

# Scheduling with conflicts: online and offline algorithms

Guy Even · Magnús M. Halldórsson · Lotem Kaplan · Dana Ron

Received: 22 January 2008 / Accepted: 1 September 2008 / Published online: 28 October 2008  
© Springer Science+Business Media, LLC 2008

**Abstract** We consider the following problem of scheduling with conflicts (SWC): Find a minimum makespan schedule on identical machines where conflicting jobs cannot be scheduled concurrently. We study the problem when conflicts between jobs are modeled by general graphs.

Our first main positive result is an *exact* algorithm for two machines and job sizes in  $\{1, 2\}$ . For jobs sizes in  $\{1, 2, 3\}$ , we can obtain a  $\frac{4}{3}$ -approximation, which improves on the  $\frac{3}{2}$ -approximation that was previously known for this case. Our main negative result is that for jobs sizes in  $\{1, 2, 3, 4\}$ , the problem is APX-hard.

Our second contribution is the initiation of the study of an online model for SWC, where we present the first results in this model. Specifically, we prove a lower bound of  $2 - \frac{1}{m}$  on the competitive ratio of any deterministic online algorithm for  $m$  machines and unit jobs, and an upper bound of 2 when the algorithm is not restricted computationally. For three machines we can show that an efficient greedy algorithm achieves this bound. For two machines we present a more complex algorithm that achieves a competitive ratio of  $2 - \frac{1}{7}$  when the number of jobs is known in advance to the algorithm.

**Keywords** Scheduling with conflicts · Mutual exclusion scheduling · Approximation algorithms · Online algorithms

## 1 Introduction

We consider the problem of *scheduling with conflicts* (SWC), defined as follows. There are  $m$  identical machines. The input consists of a set  $J$  of  $n$  jobs with processing times  $\{p_j\}_{j \in J}$ , and a conflict graph  $G = (J, E)$  over the jobs. Each edge in  $E$  models a pair of conflicting jobs that cannot be scheduled concurrently (on different machines). A *schedule* is an assignment of time intervals on the  $m$  machines to the jobs that satisfies the following conditions: (i) each job  $j \in J$  is assigned an interval of length  $p_j$  on one machine; (ii) intervals on the same machine do not overlap; and (iii) intervals assigned to conflicting jobs do not overlap. The *makespan* of a schedule is the largest endpoint of an interval assigned to a job. We are interested in schedules that minimize the makespan.

Scheduling with conflicts generally arises as resource-constrained scheduling (Garey and Graham 1975). In this setting there is a set of resources, each with a certain supply. Each job has a specified demand for each resource. A conflict arises between a subset of jobs if their cumulative demand of a resource exceeds its supply. In general, this setting can be modeled by a conflict hypergraph. In special cases, a conflict graph suffices, e.g., if the resources are nonsharable. In Baker and Coffman (1996), an application of this type is presented for balancing the load in a parallel computation. In Halldórsson et al. (2003), other applications are mentioned in traffic intersection control, frequency assignment in cellular networks, and session management in local area networks. An application derived from a problem of assigning operations to processors, where the operations

---

G. Even (✉) · L. Kaplan · D. Ron  
School of Electrical Engineering, Tel-Aviv Univ., Tel-Aviv 69978,  
Israel  
e-mail: [guy@eng.tau.ac.il](mailto:guy@eng.tau.ac.il)

L. Kaplan  
e-mail: [lotem@eng.tau.ac.il](mailto:lotem@eng.tau.ac.il)

D. Ron  
e-mail: [danar@eng.tau.ac.il](mailto:danar@eng.tau.ac.il)

M.M. Halldórsson  
Dept. of Computer Science, Faculty of Engineering, University  
of Iceland, Reykjavik, Iceland  
e-mail: [mmh@hi.is](mailto:mmh@hi.is)

are given in a flow graph, is described in Bodlaender and Jansen (1995).

In the online version of SWC, each job  $j$  has a release time  $r_j$  that specifies when the job arrives. A job  $j$  arrives together with the conflict edges to all jobs whose release time is at most  $r_j$ . Hence, the conflict graph is revealed as the jobs arrive. A job  $j$  can be scheduled starting any time  $t \geq r_j$ . Scheduling decisions at time  $t$  can only depend on jobs that have arrived before or at time  $t$ . This model is often called *scheduling over time*. We compare the makespan of the schedule computed by an online algorithm to the makespan of an optimal offline algorithm whose input consists of the whole conflict graph and the release times of all the jobs. We are interested in bounding the competitive ratio, i.e., the worst case ratio between the online makespan and the optimal offline makespan.

Problems involving conflict scheduling can be classified according to the following parameters: (i) The objective: minimum makespan, minimum sum of completion times, maximum response time (latency). (ii) Job model: unit jobs, arbitrary processing times with/without preemption, batch scheduling. (iii) Number of machines  $m$ : fixed or unlimited. In this paper we focus on the case characterized by the minimum makespan objective, nonpreemptive processing, and a fixed number of machines. We consider the cases of unit jobs, short jobs, and arbitrary processing times. One may further characterize problems by: (iv) Online/Offline model: In the online model, jobs arrive at different release times. (v) Conflict graph: general or belonging to specific classes (e.g., trees, interval graphs, etc.) We consider both the online and offline models, and treat general graphs.

### 1.1 Previous results

A more general case of scheduling with conflicts (i.e., conflicts might not be representable by a graph) was studied by Garey and Graham (1975). They analyzed a type of greedy list scheduling algorithm, and their analysis implies an approximation ratio of  $\frac{m+1}{2}$  for SWC. We prove the tightness of this upper bound by presenting examples with unit jobs in which the conflicts can be represented by a graph and the algorithm is forced to have an approximation ratio of  $\frac{m+1}{2}$ .<sup>1</sup> Baker and Coffman (1996) considered SWC with unit jobs under the name *mutual exclusion scheduling*. An optimal algorithm for two machines and unit jobs based on matching is described in Garey and Johnson (1975), Baker and Coffman (1996).

By a reduction from PARTITION (Garey and Johnson 1979), SWC is weakly NP-hard for two machines and arbitrary processing times even if the conflict graph is empty.

<sup>1</sup>Garey and Graham (1975) proved a lower bound of  $\frac{m+1}{2}$  for a more general problem and, in particular, their lower bound construction does not correspond to a graph.

Strong NP-hardness is known to hold in the unit case for  $m \geq 3$  in general graphs (Baker and Coffman 1996) and for various special graphs, including complements of comparability graphs for  $m \geq 3$  (Lonc 1992), interval graphs for  $m \geq 4$  (Bodlaender and Jansen 1993), and permutation graphs for  $m \geq 6$  (Jansen 2003).

SWC with unit jobs on  $m$  machines is equivalent to finding a minimum coloring of the conflict graph in which every color class contains at most  $m$  vertices, known as *m-bounded coloring*. The correspondence between these problems is based on assigning a different color to each time unit in the schedule, thus the number of colors equals the makespan of the schedule. This case was considered by Baker and Coffman (1996), who gave a greedy algorithm that yields a schedule whose makespan is within an additive  $d_k$  of the optimal, where  $k = \lceil n/m \rceil + 1$  and  $d_k$  is the  $k$ th largest degree in the graph. However, this result does not give a nontrivial approximation ratio if the  $k$ th largest degree is close to  $n$ . Special graphs for which the  $m$ -bounded coloring problem (SWC with unit jobs) is polynomially solvable include forests (Baker and Coffman 1996), split graphs and complements of interval graphs (Lonc 1992), cographs and bipartite graphs (Bodlaender and Jansen 1993; Hansen et al. 1993), constant treewidth (Kaller et al. 1995), and line graphs (Alon 1983). In Baker and Coffman (1996), a constant approximation algorithm is presented for the case in which the conflict graph is obtained from a two-dimensional domain decomposition problem.

### 1.2 Observations based on previous results

SWC with unit jobs can be formulated as a set cover problem with sets of size at most  $m$ . Applying the greedy algorithm for set cover gives an approximation ratio of  $\mathcal{H}_m$ . This approximation ratio can be improved (for small values of  $m$ ) to  $\mathcal{H}_m - \frac{1}{2}$  due to a result of Duh and Fürer (1997). By applying scaling, the approximation ratio for unit jobs can be extended to arbitrary processing times. Scaling incurs an increase in the approximation ratio that is logarithmic in the ratio between the maximum processing time and the minimum processing time. If this ratio is larger than  $\log n$ , then additional scaling can be used to reduce the approximation ratio to  $O(\log n \cdot \log m)$ .

On the hardness side, the problem is already APX-hard in the case of unit jobs and  $m = 3$ . Petrank (1994) proved that it is NP-hard to distinguish between instances of 3-Dimensional-Matching that admit a perfect matching and instances that admit only matchings that cover a  $(1 - \epsilon)$ -fraction of the elements. The same proof also implies a hardness gap for maximum packing of triangles in a graph. This, in turn, implies a hardness gap for scheduling with conflicts of unit jobs on 3 machines.

If we view  $m$  as a nonconstant parameter, then the problem is highly non-trivial on general graphs. In order to

**Table 1** Approximation ratios for scheduling with conflicts. The first two algorithms have running time exponential in  $m$  while the third has running time polynomial in  $m$ . Lines above the double rule refer to results based on previous work

Processing times $p_j$	#machines $m$	Approximation ratio	Technique
Unit	$m$	$\mathcal{H}_m - \frac{1}{2}$	$m$ -set-cover (based on Duh and Fürer 1997)
Arbitrary	$m$	$O(\log \min\{\frac{\max_j p_j}{\min_j p_j}, n\} \cdot \log m)$	$m$ -set-cover + scaling
Arbitrary	$m$	$\frac{m+1}{2}$	Greedy (list scheduling) (Garey and Graham 1975)
{1, 2}	2	1 (optimal)	Maximum matching
{1, 2, 3}	2	4/3	Reduction to {1, 2}
{1, 2, 3, 4}	2	APX-hard	

**Table 2** Lower and upper bounds on the competitive ratio for SWC in the online model. Results in lines above the double rule are based on recomputing an offline solution of the pending jobs in several phases

Processing times $p_j$	#machines $m$	Competitive ratio	Comment
Arbitrary	$m$	2	Exp. time (Shmoys et al. 1995)
Arbitrary	$m$	$\rho_m + 2$ ( $\rho_m + 1$ for unit jobs)	$\rho_m$ -approximate offline alg.
Unit	$m$	$\geq 2 - \frac{1}{m}$	
Arbitrary	$m$	$\frac{m+2}{2}$ ( $\frac{m+1}{2}$ for unit jobs, $m \geq 3$ )	Greedy alg.
Unit	2	$2 - \frac{1}{7}$	Non-oblivious model

schedule  $m$  jobs in a single round, you need to find an independent set of size  $m$  in the conflict graph. There are no algorithms known for this that run in time  $n^{o(m)}$ ; the fact that the Independent Set problem is complete for the complexity class  $W[1]$  (Downey and Fellows 1995) suggests that improvements may not be possible. And even if we are satisfied with smaller sets, there currently are no known polynomial time algorithms that find  $\omega(1)$ -size independent sets, even when the independence number is  $\sqrt{n}$ . Thus, we cannot expect an  $o(m)$ -approximation factor, even for unit jobs, in time polynomial independent of  $m$ . Further, for unbounded  $m$ , the unit jobs case is equivalent to the classic graph coloring problem, which is NP-hard to approximate within  $n^{1-\epsilon}$ -ratio (Feige and Kilian 1998; Zuckerman 2006), for any  $\epsilon > 0$ .

### 1.3 Our results: the offline model

The approximation ratios in the offline model are summarized in Table 1. Given the hardness results mentioned previously, we focus on the basic case of  $m = 2$ .

On the positive side we give an *optimal* algorithm for two machines with processing times  $p_j \in \{1, 2\}$  and a 4/3-approximation algorithm for  $p_j \in \{1, 2, 3\}$ . Note that a factor of 4/3 improves on the 3/2-approximation that can be obtained using Garey and Graham (1975). The first result is based on constructing an appropriate auxiliary graph whose vertices are jobs or “slices” of jobs and finding a maximum matching in this graph. We then show how to turn the match-

ing found into an optimal schedule. Interestingly, this algorithm is related to finding a maximum  $b$ -matching for  $b = 2$  when the  $b$ -matching is required to be bipartite. The second result (for  $p_j \in \{1, 2, 3\}$ ) is obtained by a reduction to the first result. For a direct proof of this approximation ratio see Kaplan (2007). On the negative side, we show that for  $p_j \in \{1, 2, 3, 4\}$  the problem is APX-hard. We leave as an open problem the question of the exact status of the case  $p_j \in \{1, 2, 3\}$ .

### 1.4 Our results: the online model

We present the first results for SWC in the online model (summarized in Table 2). These results include lower and upper bounds for the competitive ratio of deterministic online algorithms. We begin by analyzing a greedy algorithm for unit jobs and  $m = 2$  machines. In contrast to the greedy list scheduling algorithm (Garey and Graham 1975) (in the offline model), the greedy algorithm (in the online model) is shown to be no better than the trivial algorithm that uses a single machine. Namely, we prove that the competitive ratio of the greedy algorithm is 2 regardless of whether it uses a First In First Out (FIFO) or Last In First Out (LIFO) policy. For  $m > 2$ , a competitive ratio of 2 is achievable for arbitrary processing times in exponential time by computing an optimal schedule of the pending jobs in phases.<sup>2</sup> For

<sup>2</sup>This follows from Shmoys et al. (1995) and we include a proof for completeness.

$m = 3$  and unit jobs, a 2-competitive ratio can be obtained in polynomial time. We nearly match this upper bound for unit jobs by a lower bound of  $2 - \frac{1}{m}$ . In general, an offline  $\rho$ -approximation implies an online  $\rho + 2$  competitive ratio (and  $\rho + 1$  for unit jobs).

The true value of the best possible competitive ratio is an intriguing open question. We make a step in this direction by giving a nontrivial deterministic online algorithm for  $m = 2$  with a competitive ratio of  $2 - \frac{1}{7}$ . The algorithm is “non-oblivious” in the sense that it is given the number of jobs in advance; this assumption can be removed at the cost of slightly increasing the competitive ratio and flipping a single coin. The algorithm is based on the observation from our lower bounds that FIFO behaves badly when many jobs appear early in the schedule, while LIFO behaves badly when the jobs appear at a relatively slow rate. Roughly speaking, the algorithm decides to follow either (a variant of) the FIFO policy or (a variant of) the LIFO policy depending on the rate at which the jobs arrive.

### 1.5 Other related work

A special type of resource-constrained problem, in fact, a subproblem of SWC, is the scheduling of multiprocessor tasks with dedicated processors (see Drozdowski 1996 for a dated review). Each job is given with a specified set of processors that it requires the exclusive use of during its execution. Here, the processors are the resource, and the number of resources is  $m$  while the number of “machines” can be considered to be unlimited. For  $m$  fixed, there exists a linear time fully-polynomial approximation scheme by Jansen and Porkolab (2002). Note that in this case the conflict graph has a constant number of vertices.

By varying the scheduling parameters, a large body of research opens up. In the case of unbounded number of machines, the focus has been on special graph classes, due to the aforementioned hardness for graph coloring. In our case of nonpreemptive makespan scheduling, two classic problems are *dynamic storage allocation* (Buchsbaum et al. 2004) (corresponding to interval graphs) and the *file transfer problem* (Coffman 1985) (corresponding to line graphs). If we additionally change to unit jobs or preemptive scheduling, we get the classical graph coloring or multicoloring problems, of which there is an extensive literature. If we instead modify the objective function, then conflict (non-preemptive and preemptive) scheduling with the sum of completion times measure was introduced in Bar-Noy et al. (2000) and studied in several recent works.

Bodlaender et al. (1994) considered a complementary problem called scheduling incompatible jobs (CIJ). In CIJ, incompatible jobs cannot be processed by the same machine. They gave an approximation algorithm for CIJ with arbitrary processing times whose performance is good in the case that

the graph can be colored with few colors  $k < m$ . The ratio obtained approaches 2 as  $m/k$  increases, and is at most  $(m + 1)/2$  for  $k = m - 1$ . CIJ was shown to be hard to approximate within any factor less than 2, for any  $m \geq 3$ . They also gave approximations for the special classes of bipartite, bounded-treewidth, and complete graphs.

Somewhat related are papers on bin-packing with conflicts, where in our terms the jobs are nonunit, the makespan is restricted, and the goal is to minimize the number of machines (Jansen 1999; Epstein and Levin 2006).

Online algorithms have been studied for various coloring and multicoloring problems. However, the work closest to ours is that of Irani and Leung (2003). They studied a version involving unit jobs, unbounded number of machines, with the objective of minimizing the maximum response time (i.e., job completion time minus its release time). Further, they restricted their attention to bipartite and interval graphs.

### 1.6 Open problems

Below we list a few open problems.

1. Recall that we give an optimal algorithm for two machines and processing times  $p_j \in \{1, 2\}$  while for  $p_j \in \{1, 2, 3\}$  we give a  $4/3$ -approximation ratio. Is it possible to solve the latter case optimally, or is the problem hard? Recall that for  $p_j \in \{1, 2, 3, 4\}$  the problem is APX-hard.
2. Our algorithm for  $p_j \in \{1, 2, 3\}$  on two machines can be extended to the case of  $p_j \in \{1, \dots, k\}$ , where it gives a ratio of  $2 - \frac{2}{k}$ . However, for  $k > 3$  (and  $m = 2$ ) the algorithm of Garey and Graham (1975) has a better performance of  $3/2$ . Is it possible to beat the  $3/2$  bound for  $k > 3$ ?
3. Our online algorithm for unit jobs on two machines is based on deciding whether to run a FIFO-based policy or a LIFO-based policy according to the number of jobs that arrived up till a particular point. Is it possible to get a competitive ratio better than 2 using an oblivious algorithm that switches between a FIFO policy and a LIFO policy according to the number of pending jobs? Is it possible to improve on our bound of  $2 - \frac{1}{7}$  and possibly even match the lower bound of  $3/2$ ?
4. Is there an algorithm with ratio  $o(m)$  when  $m$  is a fixed constant?

