{r_{m+1}^0, b_j\})$ because it was accounted for this interval before job j was assigned.

Since $F_j(w, r) = F_{j-1}(w-w_j, r^0) + \Delta_j(r_m^0, r_{m+1}^0)$ and $\Delta_j(r_m^0, r_{m+1}^0) \geq 0$, we deduce that a partial solution in the state (j, w, r), at which the minimum costs $F_j(w, r)$ are attained, *dominates* all other partial solutions in the same state in the sense that if there is a solution in this state that can be extended to an optimal solution, then the dominant solution can be extended in the same way to an optimal solution as well. Note that, while the change of the total costs depend only on r_m^0 and r_{m+1}^0, these values can only be recursively calculated if the whole vector r^0 is maintained.

We are now able to formally describe algorithm **DPW1**. The initialization is $F_1(0, e^{(0)}) = 0$, $F_1(w_1, e^{(1)}) = c^{(1)}$ and $F_1(w, r) = U+1$ if $(w, r) \notin \{(0, e^{(0)}), (w_1, e^{(1)})\}$, $0 \leq w \leq W$, $r = (r_1, \ldots, r_{m+1})$, $r_i \in B_n$, $i = 1, \ldots, m+1$. For each $j = 2, 3, \ldots, n$, generate the following two sets of vectors $r = (r_1, \ldots, r_{m+1})$:

$$R_{(r_{m+1} > b_j)} = \{r | r_1 \geq \cdots \geq r_{m+1} > b_j, r_i \in B_n, i \in M\},$$

$$R_{(r_{m+1} \leq b_j)} = \{r | r_1 \geq \cdots \geq r_{i^{(j)}} = b_j > r_{i^{(j)}+1} \geq r_{i^{(j)}+2} \geq \cdots \geq r_{m+1}, r_i \in B_n, i \in M\}.$$

Denote $n_j := | \{i | r_i < b_j, r_i \in B_n, i \in M\} |$. We have

$$| R_{(r_{m+1} > b_j)} | \leq O((n-n_j)^{m+1}) \leq O(n^{m+1}), j = 2, 3, \ldots, n,$$

$$| R_{(r_{m+1}\le b_j)} | \le \sum_{i=1}^{m+1} O((n-n_j-1)^i (n_j)^{m-i}) | \le O((m+1)n^m), j = 2, 3, \ldots, n.$$

All the sets $R_{(r_{m+1}>b_j)}$ and $R_{(r_{m+1}\le b_j)}$, $j = 2, 3, \ldots, n$ can be generated in $O(n^{m+2})$ time. For each $r \in R_{(r_{m+1}\le b_j)}$ and $j \in \{2, 3, \ldots, n\}$, generate sets $S_R^{(j)}$, $S_0^{(j)}$ and $S_L^{(j)}$ satisfying Equation (3), calculate vectors r^0 satisfying Equation (4) and values $\Delta_j(r_m^0, r_{m+1}^0)$ satisfying Equation (5), which can be done in $O(m+1)$ time for fixed r and j, and hence, in $O((m+1)^2 n^{m+1})$ time for all $r \in R_{(r_{m+1}\le b_j)}$ and $j = 2, 3, \ldots, n$.

The recursion for $j = 2, 3, \ldots, n$, $0 \le w \le W$, and $r \in R_{(r_{m+1}>b_j)} \cup R_{(r_{m+1}\le b_j)}$ is as follows.

$$F_j(w,r) = \min \begin{cases} F_{j-1}(w,r), & \\ F_{j-1}(w-w_j, r) + \beta d_j, & \text{if } r \in R_{(r_{m+1}>b_j)}, \\ F^0, & \text{if } r \in R_{(r_{m+1}\le b_j)}, \end{cases}$$

where $F^0 = \min\{F_{j-1}(w-w_j, r^0) + \Delta_j(r_m^0, r_{m+1}^0) \mid r^0 = (r(S_R^{(j)}), r(S_0^{(j)} \backslash \{j\}), x, r(S_L^{(j)})), x \in B_{j-1}, \min\{r_i | i \in S_0^{(j)} \backslash \{j\}\} \ge x \ge \max\{r_i | i \in S_L^{(j)}\}\}$. Here and below the term under the minimum is skipped if the condition on the right hand side of it is not satisfied. If the minimum in the recursion is attained at the upper term, then job j is rejected. If it is attained at the middle term, then job j is accepted and it is fully processed on a third-party machine. If this minimum is attained at the lower term, then job j is accepted and the machines and the time intervals where it is processed either on an in-house machine or a third-party machine are determined by the value $\Delta_j(r_m^0, r_{m+1}^0)$ via (5).

