Omega 000 (2018) 1–13

Contents lists available at ScienceDirect

Omega

journal homepage: www.elsevier.com/locate/omega

Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs☆

Claudio Arbib[a,1,*], Giovanni Felici[b], Mara Servilio[b,c]

[a] Università degli Studi dell'Aquila, Dipartimento di Ingegneria/Scienze dell'Informazione e Matematica, via Vetoio L'Aquila 67010, Italy
[b] Consiglio Nazionale delle Ricerche, Istituto di Analisi dei Sistemi e Informatica "Antonio Ruberti", viale Manzoni, Roma, 30 - 00185, Italy
[c] Cyberdyne R&D, Via dei Peligni, Pescara, 60 - 65127, Italy

ARTICLE INFO

Article history:
Received 8 August 2016
Accepted 5 April 2018
Available online xxx

Keywords:
Scheduling
Pattern sequencing
Integer linear programming
Cover inequalities
Constrained submodular maximization

ABSTRACT

Common operation scheduling (COS) problems arise in real-world applications, such as industrial processes of material cutting or component dismantling. In COS, distinct jobs may share operations, and when an operation is done, it is done for all the jobs that share it. We here propose a 0-1 LP formulation with exponentially many inequalities to minimize the weighted number of tardy jobs. Separation of inequalities is in *NP*, provided that an ordinary min L_{max} scheduling problem is in *P*. We develop a branch-and-cut algorithm for two cases: one machine with precedence relation; identical parallel machines with unit operation times. In these cases separation is the constrained maximization of a submodular set function. A previous method is modified to tackle the two cases, and compared to our algorithm. We report on tests conducted on both industrial and artificial instances. For single machine and general processing times the new method definitely outperforms the other, extending in this way the range of COS applications.

1. Introduction

1.1. Common operation scheduling

Common operation scheduling (COS) problems [1] call for finding an optimal arrangement of the operations required by a set of jobs under the condition that, when an activity is done, it is done for all the jobs that require it. A COS model may not only envisage economy-of-scale aspects (that is, when some of the operations required by a job realize pre-conditions to the accomplishment of other jobs), but also situations depending on the very nature of the process optimized. To quote three applications, COS problems were identified in movie shooting [9,14], progressive network recovery [34] and pattern sequencing in stock cutting [7]. Prior to formally introducing the problem, let us give more examples from maintenance logistics and manufacturing (in Section 2 we will focus on stock cutting with a detailed real-world example).

Example 1. *(Maintenance logistics)*
A company sends m teams to visit n sites (e.g., power plants) for maintenance operations that require testing a set J of r functionalities. Functionality j is implemented in a subset N_j of the sites, and its test is fully accomplished when done in the whole N_j: so, an operation is a visit to a single site (where more functionalities are checked), and a job is the test of a single functionality in all the relevant sites. Functionality j should be tested in the whole set of sites within a given date d_j, and a team completes the visit of site i in one day. The company wants to assign sites to teams so as to minimize the complete functionality tests not terminated by the due date.

Example 2. *(Manufacturing)*
A mechanical assembly process produces a set J of final products (jobs) through a set N of operations. The operations must obey precedence constraints, described by a directed acyclic graph $G = (N, A)$. Jobs then correspond to terminal nodes of G. Opera-

☆ This manuscript was processed by Associate Editor Ljubic.
* Corresponding author.
E-mail addresses: claudio.arbib@univaq.it (C. Arbib), giovanni.felici@iasi.cnr.it (G. Felici), mara.servilio@cyberdyne.it (M. Servilio).
[1] The work of the first author was supported by the Italian Ministry of Education, National Research Program (PRIN) 2015, contract n. 20153TXRX9.

https://doi.org/10.1016/j.omega.2018.04.002

Fig. 1. Precedence graph in dismantling operations: jobs 1 and 2 share the operations on the encircled parts.

tions are of three types: preparation, assembly, disassembly[2] and are characterized by processing times p_i. An assembly (a disassembly) operation is the immediate successor (predecessor) of more than one operation. As disassembly produces more parts from a single one (see Fig. 1), those parts can possibly form distinct final products: that is, distinct jobs (terminals) share the operations lying on a common path of G.

Thus, in a COS model one has a set J of r *jobs* that require, altogether, a set N of n *operations*, and those required to complete $j \in J$ form a subset $N_j \subseteq N$. Different permutations of N complete the jobs of J in different times. For instance, let $J = \{1, 2, 3\}$, $N = \{a, b, c\}$. Then $N_1 = \{a, b\}, N_2 = \{a, c\}, N_3 = \{c\}, N_4 = \{b\}$ encode a problem with four jobs and three operations to do: the operation sequence *abc* completes job 1 and 4 first, then jobs 2 and 3 in the same time instant; *cab*, instead, completes job 3 first, then job 2 and finally jobs 1 and 4.

In both the above examples, we deal with a non-preemptive deterministic problem: Example 1 has a focus on parallel machines with unit processing times, Example 2 on precedence relations and non-unit processing times (the reader is referred to Pinedo [27] for definitions and notation). In general, we consider a set Q_m of m uniform machines with speeds $s_1, \ldots, s_m$: hence p_i/s_k is the processing time of operation i on machine k. A particular case is P_m (identical machines), where $s_k = 1$ for $k = 1, \ldots, m$. All operations are available from time 0 on, and a precedence relation among the operations is possibly encompassed, as in Example 2, requiring that some operations cannot start before the completion of some other. A strict precedence constraint between jobs j, $k \in J$ implies $N_j \cap N_k = \emptyset$: the constraint can be reduced to a precedence relation on N by simply imposing that all the operations of N_j precede all those of N_k.

Let f: $IR^r \to IR$ be a non-decreasing function of the job completion times C_j (in this case f is called *regular*). Stating that job j is completed as soon as all the operations of N_j are done, means that C_j is the largest among the completion times of the operations in N_j. The problem has the following generic expression:

Problem 1. Schedule N on Q_m, that is, give each operation a machine and a starting time, so that precedence relations among operations are respected and $f(C_1, \ldots, C_r)$ is minimized.

With the notation of [27], Problem 1 is denoted $Q_m|cos, prec|f(C_1, \ldots, C_r)$. In applications, regular objective functions are often related to the due dates d_j of the jobs, and may include such terms as the job tardiness $T_j = \max\{C_j - d_j, 0\}$, or 0-1 variables u_j indicating that j is not completed on time. Typical examples are the maximum lateness $L_{max} = \max_j\{C_j - d_j\}$, the weighted tardiness $\sum w_j T_j$, and the weighted number of tardy jobs $\sum w_j u_j$ where $u_j = 1$ if $T_j > 0$ and $u_j = 0$ otherwise. The latter is the case dealt with in this paper.

1.2. Relation to studied problems

Common operation scheduling clearly generalizes the associated ordinary scheduling problems: it suffices setting $N_j = \{j\}$ for all $j \in J$. Also, the COS paradigm includes popular graph ordering problems, such as Bandwidth and Optimal Linear Arrangement, see [17]: for instance, Bandwidth on graph $G = (V, E)$ is a COS problem with unit operations $N = V$, two-operations jobs $J = E$ and the objective of minimizing the largest job flow-time. Conversely, for regular f and any machine environment α, $\mathcal{P} = \alpha|cos, prec|f(C_1, \ldots, C_r)$ can easily be transformed into $\mathcal{P}' = \alpha|prec|f'(C_1, \ldots, C_s)$ with f' regular:

- $\mathcal{P}'$ has job set $J' = J \cup N$, $s = r + n$;
- Every job in $N \subseteq J'$ has the same duration as the corresponding operation, and every job in $J \subseteq J'$ has duration 0;
- $j \in J \subseteq J'$ is preceded by all the jobs in N_j, and precedence in N is the same as for the corresponding operations;
- $f'(C_1, \ldots, C_r, \ldots, C_s) = f(C_1, \ldots, C_r)$, i.e., f' does not depend on $C_{r+1}, \ldots, C_s$.

$\mathcal{P}'$ has a noticeable form: Woeginger [35] studied it for $\alpha = 1$, $f' = \sum w_j C_j$ (weighted completion time), no precedence relation in N, $p_i = 1$ for all $i \in N$, $p_j = 0$ for all $j \in J$ and $w_k = 1 - p_k$ for all $k \in J'$, showing that this apparently artificial problem cannot be better approximated (in polynomial time) than the more general case with arbitrary precedence, weights and job durations. With the above transformation we recognize, on the one hand, that Woeginger's problem is not so artificial; on the other hand, considering

[2] Disassembly processes in the automotive sector gained importance after EU Directives 2000/53/EC (Section 2 and Section 13) and 2005/64/EC (Section 2 and Section 15) [12,13] suggesting the practice of *Design for Dismantling (DfD)*. See also Toyota Vehicle Recycling Report 2016 [33].

Please cite this article as: C. Arbib et al., Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs, Omega (2018), https://doi.org/10.1016/j.omega.2018.04.002

the potential of COS to express complex scheduling or graph ordering problems, one can judge the complexity result not so surprising.

Instead of weighted completion time, Arbib et al. [1] focussed on the weighted number of tardy jobs $\sum w_j u_j$ and proved that $1|cos, p_i = 1|\sum w_j u_j$ is NP-hard also when restricted to jobs with two operations each, identical due dates and unit weights. They reformulated the problem as weighted Stable Set on a special graph G with two types of nodes: *assignment nodes*, corresponding to operation-slot pairs in N^2, and *on-time nodes*, corresponding to jobs in J. The edges of G are as follows:

- two nodes in N^2 are adjacent iff the associated pairs share either an operation, or a slot: this part of the graph is a grid with horizontal and vertical cliques;
- a node $(i, t) \in N^2$ and a node $j \in J$ are adjacent iff $i \in N_j$ and $t > d_j$.