## 2 Preliminaries

*The offline model* In a scheduling problem with conflicts on  $m$  machines, the input consists of (i) a set  $J$  of  $n$  jobs, (ii) a (simple) conflict graph  $G = (J, E)$  over the jobs, and (iii) an integral processing time  $p_j$  for each job  $j \in J$ . Adjacent jobs in  $G$  are conflicting and cannot be scheduled concurrently on different machines. Each job  $j$  consists of  $p_j$

*job slices*, each of unit length. Time is discrete and proceeds in rounds. We use the notation  $[t_1, t_2]$  for  $t_1 \leq t_2$  to denote the set  $\{t_1, t_1 + 1, \dots, t_2\}$ . We refer to a pair  $\langle i, t \rangle$ , where  $i$  is a machine and  $t$  is a round, as a *machine slot*. A set of slots  $\{\langle i, t \rangle : t \in [t_1, t_2]\}$  is called an *interval* of slots. Two slots  $\langle i_1, t \rangle$  and  $\langle i_2, t \rangle$  with the same round component are called *concurrent*.

A *schedule*  $T$  is a one-to-one assignment from job slices to slots that satisfies the following two properties: (i) Non-preemptive: the slices of a job are assigned to an interval of slots on the same machine. (ii) Nonconflicting: job slices assigned to concurrent slots do not belong to conflicting jobs. Note that since the schedule is one-to-one and preemption is prohibited, the number of slots in the interval of slots assigned to job  $j$  equals  $p_j$ .

The *makespan* of a schedule  $T$  is the last round in which a slot is assigned a job slice. We denote the makespan of a schedule  $T$  by  $\text{makespan}(T)$ . The goal in this paper is to find a schedule with a minimum (or approximately minimum) makespan.

We concentrate on *deterministic scheduling algorithms* both in the offline and the online models.

*The online model* In the online model, each job  $j$  has release time  $r_j$ . As in the offline model, time is divided into discrete rounds starting with round one. Job  $j$  arrives in the beginning of round  $r_j$ , and may be assigned to machine slots in rounds greater than or equal to  $r_j$ . This model is often called scheduling over time. A job  $j$  arrives with the conflict edges to all jobs whose release time is at most  $r_j$ . Hence, the conflict graph is revealed as the jobs arrive.

We deal mostly with unit jobs in the online model. An online scheduling algorithm for unit jobs proceeds as follows. The set of released jobs that have not already been scheduled are *pending*. In the beginning of each round  $t$ , the scheduling algorithm selects a subset  $J_t$  of at most  $m$  nonconflicting pending jobs. These jobs are scheduled in round  $t$ . In the beginning of the next round  $t + 1$ , the set of pending jobs is updated; the jobs  $J_t$  scheduled in round  $t$  are removed, and the jobs with release time  $r_j = (t + 1)$  are added. In the more general case of jobs with varying processing times, there are two modifications: (1) The decision concerning which jobs to schedule is not necessarily made at the start of each round, but rather at the start of rounds where at least one machine becomes idle. (2) For each such round  $t$ , the set of jobs selected to be scheduled starting at round  $t$  must be nonconflicting with all jobs already scheduled in round  $t$ . Note that if the total number of jobs is not known in advance to the algorithm, then it runs forever (since a new job may arrive in any round). Regardless of whether the number of jobs is known in advance, the makespan is defined to be the last round in which a job is scheduled. We assume that the time required to determine which jobs to schedule in each round is negligible compared to the duration of a round.

We compare the makespan of the schedule computed by an online algorithm to the makespan of an optimal offline algorithm whose input consists of the whole conflict graph and the release times of all the jobs. We are interested in bounds on the worst case ratio between the online makespan and the optimal offline makespan.

*Additional terminology* The complement of the conflict graph  $G$  is the *agreement graph*  $\overline{G} = (J, \overline{E})$ . Namely, the agreement graph is a simple graph in which  $(j_1, j_2) \in \overline{E}$  if and only if  $(j_1, j_2) \notin E$ . Since independent sets in the conflict graph correspond to cliques in the agreement graph, jobs that form a clique of size  $m$  in the agreement graph  $\overline{G}$  can be scheduled concurrently.

A slot that is not assigned a job slice is called *hole*. The set of holes in a schedule  $T$  is denoted by  $\text{holes}(T)$ , and the number of holes is denoted by  $|\text{holes}(T)|$ . A round in which all slots are holes is said to be *vacant*.

Let  $p(J') \triangleq \sum_{j \in J'} p_j$ . In particular, the total amount of work that needs to be processed is  $p(J)$ .

For a given input, let  $OPT$  denote a schedule with minimum makespan and let  $|OPT|$  denote its makespan.

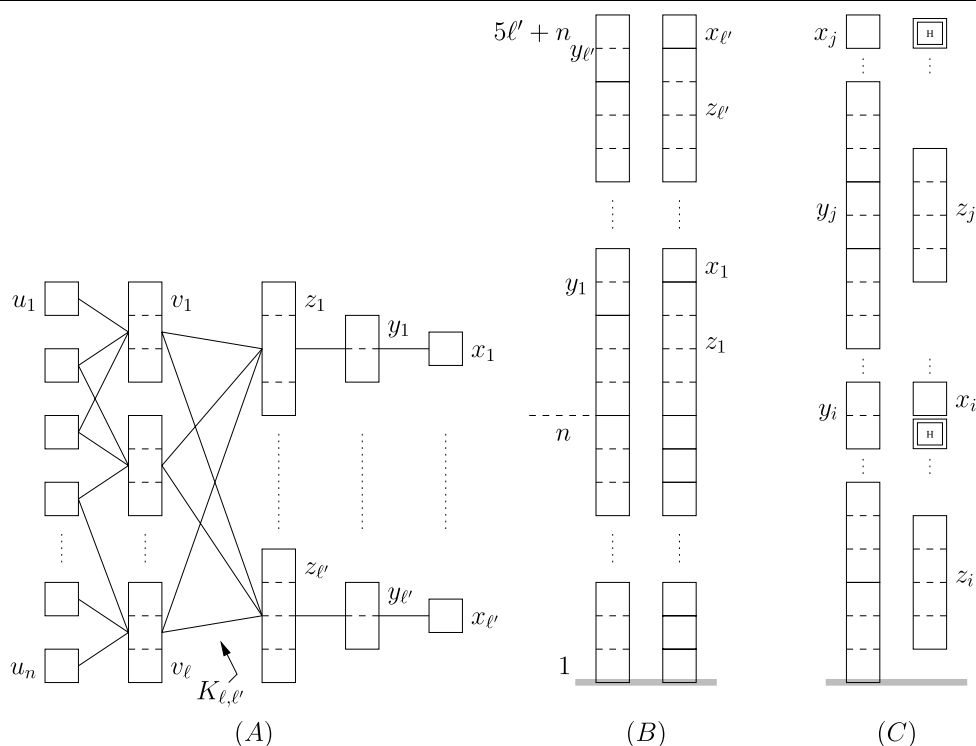
### 3 Results for the offline model

In this section we present three results for scheduling on two machines. The first result is an APX-hardness proof for two machines and  $p_j \in \{1, 2, 3, 4\}$ . The second and third results improve on the upper bound of  $\frac{m+1}{2} = \frac{3}{2}$  that the simple greedy algorithm of Garey and Graham (1975) (see also Appendix) achieves on two machines (and arbitrary job sizes). Specifically, the second result is an optimal algorithm for the case that the processing times satisfy  $p_j \in \{1, 2\}$ . The third result is a 4/3-approximation algorithm for the case that  $p_j \in \{1, 2, 3\}$ . These algorithms rely on computing a maximum matching in an appropriate auxiliary graph. We first recall what is known for unit jobs and two machines.

*Unit jobs* Garey and Johnson presented the following optimal algorithm for the case of unit jobs and two machines (Garey and Johnson 1975) (see also Baker and Coffman 1996): Compute a maximum matching  $M$  in the agreement graph  $\overline{G}$ , and schedule matched pairs concurrently but unmatched jobs as singletons. The optimality of this algorithm is based on the following observation.

**Observation 3.1** *In the case of unit jobs and two machines, every matching  $M$  in the agreement graph induces a schedule with makespan of  $|J| - |M|$ . Correspondingly, every schedule  $T$  without vacant rounds induces a matching  $M_T$  in  $\overline{G}$ , such that  $|M_T| = |J| - \text{makespan}(T)$ .*

**Fig. 1** An illustration of the construction for the APX-hardness result on 2 machines (Theorem 1). (A) An illustration of the agreement graph  $\overline{G}(U, \mathcal{C})$ . (B) The schedule when the input instance for set cover has a perfect packing. (C) An example of a schedule with holes



3.1 APX-Hardness for  $m = 2$

**Theorem 1** SWC is APX-hard for  $m = 2$  machines. This holds even if jobs are of length at most 4.

*Proof* We give a reduction from 3-set packing. Namely, given is an instance  $(U, \mathcal{C})$ , where  $U$  is a set of  $n$  base elements, and  $\mathcal{C}$  is a collection of  $\ell$  triplets (subsets of size 3) from  $U$ . By Petrank’s result (Petrank 1994) for 3-dimensional matching (which is a special case of 3-set packing), it is NP-hard to distinguish between the following two cases: a) There is an exact disjoint cover of  $U$  with  $n/3$  sets from  $\mathcal{C}$ , or b) There is no collection of  $(1 - \epsilon)n/3$  disjoint sets from  $\mathcal{C}$ . This holds even if each element occurs in at most 5 different triplets, so that  $\ell \leq 5n/3$ . We assume that  $n$  is divisible by 3; otherwise, it is obviously not the first case, namely, does not admit an exact packing.

Given an instance  $(U, \mathcal{C})$ , we form an agreement graph  $\overline{G}(U, \mathcal{C})$  over three groups of jobs (represented by the graph vertices): *element* jobs  $u_1, \dots, u_n$  of unit length, *set* jobs  $v_1, \dots, v_\ell$  of length 3, and  $\ell' = \ell - n/3$  triplets of *dummy* jobs. Each triplet of dummy jobs consists of three jobs:  $x_i, y_i$ , and  $z_i$ , of length 1, 2, and 4, respectively. Note that  $p(J) = n + 3\ell + 7\ell' = 10\ell - (4/3)n = 10\ell' + 2n = O(n)$  (where  $J$  is the set of all jobs). The edges of  $\overline{G}(U, \mathcal{C})$  are determined as follows: There is an edge between an element job  $u_i$  and a set job  $v_j$  whenever the corresponding element is in the set; there is an edge between every pair  $v_j$  and  $z_i$ ;

and for every  $i = 1, \dots, \ell'$  we have the edges  $(x_i, y_i)$  and  $(y_i, z_i)$  (see Fig. 1(A)).

In what follows, when we say that two jobs are scheduled *opposite to each other* we mean that there is at least one round in which a slice of one job is scheduled concurrently with a slice of the other job. We say that a job is *fully* opposite another job if each of its slices is scheduled opposite to the other job. Suppose there is a perfect packing in  $(U, \mathcal{C})$ , that is, there are  $n/3$  disjoint sets of elements in  $\mathcal{C}$ . Then we can form a schedule of makespan  $p(J)/2 = 5\ell' + n$  by scheduling each of the  $n/3$  set jobs corresponding to the disjoint sets concurrently with the three-element jobs it contains. The remaining  $\ell'$  set jobs are scheduled with the  $\ell'$  triplets of dummy jobs as follows. Group each dummy triplet together with one set job, and schedule them using five rounds as depicted in Fig. 1(B). This gives a perfect schedule of makespan  $p(J)/2 = 5\ell' + n$ .

We now turn to the case in which there is no collection of  $(1 - \epsilon)n/3$  disjoint sets from  $\mathcal{C}$ , and show that in this case every schedule must have makespan  $(p(J)/2)(1 + \delta)$  for  $\delta = \Omega(\epsilon)$ . We shall need the following claim.

**Claim 3.2** Let  $h$  be the number of holes in a schedule for the agreement graph  $\overline{G}(U, \mathcal{C})$ . Then the number of set jobs that are scheduled concurrently with three element jobs is at least  $n/3 - h$ .

*Proof* We prove the equivalent claim that there are at most  $\ell - (n/3 - h) = \ell' + h$  set jobs that have *less than* three el-

elements scheduled opposite to them. To this end we consider holes of two types: holes opposite set jobs, whose number we denote by  $h_s$ , and holes opposite dummy jobs (that is either an  $x$ -,  $y$ -, or a  $z$ -job), whose number we denote by  $h_d$ . Consider a dummy job  $z_i$ . Such a job is scheduled opposite at most two set jobs. Let  $a_i, i = 1, 2$ , be the number of  $z$ -jobs scheduled opposite to  $i$  jobs. Then,

$$a_1 + a_2 \leq \ell'. \tag{1}$$

For each  $z$ -job scheduled opposite to two set jobs, there is at least one hole (opposite the corresponding  $y$ - or  $x$ -job). Therefore,

$$a_2 \leq h_d. \tag{2}$$

Let  $b$  denote the number of set jobs that are scheduled opposite less than 3 element jobs and are not scheduled opposite any  $z$ -job. Then,

$$b \leq h_s. \tag{3}$$

Now, the total number of set jobs that are scheduled opposite less than 3 element jobs is at most

$$\begin{aligned} a_1 + 2a_2 + b &= (a_1 + a_2) + a_2 + b \\ &\leq \ell' + h_d + h_s \quad \text{by inequalities (1)–(3)} \\ &\leq \ell' + h, \end{aligned}$$

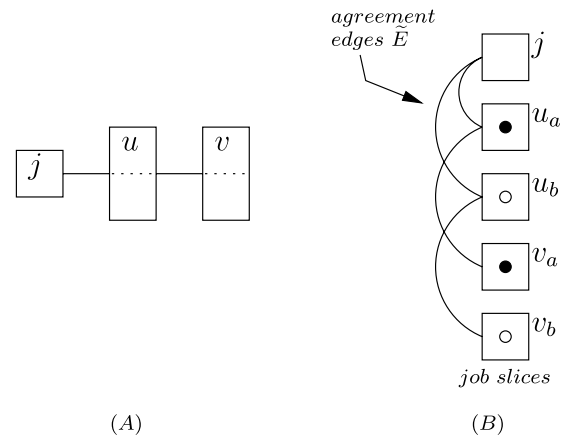
as claimed. □

If there is no 3-set packing of size  $(1 - \epsilon)\frac{n}{3}$ , then the number of set jobs that are scheduled concurrently with three element jobs can be at most  $(1 - \epsilon)\frac{n}{3} = \frac{n}{3} - \frac{\epsilon n}{3}$ . By Claim 3.2 this implies that for every schedule,  $h \geq \frac{\epsilon n}{3}$ , so that the total makespan is at least  $(p(J) + h)/2 = (p(J)/2)(1 + \frac{\epsilon n}{3p(J)})$ . Since  $\epsilon$  is a constant and  $p(J) = O(n)$ , we get a lower bound of  $(p(J)/2)(1 + \delta)$  for  $\delta = \Omega(\epsilon)$ . □

### 3.2 An optimal algorithm for processing times $p_j \in \{1, 2\}$

*Intuition* Consider a schedule on two machines and  $p_j \in \{1, 2\}$ . Such a schedule corresponds to a matching between job slices (of unit size, as defined in the preliminaries). Namely, each pair of job slices that are scheduled concurrently is matched, and a job slice opposite to a hole is left unmatched. Note that for all pairs of matched job slices, the corresponding jobs are adjacent in the agreement graph. Given this matching, the resulting makespan is exactly the sum of processing times minus the size of the matching.

In view of the above description, the high level idea of the algorithm is to create an auxiliary graph whose vertices correspond to job slices. We then find a maximum matching in this graph and transform the matching into a schedule.



**Fig. 2** (A) An agreement graph over three jobs  $j, u, v$ . The processing times are  $p_j = 1$  and  $p_u = p_v = 2$ . (B) The corresponding auxiliary graph. Job slices of type- $a$  are depicted using a filled circle while job slices of type- $b$  are depicted using an unfilled circle. Unit jobs lack a circle

Care must be taken when defining the graph that the matching found induces indeed a feasible schedule. In particular, it must be possible to schedule different slices of the same job consecutively. Details follow.

*Auxiliary graph* To simplify notation, we refer to unit jobs in this subsection with the letter  $j$ . Jobs whose processing time equals 2 are called *double size jobs* (in short, double jobs) and are named with the letters  $u$  and  $v$ .

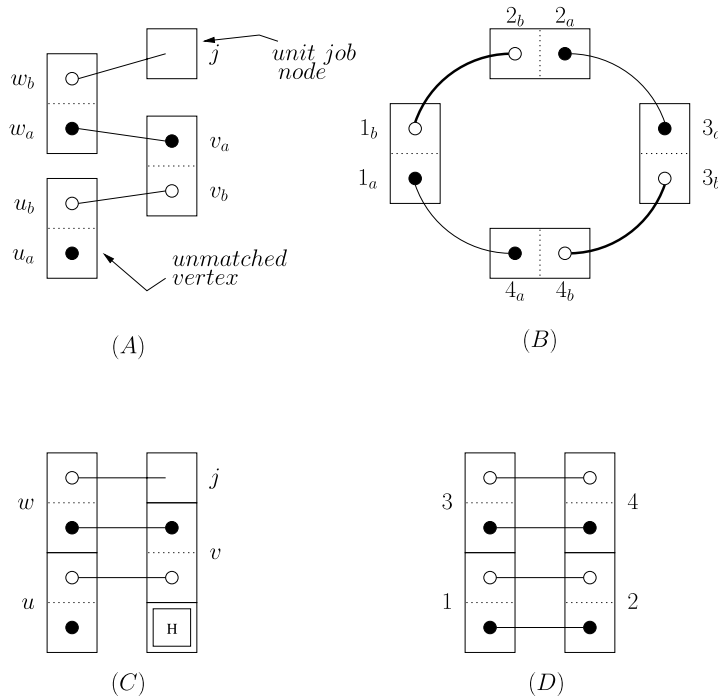
We define an auxiliary graph  $\tilde{G} = (\tilde{J}, \tilde{E})$  (see Fig. 2). The vertex set  $\tilde{J}$  contains one vertex for each unit job  $j \in J$ , and two vertices for each double job  $v \in J$  as follows. Each double job  $v$  is split into two slices: a *type- $a$*  vertex  $v_a$  and a *type- $b$*  vertex  $v_b$ . The edge set  $\tilde{E}$  consists of *agreement edges* between slices. Loosely speaking, agreement edges are induced by the agreement graph subject to the constraint that there are no agreement edges between a type- $a$  vertex and a type- $b$  vertex. Formally,

$$\begin{aligned} \tilde{E}_A \triangleq & \{(j, v_\sigma) \mid \sigma \in \{a, b\}, (j, v) \in \bar{E}\} \\ & \cup \{(v_\sigma, u_\sigma) \mid \sigma \in \{a, b\}, (v, u) \in \bar{E}\}. \end{aligned}$$

We refer to edges between type- $a$  vertices ( $v_a, u_a$ ) as *type- $a$  edges*, and to edges between type- $b$  vertices ( $v_b, u_b$ ) as *type- $b$  edges*.

