Available online at www.sciencedirect.com
ScienceDirect
Theoretical Computer Science 361 (2006) 257–274
Theoretical Computer Science
www.elsevier.com/locate/tcs

Open block scheduling in optical communication networks

A.A. Ageev[a,1,4], A.V. Fishkin[b,2], A.V. Kononov[a,*,3,4], S.V. Sevastyanov[a,4]

[a] *Sobolev Institute of Mathematics, Acad. Koptyug pr. 4, 630090 Novosibirsk, Russia*
[b] *Institut für Informatik und Praktische Mathematik, Christian-Albrechts-Universität zu Kiel, Olshausenstr. 40, 24 098 Kiel, Germany*

Abstract

In this paper the process of data transmission in star coupled optical communication networks is modelled as a shop-type scheduling problem, where channels (wavelengths) are treated as machines. We formulate an *Open Block problem* with the minimum makespan objective ($OB||C_{\max}$) in which a relation of a new type between the operations of each job is introduced: any two operations of a job have identical processing times and may be processed either simultaneously (in a common *block*) or, alternatively, at disjoint time intervals. We show that the problem is polynomially solvable for 4 machines, NP-hard for 6 machines and strongly NP-hard for a variable number of machines. For the case of a variable number of machines we present a polynomial time $\sqrt{2}$-approximation algorithm and prove that there is no polynomial time ρ-approximation algorithm with $\rho < 11/10$, unless P = NP. For the case when the number of machines is fixed, we show that the problem admits a linear time PTAS. In addition, we show that the two-machine problem with release dates is NP-hard in the strong sense.

Keywords: Scheduling; Approximation; Open shop; Data packets

1. Introduction

In this paper we study a scheduling problem that models the process of data transmission in optical communication networks. Let us briefly overview the "physics" of the process.

Wavelength Division Multiplexing (WDM) technology enables the use of multiple wavelengths, each carrying a separate "channel", in a single fiber [7]. WDM is mainly applied in wide area networks (WANs) [20]. Only very recently, a number of local area WDM testbeds have been constructed [2,10,19], and the first WDM LANs are expected to enter the market soon.

In general, a local WDM optical network may be constructed by connecting network nodes via two-way fibers to a passive star coupler, as shown in Fig. 1A. A node transmits a data packet to the star on several available wavelengths (channels) simultaneously, using a laser which produces an optical information stream. The information streams from

* Corresponding author.
E-mail addresses: ageev@math.nsc.ru (A.A. Ageev), avf@informatik.uni-kiel.de (A.V. Fishkin), alvenko@math.nsc.ru (A.V. Kononov), seva@math.nsc.ru (S.V. Sevastyanov).

[1] Supported by RFBR, Grant 06-01-00255.
[2] Supported by EU-Project CRESCO, IST-2001-33135.
[3] Supported by RFBR, Russian–Taiwan grant 05-06-90606.
[4] Supported by RFBR, Grant 05-01-00960.

0304-3975/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2006.05.014

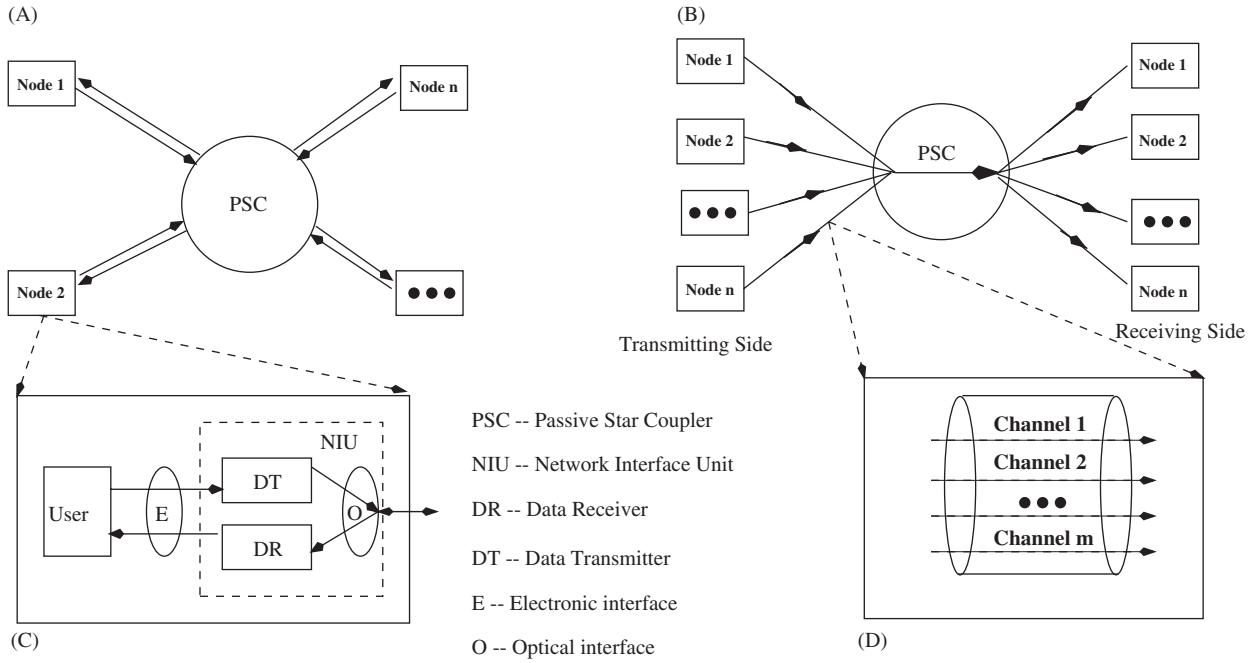

Fig. 1. A passive-star-based local optical WDM network: (A) LAN; (B) multicasting; (C) node block diagram; (D) optical fiber diagram.

multiple sources are optically combined by the star and then forwarded to all of the nodes on their receiver fibers (Fig. 1B). A node's receiver is tuned to only one of the channels; hence, it can receive a designated information stream. Communication between a source and destination nodes takes place directly, i.e., via the *single-hop method* [11], and takes time that may be neglected. What is really essential in this model is the time needed to transform information (issued from a node) from the electronic to the optical form (Fig. 1C). Also note that, when a source transmits on a particular channel, more than one receiver can be tuned to this channel, and all such receivers may pick up the information stream. Thus, the passive-star can support *multicast* services.

In this paper we concentrate on the design of efficient algorithms for the scheduling problem of a packet-switched broadcasting in such networks. We assume that each node has a receiver which is pre-tuned to a fixed channel, and has a "tunable" transmitter (laser) which can transmit on several channels simultaneously. Each node (as an element of the transmitting side, see Fig. 1B) has a multicast data packet that must be delivered to all receiving nodes in its destination set. The goal is to minimize the overall transmission time, i.e., the time needed for all data packets to be received.

We consider a multicast service with *fanout splitting*, which means that any data packet can be split into several packets, identical by their contents but having element-wise different sets of destination channels. The transmission of a data packet completes, when each destination node receives at least one copy of the packet. (For details, see [12,18].) Furthermore, data transfer in such optical networks must meet the following natural restrictions:

(a) at any point in time no two transmitters may transmit data on the same wavelength (channel);

(b) at most one data packet can be sent by a transmitter at a time (although, on several channels simultaneously). Below we formulate a new scheduling problem based on these conditions.

1.1. Open block problem (short notation: $OB||C_{\max}$)

We are given a set of n jobs $\boldsymbol{J} = \{J_1, \ldots, J_n\}$ and a set of m machines $\boldsymbol{M} = \{M_1, \ldots, M_m\}$. For each job J_j, an integer processing time p_j and a subset of indices $v_j \subseteq \{1, \ldots, m\}$ are specified, so that job J_j consists of $|v_j|$ operations that have to be processed on machines $\{M_i \mid i \in v_j\}$ and have identical processing times equal to p_j. Any two operations of a job may run either simultaneously (in a *block*), or should not overlap in time. No restrictions on the combining of operations into blocks and on the order of processing the blocks are specified.

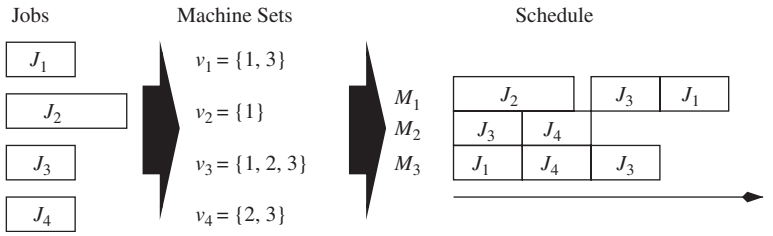

Fig. 2. A feasible open-block schedule for jobs $J_1, \ldots, J_4$ on three machines.

Each machine can process at most one operation at a time. Preemption is not allowed. The goal is minimizing the makespan.

In the above model channels correspond to machines, while data packets correspond to jobs. A job has an operation on a machine, if the data packet corresponding to that job must be transmitted on the channel corresponding to the machine. Fig. 2 illustrates a feasible schedule for a given instance of the OB-problem with jobs $\{J_1, \ldots, J_4\}$ and machines M_1, M_2, M_3. One can see that, e.g., job J_3 is scheduled in two blocks, while both operations of job J_4 are processed in a single block.

Thus, a novel feature of the above problem is a relation of a new type between the operations: now any two operations of a job may be processed either completely simultaneously (like a job in multiprocessor scheduling), or at disjoint time intervals (which is typical for classical models). Informally, here a job is an *open block* which can "break" into several "sub-blocks" not overlapping in time. To our best knowledge, no similar scheduling models have been considered in the literature.

1.2. Related problems and results

Clearly, given an instance of the OB-problem, it can also be treated either as an instance of the *Open Shop* problem [8] - on one hand or of the *Multiprocessor Dedicated Tasks* problem [4]—on the other hand. Indeed, let us consider two extremes: either no two operations of each job run overlapping, or all operations of each job run simultaneously (in parallel on several machines). This leads to two well-known scheduling problems: to a special case of the Open Shop problem (in which all operations of a job have identical processing times)—in the first case, and to the Multi-Processor dedicated Tasks scheduling problem (*MPT*)—in the second case.