In [1], the structure of G is investigated in order to get valid inequalities, and the notion of minimal cover—introduced in the next section—is also used to obtain optimality cuts.

The stable set formulation can easily be extended to $Q_m|cos, prec, p_i = 1|\sum w_j u_j$ (basically, adding a grid for each machine and suitable arcs to express precedence), and its efficiency is not questioned for unitary operation times. But, being time-indexed, is the formulation unfit for general p_i: the extension would in fact require p_i nodes per operation, thus becoming unmanageable for relatively small differences among processing times (see Sections 6.3 and 7 for numerical evidence).

1.3. This contribution

The present paper concentrates as [1] on the minimization of the *weighted number of tardy jobs*, that is:

$$f(C_1, \ldots, C_r) = \sum_{j \in J} w_j u_j \tag{1}$$

with $w_j > 0$ for all $j \in J$. We provide a methodological framework for solving Problem 1 with objective (1)—in the following, Problem 1-(1)—insisting on two special cases:

- $1|cos, prec|\sum w_j u_j$: single machine with precedence relation and general processing times;
- $Q_m|cos, p_i = 1|\sum w_j u_j$: m uniform parallel machines and unit processing times;

for which we propose an exact formulation with exponentially many constraints, to be solved by branch-and-cut.

Both cases generalize the simpler $1|cos, p_i = 1|\sum w_j u_j$ tackled in [1]. Especially the processing time generalization is important for applications. Arbitrary p_i are in fact typical of industrial applications where processing times depend on operation complexity (cuts in cutting processes, disassembly in dismantling processes etc.) and sub-operation number (e.g. pattern run lengths): to deal with p_i, model $1|cos, p_i = 1|\sum u_j$ and algorithm [1] become inefficient or inaccurate, as they must replicate each unit operation i a large number of times, see Section 6.3 below.

The paper is organized as follows. In Section 2 we develop an industrial application and explain how pattern sequencing in stock cutting can be formulated as a COS problem. In Section 3 we introduce the notion of cover in order to formulate Problem 1-(1) as Set Covering. The range of applicability of this formulation is surveyed, evaluating the complexity of cover check and separation. In Section 4 we provide a polynomial-time cover certificate for $1|cos, prec|\sum w_j u_j$ and $P_m|cos, p_i = 1|\sum w_j u_j$. The certificate is used in Section 5 to formulate the separation of cover inequalities as a combinatorial optimization problem in NP. Separation complexity is discussed, and separation algorithms—exact and heuristics—are proposed. A detailed description of a branch-and-cut algorithm is provided in Section 6, along with a numerical example. Section 7 is devoted to testing the algorithm on $P_2|cos, p_i = 1|\sum w_j u_j$ and $1|cos|\sum w_j u_j$, comparing it to the stable set model [1], and discussing advantages and disadvantages of the proposed approach.

2. An industrial application: cut sequencing

In a stock cutting problem—see e.g. [10] for a survey—one wants to obtain the best way to cut given objects (small items) from large plates (stock items). A solution is generally described as a set N of n cutting patterns, where pattern $i \in N$ specifies the way a single stock item has to be cut. A pattern is replicated a number of times on stock items, either taken individually or in packs (to minimize slitter set-ups, cuts sharing the same pattern are normally repeated consecutively and/or implemented on packs of plates). In this way, each pattern produces a specific set of small items. Fig. 2 reproduces a guillotine cutting pattern computed for a machine developed by SCM Group [30], a major Italian company of automated machine tools for wood working: this pattern produces the part codes 10, 30, 39, 44 and 49 by cutting a plate of size 2550×2100 mm.

The sequence in which cuts are done has an impact on production, and the problem of finding a good sequence has a long-standing tradition in the cutting stock research community with papers dating back as to the late '60s/early '70s [25,26]. In mid '80s, Johnston [18] included setup cost, delivery dates and work-in-progress limitations among the most relevant factors in the pattern sequencing stage. Work-in-progress has been in the spotlight for long time, and particularly open stack minimization [5,6,15,21,22,24,29,36,37] (list of references are ranked by date). In these papers, pattern sequencing is solved stand-alone, that is after an optimal cutting stock solution has been found. Other approaches integrate cutting and sequencing [2–4,8,24,28,37], and the most recent focus has moved back again on due-date related objectives: the minimization of the total (weighted) tardiness [2,28], of the maximum lateness [3,8], of the total number of tardy jobs [2]. Although an integrated approach is seemingly preferable for solution quality, there are cases in which it cannot be implemented and the two problems must be solved separately.

This is what normally happens in the furniture industry: trim-loss minimization is in fact provided as a black box algorithm by machine producers (e.g., the SCM Gabbiani series), while the machine user takes care of process scheduling. As a general practice, the implementation of a production plan starts from the orders to be fulfilled, their due-dates, and their bill of material (BOM), and follows three consecutive steps: (*i*) identification of the small items (types and amounts) to be produced to fulfill the orders; (*ii*) generation of the patterns according to which those items will be cut, trying to minimize trim loss, and identification of the patterns that contribute to produce each single order; (*iii*) determination of an optimal pattern sequence with respect to order due-dates. Step (*i*) is of no special complexity, as it just requires to explode the BOM. Step (*ii*) is the typical output of a 2-dimensional stock cutting problem. Step (*iii*) is a COS instance: the lots ordered correspond to jobs and the patterns to operations. Fig. 3 shows a small subset of the patterns forming a real production plan: items belonging to different lots are indicated in different colors.

The output time of a lot depends on the time required to implement the cuts of the relevant items, and on the sequence in which patterns are implemented. The cut time depends in turn on various factors, of which the so-called pattern run-length (i.e., the number of stock sizes cut according to that pattern) is just one element. Other factors are machine-dependent. Referring to a generic machine saw produced by SCM Group, we quote:

Please cite this article as: C. Arbib et al., Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs, Omega (2018), https://doi.org/10.1016/j.omega.2018.04.002

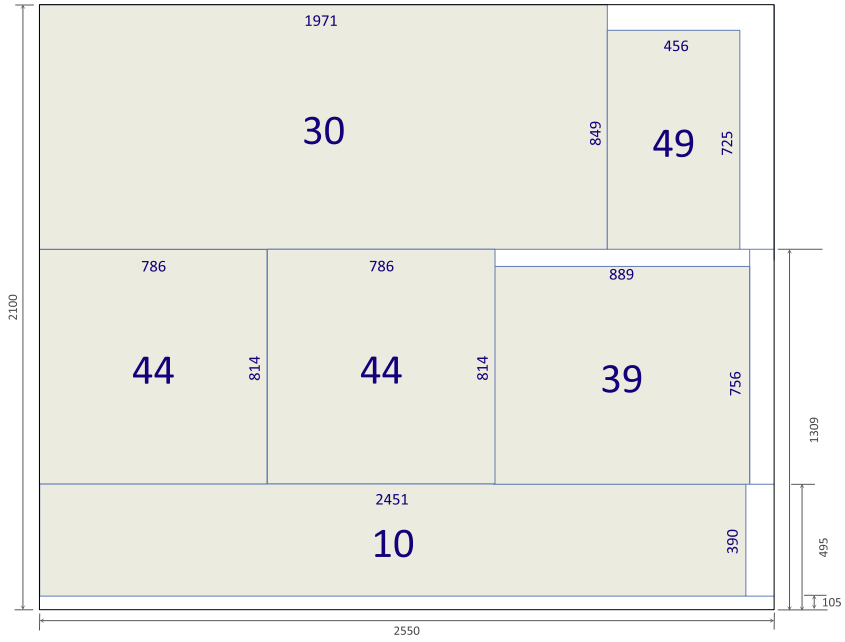

Fig. 2. A pattern implemented by a woodcutting machine (courtesy of SCM Group).