Given the auxiliary graph  $\tilde{G}$  and a matching  $M$ , let  $\hat{G}_M$  denote the (multi) graph obtained from  $\tilde{G}$  and  $M$  as follows (see Fig. 3). Consider the subgraph of  $\tilde{G}$  induced by  $M$ . For each double job  $v \in \tilde{J}$ , coalesce the vertices  $v_a$  and  $v_b$  in  $\tilde{J}$  into a single vertex  $\hat{v}$  in  $\hat{J}$ . The vertices that correspond to unit jobs remain as in  $\tilde{J}$ . For each edge in the matching  $M$ , there is an edge in  $\hat{E}$ . If an edge in  $M$  is incident to a vertex  $v_a$  or  $v_b$  in  $\tilde{J}$ , then the corresponding edge in  $\hat{E}$  is incident to the vertex  $\hat{v}$  in  $\hat{J}$ . Note that each vertex in  $\hat{G}_M$  is incident to at most two edges.

**Fig. 3** An *unfilled circle* represents a type-*a* vertex, while a *filled circle* represents a type-*b* vertex. **(A)** A path in the auxiliary graph  $\widehat{G}_{\widetilde{M}}$ . **(B)** A cycle in the auxiliary graph  $\widehat{G}_{\widetilde{M}}$ . The thick edges induce the schedule. **(C)** An induced schedule for the path in **(A)**. The unmatched job slice vertex in the path translates to a hole. **(D)** An induced schedule without holes for the cycle in **(B)**



**Claim 3.3** For any matching  $M$  in  $\widetilde{G}$ , the graph  $\widehat{G}_M$  is bipartite.

*Proof* Since vertices in  $\widehat{G}_M$  have degree at most 2, the graph is a union of disjoint paths and cycles. Each path is, clearly, bipartite. Consider a cycle  $\tau$  in  $\widehat{G}_M$  (see Fig. 3(B)). Since all vertices in the cycle  $\tau$  are incident to two edges in  $\widehat{G}_M$ , it follows that each vertex in  $\tau$  corresponds to a double job. Each edge  $(\hat{u}, \hat{v})$  in  $\tau$  corresponds either to an edge  $(u_a, v_a)$  in the matching  $M$  or to an edge  $(u_b, v_b)$  in  $M$  (since edges in  $\widetilde{E}$  do not connect slices of different types). Each coalesced vertex  $\hat{v}$  contains one type-*a* slice and one type-*b* slice. It follows that the edges of  $\tau$  alternate between type-*a* edges and type-*b* edges. Hence, the cycle has even length.  $\square$

*Scheduling algorithm*

1. Compute a maximum matching  $\widetilde{M}$  in  $\widetilde{G}$ .
2. Partition  $\widehat{G}_{\widetilde{M}}$  into paths and cycles.
3. For each path or cycle  $\tau$  in  $\widehat{G}_{\widetilde{M}}$ , obtain a schedule  $T_\tau$  of corresponding jobs as described in the proof of Claim 3.4 below.
4. Return the concatenation of the schedules  $\{T_\tau\}$ .

For any matching  $M$  in  $\widetilde{G}$  let  $\tau$  be a path or a cycle in  $\widehat{G}_M$ . Denote by  $J_\tau$  the set of jobs that correspond to vertices in  $\tau$ , and by  $M_\tau$  the set of edges in  $\tau$ .

**Claim 3.4** For any given matching  $M$  in  $\widetilde{G}$ , consider the auxiliary graph  $\widehat{G}_M$ . Each path or cycle  $\tau$  in  $\widehat{G}_M$  induces a schedule  $T_\tau$  of the jobs corresponding to the vertices in  $\tau$ , such that  $makespan(T_\tau) = p(J_\tau) - |M_\tau|$ .

*Proof* First consider a path  $\tau$  in  $\widehat{G}_M$ . Note that unit job vertices are incident to at most one edge in  $\widehat{G}_M$ . Hence, they may appear only as end vertices in the path  $\tau$ . Interior vertices in  $\tau$  correspond to double jobs. This implies that the path induces a schedule in which jobs alternate between the machines as depicted in Fig. 3(C). The only holes, if any, in this schedule are in the first and last round. A hole appears in the schedule if the path begins or ends with a double job of degree at most 1, or a unit job of degree 0. Hence, the makespan of the schedule is the number of job slices in  $\tau$  minus the number of pairs of matched slices (that are scheduled concurrently). Therefore,  $makespan(T_\tau) = p(J_\tau) - |M_\tau|$ , as claimed.

Next consider a cycle  $\tau$  in  $\widehat{G}_M$ . By Claim 3.3, the cycle  $\tau$  is of even length. Since there is an edge  $(\hat{u}, \hat{v})$  in  $\widehat{G}_M$  only if there is an edge between the corresponding double jobs  $u$  and  $v$  in the agreement graph  $\overline{G}$ , the type-*b* edges form a perfect matching in  $\overline{G}$  between the jobs that correspond to vertices on the cycle  $\tau$  (see Fig. 3(D)). And the claim follows.  $\square$

*Correctness* Optimality of the algorithm is derived from Lemmas 3.5 and 3.6, stated below.

**Lemma 3.5** Every matching  $M$  in the auxiliary graph  $\widetilde{G}$  induces a feasible schedule  $T_M$ , such that  $makespan(T_M) = p(J) - |M|$ .

Lemma 3.5 follows immediately from Claim 3.4.



**Lemma 3.6** Every schedule  $T$  induces a matching  $M_T$  in the auxiliary graph  $\tilde{G}$  with  $|M_T| \geq p(J) - \text{makespan}(T)$ .

Equality holds in Lemma 3.6 if there are no vacant rounds in  $T$ .

*Proof* We consider the rounds in  $T$  starting from the first round. We shall map each round with two concurrent job slices to an edge in  $\tilde{G}$ , so that the edges form a matching. The main issue will be in selecting the types ( $a$  or  $b$ ) for job slices of double jobs. There are three types of rounds with two concurrent job slices scheduled in them:

1. Unit jobs  $j$  and  $k$  are scheduled concurrently in the round. In this case there must be an edge in  $\tilde{G}$  between the corresponding vertices, and we select this edge.
2. Unit job  $j$  is scheduled concurrently with a job slice of a double job  $v$ . If this is the first slice of job  $v$ , or if this is the second slice of job  $v$  and the first one was scheduled opposite a hole, then we select the edge  $(j, v_a) \in \tilde{E}$ . Otherwise (it is the second slice and the first one was not scheduled opposite a hole), the previous round was mapped to an edge  $(x, v_a)$  (or  $(x, v_b)$ ), and we select  $(j, v_b)$  (or, respectively,  $(j, v_a)$ ).
3. A slice of a double job  $v$  is scheduled concurrently with a slice of another double job  $u$ . If they are both the first slice of their respective jobs, then we select  $(v_a, u_a)$ . If they are both the second slice of their respective jobs, then we select  $(v_b, u_b)$ . Otherwise, one of them is a second slice of its job, if its first slice was scheduled opposite to a hole we select  $(v_a, u_a)$ . Otherwise, the type selected for its first slice determines its type, so we select either  $(v_a, u_a)$  or  $(v_b, u_b)$  accordingly. □

**Theorem 2** The algorithm for scheduling jobs with processing times  $p_j \in \{1, 2\}$  computes a schedule with minimum makespan.

*Proof* Let  $\tilde{M}$  denote the maximum matching in the auxiliary graph found by the scheduling algorithm. Let  $T^*$  denote a schedule with minimum makespan, and let  $M_{T^*}$  denote the matching induced by  $T^*$  as defined in Lemma 3.6. Then

$$\begin{aligned} \text{makespan}(T_{\tilde{M}}) &= p(J) - |\tilde{M}| \quad (\text{by Lemma 3.5}) \\ &\leq p(J) - |M_{T^*}| \quad (\tilde{M} \text{ is maximum}) \\ &\leq \text{makespan}(T^*) \quad (\text{by Lemma 3.6}). \quad \square \end{aligned}$$

*On the relation to  $b$ -matching* Loosely speaking, a  $b$ -matching is a multi-set of edges  $F$ , where the degree of each vertex  $v$  with respect to  $F$  is at most  $b_v$  (an edge can be chosen more than once). The algorithm we showed above solves the problem of finding a maximum bipartite  $b$ -matching where  $b_j = p_j$ . This algorithm works only when  $p_j \in \{1, 2\}$ .

### 3.3 $\frac{4}{3}$ -approximation for processing times $p_j \in \{1, 2, 3\}$

We now give an algorithm for this case, with a  $4/3$ -approximation ratio. This result is a reduction to the previous optimal algorithm for  $p_j \in \{1, 2\}$ . A direct proof of this approximation ratio, which extends the construction from Sect. 3.2, can be found in Kaplan (2007).

Given an input to SWC consisting of a conflict graph  $G = (J, E)$  and processing times  $p_j$  for each  $j \in J$ , we define the  $q$ -squeezed instance of this input by the same conflict graph, but where job  $j$  is truncated at  $\min(q, p_j)$ . Consider the algorithm for  $m = 2$  and  $p_j \in \{1, 2, 3\}$  that finds an optimal solution of the 2-squeezed instance, and then “stretches” the schedule to fit the original triplet jobs. The process of “stretching” can be described in two stages. First, every even-numbered round that contains at least one squeezed job is duplicated. This causes truncated jobs to be stretched back to their original processing time of three, as desired, but it may also cause the duplication of unit jobs, and the unnecessary stretching of double jobs. In the second stage, we simply put a hole instead of each duplicated unit job and a hole instead of the first job slice of each stretched double job. This results in a valid schedule for the original instance.

**Theorem 3** Stretching the optimal schedule of the 2-squeezed instance yields a  $4/3$ -approximation for SWC for  $m = 2$  with  $p_j \in \{1, 2, 3\}$ .

*Proof* Let  $G = (J, E)$  be the original instance, and  $n_i$ ,  $i = 1, 2, 3$ , be the number of jobs with  $p_j = i$ . Recall that  $p(J) = n_1 + 2n_2 + 3n_3$  is the total length of jobs in  $G$ . For a given schedule, we say that a round is *busy* if jobs are scheduled on both machines in this round. Let  $d$  be the number of busy rounds in an optimal schedule for  $G$ . Let  $G_2 = (J_2, E)$  be the 2-squeezed instance and let  $d_2$  be the number of busy rounds in an optimal schedule for  $G_2$ .

The optimal schedule of  $G_2$  has makespan of  $p(J_2) - d_2 = n_1 + 2(n_2 + n_3) - d_2$ . In the algorithm’s schedule for  $G$ , each triple job is stretched, introducing a hole when not matched by another triple. Thus, the makespan of the algorithm is bounded by

$$|Alg| \leq p(J_2) - d_2 + n_3 = p(J) - d_2.$$

The optimal makespan of  $G$  always satisfies

$$|OPT| \geq p(J)/2.$$

Thus, if  $d_2 \geq p(J)/3$ , then the ratio  $|Alg|/|OPT|$  is at most  $4/3$ , as claimed. To deal with the case that  $d_2 < p(J)/3$  we prove the next claim.

**Claim 3.7**

$$d_2 \geq \frac{2}{3}d.$$

*Proof* Given a schedule  $OPT$  for the original instance with  $d$  busy rounds, we show how to form a schedule  $OPT_2$  for the squeezed instance with  $d_2 \geq \frac{2}{3}d$  busy rounds. Consider the first triple job  $u$  in  $OPT$ . All rounds preceding it are copied verbatim in  $OPT_2$ . We now argue how to handle the rounds involving  $u$ . If in at least one of these rounds  $u$  is scheduled opposite a hole, then we delete one such round and keep the other two. Otherwise, all these rounds are busy, and we consider two cases. In the first case, another triple job,  $v$ , is scheduled opposite  $u$  (where, by definition of  $u$ , the job  $v$  ends with or after  $u$ ). In this case, we delete the last round in which  $u$  is scheduled (so that both  $u$  and  $v$  are squeezed). In the second case, the jobs scheduled opposite  $u$  are either unit jobs or double jobs. In this case, we keep all three rounds, but execute  $u$  only for two of the rounds (thus creating a single hole). In the cases above, there are two busy rounds involving  $u$  in  $OPT_2$  if the corresponding number in  $OPT$  is three, and otherwise the numbers are identical. By induction, we then transform the part of  $OPT$  involving rounds following  $u$ , to complete the claim.  $\square$

The optimal makespan of  $G$ ,  $|OPT|$ , can be written as  $p(J) - d$ . With Claim 3.7 we have that

$$|OPT| = p(J) - d \geq p(J) - 3d_2/2.$$

It follows that the ratio  $|Alg|/|OPT|$  is at most  $1 + (d_2/2)/(p(J) - 3d_2/2)$ , which is at most  $4/3$  when  $d_2 \leq p(J)/3$ . Hence, in either case we get a performance ratio of  $4/3$ .  $\square$

**4 Results for the online model**

In this section, we consider online scheduling of jobs with release times. We begin with a trivial 2-competitive online algorithm for two machines and continue with the following results: (i) an exponential time 2-competitive online algorithm for  $m \geq 3$  machines and a polynomial time  $(\rho + O(1))$ -competitive online algorithm that is based a  $\rho$ -approximation algorithm for the offline problem, (ii) an  $\frac{m+1}{2}$ -competitive algorithm for unit jobs on  $m \geq 3$  machines, and  $\frac{m+2}{2}$ -competitive for arbitrary processing times, (iii) a lower bound of 2 for the competitive ratio of the greedy algorithm on two machines that uses either a FIFO or LIFO policy, and (iv) a lower bound of  $2 - 1/m$  for the competitive ratio of any deterministic online scheduling algorithm for  $m \geq 2$  machines (this lower bound holds even for unit jobs). In the next section, we give an efficient algorithm for two machines and unit jobs that has a competitive ratio better than 2.

*The online model* Recall that the release time of job  $j$  is denoted by  $r_j$ . This means that job  $j$  can be scheduled in any round  $t \geq r_j$ . We assume that job  $j$  arrives at the beginning of round  $r_j$ . We refer to jobs that have been released but not scheduled as *pending*. An algorithm is *nonlazy* if it assigns at least one job in a round whenever there are pending jobs. Call an (online) algorithm *2-nonlazy* if a job  $j$  is scheduled as a singleton only when no job agreeing with  $j$  is pending.

**4.1 A 2-competitive algorithm for two machines**

Consider the trivial algorithm that schedules all jobs on a single machine. Namely, at each round, if the set of pending jobs is not empty, the algorithm schedules only one of the pending jobs on the first machine. The following lemma is well known and easy to verify.

**Lemma 4.1** *The single machine scheduler is 2-competitive.*

Lemma 4.1 easily extends to  $m$  machines as follows.

**Lemma 4.2** *The competitive ratio of any nonlazy algorithm on  $m$  machines is at most  $m$ .*

**4.2 A 2-competitive exponential time online algorithm for  $m \geq 3$  machines**

For a conflict graph  $G = (J, E)$  and a subset of jobs  $J' \subseteq J$ , we use  $OPT(J')$  to denote an optimal schedule for the subgraph of  $G$  induced by  $J'$ . We use  $|OPT(J')|$  to denote the makespan of this schedule, where we measure the number of rounds that elapse from the first release time of a job in  $J'$  till the last round in the schedule. The proof of the following theorem appears in Shmoys et al. (1995) and we include it here for completeness.

**Theorem 4** *There is an online algorithm for arbitrary processing times with competitive ratio 2.*

*Proof* Consider the following algorithm that proceeds in phases each consisting of one or more rounds. Let  $f_i$  be the last round of phase  $i$ , and thus  $f_{i-1} + 1$  is the first round of phase  $i$ . In phase  $i$ , the algorithm computes an optimal offline schedule of the pending jobs and then executes that schedule. The next phase starts when the last scheduled job finishes. Optimality of the offline algorithm in each phase implies that the processing time in the phase satisfies

$$f_i - f_{i-1} \leq |OPT(J_i)|,$$

where  $J_i$  is the set of jobs processed in phase  $i$ . Let  $\ell$  be the last phase. Since there were jobs pending at the start of

phase  $\ell$ , they were released after the start of phase  $\ell - 1$ . Thus, the makespan of the optimal schedule is at least

$$|OPT| \geq f_{\ell-2} + |OPT(J_\ell)|.$$

Then, the makespan of the algorithm equals  $f_\ell$  which is bounded by

$$\begin{aligned} f_\ell &= (f_\ell - f_{\ell-1}) + (f_{\ell-1} - f_{\ell-2}) + f_{\ell-2} \\ &= |OPT(J_\ell)| + |OPT(J_{\ell-1})| + f_{\ell-2} \\ &\leq |OPT| + |OPT(J_{\ell-1})| \\ &\leq 2 \cdot |OPT|. \quad \square \end{aligned}$$

4.3 A  $(\rho + O(1))$ -competitive polynomial time online algorithm for  $m \geq 3$  machines

Suppose we have an offline algorithm that is a  $\rho$ -approximation algorithm for the makespan. We next describe an online algorithm that is a variant of the algorithm described in Theorem 4. In the exponential-time algorithm, all pending jobs in the beginning of a phase are scheduled during the phase. In fact, the end of the phase  $i$  is marked by scheduling all the jobs that were pending in the beginning of the phase. The polynomial algorithm is different in two ways. First, it does not compute an optimal schedule but a  $\rho$ -approximate solution for the pending jobs. Let  $\Pi_i$  denote the  $\rho$ -approximate schedule computed in the beginning of phase  $i$ . Second, the end of a phase depends on the arrival of new jobs during  $\Pi_i$ . If no new jobs arrive, then  $\Pi_i$  is fully executed. However, if a new job arrives in round  $t$ , then the algorithm only schedules the prefix of  $\Pi$  consisting of jobs whose processing begins before round  $t$ . Note that in such a case, the  $i$ th phase ends no later than round  $t + \max_j \{p_j\} - 1$ .