Moreover, any schedule constructed for that instance and feasible with respect to either the MPT, or the Open Shop problem is also feasible with respect to the OB-problem, and therefore, efficient methods elaborated for those two problems can be useful to provide good solutions for the OB-problem as well.

Let us first review the known results for the $O||C_{\max}$ problem. As known from [8], the $O2||C_{\max}$ problem admits a linear-time solution, while the $O3|p_{ij} = p_j|C_{\max}$ problem is NP-hard. NP-hardness of the general $O||C_{\max}$ problem in the strong sense follows from the result proved in [21]: deciding whether there exists a schedule of length at most 4 is strongly NP-complete. This also implies that for the $O||C_{\max}$ problem with a variable number of machines and any $\rho < 5/4$ there is no ρ-approximation algorithm, unless P = NP. At the same time, it is well known that any greedy algorithm provides a 2-approximation. For the $Om||C_{\max}$ problem (when the number of machines m is constant), there is a polynomial time approximation scheme (PTAS) [16]. A number of efficient "statistically optimal" algorithms (constructing optimal schedules for almost all instances) were designed in [5,13–15] for the $O||C_{\max}$ problem.

The MPT-problem seems to be much harder. Already $MPT3||C_{\max}$ is strongly NP-hard [9]. However, in the same work it is shown that $MPTm|p_j = 1|C_{\max}$ is solvable in polynomial time. A PTAS is proposed for $MPTm||C_{\max}$ in [1]. On the other hand, $MPT|p_j = 1|C_{\max}$ cannot be approximated within a factor of $m^{1/2-\varepsilon}$, unless $\mathbb{P} = \mathbb{NP}$ [6]. Matching this, polynomial time $O(\sqrt{m})$-approximation algorithms for $MPT|p_j = 1|C_{\max}$ and $MPT||C_{\max}$ were recently proposed in [3].

1.3. Our results

It is shown that the problem can be solved in polynomial time, if the number of machines is at most 4, whereas the 6-machine problem is NP-hard. The problem with a variable number of machines is shown to be strongly NP-hard,

while adding different release times makes the problem strongly NP-hard already for two machines. The problem with a variable number of machines admits a polynomial time $\sqrt{2}$-approximation, but cannot be approximated better than within a factor of $11/10$ of the optimum, unless $\mathbb{P} = \mathbb{NP}$. The result follows from the analysis of the existence of short schedules: as shown in the paper, the existence of a schedule of length no more than 9 can be checked in polynomial time, whereas the same problem for the time interval of length 10 is NP-complete. In contrast to the above result, for the case of any fixed number of machines it can be shown that the problem admits a linear time PTAS.

As an easy corollary of results known for the Open Shop problem, we also establish that the problem becomes polynomially solvable for the set of instances with sufficiently large values of the maximum machine load—in comparison with the maximum job processing time.

The paper is organized as follows. In Section 2 we introduce notation and formulate some preliminary results. In Section 3 a few known polynomially solvable cases of the OB-problem are presented. Section 4 contains NP-hardness and inapproximability results. A few constant-factor approximation algorithms and a PTAS are presented in Section 5. In the final section we discuss open questions.

2. Preliminaries

Here, we introduce basic notation and definitions. We also formulate a few known results to be used further.

We define the *maximum processing time* of the operations as $p_{\max} = \max_j p_j$, and the *length* d_j *of job* $J_j \in \boldsymbol{J}$ as the total length of all its operations; $d_{\max} = \max_{J_j \in \boldsymbol{J}} d_j$ stands for the *maximum job length* over all jobs. The total processing time of the operations on machine M_i is denoted by ℓ_i and called the *load* of machine M_i; $\ell_{\max} = \max_i \ell_i$ and $C_{\max}(S)$ denote the *maximum machine load* and the *length of schedule* S, respectively; OPT stands for the optimum. Clearly,

$$\ell_{\max} \leqslant \mathrm{OPT} \leqslant C_{\max}(S). \tag{1}$$

The same relation also holds for any scheduling problem on dedicated machines with the standard constraint that each machine cannot process more than one operation at a time.

We say that a polynomial-time algorithm A is a *ρ-approximation*-algorithm (or provides a *ρ-approximation*) for a minimization problem, if for any instance of the problem it outputs a feasible solution of cost at most $\rho \cdot \mathrm{OPT}$. We say that a minimization problem admits a PTAS if it can be solved by a polynomial-time $(1+\varepsilon)$-approximation algorithm for any constant $\varepsilon > 0$.

2.1. Normal schedules and instances

Given an instance I of a scheduling problem, its feasible schedule S is called *normal*, if

$$C_{\max}(S) = \ell_{\max}. \tag{2}$$

Due to (1), any normal schedule is optimal. The converse is not true, in general. An instance I is called *normal*, if its optimal schedule is normal.

At first glance, such a coincidence like in (2) seems to be improbable. However, it will be shown in Section 3 that most instances of the OB-problem are normal, which approves the term.

2.2. No-idle schedules

We say that a schedule is *left no-idle* (resp., *right no-idle*), or *LNI* (resp., *RNI*), for short, if each machine $M_i \in \boldsymbol{M}$ executes all its operations within a continuous time interval $I_i = [t_i', t_i'']$ (i.e., without idle time between t_i' and t_i''), and all intervals I_i start (resp., finish) at the same point in time $t_i' = t$ (resp., $t_i'' = t$).

2.3. Simple jobs and multi-jobs

A job J_j is called a *simple job*, if it consists of only one operation. Otherwise, it is called a *multi-job*.

2.4. Gluing procedures

For a subset $v \subseteq \{1, \ldots, m\}$, the *v-class of jobs* is defined as the set of all jobs in $\boldsymbol{J}$ having the machine set equal to v: $\boldsymbol{J}(v) = \{J_j \mid v_j = v\}$. As an efficient tool of decreasing the running time of algorithms, we will use three gluing procedures described below. The goal of each such procedure is to reduce the number of jobs: at each step of the procedure we replace several jobs in a given instance by a new one without violation of some selected properties of the instance. All elements of the resulting family of jobs (obtained as the output of a gluing procedure) will be referred to as *aggregated jobs*.

The *First gluing procedure* combines all jobs in each v-class into a single job, with v as the machine set and $p(v) = \sum_{J_j \in \boldsymbol{J}(v)} p_j$ as the processing time of the new job.

Lemma 2.1. *The First gluing procedure can be implemented in* $O(m^2 n)$ *time and completes with at most* $n' = \min\{n, 2^m\}$ *aggregated jobs having pairwise different machine sets.*

Proof. We implicitly use a *balanced binary tree T* with at most n' nodes (corresponding to classes $\boldsymbol{J}(v)$, $v \subseteq \{1, \ldots, m\}$) and with the length of each path from the root to a leaf no greater than $O(m)$. As a search key we use the characteristic vector $\mathbf{v}$ of a set $v \subseteq \{1, \ldots, m\}$, two vectors being compared lexicographically. The procedure consists in scanning the list of jobs $\boldsymbol{J}$. For each job $J_j \in \boldsymbol{J}$ we either insert into T a new node (with a new vector $\mathbf{v} = \mathbf{v}_j$), or find in T the already existing node with $\mathbf{v} = \mathbf{v}_j$ (in the latter case we add p_j to $p(v)$). Since both cases require at most $\log(2^m) = m$ comparisons of vectors $\mathbf{v} \in \mathbb{R}^m$, the whole procedure requires $O(m^2 n)$ time. □

The *Second* and the *Third gluing procedures* combine jobs regardless of the difference in their machine sets. The families of aggregated jobs obtained by the procedures, normally, represent no instances of the OB-problem, because the processing times of different operations of such job may be different. However, they can be treated as instances of the Open Shop problem, enabling one to apply some of known efficient methods developed for the Open Shop problem.

The *Second gluing procedure* runs as follows: Given a *lower threshold* $\boldsymbol{p}$ on processing times of operations, we scan the list of jobs, gluing two jobs together each time that the maximum processing time of the operations of both jobs is no greater than $\boldsymbol{p}/2$. The procedure terminates, as soon as each job in the resulting instance (but maybe one job) gets at least one operation with length greater than $\boldsymbol{p}/2$.

The *Third gluing procedure* works similar to the second one, but operates with job lengths $\{d_j\}$ and a threshold $\boldsymbol{d}$ on the minimum job length instead.

The following lemma can be easily proved.

Lemma 2.2. *The Second* (*the Third*) *gluing procedure can be implemented in* $O(nm)$ *time and completes with at most* $O(m\ell_{\max}/\boldsymbol{p})$ (*at most* $O(m\ell_{\max}/\boldsymbol{d})$) *aggregated jobs.*

In [13,14], a few sufficient conditions were derived which, given an instance of the Open Shop problem satisfying one of these conditions, guarantee that a normal schedule for that instance exists and can be found in polynomial time. In Section 3 we will see that these results can be successfully applied to the Open Block problem, enabling one to construct optimal schedules for most instances in polynomial time. The above mentioned results run as follows:

Proposition 2.3. *Given an instance of the Open Shop problem with n jobs and m machines, its normal* (*optimal*) *schedule can be found*
(a) *in* $O(n^2 m^2)$ *time, as soon as* $\ell_{\max} \geqslant m^2 p_{\max}$ *holds* [14], *and*
(b) *in* $O(nm^2 \log m)$ *time, as soon as* $\ell_{\max} \geqslant (\frac{16}{3} m \log_2 m + \frac{61}{9} m - 7.4) p_{\max}$ *holds* [13].

3. Polynomial time solvable cases

Theorem 3.1. *A normal* (*optimal*) *schedule for* $OB4||C_{\max}$ *with n jobs can be found in* $O(n)$ *time.*

Proof. First of all, applying the *First gluing procedure* (defined in Section 2), we obtain an instance containing 4 *simple jobs*, 6 *double-jobs*, 4 *triple-jobs*, and a single multi-job with 4 operations.

