Anarchy in the UJ

Briskorn, Dirk; Waldherr, Stefan

published in
European Journal of Operational Research
2022

DOI (link to publisher)
10.1016/j.ejor.2021.11.047

document version
Publisher's PDF, also known as Version of record

European Journal of Operational Research 301 (2022) 815–827

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Anarchy in the UJ: Coordination mechanisms for minimizing the number of late jobs

Dirk Briskorn[a,*], Stefan Waldherr[b]

[a] Professur für BWL, insbesondere Produktion und Logistik, Bergische Universität Wuppertal, 42119 Wuppertal, Germany
[b] School of Business and Economics, Vrije Universiteit Amsterdam, the Netherlands

ARTICLE INFO

Article history:
Received 10 November 2020
Accepted 25 November 2021
Available online 30 November 2021

Keywords:
Scheduling
Distributed scheduling
Price of anarchy
Coordination mechanisms

ABSTRACT

We consider the distributed scheduling problem on parallel machines with the central objective of maximizing the number of on-time jobs. Jobs are self-interested utility-maximizers that can choose the machines they are processed on and are exclusively interested in reducing their own private objective function. Each machine processes the jobs according to a local policy. We discuss Nash equilibria in the resulting schedules and perform a thorough analysis of the resulting (absolute) prices of anarchy for various parallel machine environments, utilities of the agents, and local policies of the machines. We show that local policies that are based on simple sorting-based procedures like shortest processing time first (SPT) and earliest due date first (EDD) lead to big losses in welfare compared to the global optimum. However, when employing Moore-Hodgson's algorithm as a local policy, we can prove a price of anarchy of $(2m-1)/m$ for identical machines and a price of anarchy of 2 for related and unrelated parallel machines. Moreover, we show how these results can be used to prove approximation ratios for greedy scheduling algorithms. This paper is the first to prove approximation ratios for two greedy scheduling procedures, turning them from simple heuristics into actual approximation ratios with a provable approximation ratio.

1. Introduction

In recent years, the decentralized allocation of resources in systems comprised of large numbers of independent, autonomous entities has become ever more important. Instead of a global planner that optimizes towards a central objective, these decentralized environments consist of self-interested agents that want to maximize their own utility. Since agents in general are not interested in optimizing towards the central objective, a system designer can only guide the process by implementing protocols such that the selfish agents, by pursuing their self-interests, coordinate towards a desirable outcome. This approach is known as mechanism design and has drawn attention in economics, game theory, and operations research. In their seminal work, Nisan & Ronen (2001) present a general framework for studying algorithms in mechanism design. Famous examples of applications in mechanism design are routing in networks (Roughgarden, 2005), auctions (Roughgarden, Syrgkanis, & Tardos, 2017), school choice (Abdulkadiroğlu & Sönmez, 2003), or scheduling (Heydenreich, Müller, & Uetz, 2010; Nisan & Ronen, 2001). In these environments, the planner can use monetary payment schemes (or tolls) to influence the strategies of agents. The designer needs to have global knowledge of the system and the behavior of participants in order to calculate the best payments that guide the process. Another approach for designers is the use of coordination mechanisms (Christodoulou, Koutsoupias, & Nanavati, 2009). Within this framework, the designer only implements local policies that assign costs to the strategies of each agents. A prime application of coordination mechanisms is distributed scheduling of jobs to parallel machines (Azar, Fleischer, Jain, Mirrokni, & Svitkina, 2015; Caragiannis, 2013; Cole, Correa, Gkatzelis, Mirrokni, & Olver, 2011; Hoeksma & Uetz, 2011; Immorlica, Li, Mirrokni, & Schulz, 2009; Lee, Leung, & Pinedo, 2012). Each job belongs to a selfish agent that may choose the machine her job is assigned to. In the following, we do not distinguish between jobs and agents and associate each agent directly with the job she owns. Each machine schedules all of its jobs according to a local policy, which is determined by the mechanism designer. In some settings, the machines act strategically as well when selecting their local policy in order to optimize their local objective (Ashlagi, Tennenholtz, & Zohar, 2010). We focus on the complete information setting where every agents' properties and the local policies of the machines are

* Corresponding author.
E-mail addresses: briskorn@wiwi.uni-wuppertal.de (D. Briskorn), s.m.g.waldherr@vu.nl (S. Waldherr).

https://doi.org/10.1016/j.ejor.2021.11.047
0377-2217/

known to all participants and the machines are not strategic. Moreover, no payments are issued in order to guide the agents. We refer to the excellent survey of Heydenreich, Müller, & Uetz (2007) for a thorough discussion and comparison of complete information and private information settings as well as mechanism design with and without payments.