The rationale behind not completing the schedule  $\Pi_i$  is that the analysis does not benefit from jobs that are processed before round  $r^{\max}$ . In fact, had the algorithm known in advance the number of jobs or  $r^{\max}$ , it could simply remain idle until round  $r^{\max}$ . In this case, the offline algorithm yields a  $(\rho + 1)$ -competitive ratio.

However, the algorithm is oblivious (i.e., does not know  $n$  or  $r^{\max}$  in advance). Consider the execution of the algorithm at the start of round  $r^{\max} = \max_j \{r_j\}$ . The current phase ends after at most  $\max_j \{p_j\} - 1$  additional rounds. Since all jobs have arrived by round  $r^{\max}$ , the last phase takes at most  $\rho \cdot |OPT|$  rounds. The makespan of the online algorithm is at most  $\max_j \{r_j\} + \max_j \{p_j\} - 1 + \rho \cdot |OPT|$ . For arbitrary jobs sizes this is at most  $(\rho + 2) \cdot |OPT|$  and for unit jobs it is at most  $(\rho + 1) \cdot |OPT|$ .

In particular, suppose we use as the offline algorithm the  $m$ -set cover approximation algorithm for unit jobs based on Duh and Fürer (1997) for which  $\rho = \mathcal{H}_m - 1/2$ . We then get an online algorithm with a competitive ratio of  $\mathcal{H}_m + \frac{1}{2}$ .

By using the greedy offline algorithm of Garey and Graham (1975), which has an approximation ratio of  $(m + 1)/2$ , we get a competitive ratio of  $(m + 3)/2$  for unit jobs and  $(m + 5)/2$  for arbitrary processing times. In the next two subsections, we present better bounds for online greedy algorithms that do not recompute approximate schedules in phases.

4.4 An  $\frac{m+1}{2}$ -competitive algorithm for unit jobs on  $m \geq 3$  machines

In Sect. 4.2, we described an online algorithm for arbitrary processing times on any number of machines that has a competitive ratio of 2, where the running time of the algorithm is exponential in  $m$ . Here we show that it is possible to get the same competitive ratio for unit jobs on three machines, using a simple greedy algorithm. As we show in the next section, this result is tight when the algorithm breaks ties using either a FIFO or a LIFO policy.<sup>3</sup>

We first need the following refinement of the greedy list scheduling result of Garey and Graham (1975). The analysis of this algorithm only the fact that the algorithm is 2-nonlazy, and therefore by (29) (in the Appendix) the following fact holds.

**Fact 4.3** (Refinement of Garey and Graham 1975) *The makespan of any (offline) 2-nonlazy scheduling algorithm is bounded by  $\frac{1}{2} \cdot (p(J) + |OPT|)$ .*

**Theorem 5** *The competitive ratio of any 2-nonlazy algorithm for unit jobs on  $m \geq 3$  machines is at most  $\frac{m+1}{2}$ .*

*Proof* A singleton round is one where the algorithm schedules a job as a singleton. Observe that pending jobs in singleton rounds must form an independent set in the agreement graph. Let  $r$  be the last singleton round before  $|OPT| + 1$ , or 0 if there is no such round, and let  $X$  be the set of pending jobs at the start of round  $r$ . The first observation is that the optimal schedule must have at least  $\max(|X| - r, 0)$  pending jobs from  $X$  at the start of round  $r + 1$  because it can schedule at most a single job from  $X$  in each of the rounds  $1, \dots, r$ . The second observation is that the optimal solution can schedule at most  $m(|OPT| - r)$  jobs during rounds  $r + 1, \dots, |OPT|$  (recall that  $r \leq |OPT|$ ). Since at least  $\max(|X| - r, 0)$  of these jobs must have been released by round  $r$ , at most  $m(|OPT| - r) - \max\{|X| - r, 0\}$  jobs can be released between round  $r + 1$  and round  $|OPT|$ . By definition of  $r$ , the algorithm schedules at least two jobs in

<sup>3</sup>The lower bound in the next subsection is stated for two machines, but holds for any number of machines  $m \geq 2$ , since even if there are more than two machines, the algorithm is forced to use a single machine while the optimal schedule uses two machines.

each of these rounds. Hence, at the end of round  $|OPT|$ , the number of pending jobs left for the algorithm to schedule is at most

$$(|X| - 1) + (m(|OPT| - r) - \max\{|X| - r, 0\}) - 2(|OPT| - r).$$

If  $m \geq 3$ , then the expression above is less than  $(m - 2)|OPT|$ . From round  $|OPT| + 1$  the algorithm behaves like an offline 2-nonlazy algorithm, scheduling these jobs using at most  $((m - 2)|OPT| + |OPT|)/2 = (m - 1)|OPT|/2$  additional rounds (by Fact 4.3).  $\square$

In particular, we can achieve a competitive ratio of 2 for three machines even in the case of *resource reduction*, i.e., when the algorithm gets to use only two machines.

#### 4.5 An $\frac{m+2}{2}$ -competitive algorithm for arbitrary $p_j$ on $m \geq 3$ machines

We also show a competitive ratio of  $(m + 2)/2$  for arbitrary processing times.

**Theorem 6** *The competitive ratio of any 2-nonlazy algorithm for arbitrary processing times is at most  $(m + 2)/2$ .*

*Proof* Let  $r^{\max}$  be the release time of the last job. Let  $J'$  be the subset of jobs induced by the pending and active jobs at that time, counting only the unfinished rounds in the length of the active jobs. In other words, if a job  $j$  is of length  $p_j$  and has been executed for  $\ell_j$  rounds by round  $r^{\max}$ , then its length in  $J'$  will be  $p_j - \ell_j$ . Observe that  $p(J') \leq p(J) - r^{\max}$ , as the algorithm schedules at least one job per round, since it is nonlazy. By round  $r^{\max}$ , the algorithm behaves like an offline 2-nonlazy algorithm, producing a schedule with makespan of at most  $(p(J') + OPT(J'))/2$ , by Fact 4.3. Hence, the makespan of the algorithm's schedule is at most

$$\begin{aligned} |Alg| &\leq r^{\max} + (p(J') + OPT(J'))/2 \\ &\leq (p(J) + r^{\max} + OPT)/2 \\ &\leq \frac{(m + 2)}{2} \cdot OPT, \end{aligned}$$

where the last line follows from  $\frac{p(J)}{m} \leq OPT$  and  $r^{\max} \leq OPT$ .  $\square$

#### 4.6 A lower bound of 2 for the competitive ratio of greedy

Recall that in the offline case (that is, when all jobs have release time  $r_j = 0$ ), a simple greedy algorithm achieves a nontrivial approximation ratio of  $\frac{m+1}{2}$  (as opposed to the trivial factor  $m$  achieved by scheduling all jobs on a single machine). In particular, for the case of two machines, its

approximation ratio is  $\frac{3}{2}$ . Here we show that in the online case of unit jobs with arbitrary release times, the natural online greedy algorithm for two machines performs no better (in the worst case) than the trivial single machine algorithm. Specifically, the greedy algorithm for unit jobs that we consider proceeds as follows.

Let  $J_t$  denote the set of pending jobs in the beginning of round  $t$ . If  $J_t$  contains a pair of nonconflicting jobs, then the greedy algorithm schedules such a pair. Otherwise, if  $J_t$  is not empty, it schedules a single job. We refer to this version of the greedy algorithm as “Greedy-FIFO” since, in the case that a single job is scheduled, it selects a pending job with the minimum release time. The algorithm may choose any pair of nonconflicting pending jobs, if more than one such pair exists. The lower bound does not depend on how such pairs are selected since the adversary may force Greedy-FIFO to schedule at most one job in each round (i.e., the set  $J_t$  of pending jobs will always be an independent set in the agreement graph).

We now prove that the competitive ratio of Greedy-FIFO is 2 (it is not greater than 2 because Greedy-FIFO is never worse than the trivial single machine scheduler). Note that this lower bound holds even with respect to unit jobs.

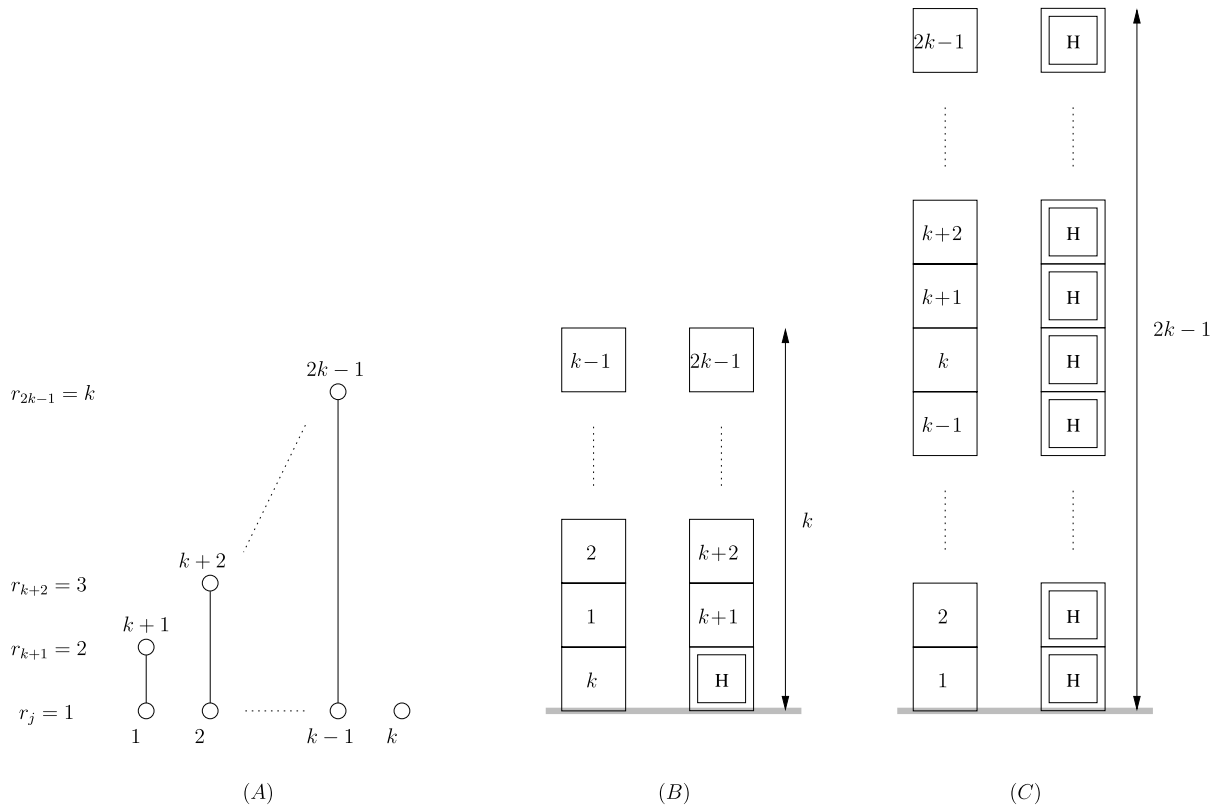
**Lemma 4.4** *The competitive ratio of Greedy-FIFO is 2.*

*Proof* We construct an agreement graph  $\overline{G} = (J, \overline{E})$  over  $2k - 1$  unit jobs, where  $k$  is an arbitrary positive integer, as follows. The graph  $\overline{G}$  is a set of  $k - 1$  disjoint edges, and one isolated vertex. The adaptive adversary maintains the invariant that in each round the set of pending jobs for Greedy-FIFO forms an independent set in  $\overline{G}$ , and therefore Greedy-FIFO can only schedule one job in each round.

At the first round, the adversary releases  $k$  jobs. These jobs are an independent set in  $\overline{G}$  (one endpoint of each edge, plus the isolated vertex). In each of the  $k - 1$  first rounds Greedy-FIFO chooses one of these jobs and schedules it. This follows from the fact that Greedy-FIFO follows a FIFO policy and from the invariant that the set of pending jobs is an independent set in  $\overline{G}$ . Let  $j$  be the job Greedy-FIFO scheduled in round  $i$ , then in round  $i + 1$  the adversary releases a job  $j'$  (so that  $r_{j'} = i + 1$ ) with an edge  $(j, j') \in \overline{E}$ . (The jobs, the agreement edges, and the release times are depicted in Fig. 4(A)).

An optimal schedule will first schedule the isolated job in round 1. In each of the following rounds, it will schedule pairs of agreeing jobs. Jobs  $j$  and  $j'$  will be scheduled in round  $i$ , if  $r_{j'} = i$  and  $(j, j') \in \overline{E}$  (see Fig. 4(B)).

The adversary forces Greedy-FIFO to schedule the first  $k$  jobs as singletons in the first  $k$  rounds. At the beginning of round  $k + 1$ , the set of pending jobs is an independent set (each of the pending jobs is an endpoint of an edge whose other endpoint was already scheduled). Hence,



**Fig. 4** (A) The agreement graph  $\overline{G}$  for the case that Greedy-FIFO schedules job  $j$  in round  $j$  for  $j \in \{1, \dots, k\}$ . The number next to each vertex is the job’s name. The vertical position of the vertex shows the

release time  $r_j$  of job  $j$ . (B) An optimal schedule. The numbers inside represent the jobs that are scheduled at that time slot. (C) Greedy-FIFO’s schedule that uses a single machine

Greedy-FIFO is forced to schedule each of these jobs in a different round as well. The makespan of Greedy-FIFO is therefore  $2k - 1$  (see Fig. 4(C)).

Note that a greedy algorithm with a LIFO policy does not do any better. The adversary can schedule jobs as depicted in Fig. 5. Again, the algorithm schedules only one job per round, while the optimal algorithm can schedule pairs of jobs in all the rounds but one.  $\square$

#### 4.7 A general lower bound of $2 - \frac{1}{m}$ for the competitive ratio

In this section we prove the following theorem.

**Theorem 7** *Let Alg be an online algorithm that schedules jobs with release times on  $m$  machines. Then the competitive ratio for the makespan of algorithm Alg is at least  $2 - \frac{1}{m}$ . In fact, this lower bound holds for unit jobs.*

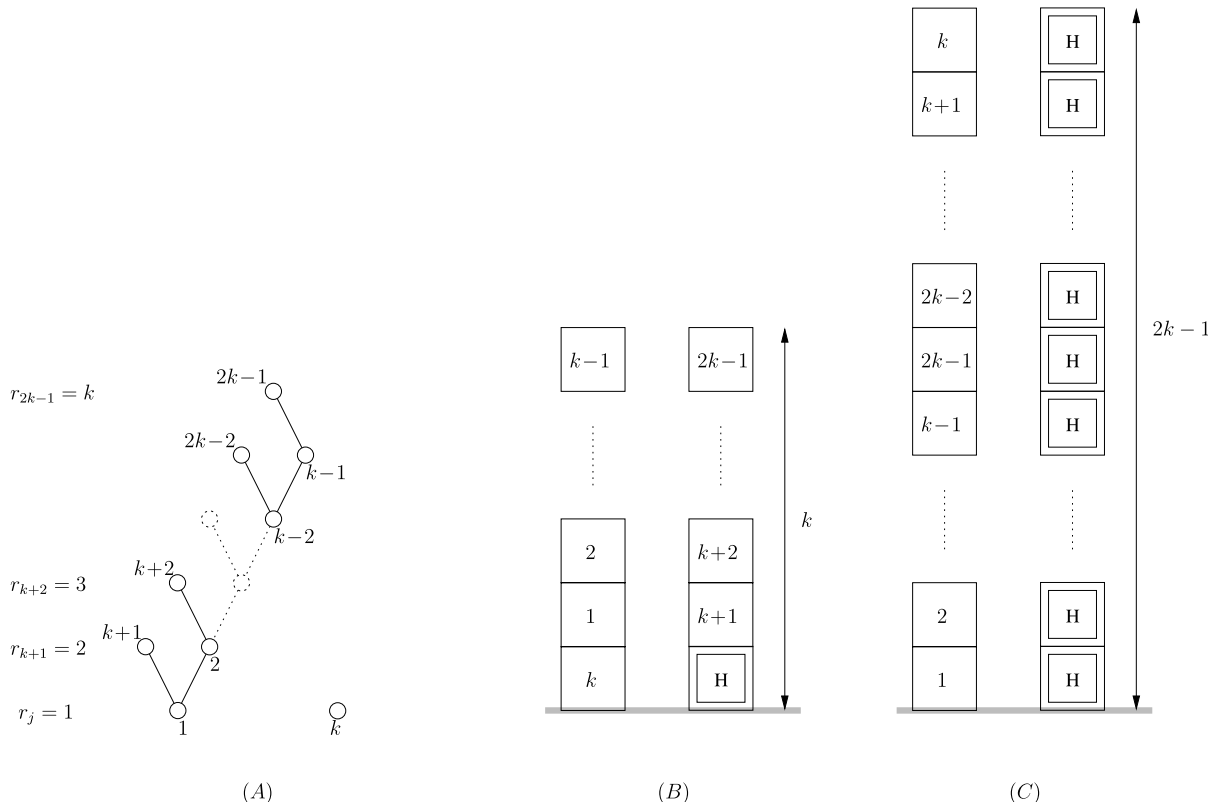
We first show that the makespan is not increased by imposing the restriction that the algorithm is nonlazy. We therefore assume without loss of generality, that the online algorithm is nonlazy.

**Lemma 4.5** *Any online algorithm  $Alg_1$  for unit jobs can be transformed into a nonlazy online algorithm  $Alg_2$  such that  $makespan(Alg_2) \leq makespan(Alg_1)$ .*

*Proof* The algorithm  $Alg_2$  mimics  $Alg_1$  with the restriction that  $Alg_2$  always tries to fill vacant slots, if possible, with a pending job. It is easy to show that, in each round, the set of pending jobs with respect to  $Alg_2$  is a subset of the set of pending jobs with respect to  $Alg_1$ . Therefore,  $makespan(Alg_2) \leq makespan(Alg_1)$ .  $\square$

We present a sequence  $\{\sigma_k\}_k$  of adversarial strategies with the following property. All released jobs are unit jobs. In addition, the makespan of Alg when the jobs arrive according to strategy  $\sigma_k$  is  $(2m - 1) \cdot k$ , while the optimal makespan is  $m \cdot k$ . The ratio for  $\sigma_k$  is  $2 - \frac{1}{m}$ , as required.