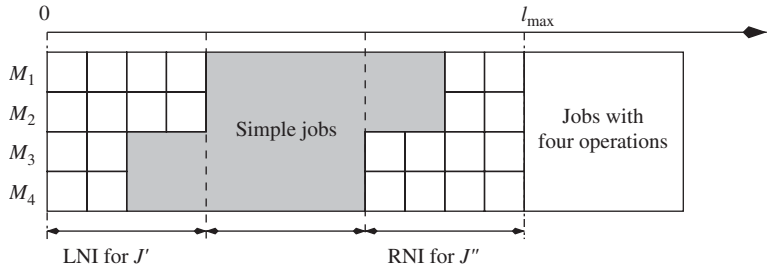

Fig. 3. A normal schedule on four machines.

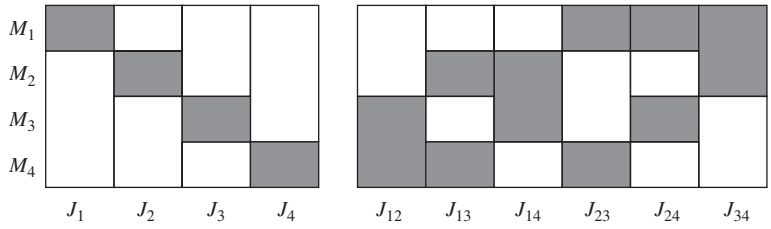

Fig. 4. Multi-jobs.

Furthermore, w.l.o.g. we may remove the latter multi-job and reduce the value of $\ell_{\max}$, respectively. After constructing a normal schedule for the remaining jobs we can simply add this removed job as a single block at the end of the schedule.

Next, we aim at constructing a normal schedule. We divide the set of all multi-jobs into two sets J' and J''. We construct a feasible LNI-schedule starting at time 0 for the jobs in J' and a feasible RNI-schedule finishing at time $\ell_{\max}$ for the jobs in J''. We remind the reader that LNI- and RNI-schedules have no interior idle time intervals. It follows that on each machine we can put the remaining simple job into the idle gap between the LNI- and RNI-schedules. For an illustration, see Fig. 3. Clearly, if we succeeded in all steps, the constructed schedule is normal.

For simplicity, we will use J_i ($i = 1, \ldots, 4$) to denote the triple job having no operation on machine M_i, and J_{ij} ($1 \leqslant i < j \leqslant 4$) to denote the double job with exactly two operations on machines M_i and M_j. For an illustration, see Fig. 4. To keep similar notation, we will also use p_i and p_{ij} to denote the processing times of jobs J_i and J_{ij}, respectively.

Consider three pairs of double-jobs: $\{J_{12}, J_{34}\}$, $\{J_{13}, J_{24}\}$, and $\{J_{14}, J_{23}\}$. One can see that two jobs in each pair require mutually complementary sets of machines. We define three values:

$$\alpha_{12} = \min\{p_{12}, p_{34}\}, \quad \alpha_{13} = \min\{p_{13}, p_{24}\}, \quad \alpha_{14} = \min\{p_{14}, p_{23}\},$$

and compute

$$\alpha^* = \max\{\alpha_{12}, \alpha_{13}, \alpha_{14}\}.$$

W.l.o.g., we may assume that $\alpha^* = \alpha_{12}$. Then we have

$$p_{12} \geqslant \min\{p_{13}, p_{24}\}, \tag{3}$$

$$p_{34} \geqslant \min\{p_{14}, p_{23}\}. \tag{4}$$

Now we divide all multi-jobs into two sets (each containing 5 jobs) as follows:

$$J' = \{J_{12}, J_{13}, J_{24}, J_3, J_4\}, \quad J'' = \{J_{34}, J_{14}, J_{23}, J_1, J_2\}.$$

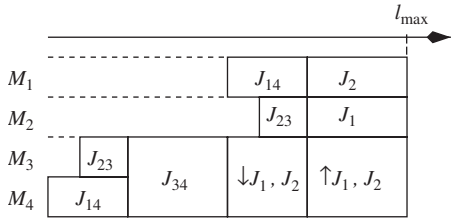

Fig. 5. The RNI-schedule for J'' with $p_{34} \geqslant \max\{p_{14}, p_{23}\}$.

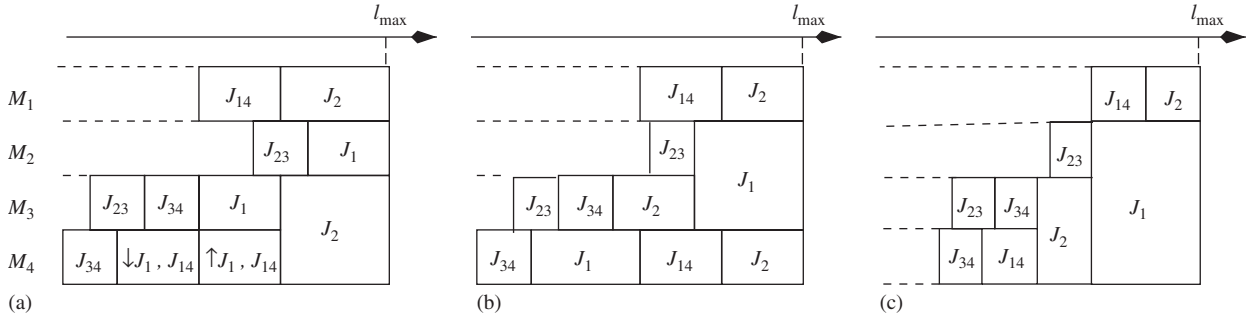

Fig. 6. The RNI-schedule for J'' with $p_{23} \leqslant p_{34} < p_{14}$. (a) $p_1 < p_2$; (b) $p_2 < p_1 \leqslant p_{14} + p_2$; (c) $p_2 + p_{14} < p_1$.

One can see that there is a symmetry between the jobs in J' and J'', including relations (3) and (4). In particular, a feasible LNI-schedule starting at time 0 for the jobs in J' can be constructed in a way similar to that we use for constructing a feasible RNI-schedule for J'' finishing at time $\ell_{\max}$. In the following we only concentrate on the latter.

First, consider the case when $p_{34} \geqslant \max\{p_{14}, p_{23}\}$. Then, the RNI-schedule depicted in Fig. 5 is clearly feasible. For two jobs A and B, we use $\downarrow A, B$ and $\uparrow A, B$ in our figures to denote the smaller and the larger job, respectively.

Now, consider the case when $\min\{p_{14}, p_{23}\} \leqslant p_{34} < \max\{p_{14}, p_{23}\}$. Clearly, the case $p_{23} > p_{14}$ reduces to the case $p_{23} < p_{14}$ by renumbering the machines: $M_1 \leftrightarrow M_2$, $M_3 \leftrightarrow M_4$. For certainty, assume that $p_{23} < p_{14}$. Then, we have

$$p_{23} \leqslant p_{34} < p_{14}. \tag{5}$$

The RNI-schedule depicted in Fig. 6 depends on which of three possible intervals $[0, p_2]$, $(p_2, p_2+p_{14}]$, (p_2+p_{14}, ∞) the value of p_1 falls. The feasibility of this schedule follows from (5).

Clearly, not mentioning the running time of the gluing procedure, all other computing operations require just a constant time. Thus, by Lemma 2.1, the overall running time of the algorithm is $O(n)$. This completes the proof of Theorem 3.1. □

Given an instance of the OB-problem, we may consider it as an instance of the Open Shop problem and propose for it a feasible *open shop schedule* in which no two operations of the same job overlap. Such a schedule is also feasible for the original OB-problem, and when it is normal, it is evidently optimal for the OB-problem. Thus, we can apply Proposition 2.3 and deduce similar conclusions for the OB-problem. Moreover, it is possible to derive better bounds on the running time of our algorithms by making use of the *Second Gluing Procedure*.

Indeed, we can implement the *Second gluing procedure* with the parameter $\boldsymbol{p} = \ell_{\max}/m^2$ or $\boldsymbol{p} = \ell_{\max}/(\frac{16}{3}m \log_2 m + \frac{61}{9}m - 7.4)$ (in cases (a) and (b) of Proposition 2.3, respectively) and obtain an instance with at most $O(m^3)$ (at most $O(m^2 \log m)$) aggregated jobs, and still satisfying the corresponding condition ((a) or (b)). Next we apply the algorithm from [14] (or the algorithm from [13]) to the family of aggregated jobs and construct a normal schedule, optimal for the OB-problem. Thus, we obtain

Theorem 3.2. *For any instance of the $OB||C_{\max}$ problem with n jobs and m machines satisfying at least one of the relations*

$$\ell_{\max} \geqslant \left(\tfrac{16}{3}\, m\, \log_2 m + \tfrac{61}{9}\, m - 7.4\right) p_{\max}, \tag{6}$$

$$\ell_{\max} \geqslant m^2 p_{\max}, \tag{7}$$

its normal schedule exists and can be found in $O(\min\{nm + m^4 \log^2 m, nm^2 \log m\})$ *time—in the case of* (6), *and in* $O(\min\{nm + m^8, n^2m^2\})$ *time—in the case of* (7).

Note 1: One can observe from the above theorem that, if all job processing times and the number of machines are bounded from above, while the number of jobs infinitely increases and the mean job processing time is positive, then $\ell_{\max}$ tends to infinity and the conditions of Theorem 3.2 are satisfied with high probability. This implies that most instances of the OB-problem are normal, and optimal schedules for such instances can be found by one of the polynomial time algorithms of Theorem 3.2.

Another interesting polynomially solvable case of the OB-problem is formulated in Theorem 3.3. It also finds a useful application in Section 5, to construct efficient approximation algorithms.

Theorem 3.3. *Let $\alpha > 0$ be a rational number, a and k_i $(i = 1, \ldots, n)$ be integers. Then, given an instance of the OB-problem with m machines and n jobs with processing times $p_j = \alpha a^{k_i}$, its optimal (normal) schedule can be found in* $O(mn + n \log n)$ *time.*

Proof. The following LNIS-algorithm is proposed for constructing the desired normal schedule.