The maximum total weight is equal to

$$W^* = \max\{w \mid F_n(w,r) \le U, w = 0, 1, \ldots, W, r = (r_1, \ldots, r_{m+1}), r_i \in B_n, i = 1, \ldots, m+1\}.$$

The corresponding set of accepted intervals and partition of each accepted interval into sub-intervals processed either on an in-house machine or a third-party machine can be found by tracing back optimal solutions of the recursive equation. Then, the $O(n\log n)$ time algorithm of Gupta et al. [16] can be used twice to find an optimal assignment of the "in-house" sub-intervals to the particular in-house machines and to find an optimal assignment of the "third-party" sub-intervals to the particular third-party machines.

The running time of the algorithm **DPW1** is

$$O\left((m+1)^2 n^{m+1} + W \sum_{j=2}^{n} \max\{R_{(r_{m+1}>b_j)} | B_{j-1} | \cdot, R_{(r_{m+1}\le b_j)}\}\right) = O\left(n^{m+2} \max\left\{W, \frac{(m+1)^2}{n}\right\}\right).$$

2.3 | Algorithm DPW2

Algorithm **DPW2** differs from **DPW1** in that the roles of the cost function $F_j(w, r)$ and the state variable w are switched. In **DPW2**, similar to **DPW1**, partial solutions are constructed by considering jobs in the order $1, \ldots, n$ and deciding for each job j of whether to accept it or reject. Once the decision has been made, it will not be changed later. Each partial solution is characterized by a *state* (j, c, r) and a *weight function* $G_j(c, r)$, where c is the total machine usage costs, and j and r are the same as in **DPW1**. The value of $G_j(c, r)$ is equal to the maximum total weight of the partial solutions in the same state (j, c, r).

Initialization is $G_1(0, e^{(0)}) = 0$, $G_1(c^{(1)}, e^{(1)}) = w_1$, and $G_1(c, r) = -\infty$ if $(c, r) \notin \{(0, e^{(0)}), (c^{(1)}, e^{(1)})\}$, $0 \le c \le U$, $r = (r_1, \ldots, r_{m+1})$, $r_i \in B_n$, $i = 1, \ldots, m+1$. Generate sets $R_{(r_{m+1}>b_j)}$, $R_{(r_{m+1}\le b_j)}$, $S_R^{(j)}$, $S_0^{(j)}$ and $S_L^{(j)}$, vectors r^0 satisfying Equation (4) and values $\Delta_j(r_m^0, r_{m+1}^0)$ satisfying Equation (5) in the same way as in algorithm **DPW1**. The recursion for $j = 2, \ldots, n$, $0 \le c \le U$ and $r \in R_{(r_{m+1}>b_j)} \cup R_{(r_{m+1}\le b_j)}$ is as follows.

$$G_j(c,r) = \max \begin{cases} G_{j-1}(c,r), & \\ G_{j-1}(c-\beta d_j, r) + w_j, & \text{if } r \in R_{(r_{m+1}>b_j)}, \text{ and } c \ge \beta d_j \\ G^0, & \text{if } r \in R_{(r_{m+1}\le b_j)} \text{ and } c \ge \Delta_j(r_m^0, r_{m+1}^0). \end{cases}$$

where $G^0 = \max\{G_{j-1}(c - \Delta_j(r_m^0, r_{m+1}^0), r^0) + w_j \mid r^0 = (r(S_R^{(j)}), r(S_0^{(j)} \backslash \{j\}), x, r(S_L^{(j)})), x \in B_{j-1}, \min\{r_i | i \in S_0^{(j)} \backslash \{j\}\} \ge x \ge \max\{r_i | i \in S_L^{(j)}\}\}$. The maximum weight is equal to

$$W^* = \max\{G_n(c,r) \mid 0 \le c \le U, r = (r_1, \ldots, r_{m+1}), r_i \in B_n, i = 1, \ldots, m+1\}$$

and the corresponding optimal solution can be found in the same way as in algorithm **DPW1**. The running time of algorithm **DPW2** is $O\left(n^{m+2} \max\left\{U, \frac{(m+1)^2}{n}\right\}\right)$.

3 | PROBLEM MAXCOVER

We begin with an important "at most two machines" property of the problem MAXCOVER and a proof of its ordinary NP-hardness for the case $m \in \{0, 1\}$ in Section 3.1. We denote special cases of the problem MAXCOVER in which $m \in \{0, 1\}$ and $m \geq 2$ as MAXCOVER($m \in \{0, 1\}$) and MAXCOVER($m \geq 2$), respectively. Dynamic programming algorithms **DPC1** and **DPC2** with running times $O(n^{m+2}L)$ and $O(n^{m+2}U)$ for MAXCOVER($m \in \{0, 1\}$) are described in Sections 3.2 and 3.3, respectively. A simple approximation algorithm for this case is presented in Section 3.4. Finally, we describe an $O(n\log n)$ time algorithm for MaxCover($m \geq 2$) in Section 3.5.