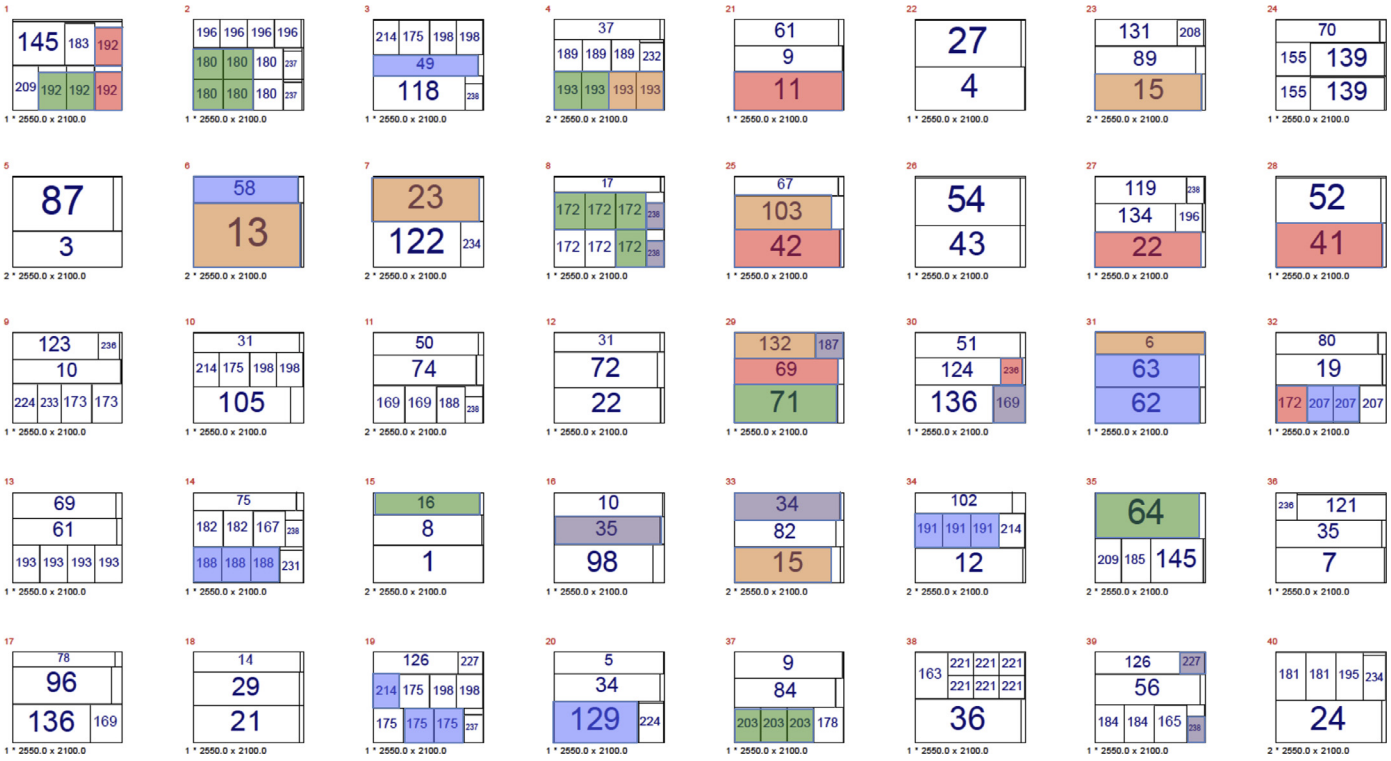

Fig. 3. Part of a production plan (courtesy of SCM Group).

- the handling time t_{hand}, needed to manipulate and possibly rotate the current piece to be cut and to place it in the initial position;
- the speed s_{push} at which the machine pushes the piece up to the correct position for cut;
- the cut speed s_{cut}, that may vary according to how many pieces (pack size) are simultaneously cut in a pack.

For example, in the pattern of Fig. 2 one first makes full width horizontal cuts (2550 mm) at positions 105, 495 and 1309 mm, then rotates the lowest piece obtained and cuts it at position 2550–2451 = 99 mm for a further cut length of 390 mm, and so on. Once a cutting plan has been elaborated, and before execution, one can evaluate by simulation the time required by the machine to implement each pattern.

Please cite this article as: C. Arbib et al., Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs, Omega (2018), https://doi.org/10.1016/j.omega.2018.04.002

The data so obtained form the input of a COS problem that can have different goals. In Section 6.3 we give a small numerical example with the objective of minimizing the total number of tardy jobs, with patterns and times taken from a real cutting stock solution. Computational experiments on this type of real-world instances are discussed in Sections 7.1 and 7.3.

3. Formulation as SET COVERING

In this section we formulate Problem 1-(1) as SET COVERING with exponentially many constraints. For any $C \subseteq J$, let

$$N_C = \bigcup_{j \in C} N_j \qquad d_C = \max_{j \in C}\{d_j\}$$

Definition 1. A subset C of J is said to be a *cover* of J if some job of C is late in any schedule of N_C.

In general, every superset of a cover is a cover too: so if J contains a cover, then J is a cover itself.

Definition 2. A cover C is said to be *minimal* if $C - \{j\}$ is not a cover for all $j \in C$.

For instance, take $C = \{1, 2, 3\}$ with $N_1 = \{a, b\}, N_2 = \{a, c\}, N_3 = \{b, c\}$, $p_i = 1$ for all $i \in N_C = \{a, b, c\}$ and $d_j = 2$ for all $j \in C$. For $m = 1$ no on-time schedule of C exists, since the total processing time of N_C exceeds d_C. Hence, C is a cover. However, C it is not minimal, because $B = \{1, 2\}$ cannot be completed before $p_a + p_b + p_c = 3 > 2 = d_B$ and therefore is a (minimal) cover properly contained in C.

For any $j \in J$, define 0-1 variable u_j so that $u_j = 1$ if and only if j is tardy, and consider the following integer program.

$$\min \sum_{j \in J} w_j u_j \tag{2}$$

$$\sum_{j \in C} u_j \geq 1 \qquad \forall C \text{ minimal cover} \tag{3}$$

$$u_j \in \{0, 1\} \tag{4}$$

According to Definition 1, inequality (3) states that not all the jobs in a minimal cover C can be done on time. Conversely, let U be the support set of $\mathbf{u} = (u_1, \ldots, u_m)$ fulfilling (3), (4), and let $S = J \setminus U$. If S cannot be scheduled on time, then S is a cover by Definition 1. But by (3), U hits all the minimal covers C, and then also a cover contained in S. Therefore $S \cap U \neq \emptyset$, a contradiction. Hence, $J \setminus U$ can be scheduled on time and, by minimizing $w(U) = \sum_{j \in U} w_j$, program (2)–(4) formulates Problem 1-(1).

A drawback of (2)–(4) is that inequalities (3) are exponentially many in r, thus cannot be listed off-line for large r. The formulation has then to be dealt with by branch-and-cut, separating infeasible solutions as long as they are encountered during computation. Online separation of an infeasible solution $\bar{\mathbf{u}}$ means finding a cover C such that

$$\sum_{j \in C} \bar{u}_j < 1 \tag{5}$$

A question then arises on the complexity of recognizing $C \subseteq J$ as a cover. To answer, we can solve an L_{max} minimization COS problem, observing that the transformation into ordinary scheduling is in this case more direct than the general one given in Section 1:

Proposition 1. *$Q_m|cos, prec|L_{max}$ is equivalent to $Q_m|prec|L_{max}$.*

Proof. Let $J_i = \{j \in J : i \in N_j\}$ contain all the jobs that require operation $i \in N$. Associate with each $i \in N$ the due date

$$d_i = \min_{j \in J_i}\{d_j\}$$

and schedule N to solve $Q_m|prec|L_{max}$. Then, an optimal schedule is also optimal for $Q_m|cos, prec|L_{max}$. In fact, for the COS problem

$$L_{max} = \max_{j \in J}\{L_j\} = \max_{j \in J}\{\max_{i \in N_j}\{L_i\}\} =$$

$$= \max_{j \in J}\{\max_{i \in N_j}\{\max\{C_i - \min_{j \in J_i}\{d_j\}, 0\}\}\} = \max_{i \in N}\{\max\{C_i - d_i, 0\}\}$$

and the latter expression gives L_{max} in the problem reformulated. □

Using Proposition 1 we then derive

Proposition 2. *C is a cover for $Q_m|cos, prec|\sum w_j u_j$ if and only if $Q_m|prec|L_{max}$ with job set N_C and due dates as in Proposition 1 has an optimal solution of positive value.*

Because $P_m||L_{max}$ is NP-hard (even for $m = 2$ and $d_j = 0$), Proposition 2 implies that separating (3) may not be in NP. On the other hand, a polynomial-time cover certificate exists for all those particular cases of $Q_m|prec$ for which minimizing L_{max} can be done in polynomial time: for instance, $1|prec$; $P_2|prec, r_i, p_i = 1$ [16]; $P_m|r_j, p_i = p$ [31]; $Q_m|p_i = 1$ [11]. The first and last case will be analysed in the following section.

4. Efficient cover certificates for $p_i = 1$ or $m = 1$

This section provides means to devise a practical separation algorithm for (2)–(4) for the two cases presented in Section 1. In general, separation is defined as a special recognition (or optimization) problem. In order to solve this problem, a first condition is that it is in NP. In the following, we prove this for either $p_i = 1$ or $m = 1$ by showing that, for any subset C of jobs, we can decide in polynomial time whether C is or is not a cover. The problem is reduced to that of computing a particular submodular function defined on operations sets.

4.1. Single machine with precedence constraints

To begin with, consider $1|prec$, namely a single machine and a precedence relation $\prec$ defined on N. Let $C \subseteq J$ and assume w.l.o.g. $C = \{1, \ldots, c\}, d_1 \leq \ldots \leq d_c$. Let then

$$D_j = \{i \in N : i \prec h, h \in N_j\}$$

for $j = 1, \ldots, c$. We say that $I_j = D_j \cup N_j$ is *implied* by j, that is, I_j contains all the operations to be completed in order to finish job j. We define

$$I_A = \bigcup_{j \in A} I_j \qquad I^k = \bigcup_{j=1}^{k} I_j$$

observing that $A \subseteq B \Rightarrow I_A \subseteq I_B$ and, consequently, $I^1 \subseteq I^2 \subseteq \ldots \subseteq I^c$.

Let us recall the following

Definition 3. For given real weights $p_1, \ldots, p_n \geq 0$, the set function $g(A) : 2^J \to R_+$ defined

$$g(A) = \sum_{i \in I_A} p_i$$

is called a *weighted coverage*.

When $p_i = 1$ for all $i \in N$, $g(A) = |I_A|$ is called a coverage. Weighted coverage functions are monotone submodular [23], that is,

i) $g(A) \leq g(B)$ for all $A \subseteq B \subseteq J$
ii) $g(A) + g(B) \geq g(A \cup B) + g(A \cap B)$ for any $A, B \subseteq J$

Please cite this article as: C. Arbib et al., Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs, Omega (2018), https://doi.org/10.1016/j.omega.2018.04.002

The following theorem uses the above function g to define a cover certificate, that is, an algorithm that answers `yes` if and only if $C \subseteq J$ is a cover of J.