LNI SCHEDULING (LNIS):

Step 1: Number the jobs in nonincreasing order of their processing times: $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n$.

Step 2: Considering the jobs in this order, schedule each operation of the current job as early as possible, without any idle time on the corresponding machine.

Clearly, the length of the LNI-schedule is equal to $\ell_{\max}$. Note that for arbitrary processing times the LNIS-algorithm can obtain an infeasible schedule. However, we show that for the considered processing times the above LNI-schedule is feasible.

By the induction on the job number k, it can be easily proved that after iteration k of Step 2, when jobs $J_1, \ldots, J_k$ have already been scheduled, each machine completes its work at a time which is a multiple of p_k.

Indeed, we have this property at the beginning of Step 2, when no jobs are scheduled yet. Suppose, this is true after the first $k-1$ jobs are scheduled. Since p_{k-1} is divisible by p_k, the completion time of each machine is also a multiple of p_k; this property, clearly, remains valid after scheduling each operation of job J_k on the corresponding machine.

Now, once each operation of job J_k starts at a time which is a multiple of p_k, any two operations of J_k either start at the same time, or do not overlap. Thus, the schedule constructed is feasible. To estimate the running time of the LNIS-algorithm, it suffices to observe that sorting the jobs requires $O(n \log n)$ time, whereas the second step can be implemented in $O(nm)$ time. □

A symmetric RNI Scheduling algorithm (RNIS) could be suggested that constructs at Step 2 an RNI-schedule finishing at point t (instead of the LNI-schedule being produced by the LNIS-algorithm).

4. Hardness and inapproximability results

In this section we prove that the OB-problem is NP-hard if the number of machines is at least 6 (Theorem 4.2). The NP-hardness of the OB-problem in the strong sense for a variable number of machines follows from Theorem 4.4 which states that checking the existence of a feasible schedule of length no more than 10 is strongly NP-complete. Another corollary of Theorem 4.4 is the inapproximability result: no ρ-approximation polynomial time algorithm may exist for the $OB||C_{\max}$ problem and any $\rho < 11/10$, unless $\mathbb{P} = \mathbb{NP}$. (And therefore, no PTAS may exist as well.) Finally, we show that adding arbitrary release dates makes the problem NP-hard in the strong sense already for two machines (Theorem 4.3).

We start with formulating sample NP-complete problems that are to be used in our proofs.

PARTITION.

Instance: k integer positive numbers $e_1, \ldots, e_k \in \mathbb{Z}^+$ and $E \in \mathbb{Z}^+$ such that $\sum_{i=1}^{k} e_i = 2E$.

Question: Is there a partition of the set of indices $\{1, \ldots, k\}$ into two disjoint subsets N_1, N_2 such that for each subset N_j we have

$$\sum_{i \in N_j} e_i = E? \tag{8}$$

In fact, the sample problem will be used in a slightly modified form, formulated below as a PARTITIONM problem (to which PARTITION can be easily reduced by adding two extra numbers: $e_{k+1} = 5E$ and $e_{k+2} = 2E$).

PARTITIONM.

Instance: k integer positive numbers $e_1, \ldots, e_k \in \mathbb{Z}^+$ and $E \in \mathbb{Z}^+$ such that $\sum_{i=1}^{k} e_i = 9E$.

Question: Is there a subset of indices $N \subset \{1, \ldots, k\}$ such that

$$\sum_{i \in N} e_i = 6E? \tag{9}$$

3-PARTITION.

Instance: $3k$ integer positive numbers $e_1, \ldots, e_{3k} \in \mathbb{Z}^+$ and $E \in \mathbb{Z}^+$ such that $E/4 < e_i < E/2$, $\forall i$ and $\sum_{i=1}^{3k} e_i = kE$.

Question: Is there a partition of the set of indices $\{1, \ldots, 3k\}$ into k disjoint subsets $N_1, \ldots, N_k$ such that for each subset N_j we have (8)?

MONOTONE-NOT-ALL-EQUAL-3SAT.

Instance: Set U of variables, collection C of clauses over U such that each clause has size 3 and contains only unnegated variables.

Question: Is there a truth assignment for U such that each clause in C has at least one true variable and at least one false variable?

A new essential condition in the MONOTONE-NOT-ALL-EQUAL-DIFFERENT-4SAT problem (or MODIFF-4SAT, for short) presented below is that all variables in each clause must be *different*. (This relates to the requirement that each job in the OB-problem may have on each machine at most one operation.) It is also essential (for the reduction of this sample problem to the recipient problem) that the number of literals in each clause is *exactly* 4.

MODIFF-4SAT.

Instance: Set U of variables, collection C of clauses over U such that each clause has size exactly 4 and contains only unnegated different variables.

Question: Is there a truth assignment for U such that each clause in C has at least one true variable and at least one false variable?

Lemma 4.1. *The* MODIFF-4SAT *problem is* NP*-complete.*

Proof. To prove the NP-completeness of the MODIFF-4SAT problem, we present a polynomial-time reduction from the MONOTONE-NOT-ALL-EQUAL-3SAT. problem. Suppose, we are given a set of variables $U = \{x_j\}$ and a collection $C_3 = \{c_1, \ldots, c_n\}$ of clauses, each being a disjunction of 2 or 3 unnegated different variables from U. (The case when there is a clause in C_3 containing only one variable is trivial: the answer is "no".) We define a collection C_4 of clauses for the MODIFF-4SAT problem as follows: add 5 *basis* variables y_i $(i = 1, \ldots, 5)$ to the set U and define 5 *basis* clauses $\hat{c}_i$, $i = 1, \ldots, 5$ (related to no variable or a clause of C_3), where $\hat{c}_i = y_{i_1} \vee y_{i_2} \vee y_{i_3} \vee y_{i_4}$ for $\{i_1, i_2, i_3, i_4\} = \{1, \ldots, 5\} \setminus \{i\}$. Next, if a clause $c_i \in C_3$ contains 2 variables: $c_i = x_j \vee x_k$, we define 10 c_i*-based* clauses $c_i^{l,m} = x_j \vee x_k \vee y_l \vee y_m$—for every pair $\{l, m\} \subset \{1, \ldots, 5\}$. If a clause $c_i \in C_3$ contains 3 variables: $c_i = x_j \vee x_k \vee x_l$, we define 4 c_i*-based* clauses $c_i^m = x_j \vee x_k \vee x_l \vee y_m$, $m = 1, 2, 3, 4$. We now prove that the desired assignment for variables in C_3 exists, *iff* it exists for variables in C_4.

$\Rightarrow$: Suppose, we have an assignment of variables $\{x_j\}$, which is NOT-ALL-EQUAL for each clause $c_i \in C_3$. To get an assignment with the same property for the variables of C_4, it suffices to assign the *true* value to any two variables y_l, y_m, retaining the *false* value for the remaining three basis variables $\{y_v\}$.

⇐: Suppose, we have an assignment of variables $\{x_j, y_v\}$, which is NOT-ALL-EQUAL for each clause $c_i \in C_4$. Let us prove that the same values of variables $\{x_j\}$ provide the NOT-ALL-EQUAL property for each clause $c_i \in C_3$. Suppose the contrary, and let $c_i = x_j \vee x_k$ be a clause of C_3 in which both its variables have the same value α. Since among any 4 basis variables $y_{i_1}, y_{i_2}, y_{i_3}, y_{i_4}$ there are two with different values (because of basis clauses $\hat{c}_i$), we can find two variables y_l, y_m having values α, which results in a clause $(x_j \vee x_k \vee y_l \vee y_m) \in C_4$ having ALL-EQUAL values of its literals—a contradiction. Similarly, suppose that all three variables of a clause $c_i = x_j \vee x_k \vee x_l$ have the same value α. Then, due to the property that at least one of the variables $y_1, \ldots, y_4$ (say, y_m) has value α, we would get a clause $c_i^m = c_i \vee y_m \in C_4$ that violates the NOT-ALL-EQUAL property—a contradiction again. Lemma 4.1 follows. □

Theorem 4.2. *The $OB6||C_{\max}$ problem is* NP*-hard.*

Proof. We prove that it is NP-complete to verify whether a given instance of the 6-machine OB-problem has a normal schedule. To this end, we construct a reduction from the PARTITIONM problem to our OB-problem. Given an instance $\{e_1, \ldots, e_n; E\}$ of the PARTITIONM problem satisfying $\sum_i e_i = 9E$, we define an instance I^* of the OB-problem as follows:

We introduce 6 machines $M_1, \ldots, M_6$ and 8 *basic* jobs: *big jobs* b_1, b_2, b_3, *medium jobs* m_1, m_2, and *small jobs* s_1, s_2, s_3 with processing times:

$$p(b_1) = 70, \quad p(b_2) = 73, \quad p(b_3) = 67, \quad p(m_1) = 17, \quad p(m_2) = 20,$$
$$p(s_1) = 13, \quad p(s_2) = 10, \quad p(s_3) = 7.$$

The machines are prescribed to execute the following jobs:

$$M_1\colon \{b_1, m_1, s_1\}, \quad M_3\colon \{b_2, m_1, s_2\}, \quad M_5\colon \{b_3, m_2, s_1\},$$
$$M_2\colon \{b_1, m_2, s_2\}, \quad M_4\colon \{b_2, m_2, s_3\}, \quad M_6\colon \{b_3, m_1, s_3\}.$$

Note that the load ℓ_i of each machine M_i $(i = 1, \ldots, 5)$ is equal to 100, while the current load of M_6 is 91. In addition to operations of basic jobs, machine M_6 is prescribed to execute n *tiny* simple jobs $\{t_1, \ldots, t_n\}$. Each job t_i has length e_i/E. Thus, the total load of machine M_6 is also 100, and so we have $\ell_{\max} = 100$.

To prove the theorem, it suffices to show that the question in the PARTITIONM problem has the positive answer, if and only if the instance I^* has a normal schedule.