Given the local policies set by the designer, each agent forms a strategy in order to maximize her own utility. The Nash equilibrium (NE) is a solution concept in game theory that describes a state in which no agent can gain utility by deviating from her strategy if the strategies of all other agents remain fixed. In this case, the strategy of each agent is a best response to the strategies of all other agents. Since the central objective of the system does not have to be aligned with the utilities of each individual agent, this may lead to agents acting in a way that is detrimental to the central objective. The *price of anarchy (POA)* (cf. Koutsoupias & Papadimitriou, 1999) measures the maximum loss in solution quality an NE can suffer with regards to the central objective as compared to the optimum solution. Thus, it quantifies the maximum benefit a global planner can achieve over an NE in the distributed scheduling system where agents are allowed to act strategically.

For a distributed minimization problem with objective function γ that allows for a set of instances $\mathcal{I}$, the POA can be formally defined as follows. For each instance $I \in \mathcal{I}$, let $N(I)$ denote the set of NEs and $OPT(I)$ denote an optimal solution. If $NE(I) \neq \emptyset$, then the POA of this coordination problem is

$$\text{POA} = \max_{I \in \mathcal{I}} \max_{\text{NE} \in N(I)} \frac{\gamma(\text{NE})}{\gamma(\text{OPT}(I))}.$$

In case of a maximization problem, the POA is defined as

$$\text{POA} = \max_{I \in \mathcal{I}} \max_{\text{NE} \in N(I)} \frac{\gamma(\text{OPT}(I))}{\gamma(\text{NE})}.$$

In distributed scheduling, an NE is the state in which - based on the current schedule - no job would want to be assigned to a different machine because its utility would not improve by switching. Mechanism design for distributed machine scheduling mostly focuses on the central objectives of minimizing the makespan, see Immorlica et al. (2009), or minimizing the total completion time, see Cole et al. (2011); Hoeksma & Uetz (2011). The utility ut_j of each job j is either assumed to be directly proportional to its own completion time C_j (the 'sequencing model') or based on the makespan of the machine the job is scheduled on (the 'congestion model', which is closely related to network congestion games). Common local policies that are considered consist of sequencing procedures where each machine sequences its assigned jobs according to their processing times, either in non-decreasing (*shortest processing time first, SPT*) or non-increasing (*longest processing time first, LPT*) order.

The POA is better the closer it is to 1 and in many cases it is tightly connected to the approximation ratio of greedy algorithms for the underlying optimization problem. Similar to the behavior of selfish agents, a greedy algorithm is based on iteratively selecting the best local option which may not be beneficial for the central objective in the end. In many cases, the POA can be derived from the approximation ratio of an algorithm for the corresponding optimization problem. For instance, for the central objective of minimizing the makespan of parallel machines, Immorlica et al. (2009) could prove that the solutions of the shortest-first greedy algorithm by Graham (1966) are equivalent to the set of pure NEs for the corresponding distributed scheduling problem when SPT is chosen as the local policy for all machines. Similar results can be obtained for minimizing total completion time (Hoeksma & Uetz, 2011; Ibarra & Kim, 1977). These results show that there is a tight correspondence between greedy heuristics and selfish behavior of agents, which themselves can be seen as deriving their strategies in a greedy fashion, based on the local policies of the machines.