3.1 | "At most two machines" property and ordinary NP-hardness for $m \in \{0, 1\}$

Property 1. *There exists an optimal solution of the problem* MAXCOVER*, in which no more than two machines are used.*

Proof. If there are three accepted intervals $(a, b]$, $(a', b']$ and $(a'', b'']$ that intersect in a point x then there are two of these intervals, one containing $[\inf\{a, a', a''\}, x]$ and one containing $[x, \max\{b, b', b''\}]$, that cover the third interval completely which can be removed without increasing the objective function. Hence, there exists an optimal solution, in which there is no accepted interval which is covered by other accepted intervals. ■

Theorem 2. *If α and β satisfy $\alpha + \beta > 0$, then* MAXCOVER($m \in \{0, 1\}$) *is NP-hard.*

Proof. A reduction from the Partition problem is used, see the definition of this problem in Theorem 1.

Case $m = 0$. For any instance of PARTITION, we construct the following instance of MAXCOVER. There are $n = k$ jobs such that job j is associated with an interval of length p_j, and the intervals do not intersect, $j = 1, \ldots, k$. We show that the instance of PARTITION has a solution if and only if the instance of MAXCOVER has a solution with the total amount of time at which at least one job is processed of at least P and the total machine usage costs of at most $(\alpha + \beta)P$.

Let a set $X \subseteq K$ be a solution of PARTITION. Construct a solution of MAXCOVER in which, if $j \in X$ then job j is accepted, else if $j \notin X$ then job j is rejected. For this solution, the total amount of time at which at least one job is processed is equal to $\sum_{j \in X} p_j = P$, and the total costs are equal to $(\alpha + \beta)\sum_{j \in X} p_j = (\alpha + \beta)P$.

Now, assume that MAXCOVER has a solution of the required quality. Let X denote the set of accepted jobs in this solution. For this solution, the total amount of time at which at least one job is processed is equal to $\sum_{j \in X} p_j$, and the total machine usage costs are equal to $(\alpha + \beta)\sum_{j \in X} p_j$. Since relations $\sum_{j \in X} p_j \geq P$ and $(\alpha + \beta)\sum_{j \in X} p_j \leq (\alpha + \beta)P$ must hold, we deduce $\sum_{j \in X} p_j = P$.

Case $m = 1$. For any instance of PARTITION, we construct an instance of MAXCOVER, in which there are $n = k$ *pairs* of jobs (intervals). The length of any interval of the same pair j is equal to $2p_j$, and the length of their common part is equal to p_j, $j = 1, \ldots, k$. Intervals of jobs from different pairs do not intersect. We show that the instance of PARTITION has a solution if and only if the instance of MAXCOVER has a solution with the total amount of time, at which at least one job is processed of at least $5P$, and the total machine usage costs of at most $(\alpha + \beta)P$.

Assume that set $X \subseteq K$ is a solution of PARTITION. Construct a solution of MAXCOVER in which, if $j \in X$ then both jobs from the pair j are accepted, else if $j \notin X$ then exactly one job of this pair is accepted. For this solution, the total amount of time at which at least one job is processed is equal to $3\sum_{j \in X} p_j + 2\sum_{j \notin X} p_j = 5P$, and the total costs are equal to $(\alpha + \beta)\sum_{j \in X} p_j = (\alpha + \beta)P$. Assume that MAXCOVER has a solution of the required quality. Let X denote the set of job pairs such that both jobs of the pair are accepted. For this solution, the total amount of time at which at least one job is processed does not exceed $3\sum_{j \in X} p_j + 2\sum_{j \notin X} p_j$, and the total machine usage costs are equal to $(\alpha + \beta)\sum_{j \in X} p_j$. Since $3\sum_{j \in X} p_j + 2\sum_{j \notin X} p_j \geq 5P$ and $(\alpha + \beta)\sum_{j \in X} p_j \leq (\alpha + \beta)P$ must be satisfied, we deduce $\sum_{j \in X} p_j = P$. ■

We now describe dynamic programming algorithms **DPC1** and **DPC2** for the problem MAXCOVER($m \in \{0, 1\}$). These algorithms are similar to the algorithms **DPW1** and **DPW2** for the problem MAXWEIGHT.