⇐: Assume that there exists a normal schedule S. Since $p(b_j) > \ell_{\max}/2$, the operations of each big job b_j must be executed as a block. Since no two different jobs have equal processing times, b_1 and b_2 can only be scheduled either at the beginning or at the end of the schedule. So the blocks of both jobs can only be scheduled either at the beginning, or at the end of the makespan. Without loss of generality we may assume that b_1 is processed at the beginning of the schedule.

We first prove that b_2 must be the last job on M_3 and M_4. Assume the contrary. Then, b_2 must be the first job on M_3 and M_4. Since $p(m_1) > \{p(s_1), p(s_2)\}$ and $p(b_1) \neq p(b_2)$, job m_1 has to be the last on both M_1 and M_3. Thus, s_2 is in the middle on M_3, and so it must be the last on M_2, while m_2 occurs there in the middle. But in this case we cannot place job m_2 on M_4 properly. This proves our claim.

We now consider three possible locations of job b_3 on machine M_6.

Case 1: b_3 is the third job on M_6 (see Fig. 7). Then it is also the third job on M_5. Since $p(m_2) > \{p(s_1), p(s_3)\}$ and $p(s_1) \neq p(s_3)$, job m_2 cannot be second neither on M_4, nor on M_5. Therefore, it must be the first job on both machines, whereas s_3 must be the second on M_4.

If m_1 is the first job on M_3 (Fig. 7A), then due to the inequality $2p(m_1) + p(b_3) > \ell_{\max}$, it also must be the first on M_6. Therefore, s_3 must be scheduled on M_6 exactly in the interval [20, 27], so as to form a block with its operation on M_4. This implies the existence of a subset N satisfying (9).

If m_1 is the second job on M_3 (Fig. 7B), it must constitute a block (scheduled in the interval [10, 27]) with its operation on M_6. The tiny jobs scheduled on M_6 between jobs m_1 and b_3 evidently determine the desired subset N satisfying (9).

Case 2: b_3 is the second on M_6 (see Fig. 8). Then it is also the second on M_5. If m_2 is the first job on M_5, then, due to $p(m_1) > p(s_1)$, job m_1 must be the first job on M_6 (see Fig. 8A). Symmetrically, if m_2 is the third job on M_5, then m_1 is the third job on M_6 as well (see Fig. 8B). In both cases the existence of a subset N satisfying (9) is evident.

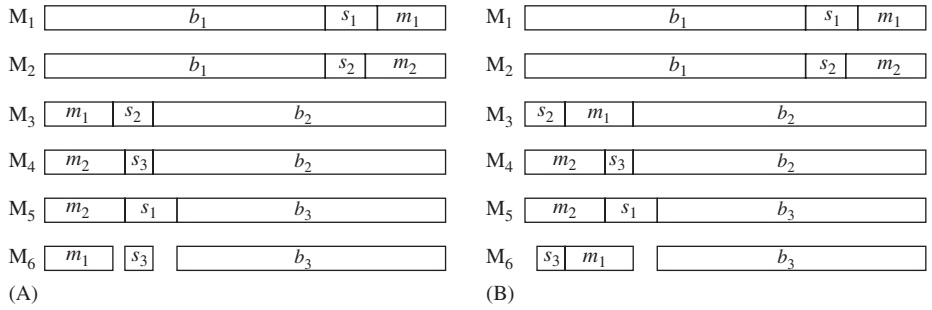

Fig. 7. Normal schedules for the basic jobs in Case 1.

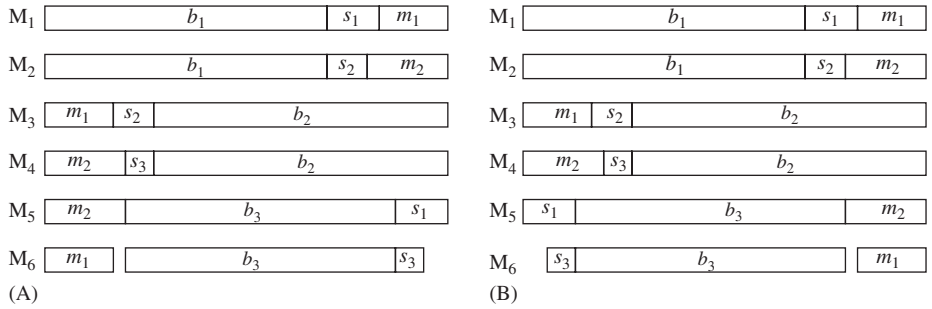

Fig. 8. Normal schedules for the basic jobs in Case 2.

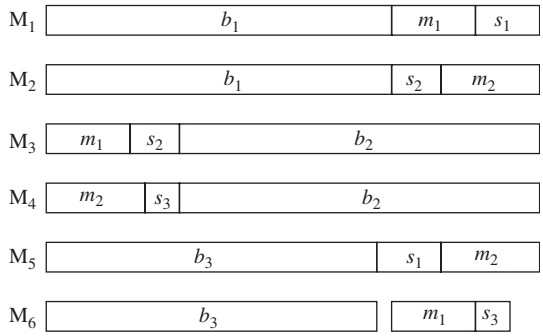

Fig. 9. A normal schedule for the basic jobs in Case 3.

Case 3: b_3 is the first on M_6 (see Fig. 9). Then it is also the first on M_5.

Clearly, m_2 must be the third job on M_2 and M_5, and therefore, s_1 is the second job on M_5. This implies that s_1 can only be the third job on M_1, whereas m_1 is the second job. This, in turn, implies that m_1 is the second job on M_6 constituting a block with its operation on M_1. This means that it must be scheduled in the interval [70, 87], requiring job s_3 to be scheduled somewhere in the interval [87, 100]. Thus, the total length of tiny jobs processed on M_6 after m_1 is equal to 6, which implies the answer "yes" to the question of the PARTITIONM problem.

$\Rightarrow$: Suppose that the answer to the question in the PARTITIONM problem is positive. Then a normal schedule for the instance I^* can easily be produced. Indeed, any of the 5 schedules depicted in Figs. 7–9 can be used for this purpose. □

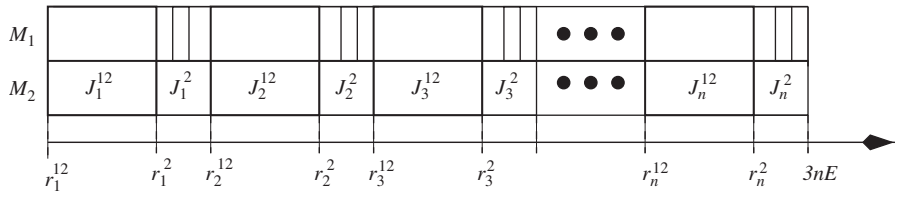

Fig. 10. A normal schedule.

If release dates are added, the two-machine Open Block problem becomes strongly NP-hard.

Theorem 4.3. *The* $OB2|r_j|C_{\max}$ *problem is* NP*-hard in the strong sense.*

Proof. We describe a polynomial reduction from the 3-PARTITION problem to our $OB2|r_j|C_{\max}$ problem. Suppose, we are given a set of indices $N = \{1, \ldots, 3n\}$, a bound $E \in \mathbb{Z}^+$, and $3n$ integer numbers e_i, $i \in N$, such that $\sum_{i\in N} e_i = nE$ and $E/4 < e_i < E/2$, for $i \in N$. We specify an instance of the $OB2|r_j|C_{\max}$ problem and show that it admits a normal schedule, if and only if there exists a partition of N into n disjoint subsets $N_1, \ldots, N_n$ such that $\sum_{i\in N_j} e_i = E$, for $j = 1, \ldots, n$.

We define $5n$ jobs as follows: Each job J_i^{12} $(i = 1, \ldots, n)$ has operations on both machines, its processing time is equal to $2E$, and its release date is $r_i^{12} = 3(i-1)E$. Each job J_i^2 $(i = 1, \ldots, n)$ has an operation on machine M_2, its processing time is equal to E, and its release date is $r_i^2 = (3i-1)E$. Each job J_i^1 $(i = 1, \ldots, 3n)$ has a single operation on machine M_1, its processing time is equal to e_i, and its release date is equal to 0. We check the existence of a normal schedule for this problem instance, i.e., a feasible schedule with makespan equal to the maximum machine load, which is equal to $3nE$.

It can be easily seen that the existence of a desired partition in the 3-PARTITION problem implies the existence of a normal schedule for the defined above instance, as shown in Fig. 10. Now let us prove the converse statement.

Suppose, we are given a normal schedule S. First of all, it can be easily observed that there is a single feasible way to schedule the operations of machine M_2—exactly as shown in Fig. 10. Next, we cannot shift the operation of job J_n^{12} on machine M_1 to the right, so, both its operations have to be scheduled as a single block (as in Fig. 10). The same is true for job J_{n-1}^{12}, and so on (by induction). Thus, all operations of jobs J_i^2 and J_i^{12} $(i = 1, \ldots, n)$ have the unique feasible schedule shown in Fig. 10.

As one can see from this schedule, the remaining idle time on machine M_1 is divided into n equal parts. So, to obtain a schedule of length $3nE$, we have to fill those "holes" completely with operations of jobs J_i^1 $(i = 1, \ldots, 3n)$. This implies the existence of a desired 3-partition. □

Note 2: By a similar reasoning, the two machine OB-problem of checking the existence of a schedule that meets given deadlines is also strongly NP-complete.

Now we turn to the following decision problem. Given an instance of the OB-problem, we would like to answer the following question: does there exist a schedule of length at most 10? To prove that the decision problem is NP-complete, we construct a reduction from the MODIFF-4SAT problem. The technique used in our reduction differs from the one used in Williamson et al. [21] in the following: instead of associating jobs and machines with literals (which obliges us to synchronize the jobs corresponding to literals of the same variable), we associate them with *variables*.

Theorem 4.4. *The problem of deciding if there is a feasible OB-schedule of length at most* 10 *is* NP*-complete.*

Proof. Suppose, we are given an instance of the MODIFF-4SAT problem, i.e., a collection $C = \{c_1, \ldots, c_n\}$ of clauses, each being a disjunction of 4 non-negated different variables. We define an instance I of the OB-problem with the maximum machine load $\ell_{\max} = 10$, and show that a feasible schedule of length 10 for that instance exists, *iff* there exists an assignment for the variables of C such that their values in each clause $c_i \in C$ are "not all equal".