*The adversarial strategy  $\sigma_k$*  The adversary constructs an agreement graph  $\overline{G} = (V, \overline{E})$ , where  $V = \{1, \dots, (2m - 1)k\}$ . The graph  $\overline{G}$  is a union of disjoint cliques. Figure 6 depicts an agreement graph where each clique is depicted as column and only part of the edges are drawn;  $\overline{E}$  is the transitive closure of the depicted edges. The number of cliques (columns) is  $k' = (m - 1) \cdot k + 1$ .



**Fig. 5** (A) The agreement graph  $\bar{G}$ . The number next to each vertex is the job's name. The vertical position of the vertex shows the release time  $r_j$  of job  $j$ . (B) An optimal schedule. The numbers inside repre-

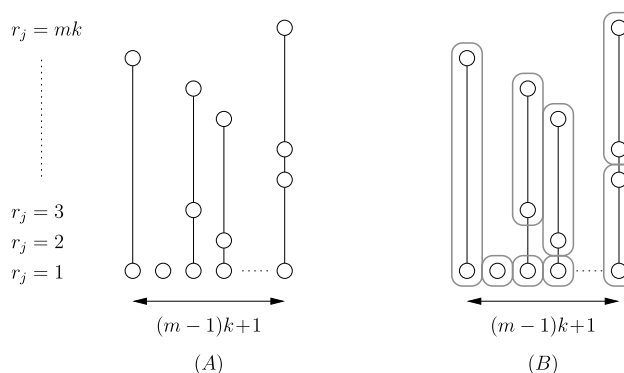
sent the jobs that are scheduled at that time slot. (C) LIFO's schedule that uses a single machine

The adversary, which is adaptive, keeps the invariant that, in each round, the set of pending jobs forms an independent set in  $\bar{G}$ . Therefore,  $Alg$  can only schedule one job per round (which it does since  $Alg$  is nonlazy). This invariant is easily satisfied as follows.

The adversary releases  $k'$  jobs in the beginning of round 1 and one job in the beginning of rounds 2 through  $m \cdot k$ . The  $k'$  jobs released in round 1 are conflicting and therefore constitute an independent set in  $\bar{G}$  (each job belongs to a different column). In each round  $t$  between 2 and  $m \cdot k$ , the released job belongs to the same column as the job that  $Alg$  scheduled in round  $t - 1$ .

Since the invariant that the pending jobs are conflicting is achieved, the makespan of  $Alg$  equals the number of jobs, i.e.,  $(2m - 1) \cdot k$ .

*An offline algorithm* We next show that there exists an offline algorithm  $A_{off}$  that, given the agreement graph, finds a schedule with a makespan of  $m \cdot k$ . The algorithm  $A_{off}$  partitions each column (clique) in  $\bar{G}$  into  $m$ -tuples, where only the first tuple may contain fewer than  $m$  jobs, as follows. Sort the jobs in each column by release time (in ascending order). Let  $r$  denote the remainder when dividing the number of jobs in the column by  $m$ . If  $r > 0$ , then the first tuple



**Fig. 6** (A) An agreement graph  $\bar{G} = (J, \bar{E})$  that the (adaptive) adversary generates. Each clique is depicted as a column and only part of the edges are drawn;  $\bar{E}$  is the transitive closure of the depicted edges. The vertical position of the vertex shows the release time  $r_j$  of job  $j$ . (B) The offline algorithm's choice of sub-cliques for  $m = 2$

consists of the first  $r$  jobs in the column. If  $r = 0$ , then the first tuple consists of the first  $m$  jobs in the column. Each of the remaining tuples consist of  $m$  consecutive jobs.

By the construction of the tuples we have the following property.

**Claim 4.6** *If a tuple is a singleton  $\{j\}$ , then  $r_j = 1$ .*

For a tuple  $\alpha$ , let  $t_\alpha \triangleq \max\{r_j \mid j \in \alpha\}$ . The following claim shows that non-singleton tuples have different maximum release times.

**Claim 4.7** *For every two distinct non-singleton tuples  $\alpha$  and  $\beta$ , it holds that  $t_\alpha \neq t_\beta$ .*

*Proof* Since  $\alpha$  and  $\beta$  contain more than one job, both  $t_\alpha$  and  $t_\beta$  are greater than 1. However, in all rounds greater than 1, only one job is released. Hence,  $t_\alpha \neq t_\beta$ , as required.  $\square$

The rule for scheduling tuples is that each tuple  $\alpha$  that contains at least two jobs is scheduled in round  $t_\alpha$ , using  $|\alpha|$  machines out of  $m$ . Claim 4.7 implies that each non-singleton tuple is scheduled in a distinct round. Thus, the last non-singleton tuple is scheduled in round  $m \cdot k$ . By Claim 4.6, the singletons are released in round 1, and are available for scheduling in all rounds. The rule for scheduling singletons is that whenever a round is not reserved for a non-singleton,  $A_{off}$  schedules a singleton.

Let  $t$  denote the number of tuples. Since singletons are available from the beginning, the last singleton will be scheduled before or during round  $t$ . To show that the makespan of  $A_{off}$  is  $m \cdot k$ , the minimum possible, it then suffices to show that  $t$  is at most  $m \cdot k$ . We shall count two types of tuples.

A tuple is *nonfull* if it contains less than  $m$  jobs, and *full* otherwise. Let  $t_{nonfull}$  ( $t_{full}$ ) be the number of nonfull (full) tuples. In each column there is at most one nonfull tuple. Hence,

$$t_{nonfull} \leq k' = (m - 1) \cdot k + 1. \tag{4}$$

Since the number of jobs in nonfull tuples is at least  $t_{nonfull}$ , we have

$$|V| \geq m \cdot t_{full} + t_{nonfull}.$$

Since  $|V| = (2m - 1) \cdot k$ , we obtain

$$t_{full} \leq \frac{1}{m} \cdot ((2m - 1) \cdot k - t_{nonfull}). \tag{5}$$

Hence, we bound the number of tuples by

$$\begin{aligned} t &= t_{full} + t_{nonfull} \\ &\leq \frac{(2m - 1) \cdot k}{m} + \frac{m - 1}{m} \cdot t_{nonfull} \quad (\text{by (5)}) \\ &\leq mk + 1 - \frac{1}{m} \quad (\text{by (4)}). \end{aligned}$$

Since  $t$  is an integer, it follows that it is bounded by  $m \cdot k$ , as required.

*Proof of Theorem 7* The makespan of  $Alg$  equals the number of jobs, namely,  $(2m - 1) \cdot k$ . By the preceding discussion, the makespan of  $A_{off}$  equals  $m \cdot k$ . The ratio is  $2 - \frac{1}{m}$ , and the theorem follows.  $\square$

### 5 An online algorithm ( $p_j = 1, r_j, m = 2$ )

The goal in this section is to present an online algorithm for scheduling unit jobs on two machines that achieves a competitive ratio  $2 - \alpha$  for a constant  $0 < \alpha < 1$ . We first assume that the number of jobs,  $n$ , is known in advance. This assumption is often called the *non-oblivious model*. We later trade this assumption with a single flip of a coin.

In order to give the intuition for the algorithm we present, we recall the worst case inputs in Sect. 4.6 for Greedy-FIFO and Greedy-LIFO. Greedy-FIFO performs badly on inputs that release a large fraction of jobs in the beginning, while Greedy-LIFO performs badly on inputs that release the jobs evenly over time. However, Greedy-LIFO does well with the worst case example for Greedy-FIFO, and vice versa. Based on this intuition the online algorithm splits into two cases depending on the fraction of the jobs that are released during the first  $\alpha n$  rounds (for a constant  $\alpha < 1$  that is optimized in the analysis). If a majority of the jobs is released during the first  $\alpha n$  rounds, then a variant of Greedy-LIFO is used; otherwise, a variant of Greedy-FIFO is used.

#### 5.1 Preliminaries

In this section we present a few assumptions that can be made without loss of generality.

Given an input graph  $G = (J, E)$  and release times  $\{r_j\}_{j \in J}$ , let  $OPT$  denote an optimal schedule for the input and let  $|OPT|$  denote the makespan of  $OPT$ . Let  $Alg$  denote an online algorithm, and let  $|Alg|$  denote its makespan on the input.

Consider a subset  $J' \subseteq J$  of the jobs. Denote by  $|OPT(J')|$  the optimal makespan for  $J'$  where the makespan is measured starting from round  $\min\{r_j \mid j \in J'\}$ .

The following claim shows that in the case of unit jobs, it suffices to deal with inputs for which  $Alg$  schedules a job in every round in the interval  $[1, |Alg|]$ .

**Claim 5.1** *Suppose that the competitive ratio of a non-oblivious scheduling algorithm  $Alg$  is bounded by  $2 - \alpha$  whenever  $Alg$  schedules a job in every round in the interval  $[1, |Alg|]$ . Then, there exists a non-oblivious scheduling algorithm  $Alg\text{-Res}$  whose competitive ratio is bounded by  $2 - \alpha$  for all inputs.*

*Proof* Consider the algorithm  $Alg\text{-Res}$  obtained from  $Alg$  by resetting  $Alg$  every time its set of pending jobs becomes

empty. Resetting  $Alg$  means that it restarts with a parameter (indicating the number of jobs) that equals  $n$  minus the number of jobs that have already been scheduled. Hence,  $Alg-Res$  is, in effect, a sequence of executions of  $Alg$  separated by vacant rounds.

If the execution of  $Alg$  lacks vacant rounds, then the execution of  $Alg-Res$  is identical to that of  $Alg$ , and the claim holds. Otherwise, consider the last vacant round  $\ell$  in the execution of  $Alg-Res$  over the remaining jobs  $\widehat{J} \triangleq \{j \mid r_j > \ell\}$ . Let  $|Alg-Res(\widehat{J})|$  be the makespan of  $Alg-Res$  on input  $\widehat{J}$ , where the makespan is measured starting from round  $\ell + 1$ . Note that, by definition, there are no vacant slots in the schedule  $Alg-Res(\widehat{J}) = Alg(\widehat{J})$  during rounds  $[\ell + 1, \ell + |Alg-Res(\widehat{J})|]$ . It follows that

$$|Alg-Res(J)| = \ell + |Alg-Res(\widehat{J})|,$$

$$|OPT(J)| = \ell + |OPT(\widehat{J})|.$$

By the premise of the claim,

$$|Alg(\widehat{J})| \leq (2 - \alpha) \cdot |OPT(\widehat{J})| + c$$

for some constant  $c$ . Therefore,

$$\begin{aligned} |Alg-Res(J)| &= \ell + |Alg-Res(\widehat{J})| \\ &= \ell + |Alg(\widehat{J})| \\ &\leq \ell + (2 - \alpha) \cdot |OPT(\widehat{J})| + c \\ &= \ell + (2 - \alpha) \cdot (|OPT(J)| - \ell) + c \\ &\leq (2 - \alpha) \cdot |OPT(J)| + c. \quad \square \end{aligned}$$

Let  $d$  denote the number of pairs scheduled by  $Alg$ . The following claim is analogous to Observation 3.1.

**Claim 5.2** *If at least one job is scheduled in every round till  $|Alg|$ , then  $|Alg| = n - d$ .*

*Proof* Let  $s$  denote the number of rounds in which a single job is scheduled. The makespan equals  $s + d = (n - 2d) + d = n - d$ .  $\square$

We say that pairs are *given precedence* over singletons if a pending singleton is scheduled only when there are no pending pairs. We say that pairs are scheduled according to a *FIFO policy* if, for every two pairs  $(j_1, j_2)$  and  $(j_3, j_4)$ , if the pair  $(j_1, j_2)$  is scheduled before the pair  $(j_3, j_4)$ , then  $\max\{r_{j_1}, r_{j_2}\} \leq \max\{r_{j_3}, r_{j_4}\}$ .

**Claim 5.3** *Without loss of generality,  $OPT$  gives precedence to pairs over singletons and schedules pairs according to a FIFO policy.*

*Proof* Simply reorder  $OPT$  so that it satisfies the assumptions. Such a reordering does not introduce violations in which a job is scheduled before its release time.  $\square$

## 5.2 An online algorithm

We describe an online algorithm in the non-oblivious model and analyze its competitive ratio. Let  $r^{\max} = \max\{r_j\}$ , namely,  $r^{\max}$  is the last round in which a job is released.

$Alg(n)$ : Non-oblivious online scheduling algorithm of  $n$  jobs on 2 machines

- 
- 1:  $\alpha \leftarrow \frac{\lfloor n/7 \rfloor}{n}$ .
  - 2: **for** round = 1 to  $\alpha n$  **do**
  - 3:   Schedule an agreeing pair or a singleton if one exists.
  - 4: **end for**
  - 5: Let  $J_1 \triangleq \{j \mid r_j \leq \alpha n\}$ .
  - 6: **if**  $|J_1| < \frac{n}{2}$  **then**
  - 7:   Starting from round  $\alpha n + 1$  till round  $r^{\max} - 1$ , use any nonlazy scheduler for the remaining jobs giving precedence to pairs over singletons, and scheduling singletons according to a FIFO policy.
  - 8:   Starting from round  $r^{\max}$ , compute an optimal schedule of the pending jobs (see Sect. 3).
  - 9: **else**
  - 10:   Compute a maximum matching  $M$  in the subgraph of the agreement graph induced by the jobs in  $J_1$  that are still pending.
  - 11:   Schedule all pairs of jobs in  $M$  during rounds  $[\alpha n + 1, \alpha n + |M|]$ .
  - 12:   Starting from round  $\alpha n + |M| + 1$ , use any nonlazy schedule that: (i) gives precedence to pairs over singletons; (ii) between pairs, precedence is given to pairs that contain a job in  $J_1$ ; and (iii) between singletons, precedence is given to jobs not in  $J_1$  (i.e., LIFO policy).
  - 13: **end if**
- 

The following theorem bounds the competitive ratio of  $Alg$ .

**Theorem 8** *The competitive ratio of  $Alg$  is at most  $2 - \alpha = 2 - 1/7$ .*

The proof of Theorem 8 is divided into two lemmas (Lemmas 5.4 and 5.7) according to the two cases of the algorithm:  $|J_1| < \frac{n}{2}$  and  $|J_1| \geq \frac{n}{2}$ . We assume for simplicity that  $n$  is divisible by 7. We later discuss the general case.

*A high-level discussion of the proof* Before giving the full proof we discuss its high level idea. Consider first the case  $|J_1| < n/2$  (which is the easier of the two). Assume for simplicity that  $r^{\max} \geq n/2$  (it can be shown that otherwise the performance of the algorithm cannot be worse). Let  $J_2 = J \setminus J_1$  (that is,  $J_2$  is the set of jobs that arrive after round  $\alpha n$ ). Recall that until round  $r^{\max}$ , the algorithm follows a FIFO policy with respect to singletons, gives precedence to pairs over singletons, and between pairs it gives



precedence to pairs that contain a job from  $J_1$ . Intuitively, the algorithm tries to “get rid” of as many of the (at most  $n/2 - 1$ ) jobs in  $J_1$  by round  $r^{\max} (\geq n/2)$ . If the algorithm succeeds in getting rid of all but at most  $\alpha n/2$  of the jobs in  $J_1$  by round  $r^{\max}$ , then we get the desired bound of  $(2 - \alpha)$ . This is true since the remaining jobs, that all belong to  $J_2$ , can be scheduled in at most  $|OPT| - \alpha n$  rounds, and the algorithm computes an optimal schedule at the start of round  $r^{\max}$ . Otherwise, we can show that the algorithm must have scheduled at least  $\alpha n/2$  pairs of jobs from  $J_2$  before round  $r^{\max}$  (these pairs “delayed” the jobs from  $J_1$ ). But this implies that the makespan of the algorithm is at most  $n - \alpha n/2$ , while the makespan of the optimal schedule is at least  $n/2$ .

We now turn to the case  $|J_1| \geq n/2$ . The first main observation is that since the algorithm computes a maximum matching  $M$  of the pending jobs from  $J_1$  in round  $\alpha n + 1$ , after the pairs in this matching are scheduled, the remaining jobs from  $J_1$ , denoted  $X$ , are an independent set in the agreement graph. Intuitively, these jobs “wait” to be paired with jobs in  $J_2$ . Assume first that the rate of arrival of jobs in  $J_2$  is such that between rounds  $\alpha n + |M| + 1$  and  $r^{\max}$  the algorithm always has pending jobs from  $J_2$ . In such a case, given the policy of the algorithm, each job in  $X$  is either scheduled together with some job from  $J_2$ , or it is scheduled as a singleton in the very last rounds of the schedule, after all jobs have arrived. We can show that if relatively many jobs from  $X$  are scheduled as singletons, then the optimal schedule must schedule many singletons as well. On the other hand, if few are scheduled as singletons, then the algorithm schedules relatively many pairs (and we can complete the argument as explained for the case of  $|J_1| < n/2$ ). If we remove the assumption that between rounds  $\alpha n + |M| + 1$  and  $r^{\max}$  the algorithm always has pending jobs from  $J_2$ , then the algorithm is forced to schedule singletons from  $X$  before all jobs arrive. The analysis in this case is more complex, but we are still able to show that if the algorithm schedules many singletons from  $X$  then the optimal schedule must schedule many singletons as well.

### 5.3 Proof of Theorem 8 for $|J_1| < n/2$

In this section we prove the following lemma (that holds for every  $0 < \alpha < 1$ ).

**Lemma 5.4** *If  $|J_1| < \frac{n}{2}$ , then the competitive ratio of the algorithm is  $2 - \alpha$ .*

We shall actually analyze a slightly different “imaginary” algorithm for the case  $|J_1| < n/2$ . This algorithm gives precedence to pairs over singletons and schedules singletons according to a FIFO policy starting at round  $\alpha n + 1$  until round  $|OPT| - 1$  (rather than until round  $r^{\max} - 1$ ). Starting from round  $OPT$ , the algorithm computes an optimal schedule of the pending jobs. Since  $|OPT| \geq r^{\max}$ , the makespan

of this imaginary algorithm is no smaller than the makespan of our algorithm (for the case  $|J_1| < n/2$ ).