3.2 | Algorithm DPC1 for MAXCOVER($m \in \{0, 1\}$)

In **DPC1**, each partial solution is characterized by a *state* (j, l, r) and a *cost function* $H_j(l, r)$, where j is the number of jobs considered so far, l is the total amount of time at which at least one job is processed, $r = (r_1, \ldots, r_{m+1})$, and r_i is the latest endpoint of the accepted intervals, which belongs to at least $i \in M$ accepted intervals. If an endpoint r_i does not exist, then $r_i := 0$ by the definition, $i \in M$. If $r_i = r_{i+1}$, $1 \leq i \leq m$, then we assume that r_i corresponds to a job accepted later, and r_{i+1} corresponds to a job accepted earlier. The value of $H_j(l, r)$ is equal to the minimum total costs of the partial solutions in the same state (j, l, r).

The initialization is $H_1(0, e^{(0)}) = 0$, $H_1(d_1, e^{(1)}) = c^{(1)}$, and $H_1(l, r) = U + 1$ if $(l, r) \notin \{(0, e^{(0)}), (d_1, e^{(1)})\}$, $0 \le l \le L$, $r = (r_1, \ldots, r_{m+1})$, $r_i \in B_n$, $i = 1, \ldots, m+1$. Generate sets $R_{(r_{m+1} > b_j)}$, $R_{(r_{m+1} \le b_j)}$, $S_R^{(j)}$, $S_0^{(j)}$ and $S_L^{(j)}$, vectors r^0 satisfying Equation (4) and values $\Delta_j(r_m^0, r_{m+1}^0)$ satisfying Equation (5) in the same way as in algorithm **DPW1**. The recursion for $j = 2, \ldots, n$, $0 \le l \le L$ and $r \in R_{(r_{m+1} > b_j)} \cup R_{(r_{m+1} \le b_j)}$ is.

$$H_j(l, r) = \min \begin{cases} H_{j-1}(l, r), & \\ H_{j-1}(l, r) + \beta d_j, & \text{if } r \in R_{(r_{m+1} > b_j)}, \\ H^0, & \text{if } r \in R_{(r_{m+1} \le b_j)}, \end{cases}$$

where

$$H^0 = \min \Big\{ H_{j-1}(l - \delta_j(x), r^0) + \Delta_j(r_m^0, r_{m+1}^0) \mid r^0 = (r(S_R^{(j)}), r(S_0^{(j)} \backslash \{j\}), x, r(S_L^{(j)})),$$
$$x \in B_{j-1}, \min\{r_i | i \in S_0^{(j)} \backslash \{j\}\} \ge x \ge \max\{r_i | i \in S_L^{(j)}\} \Big\},$$

$$\delta_j(x) = \begin{cases} 0, & \text{if } b_j < r_1, \\ \min\{d_j, r_1 - x\} & \text{if } b_j = r_1. \end{cases}$$

If the value of $H_j(l, r)$ is attained at the upper term of the minimum in the recursion, then job j is rejected, else it is accepted. The maximum total amount of time at which at least one job is processed is equal to

$$C^* = \max\{l \mid H_n(l, r) \le U, l = 0, 1, \ldots, L, r = (r_1, \ldots, r_{m+1}), r_i \in B_n, i = 1, \ldots, m+1\}$$

and the corresponding optimal solution can be found in the same way as in algorithm **DPW1**. The running time of the algorithm **DPC1** is $O(n^{m+2}L)$ for $m \in \{0, 1\}$. We remark that **DPC1** is not optimal for $m \ge 2$. If $m \ge 2$, then the function $\delta_j(x)$ needs to be corrected.

3.3 | Algorithm DPC2 for MAXCOVER($m \in \{0, 1\}$)

Algorithm **DPC2** differs from **DPC1** in that the roles of the cost function $H_j(l, r)$ and the state variable l are switched. Each partial solution is characterized by a *state* (j, c, r) and a *length function* $K_j(c, r)$, where c is the total machine usage costs, and j and r are the same as in **DPC1**. The value of $K_j(c, r)$ is equal to the maximum total amount of time at which at least one job is processed for the partial solutions in the same state (j, c, r).

Initialization of the algorithm **DPC2** is $K_1(0, e^{(0)}) = 0$, $K_1(c^{(1)}, e^{(1)}) = d_1$, and $K_1(c, r) = -\infty$ if $(c, r) \notin \{(0, e^{(0)}), (c^{(1)}, e^{(1)})\}$, $0 \le c \le U$, $r = (r_1, \ldots, r_{m+1})$, $r_i \in B_n$, $i = 1, \ldots, m+1$. Generate sets $R_{(r_{m+1} > b_j)}$, $R_{(r_{m+1} \le b_j)}$, $S_R^{(j)}$, $S_0^{(j)}$ and $S_L^{(j)}$, vectors r^0 satisfying Equation (4) and values $\Delta_j(r_m^0, r_{m+1}^0)$ satisfying Equation (5) in the same way as in algorithm **DPW1**. The recursion for $j = 2$, $\ldots, n$, $0 \le c \le U$, $r_i \in B_j$, $i = 1, \ldots, m+1$, is