Theorem 1. *Suppose $m = 1$ and $p_j > 0$ arbitrary. Then a subset C of J is a cover of J if and only if*

$$g(\{1, \ldots, k\}) > d_k \tag{6}$$

for some $k = 1, \ldots, c$.

Proof. Let (6) hold for some k. Then, not all the jobs of C can be scheduled on-time, because

$$g(\{1, \ldots, k\}) = \sum_{i \in I^k} p_i = C_k$$

Conversely, let us show that if C does not fulfill (6) for $k = 1, \ldots, c$, then all the jobs of C can be scheduled on-time. Let $R_k = I_k - \bigcup_{h=1}^{k-1} I_h = I_k - I^{k-1}$. Because (6) does not hold,

$$g(\{1, \ldots, k\}) = \sum_{i \in I^k} p_i \leq d_k$$

for all k. In an earliest due date solution that first schedules the operations of R_1, then those of R_2 and so on, precedence relations can easily be respected. Moreover, all the jobs of C are on time since

$$C_k = \sum_{i \in I^k} p_i$$

for $k = 1, \ldots, c$. □

Observe that

Proposition 4. *Certificate (6) can be checked in linear time.*

4.2. Parallel machines with unit processing times

Let us now consider the case $P_m|p_i = 1$ of identical parallel machines, no precedence constraints, and unit processing times. Certificate (6) is plainly extended as follows.

Theorem 2. *Inequality*

$$\left\lceil \frac{g(\{1, \ldots, k\})}{m} \right\rceil > d_k \tag{7}$$

is a linear time cover certificate for $P_m|cos, p_i = 1|\sum w_j u_j$.

Proof. Suppose (7) does not hold for $k = 1, \ldots, c$. Take an earliest due date solution that distributes the operations of R_1, then those of $R_2 \ldots$ as evenly as possible among the machines, starting from the least busy machine. After the kth distribution, the completion time of job k is

$$C_k = \left\lceil \frac{|N_1 \cup \ldots \cup N_k|}{m} \right\rceil = \left\lceil \frac{|N^k|}{m} \right\rceil = \left\lceil \frac{g(\{1, \ldots, k\})}{m} \right\rceil \leq d_k$$

for $k = 1, \ldots, c$. Hence all the jobs of C are on time. □

More in general, in the case $Q_m|p_i = 1$ of non-uniform machines we see that

Theorem 3. *There exists an $O(n + \log m)$ cover certificate for $Q_m|cos, p_i = 1|\sum w_j u_j$.*

Proof. In fact, $Q_m|p_j = 1|L_{max}$ can be solved in $O(n + \log m)$ time [11]. □

5. Separation of inequalities (3)

As observed in Section 3, the set covering model (2)–(4) written with all the inequalities (3) can be used for toy instances only: for practical problems one must separate violated inequalities at run time. Consider a relaxation $P(\bar{C})$ of (2)–(4) that includes only the cover inequalities in some $\bar{C}$, and let $\bar{u}$ be a (possibly fractional) solution of $P(\bar{C})$. To simplify presentation, assume d_j integer.

Let us consider $P_m|p_i = 1$ and $1|prec$. According to Theorem 2, a violated inequality corresponds to a cover C such that

$$\left\lceil \frac{g(C)}{m} \right\rceil > d_C \qquad \bar{u}(C) < 1 \tag{8}$$

with d_C as in Section 3. With integer d_j the leftmost position becomes $g(C) \geq md_C + 1$. So separation can be cast into the following maximization problem

$$\max_{C \subseteq J} \{g(C) - md_C : \bar{u}(C) < 1\} \tag{9}$$

Let $J_k = \{j \in J : d_j \leq d_k\}$. Problem (9) can be decomposed into r constrained submodular maximization subproblems with constant $d_C = d_k$

$$\max_{C \subseteq J_k} \{g(C) : \bar{u}(C) < 1\} \qquad k = 1, \ldots, r \tag{10}$$

With g as in Definition 3 and all $p_i = 1$, problem (10) is known as Budgeted Max Coverage. Khuller et al. [19] proved that (10) can be approximated within a ratio $(1 - e^{-1})$ by a modified greedy algorithm, and that no better approximation ratio can be achieved in polynomial time, unless P=NP. Sviridenko [32] generalized the result to any submodular function g, therefore including general p_i's.

Alternatively to (10), one can find a maximally violated inequality by seeking for a cover C that minimizes $\sum_{j \in C} \bar{u}_j$. Let $\mathbf{x} \in \{0, 1\}^m$ be the incidence vector of such a cover. Then $\mathbf{x}$ is an optimal solution of

$$\begin{aligned} \min & \sum_{j \in J} \bar{u}_j x_j && && (11)\\ & \sum_{i \in N} p_i z_i \geq (md_j + 1) x_j && j \in J \\ & \sum_{N_j \ni i} x_j \geq z_i && i \in N \\ & x_j, z_i \in \{0, 1\} && j \in J, i \in N \end{aligned}$$

where $\mathbf{z} \in \{0, 1\}^n$ is the incidence vector of the operations of N_C. The first inequality set states that $g(C) > md_C$, the second allows $i \in N_C$ only if i belongs to some N_j selected by C. Note that (11) does not admit $\mathbf{x} = \mathbf{z} = \mathbf{0}$ as a solution: in fact at least one z_i must be > 0 in order to fulfil the second constraint set and, consequently, the last constraint set triggers at least one x_j to a positive value. Finding a C maximizing the violation is NP-hard, because for $m = 1$ and $d_j = n - 1$ problem (11) becomes a general instance of Set Covering.

Problem (11) can be reduced by ordering the jobs by non-decreasing due date. Let $\mathbf{n}_j \in \{0, 1\}^n$ denote the incidence vector of N_j. The problem $\mathcal{P}_k$ reduced to sets N_j, $j = 1, \ldots, k$, and to the operations in $N^k = \{i \in N_j : 1 \leq j \leq k\}$ has a solution fulfilling the second and third constraints (11) only if $\mathbf{p} \cdot \sum_{j=1}^{k} \mathbf{n}_j$ is greater than md_k. Solving (11) corresponds then to solve

$$\begin{aligned} (\mathcal{P}_k) \qquad \min & \sum_{j=1}^{k} \bar{u}_j x_j && && (12)\\ & \sum_{j=1}^{k} n_{ij} x_j \geq z_i && i \in N^k \end{aligned}$$

$$\sum_{i=1}^{n} p_i z_i \geq m d_k + 1$$
$$x_j, z_i \in \{0,1\}$$

for $k = 1, \ldots, r$. In applications $\mathcal{P}_k$ may turn out to be relatively small and then solvable with a limited computational effort.

Let us finally recall that separation can be done efficiently in non-trivial special cases:

Theorem 4. *If all jobs have 2 unit operations, then inequalities (3) can be separated in polynomial time.*

Proof. See [1]. □

6. Solution algorithm

Formulation (2)–(4) is solved by branch-and-cut, first removing constraints (3), then re-inserting those violated by the current (integer or fractional) solution.

In order to reduce search, we make use of primal solutions and dual bounds. Dual bounds are obtained by solving the linear relaxation of the integer sub-problems encountered. In the following sections we focus on the separation of infeasible solutions (Section 6.1) and on the primal heuristic (Section 6.2).

6.1. Branch-and-cut

We start with a relaxation of problem (2)–(4). Each time the separation algorithm is called, a violated inequality (3) is constructed and added to the formulation. Violated inequalities are identified in different ways depending on whether the current solution $\bar{\mathbf{u}}$ is integer or not.

Initial covers. In principle, the root relaxation of problem (2)–(4) may contain no covers at all. It appears however reasonable to start it with a suitable cover subset. We tested a warm start using the covers generated by the conflict graph in [1]. In the *conflict graph* $G_C = (V_C, E_C)$, $V_C = J$ and $ij \in E_C$ if and only if there is no operation sequence that makes both i and j on time. Thus, $u_i + u_j \geq 1$ for all $ij \in E_C$, that is, E_C identifies a simple set of covers of J. Moreover, if $Q = \{1, \ldots, q\} \subseteq V_C$ is a clique of G_C, then Q identifies a cover of the form $u_1 + \ldots + u_q \geq q - 1$. For example, a clique involving three jobs 1, 2, 3 not only indicates that they cannot be pairwise on time but, moreover, that any two out of these three are surely late. This is derived by combining $u_1 + u_2 \geq 1, u_2 + u_3 \geq 1, u_1 + u_3 \geq 1$ with coefficients $\frac{1}{2}$ and then rounding the right-hand side up, obtaining $u_1 + u_2 + u_3 \geq \lceil 1.5 \rceil = 2$.