Recall that  $d$  is the number of pairs scheduled by the algorithm, and let  $d'$  be the number of those scheduled in rounds  $[1, |OPT| - 1]$ . Define  $J_2 \triangleq J \setminus J_1 = \{j \mid r_j > \alpha n\}$ . Let  $J'_1$  denote the subset of jobs from  $J_1$  that were scheduled by the algorithm in the interval  $[1, |OPT| - 1]$ . Let  $J''_1$  denote the subset of jobs from  $J_1$  that were scheduled by the algorithm starting in round  $|OPT|$ . We say that a pair of jobs is in  $(X, Y)$  if it is in the set  $X \times Y$ .

Assume first that the algorithm has pending jobs from  $J_1$  at round  $|OPT|$ , i.e.,  $|J''_1| > 0$ . Then, in every round in the interval  $[1, |OPT| - 1]$  either a job from  $J_1$  is scheduled (alone or with another job) or a pair in  $(J_2, J_2)$  is scheduled. Hence, if  $|J''_1| > 0$ , we can bound the number of jobs in  $J'_1$  by

$$|J'_1| \geq |OPT| - 1 - d'. \tag{6}$$

We now upper bound the number of jobs in  $J''_1$ .

**Claim 5.5**  $|J''_1| \leq d'$ .

*Proof* If  $J''_1 = \emptyset$ , then the claim is trivial. Otherwise, (6) holds and

$$\begin{aligned} |J''_1| &= |J_1| - |J'_1| \quad (\text{by definition}) \\ &\leq (n/2 - 1) - (|OPT| - 1 - d') \\ &\quad (\text{since } |J_1| < n/2 \text{ and by (6)}) \\ &\leq d' \quad (\text{by } |OPT| \geq n/2). \end{aligned} \quad \square$$

As a corollary of Claim 5.5 we obtain the following bound.

**Corollary 5.6**  $|Alg| \leq 2 \cdot |OPT| - 1 + d' - \alpha n$ .

*Proof* Recall that  $|OPT(J')|$  denotes the number of rounds in an optimal makespan of the set of jobs  $J' \subseteq J$ . By restricting  $OPT = OPT(J)$  to the jobs in  $J_2$ , it follows that

$$|OPT(J_2)| \leq |OPT(J)| - \alpha n = |OPT| - \alpha n. \tag{7}$$

The set of pending jobs in the beginning of round  $|OPT|$  is contained in  $J''_1 \cup J_2$ . Since the algorithm computes an optimal schedule of these pending jobs, the first line in the following equation follows:

$$\begin{aligned} |Alg| &\leq |OPT| - 1 + |J''_1| + |OPT(J_2)| \\ &\leq |OPT| - 1 + d' + |OPT| - \alpha n \\ &\quad (\text{by Claim 5.5 and (7)}) \\ &= 2 \cdot |OPT| - 1 + d' - \alpha n. \end{aligned} \quad \square$$

*Proof of Lemma 5.4* By Claim 5.2 and since  $d' \leq d$ , it follows that  $|Alg| \leq n - d'$ . By Corollary 5.6, we obtain  $|Alg| \leq \min\{n - d', 2 \cdot |OPT| + d' - \alpha n\}$ . Hence, the competitive ratio satisfies:

$$\begin{aligned} \frac{|Alg|}{|OPT|} &\leq \frac{\frac{1}{2}(n - d' + 2 \cdot |OPT| + d' - \alpha n)}{|OPT|} \\ &\quad (\text{min} \leq \text{average}) \\ &= 1 + \frac{(1 - \alpha) \cdot \frac{n}{2}}{|OPT|} \\ &\leq 2 - \alpha \quad (|OPT| \geq n/2). \quad \square \end{aligned}$$

5.4 Proof of Theorem 8 for  $|J_1| \geq n/2$

In this section we prove the following lemma (that holds for  $\alpha \leq 1/7$ ).

**Lemma 5.7** *If  $|J_1| \geq \frac{n}{2}$ , then the competitive ratio of the algorithm is  $2 - \alpha$ .*

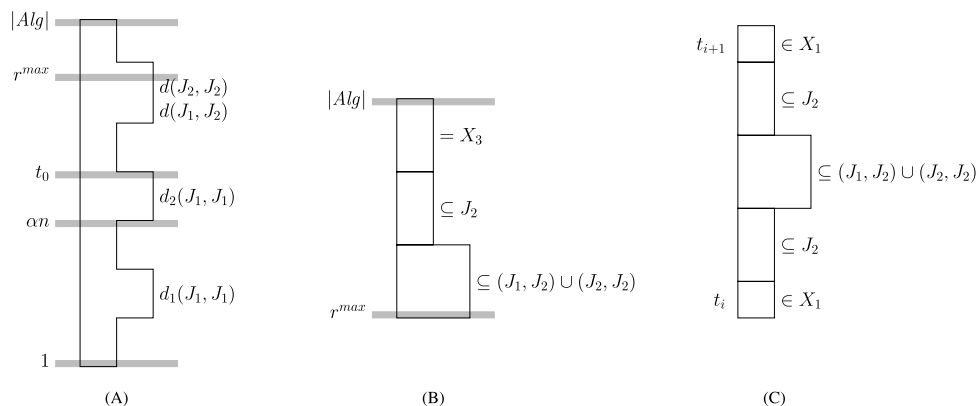
*Notation* The proof in this section uses the following notation. For a schematic illustration see Fig. 7. Let  $t_0 = \alpha n + |M|$  (this is the round in which the last pair of matching jobs in  $J_1$  is scheduled). Let  $X \subseteq J_1$  denote the set of pending jobs in the beginning of round  $t_0 + 1$  (note that every two jobs in  $X$  conflict). We partition  $X$  into three subsets  $X = X_1 \cup X_2 \cup X_3$ , as follows:  $X_1$  consists of jobs in  $X$  that are scheduled as singletons before round  $r^{\max}$ ;  $X_2$  consists

of jobs in  $X$  that are scheduled in pairs from  $(X, J_2)$ ;  $X_3$  consists of jobs in  $X$  that are scheduled as singletons starting from round  $r^{\max}$ . Let  $x = |X|$  and  $x_i = |X_i|$ .

For an interval of rounds  $[t_1, t_2]$ , let  $J_{[t_1, t_2]}$  denote the subset of jobs that arrive in this interval, that is,  $J_{[t_1, t_2]} = \{j \mid t_1 \leq r_j \leq t_2\}$ . In particular,  $J_1 = J_{[1, \alpha n]}$  and  $J_2 = J_{[\alpha n + 1, r^{\max}]}$ . Let  $J_{[t_1, t_2]}^{Alg}$  denote the subset of jobs that are scheduled by  $Alg$  in the interval  $[t_1, t_2]$ . Similarly, for a round  $t$ , denote by  $J_{\leq t}^{Alg}$  and  $J_{> t}^{Alg}$  the jobs scheduled by  $Alg$  up till round  $t$  and after round  $t$ , respectively. For subsets  $Z, Y \subseteq J$ , let  $d(Z, Y)$  denote the number of pairs in  $(Z, Y)$  that are scheduled by  $Alg$  throughout its execution. Recall that  $d$  denotes the total number of pairs scheduled by  $Alg$  (and so  $d = d(J_1, J_1) + d(J_1, J_2) + d(J_2, J_2)$ ). Let  $d_{[t_1, t_2]}(Z, Y)$  denote the number of pairs in  $(Z, Y)$  that are scheduled by  $Alg$  in the interval  $[t_1, t_2]$ . We use the shorthand  $d_1(J_1, J_1)$  to denote  $d_{[1, \alpha n]}(J_1, J_1)$ , and the shorthand  $d_2(J_1, J_1)$  to denote  $d_{[\alpha n + 1, |Alg|]}(J_1, J_1)$ .

Turning to the optimal schedule,  $OPT$ , recall that by Claim 5.3 we may assume without loss of generality that all pairs of jobs in  $(J_1, J_1)$  that are scheduled by  $OPT$  from round  $\alpha n$  and onward are scheduled consecutively starting at this round. Let  $\ell_0$  be the last round in which  $OPT$  schedules a pair of jobs from  $(J_1, J_1)$ . We denote by  $q$  the total number of rounds in which  $OPT$  schedules either a singleton job or no job at all, where  $q_1$  is the number of such rounds until round  $\ell_0$ , and  $q_2$  is the number of such rounds after round  $\ell_0$ .

We start by giving a simple lower bound on  $x = |X|$ .



**Fig. 7** An illustration of the notations in the online algorithm’s analysis. Recall that  $J_1$  is the subset of jobs that arrive in rounds 1 to  $\alpha n$ , and  $J_2$  are the jobs that arrive after round  $\alpha n$ . **(A)** A high level view of the execution of the online algorithm  $Alg$  for the case  $|J_1| \geq \frac{n}{2}$ . Specifically, from round 1 to round  $\alpha n$  it schedules the jobs in any non-lazy manner, where the number of pairs it schedules in these rounds is  $d_1(J_1, J_1)$ . From round  $\alpha n + 1$  to round  $t_0$  it schedules a maximum matching of size  $d_2(J_1, J_1)$ . From round  $t_0 + 1$  to  $|Alg|$ , there are  $d(J_1, J_2)$  pairs of jobs from  $(J_1, J_2)$  and  $d(J_2, J_2)$  pairs of jobs from  $(J_2, J_2)$ . With the exception of the matching, the pairs of jobs in any interval are not necessarily scheduled consecutively (and are only

drawn so for simplicity). **(B)** “Zooming in” to the interval of rounds starting at round  $r^{\max}$ . Since the algorithm gives precedence to pairs over singletons, and follows a LIFO policy, it first schedules pairs from  $(J_1, J_2) \cup (J_2, J_2)$ , it then schedules singletons from  $J_2$ , and finally it schedules the jobs in  $X_3 \subseteq X$  as singletons. **(C)** “Zooming in” to a subinterval of  $[t_0 + 1, r^{\max} - 1]$  which starts at round  $t_i$  for some  $1 \leq i < x_1$ , and ends at round  $t_i + 1$ . In the first and last round, a job from  $X_1 \subseteq X$  is scheduled. In all other rounds, at least one job from  $J_2$  is scheduled (either as a singleton or paired with a job from  $J_1$  or paired with another job from  $J_2$ )

**Claim 5.8**  $x \geq (\frac{1}{2} - \alpha)n - (d_1(J_1, J_1) + 2d_2(J_1, J_1))$ .

*Proof* Each job in  $J_1$  is scheduled in one of the following ways: (i) by round  $\alpha n$ , or (ii) between rounds  $\alpha n + 1$  and  $t_0$  as part of the matched pairs in the maximum matching, or (iii) after round  $t_0$  when it belongs to the independent set  $X$ . Therefore,

$$|J_1| = (\alpha n + d_1(J_1, J_1)) + 2d_2(J_1, J_1) + x. \tag{8}$$

The claim follows from  $|J_1| \geq \frac{n}{2}$  and (8).  $\square$

We next state the main technical claim in this subsection. Recall that  $q$  denotes the number of rounds in which  $OPT$  schedules a singleton job or no job at all.

**Claim 5.9**  $q \geq (\frac{1}{4} - \alpha) \cdot n - \frac{3}{2}d$ .

In order to prove Claim 5.9, we prove several subclaims concerning  $q$ . Recall that  $x_3 = |X_3|$  where  $X_3$  is the subset of jobs in  $X$  that are scheduled by  $Alg$  as singletons in the interval of rounds  $[r^{\max}, |Alg|]$ . By definition of  $Alg$ , these jobs are scheduled consecutively in the last  $x_3$  rounds.

**Claim 5.10**  $q \geq x_3 - (\alpha n + d_1(J_1, J_1) + d_2(J_1, J_1) + d(J_1, J_2))$ .

*Proof* We upper bound the number of jobs from  $X_3$  that  $OPT$  paired with some other job. A job  $j \in X_3$  can be paired in  $OPT$  with a job in one of the following three subsets: (1) Jobs that are scheduled by the algorithm in rounds  $[1, \dots, \alpha n]$ , that is,  $J_{\leq \alpha n}^{Alg}$ ; (2) Jobs in  $J_1$  that are matched by the algorithm and scheduled in pairs from round  $\alpha n + 1$  to round  $t_0$ ; (3) Jobs in  $J_2$  that are paired with jobs in  $J_1$ . Note that a job  $j \in X_3$  cannot be paired by  $OPT$  with a job in  $J_2$  that was scheduled by the algorithm in a pair together with another job from  $J_2$ . Otherwise,  $Alg$  would have scheduled  $j$  together with a job in  $J_2$ , since it gives precedence to pairs from  $(J_1, J_2)$  over pairs from  $(J_2, J_2)$ .

The number of jobs in the first category, that is  $|J_{\leq \alpha n}^{Alg}|$ , is  $\alpha n + d_1(J_1, J_1)$ . To bound the number of jobs in the second category, recall that  $d_2(J_1, J_1)$  denotes the size of a maximum matching between jobs in  $X$ . Therefore, the number of jobs in  $X_3$  that can be matched to jobs that participate in this matching cannot be larger than  $d_2(J_1, J_1)$ . Finally, the number of jobs in the third category is  $d(J_1, J_2)$ , and the claim follows.  $\square$

We shall need the following lemma about matchings for our next lower bound on  $q$ .

**Lemma 5.11** *Let  $H = (V, E)$  denote a graph. Let  $H' = (V', E')$  denote a subgraph of  $H$  induced by  $V' \subseteq V$ . Let  $M_H$  and  $M_{H'}$  denote maximum matchings in  $H$  and  $H'$ , respectively. Then  $|M_H| \leq |M_{H'}| + |V \setminus V'|$ .*

*Proof* The edges in  $M_H$  consist of two disjoint types: those that are adjacent to at least one vertex in  $V \setminus V'$  and those that are adjacent to two vertices in  $V'$ . The number of edges in  $M_H$  of the first type is at most  $|V \setminus V'|$ , and the number of edges of the second type is at most  $|M_{H'}|$ . The lemma follows.  $\square$

Recall that  $q = q_1 + q_2$  where  $q_1$  is the number of rounds until round  $\ell_0$  in which  $OPT$  schedules either a singleton job or no job at all, and  $q_2$  is the number of such rounds after round  $\ell_0$ . Also recall that  $\ell_0$  is the last round in  $OPT$  with a pair of jobs from  $J_1$ . In order to obtain our second lower bound on  $q$ , we prove two claims that lower bound  $q_1$  and  $q_2$ .

**Claim 5.12**  $q_1 \geq \ell_0 - t_0 - d_1(J_1, J_1)$ .

*Proof* By Claim 5.3, we may assume that up to round  $\ell_0$ ,  $OPT$  schedules only jobs from  $J_1$ . Up to round  $\ell_0$ ,  $OPT$  schedules jobs either as singletons (and their number is  $q_1$ ) or as pairs in  $(J_1, J_1)$ . Let  $H$  denote the subgraph of the agreement graph induced by  $J_1$ , and let  $M_H$  denote a maximum matching in  $H$ . Since the pairs of jobs scheduled in rounds  $\ell_0 - q_1$  form a matching in  $H$ ,

$$|M_H| \geq \ell_0 - q_1. \tag{9}$$

We use Lemma 5.11 to bound  $|M_H|$ . Let  $H'$  denote the subgraph of  $H$  induced by  $J_1 \cap J_{> \alpha n}^{Alg}$ . By the definition of  $Alg$ ,  $|M_{H'}| = d_2(J_1, J_1)$ . The difference  $V(H) \setminus V(H')$  is the set of jobs  $J_{\leq \alpha n}^{Alg}$  and contains  $\alpha n + d_1(J_1, J_1)$  jobs. Apply Lemma 5.11 to obtain

$$\begin{aligned} |M_H| &\leq |M_{H'}| + |V(H) \setminus V(H')| \\ &= d_2(J_1, J_1) + \alpha n + d_1(J_1, J_1). \end{aligned} \tag{10}$$

Recall that  $t_0 = \alpha n + d_2(J_1, J_1)$ . Hence,

$$\begin{aligned} \ell_0 &\leq q_1 + |M_H| \quad (\text{by (9)}) \\ &\leq q_1 + d_2(J_1, J_1) + \alpha n + d_1(J_1, J_1) \quad (\text{by (10)}) \\ &= q_1 + t_0 + d_1(J_1, J_1) \quad (t_0 = \alpha n + d_2(J_1, J_1)), \end{aligned}$$

and the claim follows.  $\square$

Recall that  $x_1 = |X_1|$  where  $X_1$  is the subset of jobs in  $X$  that are scheduled as singletons before round  $r^{\max}$ .