$$K_j(c, r) = \max \begin{cases} K_{j-1}(c, r), & \\ K_{j-1}(c - \beta d_j, r), & \text{if } r \in R_{(r_{m+1} > b_j)} \text{ and } c \ge \beta d_j, \\ K^0, & \text{if } r \in R_{(r_{m+1} \le b_j)} \text{ and } c \ge \Delta_j(r_m^0, r_{m+1}^0), \end{cases}$$

where

$$K^0 = \max \Big\{ K_{j-1}(c - \Delta_j(r_m^0, r_{m+1}^0), r^0) + \delta_j(x) \mid r^0 = (r(S_R^{(j)}), r(S_0^{(j)} \backslash \{j\}), x, r(S_L^{(j)})),$$
$$x \in B_{j-1}, \min\{r_i | i \in S_0^{(j)} \backslash \{j\}\} \ge x \ge \max\{r_i | i \in S_L^{(j)}\} \Big\}$$

and functions $\delta_j(x)$ are the same as in **DPC1**. The maximum total amount of time at which at least one job is processed is equal to.

$$C^* = \max\{K_n(c, r) \mid 0 \le c \le U, r = (r_1, \ldots, r_{m+1}), r_i \in B_n, i = 1, \ldots, m+1\}$$

and the corresponding optimal solution can be found in the same way as in algorithm **DPW1**. The running time of the algorithm **DPC2** is $O(n^{m+2}U)$ for $m \in \{0, 1\}$.

3.4 | 1/2-approximation algorithm for MAXCOVER($m \in \{0, 1\}$)

Due to Property 1, at most two machines are used in an optimal solution of the problem MAXCOVER($m \in \{0, 1\}$). Hence, the optimal solution value C^* can be represented as $C^* = C_1^* + C_2^* - C_{12}^*$, where C_i^* is the total job processing time on machine i,

$i = 1, 2$, and C^*_{12} is the total time at which machines 1 and 2 operate simultaneously. It is convenient to use the graph-theoretic interpretation, in which intervals are vertices of an interval graph, and two vertices are linked by an edge if the corresponding intervals intersect.

A feasible solution of the problem MAXCOVER($m \in \{0, 1\}$), in which at most two machines are used, is determined by two non-intersecting *independent sets* (or *2-coloring*) of the interval graph, and vice versa. Vertices of an independent set (of the same color) are jobs processed on the same machine. Denote by C^0 the maximum total length of the intervals of any independent set of the interval graph. An independent set with the value C^0 can be found in $O(n\log n)$ time by the min-cost flow algorithms of Bouzina and Emmons [6] and Carlisle and Lloyd [7], which are intended for maximizing the weighted number of accepted jobs on identical parallel machines. For our purposes, there is a single machine and the weight of each job is equal to the length of its interval.

It is clear that $C^0 \geq \max\{C^*_1, C^*_2\}$, which implies $C^0 \geq C^*/2$. A solution in which intervals of the independent set with value C^0 are processed on a single machine is feasible for the problem MAXCOVER($m \in \{0, 1\}$). Hence, the following theorem holds.

Theorem 3. *Any of the algorithms of Bouzina and Emmons* [6] *and Carlisle and Lloyd* [7] *is a* 1/2*-approximation algorithm for the problem* MAXCOVER($m \in \{0, 1\}$).

Note that the approximation ratio 1/2 is asymptotically tight. Indeed, consider an instance, in which $m = 1$, there are two intervals of the same length $Y + 1$, the length of their intersection is 1, and $U = \alpha + \beta$, which implies that at most one third-party machine can be used during one time unit. The algorithm in Bouzina and Emmons [6] will deliver a single interval solution with value $Y + 1$, while the optimal solution includes both intervals with value $2Y + 1$. Hence, the approximation ratio is $\frac{Y+1}{2Y+1} = \frac{1}{2} + \frac{1}{4Y+2}$, which approaches 1/2 for sufficiently large Y.

3.5 | An $O(n \log n)$ time algorithm for MAXCOVER($m \geq 2$)

We now demonstrate that the problem MAXCOVER($m \geq 2$) can be solved in $O(n\log n)$ time. The solution consists of two stages. The first stage is a pre-processing algorithm, denoted as **Remove**, which removes all intervals which are inside of another one. The algorithm of Kung et al. [22] designed to find the set of maximal 2-dimensional vectors (intervals) with respect to a partial order can serve as algorithm **Remove**. Below we describe algorithm **Remove** for completeness. Due to Property 1, an optimal solution of the new problem, in which $(a_i, b_i] \not\subseteq (a_j, b_j]$ for all i and j, is an optimal solution of the original problem.