The processing time of each job in I will be either 2, 3, or 5 (according to which the jobs will be referred to as 2-, 3-, or 5-jobs). By their destination, the jobs divide into: *basis jobs* ($\tilde{J}_j^i$) and *assignment jobs* (J_j^i), where the upper index i is the processing time of the job.

Similarly, we define three types of machines by their destination: *basis machines* (denoted with *tildes*), *clause machines* (with *hats*), and *assignment machines*. By processing times of jobs that are to be scheduled on a machine, we recognize 4 types of machines: *A-machines* (with load $2+2+2+2+2$), *B-machines* ($3+3+3$), *C-machines* ($5+5$), and *D-machines* ($2+3+5$). It can be seen that the maximum load over all types of machines is 10.

There are 11 *basis jobs*: $\tilde{J}_1^2, \ldots, \tilde{J}_6^2$; $\tilde{J}_1^3, \tilde{J}_2^3, \tilde{J}_3^3$; $\tilde{J}_1^5, \tilde{J}_2^5$, and 13 *basis machines* (the jobs listed in braces are to be processed on the corresponding machine):

$$\tilde{A}: \{\tilde{J}_1^2, \ldots, \tilde{J}_5^2\}; \quad \tilde{B}: \{\tilde{J}_1^3, \tilde{J}_2^3, \tilde{J}_3^3\}; \quad \tilde{C}: \{\tilde{J}_1^5, \tilde{J}_2^5\};$$

$$\tilde{D}_i: \begin{cases} \{\tilde{J}_2^3, \tilde{J}_1^5, \tilde{J}_i^2\} & \text{for } i = 1, \ldots, 5; \\ \{\tilde{J}_3^3, \tilde{J}_1^5, \tilde{J}_{i-5}^2\} & \text{for } i = 6, \ldots, 10. \end{cases}$$

With each logic *variable* x_j ($j = 1, \ldots, N$) in collection C we associate one *assignment job* J_j^2 and one *assignment machine* $D_j: \{\tilde{J}_1^3, \tilde{J}_1^5, J_j^2\}$.

With each *clause* $c_v = x_{j_1(v)} \vee x_{j_2(v)} \vee x_{j_3(v)} \vee x_{j_4(v)}$ ($v = 1, \ldots, n$) we associate a *clause machine* $\hat{A}_v: \{\tilde{J}_6^2, J_{j_1(v)}^2, J_{j_2(v)}^2, J_{j_3(v)}^2, J_{j_4(v)}^2\}$. This completes the definition of instance I.

The constant number of basis machines is characterized by the property that it processes only basis jobs, related to no variables or clauses. They are interesting for us only in view of the following:

Lemma 4.5. *In any normal schedule for the instance I, every assignment machine D_i processes its basis jobs $\tilde{J}_1^3$ and $\tilde{J}_1^5$ at the end of the interval $[0, \ell_{\max}]$; this implies that the remaining job J_j^2 has to be scheduled on that machine in the middle.*

Proof. Due to machine $\tilde{C}$, job $\tilde{J}_1^5$ should be scheduled on each D-machine as an outermost job. This enables us, given a normal schedule S for the instance I, to divide all D-machines into two subsets: *right machines* (where $\tilde{J}_1^5$ is scheduled at the right end of the interval $[0, \ell_{\max}]$) and *left machines* (where it is scheduled first). Let us prove that job $\tilde{J}_1^3$ is also outermost on all D-machines.

W.l.o.g., we may assume that jobs $\tilde{J}_1^2, \ldots, \tilde{J}_5^2$ are processed on machine $\tilde{A}$ in this order. This implies that job $\tilde{J}_3^2$ cannot be processed on machines $\tilde{D}_3$ and $\tilde{D}_8$ in the middle. Therefore, job $\tilde{J}_2^3$ has to be scheduled in the middle on machine $\tilde{D}_3$, and job $\tilde{J}_3^3$ in the middle on machine $\tilde{D}_8$. This means that neither of the two jobs may be the second on machine $\tilde{B}$. So, this vacant place must be occupied by job $\tilde{J}_1^3$, which excludes any appearance of this job in the middle on any assignment machine D_j.

To complete the proof of Lemma 4.5, it remains to make sure that there exists a feasible schedule for basis machines. One of such schedules is presented in Fig. 11. □

Let us prove that a normal schedule for the instance I exists, *iff* there is an assignment for the variables of collection C with the "not all equal" property.

$\Rightarrow$: Suppose, we are given a normal schedule S. Once all assignment jobs J_j^2 have to be scheduled on the corresponding assignment machines in the middle, they cannot occupy the third place on clause machines $\hat{A}_v$. That is why this place on all clause machines is exclusively occupied by job $\hat{J}_6^2$ (which has no operations on assignment machines).

Next, for any clause $c_v = x_{j_1(v)} \vee x_{j_2(v)} \vee x_{j_3(v)} \vee x_{j_4(v)}$, the four assignment machines $D_{j_1(v)}, D_{j_2(v)}, D_{j_3(v)}, D_{j_4(v)}$ corresponding to the variables of clause c_v cannot be *right* machines simultaneously. Indeed, if we assume the contrary, then each of the jobs $J_{j_1(v)}^2, J_{j_2(v)}^2, J_{j_3(v)}^2, J_{j_4(v)}^2$ has to be scheduled on the corresponding assignment machine in the interval $[3, 5]$, and so, none of them could occupy the second place on the clause machine $\tilde{A}_v$. Similarly, we can prove that the above four assignment machines cannot be *left* machines simultaneously. Let us assign to each variable x_j of

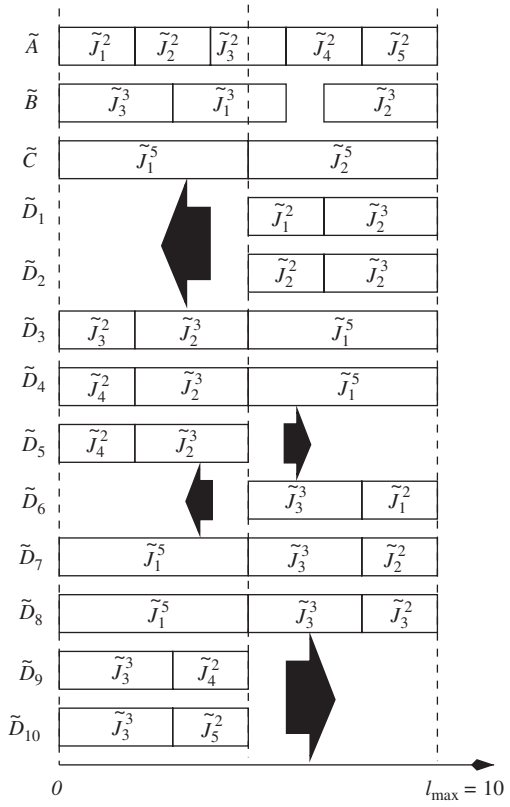

Fig. 11. A normal schedule for basis machines.

the collection C the value

$$x_j := \text{“machine } D_j \text{ is right”}. \tag{10}$$

Then for each clause c_v, the value at least one of its variable $x_{j_i(v)}$ is *true*, while the value of some other variable $x_{j_k(v)}$ is *false*, which provides the desired “not all equal” property.

$\Leftarrow$: Suppose, we are given a truth assignment for the variables of the collection C with the “not all equal” property. We first schedule the operations on basis machines as shown in Fig. 11 and the operations on assignment machines according to (10), placing all 2-jobs in the middle. Then a feasible normal schedule for each clause machine $\hat{A}_v$ can be easily defined: assign the basis job $\tilde{J}_6^2$ to the third position, one of the “false” jobs of that machine (a job $J^2_{j_k(v)}$ whose assignment machine $D_{j_k(v)}$ is *left* we know that such jobs exist) to the second position, and one of the “true” jobs of that machine (a job $J^2_{j_k(v)}$ whose assignment machine $D_{j_k(v)}$ is *right* we know that such jobs also exist) to the fourth position, while the remaining two assignment jobs occupy the vacant first and fifth positions arbitrarily. The resulting schedule of machine $\hat{A}_v$ is clearly feasible, and so, the whole schedule is normal and feasible. Theorem 4.4 follows. □

As a straightforward corollary of the above theorem, we obtain

Theorem 4.6. *For the OB-problem with a variable number of machines and any $\rho < 11/10$, there is no polynomial time ρ-approximation algorithm, unless $\mathbb{P} = \mathbb{NP}$.*

Now we should motivate, why in Theorem 4.4 we consider the time interval of length 10. (If we proved a similar result for a smaller value of $\ell_{\max}$, we could obtain a better lower bound on nonapproximability.) The answer is contained in

Theorem 4.7. *There is a polynomial time algorithm which, given an instance of the OB-problem with* $\ell_{\max} \leqslant 9$, *constructs its normal* (*and therefore, optimal*) *schedule.*

Proof. We first show this for $\ell_{\max} = 9$. In fact, it suffices for each $i \in \{2, \ldots, 9\}$ to prescribe a set of *allowed locations* for i-jobs within the interval $[0, 9]$, so that:

(a) no two locations of i-jobs overlap in time;

(b) for any feasible combination of jobs destined to be processed on one machine (i.e., such that their total length is at most 9), it should be shown that there exists a schedule for that machine in which each i-job is assigned to one of its allowed locations.

Let us define a set L_i of allowed locations for each $i \in \{2, \ldots, 9\}$ as follows: $L_9 = \{[0, 9]\}$, $L_8 = \{[1, 9]\}$, $L_7 = \{[2, 9]\}$, $L_6 = \{[3, 9]\}$, $L_5 = \{[4, 9]\}$, $L_4 = \{[0, 4], [4, 8]\}$, $L_3 = \{[0, 3], [3, 6], [6, 9]\}$, $L_2 = \{[0, 2], [2, 4], [4, 6], [6, 8]\}$ (clearly, 1-jobs may be scheduled arbitrarily).