6.1.1. Current integer solution

A current integer solution $\bar{\mathbf{u}}$ gives a set of jobs that are supposedly on-time, but because no formulation is in general available at the node, we need a feasibility check. In the single machine case the check is done via Lawler's algorithm [20], solving $1|prec|L_{max}$ for the supposed on-time jobs: the solution is feasible if and only if $L^*_{max} = 0$. For convenience of the reader we next describe Lawler's algorithm as applied to our situation:

i) Let

$$\bar{J} = \{j \in J \text{ and } \bar{u}_j = 0\} \qquad \bar{N} = \bigcup_{j \in \bar{J}} N_j \qquad \bar{J}_i = \{j \in \bar{J} : i \in \bar{N}\}$$

and for any $i \in \bar{N}$ let $d_i = \min\{d_j : j \in \bar{J}_i\}$

ii) In order to fulfill the precedence relations among operations, re-define

$$d_i = \min\{d_i, \min_{k: i \prec k}(d_k - p_k)\}$$

for any $i \in \bar{N}$. Observe that this update does not increase L_{max} in any feasible schedule.

iii) Schedule the operations of $\bar{N}$ according to EDD first and compute the associated value of L^*_{max} If $L^*_{max} = 0$ then $\bar{\mathbf{u}}$ is optimal. Otherwise, $\mathbf{1} - \bar{\mathbf{u}}$ gives the coefficients of a violated inequality (3).

In the parallel machine case, minimizing L_{max} is even simpler as we deal with unit processing times, and an optimal operation schedule is obtained as specified in the proof of Theorem 5.

6.1.2. Current fractional solution

As explained in Section 5, the separation of a fractional solution $\bar{\mathbf{u}}$ requires in general to solve Max Budgeted Coverage problems of the form (10). Heuristic algorithms [19,32] proved to be ineffective; therefore we formulate the separation problem (12) and solve it exactly.

6.1.3. Lifting

Inequalities (3) can be strengthened by sequential lifting. Exact lifting coefficients cannot be computed efficiently, so we resort to a relaxation that only includes the lifted cover inequalities found so far and removes integrality clauses (4). In this way, the computation of the lifting coefficient reduces to a linear program. Deeper cuts are generally obtained by computing up-lifting coefficients in EDD sequence. A variable has generally more chances to be up-lifted when the due-date of the relevant job is not larger than the largest due-date of the jobs considered so far.

6.2. Primal heuristic

A primal heuristic is run at the root node of the branch-and-cut tree, in order to find a global upper bound *UB*. The heuristic aims at minimizing tardy jobs in a way compliant to the current solution at root $\bar{\mathbf{u}}$, and is run every time a violated inequality and a new solution are found. Define

$$J^1 = \{j \in J : \bar{u}_j = 1\},\ N^1 = \bigcup_{j \in J^1} N_j$$
$$\bar{J} = \{j \in J : \bar{u}_j \neq 1\},\ \bar{N} = \bigcup_{j \in \bar{J}} N_j.$$

Note that for $j \in \bar{J}$, variables $\bar{u}_j$ are either 0 or assume a fractional value.

Compute then the *UB* at the current node as follows:

1. Initialize $UB := \sum_{j \in J^1} w_j$
2. Associate a due date with each operation of $\bar{N}$ as in steps *i*) and *ii*) of Lawler's algorithm in Section 6.1.
3. Order the operations of $\bar{N}$ according to EDD first.
4. Try to schedule the first operation, say l, of $\bar{N}$.
5. If the schedule violates d_l, define $J^{d_l} = \{j \in \bar{J} : l \in N_j \text{ and } d_j = d_l\}$.
   (a) For each $j \in J^{d_l}$, update $UB := UB + w_j$;
   (b) Update $\bar{J} := \bar{J} \setminus J^{d_l}$;
   (c) For each operation $k \in N_j$, $k \neq l$ and $j \in J^{d_l}$, check if $k \in N_h$, for some $h \in \bar{J}$;
   - if $k \notin N_h$, for any $h \in \bar{J}$, then update $L := L \setminus \{k\}$ and go to step (d);
   - if otherwise $k \in N_h$, for some $h \in \bar{J}$, then go to step (d);

   (d) Check if there exists some other job $h \in \bar{J}$ such that $l \in N_h$ and $d_h \neq d_l$;
   - if $h \in \bar{J}$, then re-compute the due date to operations in $\bar{N}$ and re-order them according to EDD;
   - if otherwise no such h exists, then $L = L \setminus l$ and go to step 4;

   Otherwise, go to step 4.

Please cite this article as: C. Arbib et al., Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs, Omega (2018), https://doi.org/10.1016/j.omega.2018.04.002

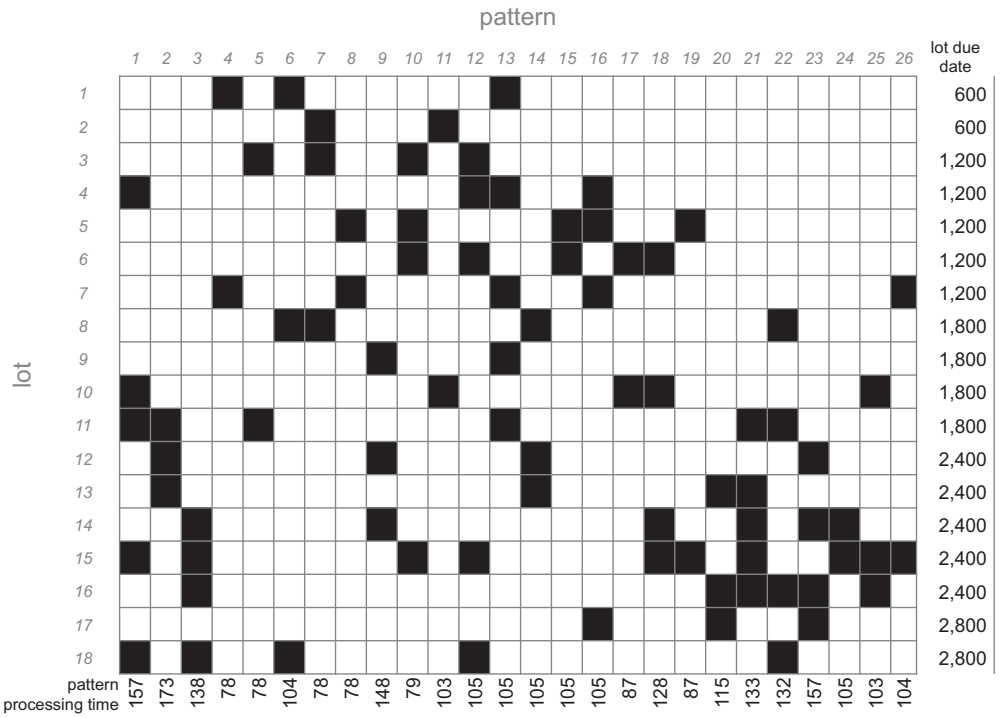

Fig. 4. A lot vs. pattern matrix from an industrial case: black cells give which pattern (column) feeds which lot (row). The last column reports lot due dates; the last row, pattern processing times.

6.3. A numerical example

In this example we refer to the industrial application described in Section 2. We start from a solution of a real cutting stock problem with $n = 26$ patterns. These produce 49 distinct part types in various amounts distributed in $r = 18$ lots.

The matrix in Fig. 4 represents which lots are fed by which pattern: lots are indicated by row, black cells hit the patterns that feed each single lot. Pattern processing times are obtained by simulating the corresponding cuts: the total processing time amounts to 2890 units; individual values for each pattern are reported in the last row of Fig. 4 in nominal units (actual times are confidential). Five different lot due dates are considered, at time 600 (lots 1 and 2), 1200 (lots 3–7), 1800 (lots 8–11), 2400 (lots 12–16) and 2800 (lots 17 and 18). The weights w_j of the objective function are assumed identical.

We solved the problem with the algorithm of Section 6 and with the Stable Set approach [1] adapted to general processing time by pseudo-polynomial transformation of the graph. An optimum solution has 6 late jobs, that is lots 3, 4, 5, 6, 9 and 14.

With the former approach, an integer optimum is found in 2.96 s CPU after separation of 121 fractional and 19 integer infeasible solutions. With the latter, the gap cannot be closed within the time limit: to close it, one can divide processing times and due dates by a scaling factor, say 60, and then round to the closest integer. By doing so, processing times range between 1 and 3, and both approaches find an optimum in fractions of seconds. But the solution found is completely different from the one obtained with non-scaled times, as it has no tardy job.

7. Computational experience

We tested the branch-and-cut algorithm of Section 6 on $1|cos|\sum w_j u_j$ and $P_2|cos, p_i = 1|\sum w_j u_j$ using both an artificial test bed created on purpose, and data derived from industrial cases. The algorithm was coded in standard C under GNU GCC MinGV compiler Version 3.81, developed within the Code::Blocks environment. IBM Ilog Cplex Version 12.6 was adopted as LP and MIP solver; Gomory and $(0, \frac{1}{2})$-cuts were deactivated as they only slowed runs down, while all other settings were kept at default values. The code was optimized with -O optimization option. Experiments were run on a 4-Core 3 GHz Intel I-7 processor with 24GB RAM and Linux Debian Kernel Version 2.6.26-2-amd64. A time limit of 7200 s was allowed for each run.

Details on algorithm implementation are given in Section 7.2. Some of its features were tested using the same data set as in [1]. To properly extend the range of the computational evidence, new problems were also created with the aim of identifying instances that are specifically challenging for the algorithm proposed here. The way these new classes of instances are generated is described in Section 7.1, alongwith details on a smaller testbed formed, with industrial data, by twenty real pattern sequencing problems. The outcome of the experiments is commented in Section 7.3.

7.1. Features of the datasets

The main test bed for the experiments consists of an *artificial dataset* and a set of *real industrial problems*.[3]

To form the artificial dataset, we made a preliminary run in order to detect the features that make an instance significant. Generally speaking, the completion time of any job j lies in the interval

$$T_j = [C_j^{min}, C_{max}] = \left[\sum_{i \in N_j} p_i, \sum_{i \in N} p_i\right]$$

where operation times p_i are either unitary ($P_2|cos, p_i = 1|\sum w_j u_j$) or uniformly generated in the integer interval [1, 100] ($1|cos|\sum w_j u_j$).