The second stage is an algorithm, denoted as **Greedy**, which solves the new problem as follows. Firstly, an interval with the earliest endpoint is accepted. Secondly, intervals are processed in a non-decreasing order of their left endpoints. Thirdly, among intervals that intersect with the interval accepted last, the interval with the latest endpoint is accepted. If no unprocessed interval intersects with the last accepted interval, then the interval with the earliest endpoint is accepted.

Algorithm Remove.

Initialization. Denote by V a maximal set of intervals from N such that $(a_i, b_i] \not\subseteq (a_j, b_j]$ for $i \in V$ and $j \in V$. Re-number intervals from N according to $a_1 \leq \cdots \leq a_n$ breaking ties so that $b_i \leq b_{i+1} \leq \cdots \leq b_j$ if $a_i = a_{i+1} = \cdots = a_j$. Initialize $V := \{(a_s, b_s]\}$, where $s = \max\{j \in N | a_j = a_1\}$. Let $(a^*, b^*]$ denote the interval included last into the set V. Process intervals in the order $s, s+1, \ldots, n$.

General step. If $s = n$ then stop: set V is constructed. If $s \leq n-1$, then perform the following computations. If $b_{s+1} \leq b^*$, then re-set $s := s+1$ and repeat General step. If $b^* < b_{s+1}$, then re-set $V := V \cup (a_t, b_t]$, $a^* := a_t$, $b^* = b_t$, where $t = \max\{j | a_j = a_{s+1}\}$, re-set $s := t$ and repeat General step. ■

The running time of algorithm **Remove** is $O(n \log n)$.

Algorithm Greedy.

Input: Set V (maximal set of intervals from N such that $(a_i, b_i] \not\subseteq (a_j, b_j]$ for $i \in V$ and $j \in V$).

Initialization. Re-number intervals of the set V such that $a_1 < \cdots < a_v$. Initialize optimal set of accepted intervals $O = \{(a_1, b_1]\}$. Let $(a^*, b^*]$ denote the interval accepted last. Set $(a^*, b^*] = (a_1, b_1]$ and $s = 1$.

General step. If $s = v$ then stop: O is the optimal set of accepted intervals. If $s \leq v-1$, then perform the following computations. If $a_{s+1} > b^*$, then re-set $O := O \cup (a_{s+1}, b_{s+1}]$, $(a^*, b^*] := (a_{s+1}, b_{s+1}]$, re-set $s := s+1$, and repeat General step. If $a_{s+1} \leq b^*$, then determine largest i, $s+1 \leq i \leq v$, such that $a_i \leq b^*$. Intervals $(a_k, b_k]$, $k = s+1, s+2, \ldots, i$, are all intervals $(a_i, b_i]$, $i \geq s+1$, intersecting with $(a^*, b^*]$. Among them, select the interval with the latest endpoint b_r. Re-set $O := O \cup \{(a_r, b_r]\}$, $a^* := a_r$, $b^* = b_r$, $s := i$, and repeat General step. ■

Let us show that algorithm **Greedy** is optimal for the problem MAXCOVER($m \geq 2$) with the set of intervals V. Denote by O^* an arbitrary optimal set of accepted intervals. It is obvious that inclusion $(a_1, b_1] \in O^*$ must be satisfied. Assume that O^*

contains intervals selected by **Greedy** up to the interval $(a_s, b_s]$. If $a_{s+1} > b_s$, then inclusion $(a_{s+1}, b_{s+1}] \in O^*$ must be satisfied, else inclusion $(a_t, b_t] \in O^*$ must be satisfied, where $(a_t, b_t]$, $t \geq s+1$, is the interval intersecting with $(a_s, b_s]$, which has the latest endpoint. Observe that $a_{t+1} > b_s$, which implies that, among the intervals which intersect with $(a_s, b_s]$, only $(a_t, b_t]$ is present in O^*. Hence, two machines are enough to process these intervals without intersection. Continuing in the same fashion, we see that **Greedy** outputs an optimal set of accepted intervals. The running time of this algorithm is $O(n\log n)$ because in General step each interval is considered at most twice: when comparing with b^* and when selecting the interval with the latest endpoint b_r. The following theorem holds.

Theorem 4. *Sequential application of algorithms* **Remove** *and* **Greedy** *constitutes an optimal algorithm for* MAXCOVER($m \geq 2$). *Both algorithms run in* $O(n\log n)$ *time*.

4 | CONCLUSIONS

We have proved that the problems MAXWEIGHT and MAXCOVER($m \in \{0, 1\}$) are NP-hard in the ordinary sense and developed dynamic programming algorithms for them which are pseudo-polynomial if the number of in-house machines m is a constant. An $O(n\log n)$ time 1/2-approximation algorithm is designed for MAXCOVER($m \in \{0, 1\}$). Surprisingly, the problem MAXCOVER($m \geq 2$) is solvable in $O(n\log n)$ time.