**Claim 5.13**  $q_2 \geq x_1 + t_0 - \ell_0 - d(J_2, J_2)$ .

*Proof* Let  $\{t_1, t_2, \dots, t_{x_1}\}$  denote the set of rounds during which  $Alg$  schedules singletons in  $X_1$ . In each round  $t_i$ , the set of pending jobs does not contain jobs in  $J_2$  (since the LIFO policy of  $Alg$  implies that precedence is given to  $J_2$

over  $J_1$ ). In each round during the interval  $(t_i, t_{i+1})$ , the algorithm schedules at least one job from  $J_2$ . By the end of this interval, no job from  $J_2$  is pending. Hence,

$$|J_{[t_i, t_{i+1}]}| = t_{i+1} - t_i - 1 + d_{[t_i, t_{i+1}]}(J_2, J_2) \tag{11}$$

and

$$|J_{[\alpha n + 1, t_1]}| = t_1 - t_0 - 1 + d_{[\alpha n + 1, t_1]}(J_2, J_2). \tag{12}$$

Note that  $J_{[t_{i-1}, t_i]}$  and  $J_{[t_i, t_{i+1}]}$  are disjoint since no job arrives in round  $t_i$  (or else the algorithm would schedule that job and not a job from  $X_1$ ). We now sum (11) (for  $1 \leq i < x_1$ ) and (12) to obtain

$$\begin{aligned} |J_{[\alpha n + 1, t_{x_1}]}| &= |J_{[\alpha n + 1, t_1]}| + \sum_{i=1}^{x_1-1} |J_{[t_i, t_{i+1}]}| \\ &= t_1 - t_0 - 1 + d_{[\alpha n + 1, t_1]}(J_2, J_2) \\ &\quad + \sum_{i=1}^{x_1-1} (t_{i+1} - t_i - 1 + d_{[t_i, t_{i+1}]}(J_2, J_2)) \\ &= t_{x_1} - t_0 - x_1 + d_{[\alpha n + 1, t_{x_1}]}(J_2, J_2) \\ &\leq t_{x_1} - t_0 - x_1 + d(J_2, J_2). \end{aligned} \tag{13}$$

In each round in the interval  $[\ell_0 + 1, t_{x_1}]$ ,  $OPT$  schedules either a singleton (counted in  $q_2$ ) or a pair from  $(J_1, J_2)$  or from  $(J_2, J_2)$ . That is, in each pair there is at least one job from  $J_2$ . Hence,

$$t_{x_1} - \ell_0 \leq q_2 + J_{[\alpha n, t_{x_1}]}, \tag{14}$$

from which it follows that

$$\begin{aligned} q_2 &\geq t_{x_1} - \ell_0 - J_{[\alpha n, t_{x_1}]} \\ &\geq t_{x_1} - \ell_0 - (t_{x_1} - t_0 - x_1 + d(J_2, J_2)) \quad (\text{by (13)}) \\ &= t_0 + x_1 - \ell_0 - d(J_2, J_2). \end{aligned} \quad \square$$

We are now ready to prove Claim 5.9, and following it, Lemma 5.7.

*Proof of Claim 5.9* We use Claims 5.12 and 5.13 to bound the total number of singletons  $q$  in  $OPT$ :

$$\begin{aligned} q &= q_1 + q_2 \\ &\geq (\ell_0 - t_0 - d_1(J_1, J_1)) + (t_0 + x_1 - \ell_0 - d(J_2, J_2)) \\ &= x_1 - d_1(J_1, J_1) - d(J_2, J_2). \end{aligned} \tag{15}$$

By combining (15) and Claim 5.10, the number of singletons in  $OPT$  is at least

$$\begin{aligned} q &\geq \max\{x_1 - (d_1(J_1, J_1) + d(J_2, J_2)), \\ &\quad x_3 - (\alpha n + d_1(J_1, J_1) + d_2(J_1, J_1) + d(J_1, J_2))\}. \end{aligned} \tag{16}$$

We further develop the lower bound on  $q$  as follows:

$$\begin{aligned} q &\geq \frac{1}{2} \cdot (x_1 + x_3 - (\alpha n + 2d_1(J_1, J_1) + d_2(J_1, J_1) \\ &\quad + d(J_1, J_2) + d(J_2, J_2))) \quad (\text{max} \geq \text{average}) \\ &= \frac{1}{2} \cdot (x - x_2 - (\alpha n + 2d_1(J_1, J_1) + d_2(J_1, J_1) \\ &\quad + d(J_1, J_2) + d(J_2, J_2))) \quad (x = x_1 + x_2 + x_3) \\ &= \frac{1}{2} \cdot (x - (\alpha n + 2d_1(J_1, J_1) + d_2(J_1, J_1) \\ &\quad + 2d(J_1, J_2) + d(J_2, J_2))) \quad (x_2 = d(J_1, J_2)) \\ &\geq \frac{1}{2} \cdot \left( \left( \frac{1}{2} - 2\alpha \right) \cdot n - (3d_1(J_1, J_1) + 3d_2(J_1, J_1) \right. \\ &\quad \left. + 2d(J_1, J_2) + d(J_2, J_2)) \right) \quad (\text{by Claim 5.8}). \end{aligned}$$

Since  $d$  equals the total number of pairs scheduled by  $Alg$ , it follows that

$$q \geq \left( \frac{1}{4} - \alpha \right) \cdot n - \frac{3}{2}d, \tag{17}$$

as required.  $\square$

*Proof of Lemma 5.7* The makespan of  $OPT$  equals the number of rounds  $q$  in which  $OPT$  schedules a singleton job or no job at all, plus the number of rounds in which  $OPT$  schedules pairs. Since the number of pairs  $OPT$  schedules is at least  $\frac{1}{2}(n - q)$ , by (17),

$$|OPT| \geq \frac{n - q}{2} + q = \frac{n + q}{2} \geq \left( \frac{5}{8} - \frac{1}{2}\alpha \right) n - \frac{3}{4}d. \tag{18}$$

By Claim 5.2,  $|Alg| = n - d$ , and since  $|OPT| \geq n/2$  it follows that the competitive ratio is at most  $\frac{n-d}{n/2}$ . We consider two cases: (1) If  $d \geq \frac{1}{2}\alpha n$ , then  $\frac{n-d}{n/2} \leq 2 - \alpha$ . (2) If  $d < \frac{1}{2}\alpha n$ , we bound the competitive ratio  $\rho$ , using (18), by

$$\rho \leq \frac{n - d}{(5/8 - (1/2)\alpha)n - \frac{3}{4}d}. \tag{19}$$

We compare the right hand side of (19) with  $2 - \alpha$ . By solving a quadratic equation, we obtain that if  $\alpha \leq 1/7$ , then  $\rho \leq 2 - \alpha$ , as required.  $\square$

*Dealing with the case that  $n$  is not divisible by 7* In the general case where  $n$  is not divisible by 7,  $\alpha$  is bounded by  $\frac{1}{7} - \frac{6}{n} \leq \alpha \leq \frac{1}{7}$ . Recall that Lemma 5.4 is applicable for

every  $0 < \alpha < 1$ , and Lemma 5.7 is applicable for  $\alpha \leq \frac{1}{7}$ . Hence, we get the following bound on the competitive ratio:

$$\begin{aligned} |Alg| &\leq (2 - \alpha) \cdot |OPT| \\ &\leq \left(2 - \left(\frac{1}{7} - \frac{6}{n}\right)\right) \cdot |OPT| \quad \left(\alpha \geq \frac{1}{7} - \frac{6}{n}\right) \\ &= \left(2 - \frac{1}{7}\right) \cdot |OPT| + \frac{6}{n} \cdot |OPT| \\ &\leq \left(2 - \frac{1}{7}\right) \cdot |OPT| + 6 \quad (|OPT| \leq n). \end{aligned}$$

### 5.5 A “slightly” randomized algorithm

In this subsection we remove the assumption that the algorithm knows  $n$  by allowing the algorithm to flip one random bit. We analyze the expected competitive ratio of the randomized algorithm, where the expectation is taken over the random bit.

We first observe that Claim 5.1 still holds. That is, we may assume that the (randomized) online algorithm  $Alg$  is never left without any pending jobs. Recall that the argument in the proof of Claim 5.1 was that, otherwise, we define another algorithm  $Alg\text{-}Res$ , that restarts  $Alg$  each time it is left with no pending jobs. This seems to imply that we would need a new, independent, random bit each time we restart  $Alg$ . However, the bound on the competitive ratio of  $Alg\text{-}Res$  was derived based on the very last execution of  $Alg$ , and hence it is possible to “reuse” the same random bit in all executions.

Next we observe that the online algorithm  $Alg$  described in Sect. 5.2 actually consists of two algorithms:  $Alg_1$  for the case that  $|J_1| < n/2$  and  $Alg_2$  for the case that  $|J_1| \geq n/2$ . The idea is that the randomized algorithm, which does not know  $n$ , will simply flip a coin and, according to its outcome, choose one of the algorithms. However, we shall first need to slightly modify the algorithms so that they do not rely on knowing  $n$ . During the first  $\alpha n$  rounds  $Alg_1$  and  $Alg_2$  are simply nonlazy algorithms. The difference between the algorithms is in their scheduling policy starting from round  $\alpha n + 1$ .

#### 5.5.1 The first algorithm

The first algorithm, which we refer to as  $Alg_1$ , gives precedence to pairs over singletons, and schedules singletons according to a FIFO policy. When it reaches round  $r^{\max}$ , it computes an optimal schedule of the pending jobs. First note that  $Alg_1$  does not actually need to know when round  $\alpha n$  arrives, and can simply use the same policy (precedence to pairs plus FIFO over singletons) from the start. It seems from its description that it needs to know when round  $r^{\max}$  arrives (that is, to know that all  $n$  jobs arrived). However, we

can do away with this knowledge as follows, where we call the modified algorithm  $Alg'_1$ .

At the start of each round,  $Alg'_1$  computes an optimal schedule of the current pending jobs, that is, a maximum matching in the agreement graph induced by the pending jobs (see Observation 3.1). If there is any matched pair then it schedules this pair; otherwise, it schedules a singleton job according to a FIFO policy. Thus, on one hand, it still gives precedence to pairs over singletons and schedules singletons according to a FIFO policy. On the other hand, once it reaches round  $r^{\max}$  it schedules the pending jobs optimally.  $Alg'_1$  actually belongs to the set of algorithms that  $Alg_1$  defines for the case of  $|J_1| < n/2$ . Therefore, Lemma 5.4 holds for  $Alg'_1$  and we get:

**Lemma 5.14** *For any choice of  $\alpha$ , if  $|J_1| < n/2$  then the competitive ratio of  $Alg'_1$  is  $2 - \alpha$ .*

#### 5.5.2 The second algorithm

The second algorithm, denoted  $Alg_2$ , was defined as follows (starting from round  $\alpha n + 1$ ). At the beginning of round  $\alpha n + 1$  it finds a maximum matching  $M$  of the pending jobs from  $J_1$ , and schedules the pairs in this matching in rounds  $[\alpha n + 1, t_0 = \alpha n + |M|]$ . From round  $t_0 + 1$  and onward, it continues by giving precedence to pairs over singletons, where pairs in  $(J_1, J_2)$  have precedence over pairs in  $(J_2, J_2)$ , and singletons are scheduled according to a LIFO policy. We next modify  $Alg_2$  so that it will be oblivious to  $n$  ( $\alpha n$ ). We will show that the resulting algorithm, denoted  $Alg'_2$ , has a competitive ratio of  $2 - \alpha'$  when  $|J_1| \geq n/2$ , for  $\alpha' \leq \frac{1}{9}$ .

The modified algorithm,  $Alg'_2$  is defined as follows: (i) It is nonlazy and gives precedence to pairs over singletons; (ii) It schedules pairs according to a lexicographic FIFO policy; (iii) It schedules singletons according to a LIFO policy. The main difference between  $Alg_2$  and  $Alg'_2$  is that while the former schedules a *maximum* matching of the pending jobs in  $J_1$  starting from round  $\alpha n + 1$ , the latter schedules a *maximal* matching of the pending jobs (because it gives precedence to pairs over singletons and schedules pairs according to a FIFO policy).

We show (recall that  $J_1$  is defined based on  $\alpha$ ):

**Lemma 5.15** *For  $\alpha \leq 1/9$ , if  $|J_1| \geq n/2$  then the competitive ratio of  $Alg'_2$  is  $2 - \alpha$ .*

*Proof* We follow the proof of Lemma 5.7 and only note what needs to be changed. The notation used in the proof remains the same with one exception:  $d_2(J_1, J_1)$  is no longer the size of the maximum matching  $M$  whose pairs (in  $(J_1, J_1)$ ) are scheduled by  $Alg_2$  starting from round  $\alpha n + 1$ , but is the size of the maximal matching of pairs in  $(J_1, J_1)$  scheduled by  $Alg'_2$  starting from round  $\alpha n + 1$ .

We first observe that Claim 5.8 remains unchanged. We next turn to the (sub)claims in which  $q$  (the number of rounds in which  $OPT$  schedules a singleton job or no job at all) is bounded (and which leads to Claim 5.9). The bound in Claim 5.10 is modified from  $q \geq x_3 - (\alpha n + d_1(J_1, J_1) + d_2(J_1, J_1) + d(J_1, J_2))$  to:

$$q \geq x_3 - (\alpha n + d_1(J_1, J_1) + 2d_2(J_1, J_1) + d(J_1, J_2)). \tag{20}$$

The reasoning is that, since  $Alg'_2$  schedules a maximal matching of size  $d_2(J_1, J_1)$  rather than a maximum matching, jobs in  $X_3$  may be paired by  $OPT$  with all  $2d_2(J_1, J_1)$  jobs from  $J_1$  that participate in the matching. Note that it is still true that  $OPT$  cannot pair jobs in  $X_3$  with jobs that were scheduled by  $Alg'_2$  in pairs from  $(J_2, J_2)$  because of the lexicographic FIFO policy that  $Alg'_2$  applies to pairs.

Since the size of a maximum matching in a graph is at most twice the size of any maximal matching, the bound in Claim 5.12 is changed from:  $q_1 \geq \ell_0 - t_0 - d_1(J_1, J_1)$  to

$$q_1 \geq \ell_0 - t_0 - d_1(J_1, J_1) - d_2(J_1, J_1). \tag{21}$$

The bound in Claim 5.13 remains unchanged, that is:

$$q_2 \geq x_1 + t_0 - \ell_0 - d(J_2, J_2) \tag{22}$$

(where  $t_0 = \alpha n + d_2(J_1, J_1)$  as before, that is, it is the last round in which  $Alg'_2$  schedules pairs in  $(J_1, J_1)$ ). By combining (21) and (22) we obtain:

$$q \geq x_1 - d_1(J_1, J_1) - d_2(J_1, J_1) - d(J_2, J_2), \tag{23}$$

and by combining (23) with (20) we get:

$$q \geq \max \left\{ x_1 - (d_1(J_1, J_1) + d_2(J_1, J_2) + d(J_2, J_2)), x_3 - (\alpha n + d_1(J_1, J_1) + 2d_2(J_1, J_1) + d(J_1, J_2)) \right\}. \tag{24}$$

By developing this expression (similarly to what was done in the proof of Claim 5.9) we obtain:

$$q \geq \left( \frac{1}{4} - \alpha' \right) \cdot n - \frac{5}{2}d. \tag{25}$$

By continuing in a manner analogous to the proof of Lemma 5.7 we get that the competitive ratio of  $Alg'_2$  (conditioned on  $|J_2| \geq n/2$ ) is upper bounded by  $2 - \alpha'$  for  $\alpha' \leq 1/9$ .  $\square$

### 5.5.3 The randomized algorithm

Let  $Alg'$  be a randomized algorithm that flips a single fair coin and according to its outcome decides whether to execute  $Alg'_1$  or  $Alg'_2$ . By combining Lemmas 5.14 and 5.15 (and using the fact that the competitive ratio of  $Alg'_1$  and  $Alg'_2$  is at most 2), we get:

**Theorem 9** *The competitive ratio of  $Alg'$  is  $2 - \frac{1}{2}\alpha' = 2 - \frac{1}{18}$  in expectation.*

### 5.5.4 Lower bound for randomized online algorithms

**Theorem 10** *The competitive ratio of any randomized online algorithm is at least 5/4.*

*Proof* At step 1,  $n$  vertices that form an independent set in the agreement graph are released. At step  $n/2 + 1$ ,  $n/2$  additional vertices are released; they form a matching with half of the earlier vertices. This completes the construction. The optimal solution is to schedule first the  $n/2$  non-matched isolated vertices, followed by the  $n/2$  pairs, for a makespan of  $n$ .

Any randomized algorithm can schedule at most half of the non-matched vertices during the first  $n/2$  steps, in expectation. Thus, at step  $n/2 + 1$ , it has at least expected  $n/4$  non-matched vertices left, along with  $n/2$  pairs (or parts thereof). Hence, its makespan is at least expected  $5n/4$ , for a performance ratio of  $5/4$ .  $\square$

## 6 A reduction to $k$ -set-cover

Consider the problem of scheduling unit jobs with conflicts on  $m$  machines, where  $m$  is constant. We present a reduction of this problem to finding a minimum set cover for the case in which all sets have cardinality at most  $m$  (in short,  $m$ -set-cover).

Let  $G = (J, E)$  denote the conflict graph of a scheduling problem with unit jobs on  $m$  machines. We reduce  $G$  to a set system  $\langle S, U \rangle$  in which every set contains at most  $m$  elements as follows. The set of elements  $U$  is simply  $J$ . The collection of subsets  $S$  is the set of all cliques of size at most  $m$  in the agreement graph  $\overline{G}$ . Note that if  $A \in S$ , then every subset  $A' \subset A$  is also in the collection  $S$ . There is a one-to-one correspondence between covers  $S' \subseteq S$  of  $U$  by disjoint sets and schedules  $T$  of jobs in  $J$  using  $m$  machines (here we ignore the order of rounds in a schedule and regard two schedules to be the same if one can be obtained from the other only by permuting the order of the rounds and the assignment of jobs to machines within each round). This correspondence maps a disjoint cover  $S'$  to a schedule  $T_{S'}$  such that the makespan of  $T_{S'}$  equals the cardinality of  $S'$ .

The  $k$ -set-cover problem has an approximation algorithm whose ratio equals  $\mathcal{H}_k - \frac{1}{2}$  (Duh and Fürer 1997). The reduction to  $k$ -set-cover implies that scheduling unit jobs on  $k$  machines is not harder to approximate than  $k$ -set-cover.

**Claim 6.1** *The problem of scheduling unit jobs with conflicts on  $m$  machines (where  $m$  is constant) has an approximation algorithm whose approximation ratio  $\mathcal{H}_m - \frac{1}{2}$ . In particular, for  $m = 3$ , a  $\frac{4}{3}$ -approximation exists.*

We remark that the tight example in Duh and Fürer (1997) for  $m = 3$  also holds for scheduling unit jobs with conflicts. It remains an open problem whether minimum makespan scheduling of unit jobs with conflicts on  $m$  machines is as hard to approximate as  $m$ -set-cover ( $SC_m$ ).

The following lemma, for the case of jobs with varying sizes, is proved by scaling.