[3] All problem instances can be downloaded from ftp://bioinformatics.iasi.cnr.it/public/Common-Operation-Scheduling-Instances/.

Problems are then classified by four parameters: number n of operations, number r of jobs, *density* and *looseness*.

The density δ of a problem is that of its operation-job matrix, that is,

$$\delta = \frac{\sum_{j\in J} |N_j|}{nr}$$

We define the looseness of job j as the ratio $\lambda_j = \frac{d_j - C_j^{min}}{C_{max} - C_j^{min}}$: the larger the λ_j, the less operations of j need to be anticipated if one wants the job on time (clearly, $\lambda_j \in [0, 1]$ because jobs with $d_j < C_j^{min}$ cannot be on-time and can therefore be dropped). Note that the due date of job j is completely defined by the job looseness. We then define the looseness λ of an instance as the average looseness of its jobs,

$$\lambda = \frac{\sum_{j\in J} \lambda_j}{r}$$

and say in particular that an instance is *uniformly loose with looseness* λ if all of its jobs have looseness $\lambda_j = \lambda$.

It is reasonable to expect that the easiest instances, for a given problem size and density, will be those for which the looseness is either (*i*) very small or (*ii*) very large. The former will tend to have no job on time, the opposite for the latter. To validate this intuition, we run a preliminary test with the aim of estimating the relation between looseness and density on 132 uniformly loose instances with 40 operations and 40 jobs. Based on the outcome obtained, we settled the test bed described below.

The artificial dataset comprises 375 non-uniformly loose instances of different sizes and densities. Looseness varies in a way that differs from the first test: each instance is constructed referring to an interval in which due dates are uniformly generated at random with maximum looseness λ_{max}. Specifically, d_j is picked between its minimum possible completion time C_j^{min} and $C_j^{min} + (C^{max} - C_j^{min})\lambda_{max}$. Randomization implies that each job j has an individual looseness λ_j. The experiment setting is described below:

size: $n = 40, 80, 100$, $r = 40, 80, 100$ in five combinations $n \times r$:
- small (S): 40×40;
- medium-chubby (MC): 80×40;
- medium-thin (MT): 40×80;
- large (L): 80×80;
- extra large (XL): 100×100.

δ: three values from sparser ($\delta = 20\%$) to denser ($\delta = 40\%$) matrices.

λ_{max}: five values from very tight ($\lambda_{max} = 0.1$) to very loose ($\lambda_{max} = 0.9$) due dates;

obj: $w_j = 1$ *for all* $j \in J$.

For completeness, we also tested 20 instances of direct industrial derivation. Those instances were obtained, in the way described in Section 2, from 10 solutions of cutting stock problems presented by clients of the SCM Group [30]. Jobs correspond to orders of parts, operations to the cutting patterns that form the solution, and the technical features of the machine (Gabbiani series) used in the production line impose the time length of each cut and of the whole process.

These instances are classified as small, medium and large according to the number of patterns and orders. Original instances 3.2 and 3.11, respectively with 8 and 7 patterns, are too small and were therefore merged into a single instance 3.2 + 11 with 15 patterns and 9 orders generated by random part mixes. Similarly, instances 3.28 (small, with 26 patterns) and 3.38 (medium, 32 patterns) were merged so as to obtain the new instance 3.28 + 38 with 58 patterns and 20 orders. The whole cutting process duration ranges from 1889 to 4096 time units for small instances, and from 4205 to 8301 for medium ones; the duration is 16,350 time units for problem 3.19 and 23,577 time units for problem 3.41.

We could not access the company delivery plans for those instances, so we created due dates in two different ways: *regular*, i.e. equally spaced, and *random*, i.e. randomly placed in the process time interval. The due dates generated in each way were then randomly associated with jobs. The resulting testbed has 20 instances and is depicted in Table 1.

7.2. Algorithm implementation

The final implementation of the algorithm is the result of choices made on few possible variants. The major ones, described below, are the outcome of tests done before setting the main testbed described in Section 7.1.

A first test on a subset of reference instances was done to identify the best initial cover set before separation (see Section 6.1). We tested four different options:

i) Empty formulation.
ii) Only 2-job covers associated with arcs of the job conflict graph.
iii) Option (*ii*) plus cover inequalities obtained by random search.
iv) Option (*ii*) plus cover inequalities associated with cliques of the job conflict graph.

The above options were compared on the basis of root gap and CPU time, and the trade-off that emerged indicated (*ii*) as the best option: all the additional strategies to enlarge the initial cover set had in fact no measurable or monotonic effect on performance. Root gaps for the formulation presented in the following will thus refer to an initial LP with only covers formed by 2 jobs.

Separation is invoked by a callback subroutine at each node of the branch-and-bound tree. We noted how a less frequent invocation of the separation callback implied additional branching work on an incomplete formulation, with negative effect of final solution times. At each iteration of the separation algorithm, only the most violated cut is added to the formulation. Separation of integer solutions is done by the simple algorithms described in Section 6.1. As noticed, heuristic separation of fractional solutions via the methods in [19,32] turned out to be ineffective, so we performed exact separation via MIP (12).

We recorded computation time with and without the lifting procedure described in Section 6.1. We did not observe any improvement of efficiency by lifting, as the CPU time required by each up-lifting coefficient (an LP solved by Cplex) was not at all compensated by the cut improvement. For this reason, all experiments reported in the following refer to the implementation where lifting is not used.

Another preliminary test concentrates on the effects of the primal heuristic of Section 6.2 in terms of root gap reduction, CPU time and number of nodes. Table 2 reports such values for six problems of small and medium size with different values of density δ and looseness λ. The primal heuristic (denoted as *p.h.* in the table) is of course important to quickly determine a primal solution, but, as far as gap reduction and CPU time are concerned, its contribution does not have a straightforward interpretation: as a matter of fact, Table 2 shows cases where CPU time increases.

A final observation regards the level of violation of the cover inequalities generated. In general, cover violation at run time has a predictable sawtooth pattern, see the example depicted in Fig. 5 for instance C of Table 2. This behavior is typical of a dynamic formulation where inequalities are generated as soon as infeasible integer solutions pop up, and was found in the all runs described in this section. To avoid excessive tailing we set a cut-off threshold of 0.01.

Please cite this article as: C. Arbib et al., Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs, Omega (2018), https://doi.org/10.1016/j.omega.2018.04.002

Table 1
Computational results of 20 industrial instances of the pattern sequencing problem. Number of operations, jobs, average number of pieces per job and matrix density are indicated in columns 3–6. Each instance was run with two due date configurations (*regular*, *random*). Fractional/integer cuts and CPU time in seconds are reported in columns 8–10.

Instance class	Problem name	Op. n	Jobs r	#items per job	Density δ	Due dates	*#frac.* cuts	#int. cuts	Time (s)	Obj. value
Small	3.2+11	15	10	7.1	0.38	regular	1	4	0.010	3
						random	1	2	0.010	3
Small	3.29	15	10	6.6	0.33	regular	3	5	0.010	3
						random	8	15	0.050	5
Small	3.12	17	12	7.25	0.36	regular	3	14	0.030	2
						random	0	2	0.005	1
Small	3.28	26	18	6.33	0.22	regular	5	7	0.030	4
						random	12	50	0.190	5
Medium	3.38	32	20	5.7	0.16	regular	1	4	0.010	3
						random	11	3	0.250	3
Medium	3.20	39	26	7.42	0.17	regular	37	94	1.290	8
						random	10	12	0.370	4
Medium	3.28+38	58	20	11.4	0.18	regular	11	53	0.330	9
						random	17	24	0.460	8
Large	3.39	123	26	7.46	0.62	regular	1	0	0.020	19
						random	1	0	0.010	18
Large	3.41	123	26	23.23	0.17	regular	24	59	1.630	8
						random	56	128	6.410	9
Large	3.19	139	26	19.34	0.12	regular	6	10	0.5	5
						random	4	4	0.860	5

Table 2
CPU times, root gap and number of branch nodes with and without the primal heuristic (p.h.) of Section 6.2.

Prob.	Op.	Jobs	Density	Looseness	Root gap		CPU time (s)		# of nodes	
id.	n	r	δ	λ	p.h.	no p.h.	p.h.	no p.h.	p.h.	no p.h.
A	40	40	20%	0.5	0.0 %	0.0%	195.09	205.73	1806	2058
B	40	40	20%	0.9	0.0 %	0.0%	414.83	293.51	784	512
C	40	40	40%	0.9	97.37%	97.37%	90.26	97.15	496	452
D	40	80	30%	0.1	0.0 %	0.0%	45.65	43.81	243	205
E	40	80	30%	0.3	0.0 %	0.0%	117.89	132.57	1336	1381
F	40	80	40%	0.5	97.62%	97.62%	214.68	239.50	2212	2390

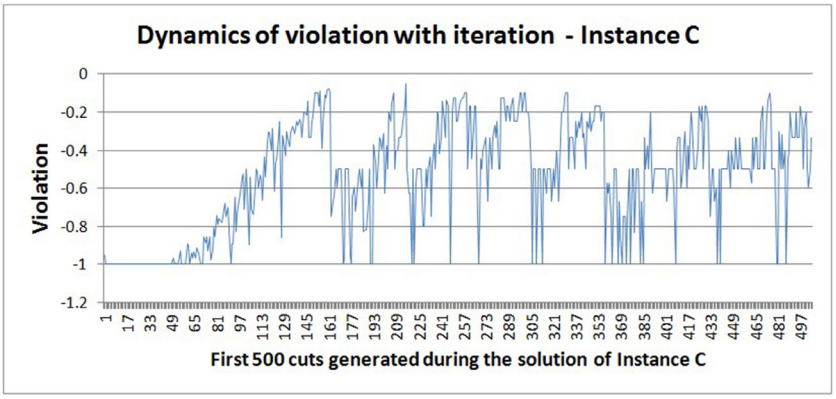

Fig. 5. Dynamics of cover violation for the first 500 separations in Instance C of Table 3.