For future research, it is interesting to develop faster optimal algorithms and (Fully) Polynomial Time Approximation Schemes for the problems MAXWEIGHT and MAXCOVER($m \in \{0, 1\}$) as well as to prove or disprove their NP-hardness in the strong sense and *fixed parameter tractability* (Cygan et al. [10], Mnich and Wiese [24]). Considering time windows instead of fixed intervals and different but still realistic cost functions for using third-party machines are other promising research directions.

We conjecture that the dynamic programming algorithms **DPW1, DPW2, DPC1** and **DPC2** can be modified to solve more general problems than MAXWEIGHT and MAXCOVER($m \in \{0, 1\}$). One way of modification is to introduce costs for the activation and deactivation of third-party machines. Using in-house machines entails no costs regardless of the time as assumed in this paper, while using x_{ab} third-party machines in a time interval $(a, b]$, $b > a$ would incur costs

$$u(x_{ab})(b-a) + h(x_{a-1,a}, x_{a,a+1}) + f(x_{b-1,b}, x_{b,b+1}), \quad \text{where}$$

- $u(x_{ab})$ are the costs for using x_{ab} third-party machines a time unit such that $u(0) = 0$ and $u(x_{ab}) = \alpha + \beta x_{ab}$ if $x_{ab} > 0$, as it is in the problems MAXWEIGHT and MAXCOVER,
- $h(x_{a-1,\,a}, x_{a,\,a+1})$ are activation costs such that $h(x_{a-1,\,a}, x_{a,\,a+1}) > 0$ if $x_{a-1,\,a} = 0$ and $x_{a,\,a+1} > 0$, and $h(x_{a-1,\,a}, x_{a,\,a+1}) = 0$ if $x_{a-1,\,a} > 0$,
- $f(x_{b,\,b+1})$ are deactivation costs such that $f(x_{b,\,b+1}) > 0$ if $x_{b-1,\,b} > 0$ and $x_{b,\,b+1} = 0$, and $f(x_{b,\,b+1}) = 0$ if $x_{b,\,b+1} > 0$.

ACKNOWLEDGMENT

The research of the first author was supported by the Friede Springer foundation, grant 6000108. Open access funding enabled and organized by Projekt DEAL.

ORCID

Mikhail Y. Kovalyov https://orcid.org/0000-0003-0832-0829
Erwin Pesch https://orcid.org/0000-0003-0182-870X

REFERENCES

[1] E. Angelelli, N. Bianchessi, and C. Filippi, *Optimal interval scheduling with a resource constraint*, Comput. Oper. Res. **51** (2014), 268–281.
[2] Y. Bartal, S. Leonardi, A.M. Spaccamela, J. Sgall, and L. Stougie, *Multi-processor scheduling with rejection*, SIAM J. Discrete Math. **13** (2000), 64–78.
[3] M. Bentert, R. van Bevern, and R. Niedermeier, *Inductive k-independent graphs and* c*-colorable subgraphs in scheduling: A review*, J. Sched. **22**(1) (2019), 3–20.
[4] R. van Bevern, M. Mnich, R. Niedermeier, and M. Weller, *Interval scheduling and colorful independent sets*, J. Sched. **18**(5) (2015), 449–469.
[5] L. Bodin, B. Golden, A. Assad, and M. Ball, *Routing and scheduling of vehicles and crews: The state of the art*, Comput. Oper. Res. **10**(2) (1983), 62–212.
[6] K.I. Bouzina and H. Emmons, *Interval scheduling on identical machines*, J. Global Optim. **9** (1996), 379–393.
[7] M.C. Carlisle and E.L. Lloyd, *On the k-coloring of intervals*, Discrete Appl. Math. **59** (1995), 225–235.