**Lemma 6.2** *The problem of scheduling jobs with conflicts on  $m$  machines (where  $m$  is constant) has an approximation algorithm whose approximation ratio  $2 \cdot \lceil \log_2 \frac{\max_j p_j}{\min_j p_j} \rceil \cdot (\mathcal{H}_m - \frac{1}{2})$ .*

*Proof* The algorithm, which we denote by Scaled- $m$ -Set-Cover ( $SSC_m$ ), works as follows. First it divides the jobs into bins  $J_i$  such that  $j \in J_i$  if  $2^{i-1} \leq p_j \leq 2^i$ , for  $i = \lceil \log_2(\min_j p_j) \rceil + 1, \dots, \lceil \log_2(\max_j p_j) \rceil$  (a job  $j$  with  $p_j = 2^i$  that can be in  $J_i$  or in  $J_{i+1}$  will arbitrarily be in one of them). Denote by  $\tilde{J}_i$  the set of jobs in  $J_i$  when the size of each is rounded up to  $2^i$ . Namely, for each  $j_\ell \in J_i$  there is a job  $\tilde{j}_\ell$  of size  $2^i$  in  $\tilde{J}_i$ , where the edges between the jobs are as in the original agreement graph. For each bin  $J_i$ ,  $SSC_m$  runs  $m$ -set-cover on  $\tilde{J}_i$ . Let  $SC_m(\tilde{J}_i)$  be the corresponding schedule. For each subset  $\{\tilde{j}_1, \dots, \tilde{j}_t\}$  of  $SC_m(\tilde{J}_i)$ ,  $SSC_m$  schedules  $\{j_1, \dots, j_t\}$  concurrently (in time at most  $2^i$ ).

Recall that  $OPT(J_i)$  denotes the optimal makespan for  $J_i$ . We now bound the makespan of the schedule  $SSC_m(J_i)$ .

$$\begin{aligned} |SSC_m(J_i)| &\leq |SC_m(\tilde{J}_i)| \quad (\text{by rounding}) \\ &\leq \left(\mathcal{H}_m - \frac{1}{2}\right) \cdot |OPT(\tilde{J}_i)| \\ &\quad (\text{approximation of } m\text{-set-cover}) \\ &\leq 2 \cdot \left(\mathcal{H}_m - \frac{1}{2}\right) \cdot |OPT(J_i)| \\ &\quad (\text{rounding up only doubles the makespan}). \end{aligned}$$

Let  $i_{\min} \triangleq \lceil \log_2(\min_j p_j) \rceil$  and  $i_{\max} \triangleq \lceil \log_2(\max_j p_j) \rceil - 1$ . The makespan of  $SSC_m$  is the sum of the makespans of the schedules  $SSC_m(J_i)$  for  $i = i_{\min}, \dots, i_{\max}$

$$\begin{aligned} |SSC_m| &= \sum_{i=i_{\min}}^{i_{\max}} |SSC_m(J_i)| \\ &\leq \sum_{i=i_{\min}}^{i_{\max}} 2 \cdot \left(\mathcal{H}_m - \frac{1}{2}\right) \cdot |OPT(J_i)| \\ &\leq \sum_{i=i_{\min}}^{i_{\max}} 2 \cdot \left(\mathcal{H}_m - \frac{1}{2}\right) \cdot |OPT| \end{aligned}$$

$$\begin{aligned} &= 2 \cdot (i_{\max} - i_{\min} + 1) \cdot \left(\mathcal{H}_m - \frac{1}{2}\right) \cdot |OPT| \\ &= 2 \cdot \left\lceil \log_2 \frac{\max_j p_j}{\min_j p_j} \right\rceil \cdot \left(\mathcal{H}_m - \frac{1}{2}\right) \cdot |OPT|. \quad \square \end{aligned}$$

Scaling can be further extended to yield the following approximation ratio.

**Claim 6.3** *The problem of scheduling jobs with conflicts on  $m$  machines (where  $m$  is constant) has an approximation algorithm whose approximation ratio  $O(\log \min\{\frac{\max_j p_j}{\min_j p_j}, n\}) \cdot \log m$ .*

*Proof* Partition the jobs into tiny jobs, whose processing times satisfy  $p_j < \frac{1}{n^2} \cdot \frac{1}{m} \cdot p(J)$ , and to normal jobs, whose processing time satisfies  $p_j \geq \frac{1}{n^2} \cdot \frac{1}{m} \cdot p(J)$ . Apply Lemma 6.2 to the normal jobs. Since the ratio  $\frac{\max_j p_j}{\min_j p_j}$  for normal jobs is bounded by  $m \cdot n^2$ , the approximation ratio of the makespan obtained for normal jobs is bounded by  $O(\log n \cdot \log m)$ . Schedule the tiny jobs separately on a single machine. The makespan for the tiny jobs is bounded by  $\frac{1}{n} \cdot \frac{p(J)}{m} \leq \frac{1}{n} \cdot |OPT|$ .  $\square$

**Acknowledgement** We thank Boaz Patt-Shamir and Hadas Shachnai for their useful comments. We thank the reviewers for their helpful comments.

**Appendix:  $m$  machines and arbitrary processing times**

In this section we analyze the makespan of the greedy mutual exclusion (GME) algorithm in the case of  $m$  machines and jobs of arbitrary (integral) processing times. This is essentially the algorithm that is derived from the work of Garey and Graham (1975), and we present its analysis for completeness. We note that the algorithm and its analysis can be extended to non-integral processing times. We prove that GME obtains an  $(m + 1)/2$ -approximation of the makespan, and show this ratio to be tight for this algorithm.

A listing of the GME algorithm appears below. Note that, in the description of the algorithm, after a hole is scheduled in a machine’s slot, then the machine is not considered vacant during this slot anymore.

**GME( $G = (J, E), m$ ):** Greedy Mutual Exclusion schedules the jobs  $J$  with conflict graph  $G$  on  $m$  machines

- 1: **while**  $J \neq \emptyset$  **do**
- 2:   Let  $t$  denote the first round that contains a vacant slot.
- 3:   **if** exists a job  $j \in J$  that agrees with all the jobs already scheduled in round  $t$  **then**
- 4:     Pick a machine that has a vacant slot in round  $t$  and schedule  $j$  on that machine during rounds  $[t, t + p_j - 1]$ .

```

5:    $J \leftarrow J \setminus \{j\}$ 
6:   else
7:     Schedule a hole in round  $t$  on each machine that
       has a vacant slot in round  $t$ .
8:   end if
9: end while

```

For simplicity, we described GME round by round, so that the running time of the algorithm, as described, is pseudo-polynomial. The algorithm can quite easily be modified to run in (strictly) polynomial time by working in phases as follows. In each phase, we consider a machine with minimal load (that is, the smallest number of occupied slots). If we can schedule a job on that machine, we do so. Otherwise, we schedule a *block (sequence) of holes* on this machine (and all other machines that have the same load), so that their load now equals the second largest load. It follows that a job is scheduled every  $m' \leq m$  phases.

*Analysis of GME’s makespan* We now prove the following lemma.

**Lemma 7.1** *GME is an  $(\frac{m+1}{2})$ -approximation algorithm for the makespan.*

*Proof* Fix an input and consider the schedule that is computed by an execution of GME. Let  $slices(t)$  denote the set of job slices that are scheduled in round  $t$  by GME. Let  $A_k$  denote the set of rounds in which  $k$  jobs are scheduled (namely,  $t \in A_k$  iff  $|slices(t)| = k$ ). Let  $A_{\geq 2} \triangleq \bigcup_{i=2}^m A_i$  (note that  $|A_{\geq 2}| = \sum_{i=2}^m |A_i|$  since the sets  $A_i$  are disjoint).

The proof is based on the following two equations:

$$|A_1| \leq |OPT|, \tag{26}$$

$$|A_{\geq 2}| \leq \frac{1}{2}(p(J) - |A_1|). \tag{27}$$

Equation (26) holds because the set of jobs that have slices in  $\bigcup_{t \in A_1} slices(t)$  form an independent set in the agreement graph  $\overline{G}$ . Indeed, suppose  $j_1$  and  $j_2$  contain slices that are scheduled in rounds  $t_1, t_2 \in A_1$ , respectively, where  $t_1 < t_2$ . If  $j_1$  and  $j_2$  are not conflicting jobs, then  $j_2$  would have been scheduled by GME in round  $t_1$  together with  $j_1$ . In this case, round  $t_1$  would not be in  $A_1$ .

Equation (27) holds because GME schedules  $p(J) - |A_1|$  job slices during the rounds  $A_2 \cup \dots \cup A_m$ , at least two per round. Observe that the sum  $p(J) = \sum_j p_j$  of processing times satisfies

$$p(J) = m \cdot |OPT| - |holes(OPT)|. \tag{28}$$

GME’s makespan equals  $|A_1| + |A_{\geq 2}|$ , which we bound as follows:

$$\begin{aligned} |A_1| + |A_{\geq 2}| &\leq \frac{1}{2} \cdot (p(J) + |A_1|) \quad (\text{by (27)}) \\ &\leq \frac{1}{2} \cdot (p(J) + |OPT|) \quad (\text{by (26)}) \\ &\leq \left(\frac{m+1}{2}\right) \cdot |OPT| - \frac{|holes(OPT)|}{2} \\ &\quad (\text{by (28)}) \\ &\leq \left(\frac{m+1}{2}\right) \cdot |OPT|, \end{aligned} \tag{29}$$

and the lemma follows. □

The proof of Lemma 7.1 implies that holes in the optimal schedule (i.e.,  $|holes(OPT)| > 0$ ) reduce the approximation ratio of GME. For example, if  $|holes(OPT)| \geq \frac{m}{2} \cdot |OPT|$ , then the approximation ratio of GME is reduced to  $\frac{m}{4} + \frac{1}{2}$ .

We also note that the analysis of GME was based on the fact that it is 2-nonlazy, and so the bound obtained holds for any 2-nonlazy algorithm.

*Tightness of the approximation of the makespan* We present an example with  $m^2$  unit jobs over  $m$  machines with the following property. All optimal schedules have a makespan  $m$  and lack holes. However, GME might compute a “bad” schedule in which only two machines are utilized. Moreover, in this bad schedule there are  $m$  jobs that GME schedules alone. This implies that the ratio of  $(m + 1)/2$  is indeed tight.

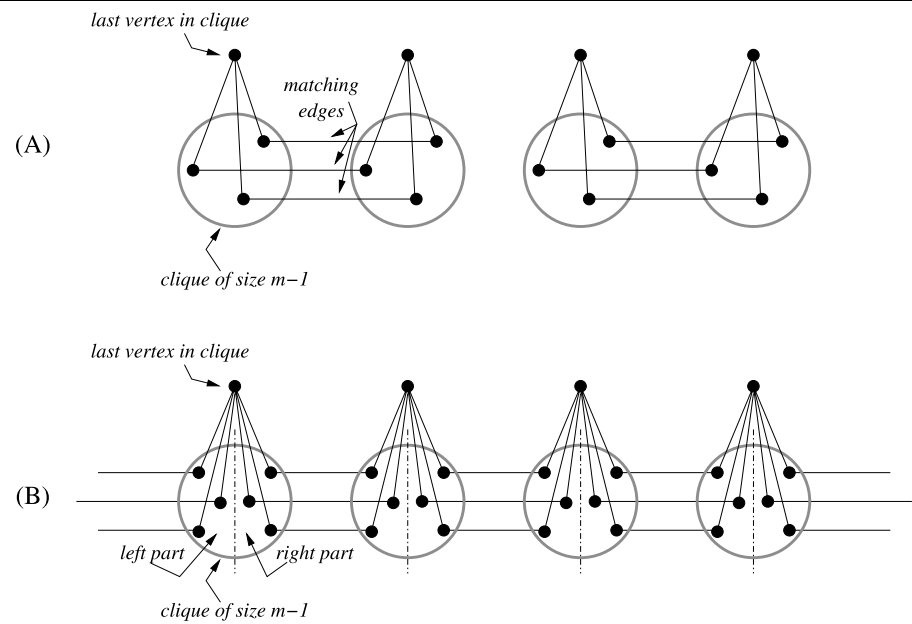
The set of jobs is  $J = \{(i, \ell) \mid (i, \ell) \in [0, \dots, m - 1] \times [0, \dots, m - 1]\}$ . The edges of the agreement graph  $\overline{G} = (J, \overline{E})$  form  $m$  cliques of size  $m$ , where the  $i$ th clique is over the set of vertices  $\{i\} \times [0, \dots, m - 1]$ . Suppose that  $m$  is even. We add “matching” edges  $((2i, \ell), (2i + 1, \ell))$ , for each  $i \in [0, \dots, m/2 - 2]$  and each  $\ell \in [0, \dots, m - 2]$ . Note that, all the vertices, except the “last” one, are matched in each clique. Figure 8(A) depicts the agreement graph.

An optimal schedule can schedule each clique in a single round, thus completing the schedule in  $m$  rounds. GME, on the other hand, might select pairs of jobs that are connected by matching edges. Each such pair is a maximal clique in the agreement graph, namely, no other job can be scheduled concurrently. After scheduling  $m(m - 1)/2$  pairs of matching jobs, GME is left with  $m$  independent jobs in  $\overline{G}$ . These jobs are then scheduled as singletons, as claimed above.

A similar construction exists for odd values of  $m$ . Here, each clique is partitioned into 3 parts: (a) the left part  $\{i\} \times [0, \dots, (m - 1)/2 - 1]$ , (b) the right part  $\{i\} \times$



**Fig. 8** Tight constructions for the greedy algorithm.  
 (A) A construction for the case that  $m$  is even.  
 (B) A construction for odd values of  $m$ . The  $m$  cliques are arranged in a cycle



$[(m - 1)/2, m - 2]$ , and (c) the last vertex  $(i, m - 1)$ . Note that the left and right parts each contain  $(m - 1)/2$  vertices. Now, we add edges that form a perfect matching from the right part of clique  $i$  to the left part of clique  $i + 1 \pmod m$ . Figure 8(B) depicts the agreement graph in this construction. A similar argument shows that the greedy algorithm might compute a schedule whose makespan is  $m + m(m - 1)/2$ .

**References**

Alon, N. (1983). A note on the decomposition of graphs into isomorphic matchings. *Acta Mathematica Hungarica*, 42, 221–223.  
 Baker, B. S., & Coffman, Jr., E. G. (1996). Mutual exclusion scheduling. *Theoretical Computer Science*, 162(2), 225–243.  
 Bar-Noy, A., Halldórsson, M. M., Kortsarz, G., Salman, R., & Shachnai, H. (2000). Sum multicoloring of graphs. *Journal of Algorithms*, 37, 422–450.  
 Bodlaender, H. L., & Jansen, K. (1993). On the complexity of scheduling incompatible jobs with unit-times. In *MFCS'93: Proceedings of the 18th international symposium on mathematical foundations of computer science* (pp. 291–300). London, UK, 1993. Berlin: Springer.  
 Bodlaender, H. L., & Jansen, K. (1995). Restrictions of graph partition problems. Part I. *Theoretical Computer Science*, 148, 93–109.  
 Bodlaender, H. L., Jansen, K., & Woeginger, G. J. (1994). Scheduling with incompatible jobs. *Discrete Applied Mathematics*, 55, 219–232.  
 Buchsbaum, A. L., Karloff, H., Kenyon, C., Reingold, N., & Thorup, M. (2004). OPT versus LOAD in dynamic storage allocation. *SIAM Journal on Computing*, 33, 632–646.  
 Coffman, Jr. E. G., Garey, M. R., Johnson, D. S., & LaPaugh, A. S. (1985). Scheduling file transfers. *SIAM Journal on Computing*, 14(3), 744–780.  
 Downey, R. G., & Fellows, M. R. (1995). Fixed-parameter tractability and completeness i: Basic results. *SIAM Journal on Computing*, 24(4), 873–921.

Duh, R., & Fürer, M. (1997). Approximation of k-set cover by semi-local optimization. In *Proceedings of the twenty-ninth annual ACM symposium on the theory of computing* (pp. 256–264).  
 Drozdowski, M. (1996). Scheduling multiprocessor tasks—an overview. *European Journal of Operational Research*, 94, 167–191.  
 Epstein, L., & Levin, A. (2006). On bin packing with conflicts. In *Proceedings of the 4th workshop on approximation and online algorithms (WAOA)* (pp. 160–173).  
 Feige, U., & Kilian, J. (1998). Zero knowledge and the chromatic number. *JCSS*, 57, 187–199.  
 Garey, M. R., & Graham, R. L. (1975). Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4, 187–200.  
 Garey, M. R., & Johnson, D. S. (1975). Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal on Computing*, 4, 397–411.  
 Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability*. New York: Freeman.  
 Hansen, P., Hertz, A., & Kuplinsky, J. (1993). Bounded vertex colorings of graphs. *Discrete Mathematics*, 111(1–3), 305–312.  
 Halldórsson, M. M., Kortsarz, G., Proskurowski, A., Salman, R., Shachnai, H., & Telle, J. A. (2003). Multicoloring trees. *Information and Computation*, 180(2), 113–129.  
 Irani, S., & Leung, V. (2003). Scheduling with conflicts on bipartite and interval graphs. *Journal of Scheduling*, 6(3), 287–307.  
 Jansen, K. (1999). An approximation scheme for bin packing with conflicts. *Journal of Combinatorial Optimization*, 3(4), 363–377.  
 Jansen, K. (2003). The mutual exclusion scheduling problem for permutation and comparability graphs. *Information and Computation*, 180(2), 71–81.  
 Jansen, K., & Porkolab, L. (2002). Polynomial time approximation schemes for general multiprocessor job shop scheduling. *Journal of Algorithms*, 45, 167–191.  
 Kaplan, L. (2007). *Scheduling with conflicts*. Master's thesis, Tel-Aviv University.  
 Kaller, D., Gupta, A., & Shermer, T. (1995). The  $\chi_t$  coloring problem. In *Lecture notes in computer sciences: Vol. 900. Symposium on theoretical aspects of computer science, STACS 95*. Berlin: Springer.

- Lonc, Z. (1992). On complexity of some chain and antichain partition problems. In *WG'91: Proceedings of the 17th international workshop* (pp. 97–104). London, UK, 1992. Berlin: Springer.
- Petranc, E. (1994). The hardness of approximation: Gap location. *Computational Complexity*, 4, 133–157.
- Shmoys, D. B., Wein, J., & Williamson, D. P. (1995). Scheduling parallel machines on-line. *SIAM Journal on Computing*, 24(6), 1313–1331.
- Zuckerman, D. (2006). Linear degree extractors and the inapproximability of max clique and chromatic number. In *Proceedings of the thirty-sixth annual ACM symposium on the theory of computing* (pp. 681–690).