7.3. Test results

We compared the branch-and-cut method of Section 7.2 and the extended Stable Set approach [1] on S-instances, with operation times either unitary ($P_m|cos, p_i = 1$) or generated as in Section 7.1 ($1|cos$).

In the parallel machines case, Stable Set outperforms branch-and-cut in all the instances tested. On the contrary, in the single machine case with general processing times, Stable Set does not close the gap within the time limit in any of the problems tested. Branch-and-cut solves instead all S-instances in less than 1800 s; the max CPU time, 1688.09 s, occurred with $\delta = 20\%$ and $\lambda_{max} = 0.9$.

A similar result is obtained for industrial instances. Table 1 reports, in columns 8–11, the branch-and-cut performance on this test bed: most problems were solved at optimality in fractions of seconds, and always in less than 2 s. On the other hand, similarly to the example in Section 6.3, none of them could be solved in reasonable time (7200 s) using the Stable Set approach [1].

Table 3 reports the general branch-and-cut performance in classes from S to XL. Each row corresponds to a pair $\langle$size, $\delta\rangle$ and to 25 runs, providing values for five runs per five values of λ_{max} on problems generated as described. Rows are three-partitioned

Please cite this article as: C. Arbib et al., Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs, Omega (2018), https://doi.org/10.1016/j.omega.2018.04.002

Table 3
Each line reports, for a different problem size and matrix density, average values of 5 problems generated at random. **Row 1**: number of optima found. **Rows 2-3**: average and standard deviation of solution times. A dash means that time limit expired before closing the gap. Times in seconds.

Size	Operations	Jobs	Density	Maximum looseness λ_{max}				
class	n	r	δ	0.1	0.3	0.5	0.7	0.9
				5	5	5	5	5
S	40	40	40%	11.55	12.36	28.90	81.83	141.44
				0.72	1.32	6.77	18.39	43.61
				5	5	5	5	5
S	40	40	30%	1	16.08	65.81	221.99	373.25
				1.40	3.53	19.19	74.42	159.10
				5	5	5	5	5
S	40	40	20%	9.32	39.23	252.19	858.10	756.25
				1.00	11.53	97.08	348.10	558.76
				5	5	5	5	5
MC	40	80	40%	35.21	23.63	62.47	247.95	877.08
				6.54	4.95	14.53	48.61	207.06
				5	5	5	5	5
MC	40	80	30%	24.40	35.22	201.99	793.01	2369.05
				6.33	1.23	49.34	199.49	1391.15
				5	5	5	3	2
MC	40	80	20%	16.26	98.66	923.03	4793.21	5698.11
				1.66	26.85	387.16	2366.52	2062.55
				5	5	5	5	2
MT	80	40	40%	74.00	97.35	374.43	2059.62	5528.41
				9.66	12.37	134.18	1010.41	2293.65
				5	5	5	1	0
MT	80	40	30%	75.77	175.98	1724.68	652	–
				19.70	55.42	901.18	1520.53	
				5	5	1	0	0
MT	80	40	20%	57.82	741.88	6628.87	–	–
				12.42	334.91	1277.09		
				5	5	5	4	0
L	80	80	40%	171.58	166.42	806.37	5911.73	–
				17.63	22.30	146.01	1007.17	
				5	5	5	0	0
L	80	80	30%	132.61	281.46	3817.57	–	–
				22.92	35.46	885.26		
				5	5	0	0	0
L	80	80	20%	100.04	1606.63	–	–	–
				6.66	246.06			
				5	5	5	0	0
XL	100	100	40%	741.58	438.81	3024.90	–	–
				622.09	153.46	1054.56		
				5	5	0	0	0
XL	100	100	30%	500.62	744.64	–	–	–
				375.54	95.31			
				5	4	0	0	0
XL	100	100	20%	250.80	6477.48	–	–	–
				26.48	879.76			

and give, in the order, the number of optima found, the average CPU time observed in the five problems solved, and the relevant standard deviation. Problems generally become harder with lower density and looser due dates. Fig. 6 shows the dependance of average CPU time on the combination looseness-density for class S up to L instances (the chart for class XL is basically flat for difficult instances). Evidently, problems become in general more and more difficult as long as they get looser and sparser. Branch-and-bound nodes are always in a manageable amount: in class XL, their number averages around 3000 with peaks well below 10,000.

MT-instances have more variables and seem harder than MC: in the latter class, all problems were solved within the time limit (4582.38 s in the worst case, corresponding to $\lambda_{max} = 0.7$ and $\delta = 20\%$). The largest CPU times always occur for $\lambda_{max} = 0.9$, and with this value we could never close the optimality gap in classes L and XL.

No low-density XL-instance could be solved within a time limit of 7200 s when $\lambda \geq 0.5$. In class XL we solved slightly more than 45% of the problems within time limit, taking 716 s on average (7113.46 s maximum). In class L we solved about 59% of the problems within time limit, taking 1313 s on average (6769.10 s maximum). CPU times in the two classes are distributed as in Fig. 7.

We evaluated the effect of the primal heuristic in terms of solution quality at root. It turns out that it provides solutions of good quality for problems of small size, while its quality becomes less interesting as size increases. For randomly generated test problems, the average difference of tardy jobs in the optimal and the heuristic solution is 1.2 if we limit our attention to classes S, MC, MT. The difference rises to 9.75 for L matrices and to 12.25 for XL. Noticeably, the average increase of tardy jobs in the 20 industrial instances of Table 2 is 6, despite they are of small size and easily solvable.

Finally, we found an optimum in less than 7200 s in all problems with $\lambda_{max} = 0.1$ or 0.3 and $\delta \geq 30\%$ and, regardless of density, in 149 cases of 150: as expected, the only unsolved case is an XL-instance with $\lambda_{max} = 0.3$ and $\delta = 20\%$.

8. Conclusions

In this paper we addressed a family of scheduling problems (denoted as common operation scheduling, COS), where operations

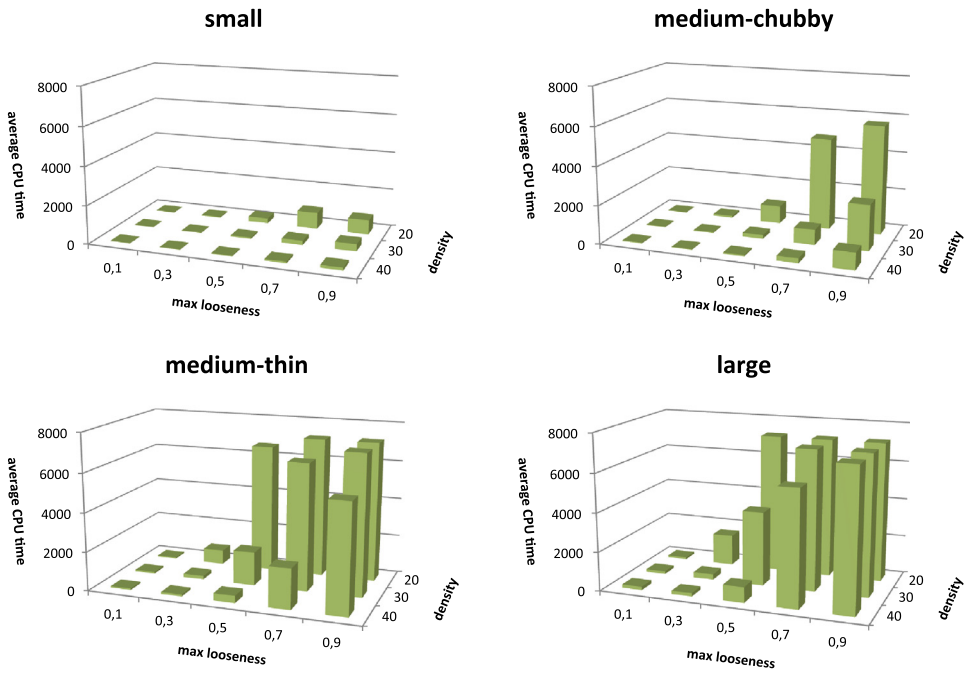

Fig. 6. CPU time (seconds) with S, MC, MT and L instances, average of five problems per each value of looseness and density.