[8] L. Chen, F. Eberle, N. Megow, K. Schewior, and C. Stein, *A general framework for handling commitment in online throughput maximization*, Math. Program. **183** (2020), 215–247.
[9] Z.-L. Chen and C.-L. Li, *Scheduling with subcontracting options*, IIE Trans. **40**(12) (2008), 1171–1184.
[10] M. Cygan, F.V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh, *Parameterized Algorithms*, Berlin: Springer, 2015.
[11] G. Dosa and Y. He, *Scheduling with machine cost and rejection*, J. Comb. Optim. **12** (2006), 337–350.
[12] D.W. Engels, D.R. Karger, S.G. Kolliopoulos, S. Sengupta, R.N. Uma, and J. Wein, *Techniques for scheduling with rejection*, J. Algorithms **49** (2003), 175–191.
[13] T. Fukunaga, M. Halldórsson, and H. Nagamochi, *"Rent-or-Buy" scheduling and cost coloring problems*, International Conference on Foundations of Software Technology and Theoretical Computer Science FSTTCS, 2007, pp. 84–95.
[14] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.
[15] F. Gnegel and A. Fügenschuh, *An iterative graph expansion approach for the scheduling and routing of airplanes*, Comput. Oper. Res. **114** (2020), 104832.
[16] U.I. Gupta, D.T. Lee, and J.Y.-T. Leung, *An optimal solution for the channel-assignment problem*, IEEE Trans. Comput. **28** (1979), 807–810.
[17] D. Hermelin, M. Pinedo, D. Shabtay, and N. Talmon, *On the parameterized tractability of single machine scheduling with rejection*, Eur. J. Oper. Res. **273**(1) (2019), 67–73.
[18] B. Hezarkhani and W. Kubiak, *Decentralized subcontractor scheduling with divisible jobs*, J. Sched. **18**(18) (2015), 497–511.
[19] A.W.J. Kolen, J.K. Lenstra, C.H. Papadimitriou, and F.C.R. Spieksma, *Interval scheduling: A survey*, Naval Res. Logist. **54** (2007), 530–543.
[20] M.Y. Kovalyov, C.T. Ng, and T.C.E. Cheng, *Fixed interval scheduling: Models, applications, computational complexity and algorithms*, Eur. J. Oper. Res. **178** (2007), 331–342.
[21] S.O. Krumke, C. Thielen, and S. Westphal, *Interval scheduling on related machines*, Comput. Oper. Res. **38**(12) (2011), 1836–1844.
[22] H.T. Kung, F. Luccio, and F.P. Preparata, *On finding the maxima of a set of vectors*, J. ACM **22**(4) (1975), 469–476.
[23] G. Lera-Romero, J.J. Miranda Bront, and F.J. Soulignac, *Linear edge costs and labeling algorithms: The case of the time-dependent vehicle routing problem with time windows*, Networks **76**(1) (2020), 24–53.
[24] M. Mnich and A. Wiese, *Scheduling and fixed-parameter tractability*, Math. Program. **154**(1–2) (2015), 533–562.
[25] S. Mohammadi, S.M.J.M. Al-e-Hashem, and Y. Rekik, *An integrated production scheduling and delivery route planning with multi-purpose machines: A case study from a furniture manufacturing company*, Int. J. Prod. Econ. **219** (2020), 347–359.
[26] H. Mokhtari and I.N.K. Abadi, *Scheduling with an outsourcing option on both manufacturer and subcontractors*, Comput. Oper. Res. **40** (2013), 1234–1242.
[27] X. Qi, *Production scheduling with subcontracting: The subcontractor's pricing game*, J. Sched. **15** (2012), 773–781.
[28] J. Ren, G. Sun, and Y. Zhang, *The supplying chain scheduling with outsourcing and transportation*, Asia-Pac. J. Oper. Res. **34**(2) (2017), 1750009.
[29] B. Sarasola and K.F. Doerner, *Adaptive large neighborhood search for the vehicle routing problem with synchronization constraints at the delivery location*, Networks **75**(1) (2020), 64–85.
[30] L. Schrage, *Formulation and structure of more complex/realistic routing and scheduling problems*, Networks **11**(2) (1981), 229–232.
[31] D. Shabtay, N. Gaspar, and M. Kaspi, *A survey on offline scheduling with rejection*, J. Sched. **16**(1) (2013), 3–28.
[32] S.A. Slotnick, *Order acceptance and scheduling: A taxonomy and review*, Eur. J. Oper. Res. **212**(1) (2011), 1–11.
[33] G.L. Vairaktarakis, *Noncooperative games for subcontracting operations*, Manuf. Serv. Oper. Manag. **15**(1) (2013), 148–158.
[34] G.L. Vairaktarakis and T. Aydinliyim, *Benchmark schedules for subcontracted operations: Decentralization inefficiencies that arise from competition and first-come-first-served processing*, Decision Sci. **48**(4) (2017), 657–690.
[35] X. Wang, Q. Zhu, and T.C.E. Cheng, *Subcontracting price schemes for order acceptance and scheduling*, Omega **54** (2015), 1–10.
[36] W. Zhong and Z. Huo, *Single machine scheduling problems with subcontracting options*, J. Comb. Optim. **26** (2013), 489–498.
[37] X. Zhong, J. Ou, and G. Wang, *Order acceptance and scheduling with machine availability constraints*, Eur. J. Oper. Res. **232**(3) (2014), 435–441.

How to cite this article: Fridman I, Kovalyov MY, Pesch E, Ryzhikov A. Fixed interval scheduling with third-party machines. *Networks.* 2021;77:361–371. https://doi.org/10.1002/net.21973