It can be easily checked that, given a family of jobs $\{J_1, \ldots, J_k\}$ with total length $\leqslant 9$ containing a "long job" (of length $\geqslant 5$), it can be scheduled without overlapping within the time interval $[0, 9]$, so that each job gets one of its allowed locations. All other combinations of jobs with total length 9 can be sequenced as follows: $4 + 4 + 1$, $4 + 2 + 3$, $4 + 2 + 2 + 1$, $3 + 3 + 3$, $2 + 1 + 3 + 3$, $2 + 2 + 2 + 3$, $2 + 2 + 2 + 2 + 1$.

As one can observe, for each i we assigned allowed locations to i-jobs successively, one right after another, starting from either the left, or the right end of the interval $[0, 9]$. Depending on this left-to-right or right-to-left orientation, the set of allowed locations for $i \in \{2, \ldots, 9\}$ will be denoted by either $\overrightarrow{i}$, or $\overleftarrow{i}$. In this notation, the sets $\{L_i\}$ defined above for the case of $\ell_{\max} = 9$ can be shortly written as: $\overleftarrow{8}, \overleftarrow{7}, \overleftarrow{6}, \overleftarrow{5}, \overrightarrow{4}, \overleftarrow{3}, \overrightarrow{2}$. It can be checked that the sets $\{L_i\}$ with exactly the same orientation can be defined for i-jobs in all remaining cases of $\ell_{\max} \leqslant 9$ (with the only exception for $\ell_{\max} = 8$, where we change $\overleftarrow{5}$ to $\overrightarrow{5}$). Theorem 4.7 follows. □

5. Approximation algorithms

In this section, we first propose a few constant approximation algorithms. After that, a linear time PTAS will be presented for the case of an arbitrary fixed number of machines.

5.1. Rounding technique

A simple 2-approximation algorithm can be deduced by applying the LNIS-algorithm from the proof of Theorem 3.3.

2-ROUNDED LNI SCHEDULING (2R-LNIS):

Step 1: For each job $J_j \in \boldsymbol{J}$, round its processing time p_j up to the nearest integer power of 2.

Step 2: Apply the LNIS-algorithm (see page 11).

The first step ensures an increasing of the maximum machine load ($\ell_{\max}$) by at most a factor of 2, whereas Step 2 provides (by Theorem 3.3) the computing of an LNI-schedule with length equal to the new $\ell_{\max}$. As a result, we have a feasible schedule for the original instance with length at most a factor of 2 larger than the optimum.

Another algorithm with better approximation can be suggested, if to implement a *division* trick. The idea of the algorithm can be formulated as follows:

5.2. Division technique

ROUND-AND-DIVIDE SCHEDULING (RDS):

Step 1: Round p_j ($J_j \in \boldsymbol{J}$) up to the nearest integer power of $\beta \doteq \sqrt{2}$; let $\bar{p}_j = \beta^{\alpha_j}$ be the new processing time of job J_j. Compute $\ell_{\max}$ for the instance with the new processing times.

Step 2: Divide the set of jobs into $\boldsymbol{J}_{\text{odd}} = \{J_j \in \boldsymbol{J} \mid \alpha_j \text{ is odd}\}$ and $\boldsymbol{J}_{\text{even}} = \{J_j \in \boldsymbol{J} \mid \alpha_j \text{ is even}\}$.

Step 3: Apply the LNIS-algorithm to $\boldsymbol{J}_{\text{odd}}$.

Step 4: Apply the RNIS-algorithm with finishing time $\ell_{\max}$ to $\boldsymbol{J}_{\text{even}}$.

The $\sqrt{2}$-approximation of the above algorithm is ensured by two factors. Firstly, the rounding procedure increases $\ell_{\max}$ at Step 1 by at most a factor of $\sqrt{2}$. And secondly, the RNIS-algorithm succeeds (due to Theorem 3.3) in constructing a feasible RNI-schedule for the set of jobs $\boldsymbol{J}_{\text{even}}$, since the processing times of all jobs $J_j \in \boldsymbol{J}_{\text{even}}$ represent integer

powers of 2. Similarly, all processing times of jobs $J_j \in \boldsymbol{J}_{\text{odd}}$ represent integer powers of 2 multiplied by β, and so, the LNIS-algorithm also succeeds in constructing a feasible LNI-schedule for the jobs $J_j \in \boldsymbol{J}_{\text{odd}}$.

The above algorithm would be correct, if we could compute the value of $\sqrt{2}$ exactly. But if in the above algorithm we use an approximate value of $\sqrt{2}$, the rounded processing times $\bar{p}_j$ of jobs in $\boldsymbol{J}_{\text{odd}}$ (as well as those of jobs in $\boldsymbol{J}_{\text{even}}$) lose their property of being multiples of each other (because β^2 is not an integer any more). As a result, Theorem 3.3 cannot guarantee now the feasibility of the LNI-schedule produced by the LNIS-algorithm. In the algorithm below we solve this problem, and furthermore, manage to decrease the running time down to a function linear in n.

SQUARED-ROUND-AND-DIVIDE SCHEDULING (SRDS):

Step 1: Applying the *First gluing procedure*, we get an instance with $n' \leqslant \min\{n, 2^m\}$ aggregated jobs $\{\hat{J}_j\}$ with processing times $\hat{p}_j$. By Lemma 2.1, this can be done in $O(m^2 n)$ time.

Step 2: Round the squared processing time $\hat{p}_j^2$ of each aggregated job $\hat{J}_j$ up to the nearest value 2^{α_j} for an integer α_j. Divide the whole set $\hat{J}$ of aggregated jobs into two subsets: $\boldsymbol{J}_{\text{odd}}$ (the set of jobs with odd exponents α_j) and $\boldsymbol{J}_{\text{even}}$ (with even exponents α_j). For each $\{J_j \in \boldsymbol{J}_{\text{odd}}\}$ define $p'_j = 2^{(\alpha_j-1)/2}$; set $\beta = \max_{J_i \in \boldsymbol{J}_{\text{odd}}} \frac{\hat{p}_i}{p'_i}$. Define instance I'_{odd} by jobs $\{J_j \in \boldsymbol{J}_{\text{odd}}\}$ with new processing times $\bar{p}_j = \beta p'_j$ and instance I''_{even} by jobs $\{J_j \in \boldsymbol{J}_{\text{even}}\}$ with new processing times $p''_j = 2^{\alpha_j/2}$. Compute the loads ℓ'_i and ℓ''_i of each machine M_i in the instances I'_{odd} and I''_{even}. Compute $\bar{\ell}_{\max} = \max_i(\ell'_i + \ell''_i)$.

Step 3: Applying the LNIS-algorithm, construct the LNI-schedule S' for the instance I'_{odd}.

Step 4: Applying the RNIS-algorithm, construct the RNI-schedule S'' (finishing at time $\bar{\ell}_{\max}$) for the instance I''_{even}.

Step 5: Round the starting time of each operation in schedule $\bar{S} = S' \cup S''$ down to the nearest integer, providing an integer-valued schedule $\hat{S}$ for the instance with aggregated jobs.

Step 6: Disaggregate each job $\hat{J}_j$ into the original jobs, preserving identical orders of those jobs in every aggregated operation of $\hat{J}_j$. The resulting schedule S is the desired one.

Theorem 5.1. *For any instance of the OB-problem there exists a feasible integer-valued schedule S with length $C_{\max}(S) < \sqrt{2}\,\ell_{\max}$; the schedule can be found by the SRDS-algorithm in $O(m^2 n)$ time, where m is the number of machines and n is the number of jobs.*

Proof. Due to Theorem 3.3, the LNI-schedule S' is feasible for the instance I'_{odd}, and the RNI-schedule S'' is feasible for the instance I''_{even}. The overall schedule $\bar{S} = S' \cup S''$ is also feasible for the whole instance $I'_{\text{odd}} \cup I''_{\text{even}}$, because on each machine M_i the jobs of the instances I'_{odd} and I''_{even} are scheduled within not overlapping intervals $[0, \ell'_i]$ and $[\ell'_i, \bar{\ell}_{\max}]$.

Since we have $\bar{p}_j \geqslant \hat{p}_j$ for every $J_j \in \boldsymbol{J}_{\text{odd}}$ and $p''_j \geqslant \hat{p}_j$ for every $J_j \in \boldsymbol{J}_{\text{even}}$, schedule $\bar{S}$ remains feasible for the instance with (smaller) aggregated processing times $\{\hat{p}_j\}$. Next, since all $\{\hat{p}_j\}$ are integers, rounding each starting time down to the nearest integer does not break the feasibility of the schedule. Thus, schedule $\hat{S}$ is also feasible. Finally, the feasibility of schedule S follows from the following two properties:

(a) any two aggregated operations of an aggregated job $\hat{J}_j$ are processed in schedule $\hat{S}$ either simultaneously, or with no overlap in time;
(b) original jobs making up an aggregated job $\hat{J}_j$ are sequenced identically in each of its aggregated operations.

Since $p'_j < \hat{p}_j < \sqrt{2}p'_j$, we have $\beta < \sqrt{2}$. This implies the relation $\bar{p}_j = \beta p'_j < \sqrt{2}\hat{p}_j$ for each job $J_j \in \boldsymbol{J}_{\text{odd}}$. Similar relations $p_i'' < \sqrt{2}\hat{p}_j$ hold for jobs $J_j \in \boldsymbol{J}_{\text{even}}$. This implies $\bar{\ell}_{\max} < \sqrt{2}\ell_{\max}$, and therefore

$$C_{\max}(S) \leqslant C_{\max}(\bar{S}) = \bar{\ell}_{\max} < \sqrt{2}\ell_{\max}.$$

It remains to estimate the running time of the algorithm. By Lemma 2.1, Step 1 can be implemented in $O(nm^2)$ time. Time $O(nm)$ is clearly sufficient for Steps 2, 5 and 6. Finally, since $O(n' \log n') \leqslant O(n \log 2^m) = O(nm)$, by Theorem 3.3 the running time of Steps 3 and 4 is $O(nm)$. □

It follows from Theorem 4.6, that no PTAS can be designed for the OB-problem (unless $\mathbb{P} = \mathbb{NP}$) in the case that the number of machines is treated as a variable. An opposite result can be derived, if we treat the number of machines

as a constant. In this case, a linear time PTAS similar to that proposed by Sevastianov and Woeginger [17] for the Open Shop problem exists.