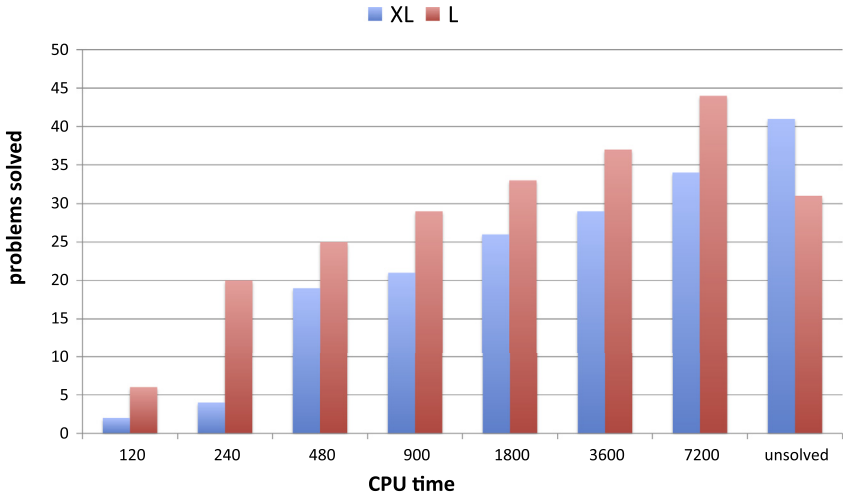

Fig. 7. Number of L and XL instances solved/unsolved within time limit (seconds).

can be shared among a set of jobs with individual due dates. Potential applications range from progressive network recovery to pattern sequencing in cutting stock, and from maintenance logistics to disassembly processes in manufacturing. We focussed on the minimization of the weighted sum of tardy jobs, also considered by Arbib et al. [1] for a particular case, and developed a set covering formulation with inequalities increasing exponentially with the number of jobs. Separation/lifting of cover inequalities was addressed and its complexity stated in different cases.

In general, the method is able to solve problems of two types, in both cases, with general precedence relations among operations: single machine with general processing times $(1|cos|\sum w_j u_j)$, or parallel machines with unit processing times $(Q_m|cos, p_i = 1|\sum w_j u_j)$.

A primal heuristic and a branch-and-cut algorithm were then proposed and tested on a significant sample of COS problems partly derived from industrial applications and partly generated at random, identifying the main features of difficult instances. At the same time, the algorithm in [1] (based on a Stable Set formulation) was extended to tackle the two cases. While for unit processing times the Stable Set approach maintains its competitiveness also with parallel machines, it is of no help for general processing times, where instead the new approach is able to solve problems with up to 100 jobs and 100 operations.

Various options were evaluated to optimize the method proposed, among which heuristic cover separation, sequential lifting and primal heuristic. Cover separation is a Budgeted Max Coverage problem: available heuristics for this problem fail to be efficient separators. The procedure we adopted for sequential lifting requires the solution of a linear program for each coefficient, but does not provide speed-up. Finally, although the primal heuristic provides feasible solutions in very short time, its contribution to algorithm efficiency varies depending on the problem instance. Thus, directions for future research may include testing different heuristics for separation, lifting and primal solution.

Please cite this article as: C. Arbib et al., Common operation scheduling with general processing times: A branch-and-cut algorithm to minimize the weighted number of tardy jobs, Omega (2018), https://doi.org/10.1016/j.omega.2018.04.002

Finally, an open problem is the design of an effective exact method for the more general case of COS problems on parallel machines with general processing times.

Acknowledgement

The authors are grateful: to the Area and Associate Editors and the anonymous Referees, who thoroughly revised the preliminary drafts and gave us stimulating suggestions to improve the quality of the paper; to Franca Rinaldi (Università di Udine), who pointed out the relation between COS and ordinary precedence constrained scheduling. Special thanks to dr. eng. Fabrizio Ratti and his technical staff at SCM Group, and to Fabrizio Marinelli (Università Politecnica delle Marche), who provided us with real data for the pattern sequencing application.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.omega.2018.04.002.

References

[1] Arbib C, Felici G, Servilio M. Sorting common operations to minimize the number of tardy jobs. Networks 2014;64:306–20.
[2] Arbib C, Marinelli F. On cutting stock with due dates. Omega Int J of Manage Sci 2014;46:11–20.
[3] Arbib C, Marinelli F. Maximum lateness minimization in one-dimensional bin packing. Omega Int J of Manage Sci 2017;68:76–84.
[4] Arbib C, Marinelli F, Ventura P. One-dimensional cutting stock with a limited number of open stacks: bounds and solutions from a new integer linear programming model. Int Trans Oper Res 2016;23(1-2):47–63.
[5] Baptiste P. Simple MIP formulations to minimize the maximum number of open stacks. In: Smith BM, Gent IP, editors. Constraint modelling challenge – IJCAI 2005, Edinburgh, Scotland; 2005. p. 9–13. July 31
[6] Becceneri JC, Yanasse HH, Soma NY. A method for solving the minimization of the maximum number of open stacks problem within a cutting process. Comput Oper Res 2004;31:2315–32.
[7] Belov G, Scheithauer G. Setup and open-stacks minimization in one-dimensional stock cutting. INFORMS J Comput 2007;19(1):27–35.
[8] Bennell JA, Lee LS, Potts CN. A genetic algorithm for two-dimensional bin packing with due dates. Int J Prod Econ 2013;145(2):547–60.
[9] Cheng TCE, Diamond J, Lin BMT. Optimal scheduling in film production to minimize talent hold cost. J Optim Theory Appl 1993;79:197–206.
[10] Cheng CH, Feiring BR, Cheng TCE. The cutting stock problem – a survey. Int J Prod Econ 1994;36(3):291–305.
[11] Dessouky MI, Lageweg BJ, Lenstra JK, van de Velde SL. Scheduling identical jobs on uniform parallel machines. Statistica Neerlandica 1990;44(3):115–23.
[12] EC directive. 2000/53/EC on end-of life vehicles; 2000. http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CONSLEG:2000L0053:20050701:EN:PDF September 18.
[13] EC directive. 2005. 2005/64/EC on the type-approval of motor vehicles with regard to their reusability, recyclability and recoverability and amending Council Directive 70/156/EEC, http://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX&percent;3A32005L0064 October 26.
[14] Fink A, Voß S. Applications of modern heuristic search methods to pattern sequencing problems. Comput Oper Res 1999;26:17–34.
[15] de la Banda M G, Stuckey PJ. Dynamic programming to minimize the maximum number of open stacks. INFORMS J Comput 2007;19(4):607–17.
[16] Garey MR, Johnson DS. Two-processor scheduling with start-times and deadlines. SIAM J Comput 1977;6:416–26.
[17] Garey MR, Johnson DS. Computers and intractability. A Guide to the Theory of NP-completeness, San Francisco. Freeman WH, editor; 1979.
[18] Johnston RE. Cutting pattern and cutter schedules. ASOR Bull 1984;4(1):3–15.
[19] Khuller S, Moss A, Naor J. The budgeted maximum coverage problem. Inf Process Lett 1999;70(1):39–45.
[20] Lawler EL. Optimal sequencing of a single machine subject to precedence constraints. Manage Sci 1973;19, 5:544–6.
[21] Linhares A, Yanasse HH. Connections between cutting-pattern sequencing, VLSI design, and flexible machines. Comput Oper Res 2002;29(12):1759–72.
[22] Lopes IC, de Carvalho J M V. An integer programming framework for sequencing cutting patterns based on interval graph completion. In: Proc. of the VII ALIO-EURO workshop on applied combinatorial optimization. Portugal: Porto; 2011. p. 47–50. May 4-6
[23] Lovász L. Submodular functions and convexity. In: Bachem A, Grötschel M, Korte B, editors. Mathematical programming - the state of the art. Springer-Verlag, Berlin Heidelberg; 1983. p. 235–57.
[24] Munawar SA, Kapadi MD, Patwardhan SC, Madhavan KP, Pragathieswaran S, Lingathurai P, Gudi RD. Integration of planning and scheduling in multi-site plants: application to paper manufacturing. In: Puigjaner L, Espuña A, editors. European symposium on computer aided process engineering. Amsterdam: Elsevier BV; 2005. p. 1621–6.
[25] Pegels CC. A comparison of scheduling models for corrugator production. J Indus Engs 1967;18(8):466–72.
[26] Pierce JF. Pattern sequencing and matching in stock cutting operations. TAPPI J 1970;53(4):668–78.
[27] Pinedo ML. Scheduling: theory, algorithms, and systems. Springer Science+Business Media; 2008.
[28] Reinertsen H, Vossen TWM. The one-dimensional cutting stock problem with due dates. Eur J Oper Res 2010;201(3):701–11.
[29] Respicio A, Captivo ME. Bi-objective sequencing of cutting patterns. In: Ibaraki T, Nonobe K, Yagiura M, editors. Metaheuristics: progress as real problem solvers, operations research/computer science interfaces series, vol 32. Boston, MA: Springer; 2005. p. 227–41.
[30] SCM Group. https://www.scmgroup.com/it.
[31] Simons BB, Warmuth M. A fast algorithm for multiprocessor scheduling of unit-length jobs. SIAM J Comput 1989;18(4):690–710.
[32] Sviridenko M. A note on maximizing a submodular set function subject to a knapsack constraint. Oper Res Lett 2004;32(1):41–3.
[33] Toyota Sustainability Report. 2016. Vehicle Recycling. http://www.toyota-global.com/sustainability/report/vehicle_recycling/pdf/vr_p4-p6.pdf.
[34] Wang J, Qiao C, Yu H. On progressive network recovery after a major disruption. In: Proc. of IEEE INFOCOM 2011; 2011. p. 1925–33. Shanghai, P.R. China April 10-15
[35] Woeginger GJ. On the approximability of average completion time scheduling under precedence constraints. Discrete Appl Math 2003;131, 1:237–52.
[36] Yanasse HH. On a pattern sequencing problem to minimize the maximum number of open stacks. Eur J Oper Res 1997;100(3):454–63.
[37] Yanasse HH, Pinto Lamosa MJ. An integrated cutting stock and sequencing problem. Eur J Oper Res 2007;183(3):1353–70.