5.3. PTAS for the OB-problem with a fixed number of machines

For any $\varepsilon > 0$, we present an algorithm A_ε that outputs a schedule S with makespan $C_{\max}(S) \leqslant (1+\varepsilon)\,OPT$. The running time of A_ε is polynomially bounded in n (but exponentially in ε and m).

The desired $(1+\varepsilon)$-approximation can be easily attained in the case that $d_{\max} \leqslant \varepsilon \cdot \ell_{\max}$: a simple greedy algorithm [17] finds a schedule S with makespan

$$C_{\max}(S) \leqslant \ell_{\max} + d_{\max} \leqslant (1+\varepsilon)\ell_{\max} \leqslant (1+\varepsilon)\,OPT$$

in time $O(m^2 n)$. If we first apply the *Third gluing procedure* with $\boldsymbol{d} = \varepsilon \cdot \ell_{\max}$ (so as to reduce the number of jobs to $O(m/\varepsilon)$), we can reduce the overall running time downto $O(nm + C(m,\varepsilon))$, where $C(m,\varepsilon)$ is a function of m and ε.

In the case $d_{\max} > \varepsilon \cdot \ell_{\max}$ we divide the set of jobs $\boldsymbol{J}$ into three subsets L, M, S of *large*, *medium*, and *small* jobs, respectively. For given $\alpha' > \alpha'' > 0$, these sets are defined as follows:

$$L = \{J_j \in \boldsymbol{J} \mid d_j \geqslant \alpha' \ell_{\max}\}, \quad M = \{J_j \in \boldsymbol{J} \mid \alpha'' \ell_{\max} \leqslant d_j < \alpha' \ell_{\max}\},$$

$$S = \{J_j \in \boldsymbol{J} \mid d_j < \alpha'' \ell_{\max}\}.$$

The values of α' and α'' are chosen with respect to the value of ε so that:
(a) the number $|L|$ of large jobs is bounded above by a constant;
(b) the total length of medium jobs is at most $\varepsilon \cdot \ell_{\max}$;
(c) the ratio α''/α' is small enough, so as to meet $\alpha' + \alpha'' + \alpha''|L| \leqslant \varepsilon$.

As proved in [17], the numbers α' and α'' with the above properties exist, the value of α'' being at least $\alpha^* \doteq \varepsilon/(e^{1.25} \cdot 2^{m/\varepsilon})$. The latter bound enables us to reduce the total number of jobs to at most a constant number $2m/\alpha^*$ by applying the *Third gluing procedure* with the threshold $\boldsymbol{d} = \alpha^* \ell_{\max}$. Next, we compute an optimal schedule $OPT(L)$ for the set of large jobs, and finally, using the greedy algorithm [17], we complete the schedule for the remaining jobs, trying to fill up the "holes" in the $OPT(L)$ schedule with medium and (glued) small jobs as densely as possible.

Let us present a formal description of the algorithm A_ε in the case $d_{\max} > \varepsilon \cdot \ell_{\max}$.

ALGORITHM A_ε:

Step 1: Apply the *Third gluing procedure* with $\boldsymbol{d} = \varepsilon \cdot e^{-1.25} \cdot 2^{-m/\varepsilon} \cdot \ell_{\max}$. Number the jobs with $d_j > \boldsymbol{d}$ in nonincreasing order of d_j.

Step 2: Find a partition of the job set into subsets L, M, and S satisfying (a)–(c).

Step 3: Construct an optimal schedule $OPT(L)$ for the jobs in L.

Step 4: Use the greedy algorithm [17] to place the jobs of M and S into $OPT(L)$.

Similar to [17], the following result can be proved.

Theorem 5.2. *For any $\varepsilon > 0$ and any instance of the $OB||C_{\max}$ problem with n jobs and m machines, algorithm A_ε constructs a schedule S with makespan*

$$C_{\max}(S) \leqslant (1+\varepsilon)\,OPT.$$

The algorithm runs in time $O(nm + C(m,\varepsilon))$, where $C(m,\varepsilon)$ is the time needed to find an optimal schedule for at most $O(2^{m/\varepsilon})$ so-called "large" jobs.

6. Concluding remarks

In our paper we provided a certain basis to design good performing scheduling algorithms for multicast star coupled WDM LANs. On the other hand, we introduced a new scheduling model based on a new type relation between the operations of each job. The simplest scheduling problem of this type (called an OB-problem) was considered, and a

first (yet quite comprehensive) complexity analysis of the problem was performed. However, a lot of questions remain open. Some of the most interesting ones are listed below.

1. Does there exist a pseudo-polynomial algorithm for solving the OB-problem with any fixed number of machines?
2. What is the complexity status of the 5-machine OB-problem? (We know that the 4-machine problem is polynomially solvable, while the one with 6 machines is NP-hard.)
3. What is the exact polynomial time approximation threshold ρ^* for the OB-problem with variable number of machines? (We know that $1.1 \leqslant \rho^* \leqslant \sqrt{2}$.)
4. What is the complexity status of the OB-problem with at most 2 operations per job?

Acknowledgment

The authors would like to express their gratitude to Olga Gerber for her kind assistance in preparing the paper.

References

[1] A.K. Amoura, E. Bampis, C. Kenyon, Y. Manoussakis, Scheduling independent multiprocessor tasks, in: Proc. Fifth European Symp. on Algorithms (1997), Lecture Notes in Computer Science, Vol. 1284, Springer, Berlin, 1997, pp. 1–12.
[2] I. Baldine, L. Jackson, G. Rouskas, Helios: a broadcast optical architecture, in: Proc. Second Internat. IFIP-TC6 Networking Conf. (Piza, Italy, 2002), Lecture Notes in Computer Science, Vol. 2345, Springer, Berlin, 2002, pp. 887–898.
[3] E. Bampis, M. Caramia, J. Fiala, A.V. Fishkin, A. Iovanella, On scheduling of independent dedicated multiprocessor tasks, in: Proc. 13th Internat. Symp. on Algorithms and Computation, Vancouver, BC, Canada, 2002, Lecture Notes in Computer Science, Vol. 2518, Springer, Berlin, 2002, pp. 391–402.
[4] P. Brucker, Scheduling Algorithms, Springer, Berlin, Heidelberg, New York, 1995.
[5] T. Fiala, An algorithm for the open-shop problem, Math. Oper. Res. 8 (1983) 100–109.
[6] A.V. Fishkin, K. Jansen, L. Porkolab, On minimizing average weighted completion time of multiprocessor tasks with release dates, in: Proc. 28th Internat. Colloq. on Automata, Languages and Programming (Crete, 2001), Lecture Notes in Computer Science, Vol. 2076, Springer, Berlin, 2001, pp. 875–886.
[7] O. Gerstel, B. Li, A. McGuire, G.N. Rouskas, K. Sivalingam, Z. Zhang (Eds.), Special issue on protocols and architectures for next generations optical wdm networks, IEEE J. Selected Areas in Commun. 18 (2000).
[8] T. Gonzalez, S. Sahni, Open shop scheduling to minimize finish time, J. ACM 23 (1976) 665–679.
[9] J.A. Hoogeveen, S.L.V. de Velde, B. Veltman, Complexity of scheduling multiprocessor tasks with prespecified processor allocations, Discrete Appl. Math. 55 (1994) 259–272.
[10] M. Kuznetsov, N. Froberg, S. Henion, H. Rao, J. Korn, K. Rauschenbach, E. Modiano, V. Chan, A next-generation optical regional access networks, IEEE Commun. Mag. 38 (2000) 66–72.
[11] B. Mukherjee, WDM-based local lightwave networks part I: single-hop systems, IEEE Network Mag. (1992), 12–27.
[12] G.N. Rouskas, Scheduling algorithms for unicast, multicast, and broadcast, in: K. Sivalingam, S. Subramanian (Eds.), Optical WDM Networks: Principles and Practice, Kluwer Academic Publishers, Dordrecht, 2000, pp. 171–188.
[13] S.V. Sevast'janov, Polynomially solvable case of the open-shop problem with arbitrary number of machines, Cybernet. Systems Anal. 28 (1992) 918–933 (Translated from Russian).
[14] S.V. Sevast'janov, Vector summation in Banach space and polynomial algorithms for flow shops and open shops, Math. Oper. Res. 20 (1995) 90–103.
[15] S. Sevastianov, Nonstrict vector summation in multi-operation scheduling, Ann. Oper. Res. 83 (1998) 179–211.
[16] S.V. Sevastianov, G.J. Woeginger, Makespan minimization in open shops: a polynomial time approximation scheme, Math. Programming 82 (1998) 191–198.
[17] S.V. Sevastianov, G.J. Woeginger, Linear time approximation scheme for the multiprocessor open shop problem, Discrete Appl. Math. 114 (2001) 273–288.
[18] D. Thaker, G.N. Rouskas, Multi-destination communication in broadcast WDM networks: a survey, Technical Report 2000-08, North Caroline State University, 2000.
[19] The NGI Helious project: Regional Testbed Optical Access Network For IP Multicast and Differentiated Services. ⟨http://projects.anr.mcnc.org/Helios/⟩, 2000.
[20] R.E. Wagner, R.C. Alferness, A.A.M. Saleh, M.S. Goodman, MONET: multiwavelength optical networking, J. Lightwave Technol. 14 (1996) 1349.
[21] D.P. Williamson, L.A. Hall, J.A. Hoogeveen, C.A.J. Hurkens, J.K. Lenstra, S.V. Sevastianov, D.B. Shmoys, Short shop schedules, Oper. Res. 45 (1997) 288–294.