In addition to its processing times, a job may also have a publicly known due date d_j until which it has to be completed. Approximation ratios as well as the POA are not well-defined for objective functions that deal with the tardiness of jobs, since the optimal objective value may be zero (when there are no tardy jobs) even for non-trivial instances. In this case, these ratios immediately become unbounded unless an approximation algorithm guarantees an optimum solution for those instances. Therefore, Lee et al. (2012) propose an alternative metric to measure the performance loss in these cases: the absolute price of anarchy (APOA) which is calculated for a minimization problem as

$$\text{APOA} = \max_{I \in \mathcal{I}} \max_{\text{NE} \in N(I)} \gamma(\text{NE}) - \gamma(\text{OPT}(I)).$$

In case of a maximization problem function, analogous to the POA, we can then define the APOA as

$$\text{APOA} = \max_{I \in \mathcal{I}} \max_{\text{NE} \in N(I)} \gamma(\text{OPT}(I)) - \gamma(\text{NE}).$$

Similar to the close relation between the POA and approximation algorithms that yield relative approximation guarantees, the absolute price of anarchy can be linked to approximation algorithms that have an absolute, additive approximation factor. Up to now, Lee et al. (2012) were the only authors to discuss coordination mechanisms for central objectives based on tardiness, namely minimization of maximum tardiness, total tardiness, and (weighted) number of tardy jobs. In addition to SPT and LPT, they also considered the local policy of sequencing jobs in non-decreasing order of their due dates (*earliest due date first, EDD*). The authors provide APOAs for all tardiness-based objective functions discussed above for identical parallel machines, utility function $ut_j = -C_j$ of the agents, and local policies based on sequencing jobs in SPT, LPT, and EDD. Their APOAs all show a very large loss in solution quality in the case of distributed scheduling, especially for central objectives of minimizing total tardiness and number of tardy jobs.

SPT leads to low losses in welfare when designing a protocol for makespan and total completion time objectives because SPT is actually an optimal scheduling procedure for the respective single machine problems. Similarly, EDD is optimal for minimizing the maximum tardiness on a single machine and leads to reasonably good results when used as a local policy in the distributed environment with maximum tardiness as the central objective. However, for minimizing other tardiness-related objective functions, these simple sorting based procedures lead to very bad schedules even in the single machine case and hence constitute poor local policies in the distributed environment.

In this paper, we focus on the central objective of minimizing the number of tardy jobs. We show that not only SPT, LPT, and EDD, but any such simple sorting based policy leads to very bad results. Hence, such policies should be avoided by a designer. Minimizing the number of tardy jobs becomes an *NP*-hard optimization problem for a global planner in the case of two or more identical parallel machines. For a single machine, however, the problem can be solved in polynomial time with the algorithm of Moore (1968). Therefore, given that optimal single machine scheduling procedures turned out to be good local policies for other objectives it seems reasonable to apply the Moore-Hodgson algorithm as a local policy. We investigate this local policy and show that it leads to reasonably good results.

Since ratios for POA and approximation algorithms can become unbounded for minimizing $\sum_j U_j$, the scheduling literature sometimes focuses on maximizing the number of on-time jobs $\sum_j (1 - U_j)$ (Chen, Potts, & Woeginger, 1998; Graham, Lawler, Lenstra, & Kan, 1979). For the single machine case, Sahni (1976) gave a fully

polynomial time approximation scheme which is even applicable for minimizing the weighted number of tardy jobs $\sum w_j U_j$ (or, equivalent, maximizing the weighted number of on-time jobs). For parallel machines, no such approximation scheme exists. For two identical machines, Leung & Yu (1994) showed that a simple heuristic achieves a 4/3-approximation. Later, Bar-Noy, Guha, Naor, & Schieber (2001) designed an algorithm that achieves an $(1+1/m)^m/((1+1/m)^m-1)$ - approximation ratio for m identical parallel machines and a 2 - approximation for unrelated parallel machines. Several other authors focused on designing heuristics and evaluated their performance empirically against each other and against optimal solutions without providing approximation guarantees (Chen & Powell, 1999; Ho & Chang, 1995; M'Hallah & Bulfin, 2005).

Contribution

In this paper, we present several theoretical contributions to scheduling jobs on parallel machines with the central objective to minimize the number of tardy jobs (or, equivalent, to maximize the number of non-tardy jobs), both in the distributed as well as in the classical scheduling environment. We perform a thorough analysis of the (A)POA in the distributed scheduling environment. We consider identical, related, as well as unrelated parallel machines and a variety of local policies and job utilities. For the latter, we extend the line of research from the literature where the utility function of a job depends on the job's completion time only. In many environments, a job might not care about its specific completion time as long as it is processed on time and, hence, may not switch the machine unless it is tardy now and becomes on time after switching. To the best of our knowledge we derive the first results for tardiness related utility functions.

Literature on coordination mechanisms in scheduling, especially for due-date related objective function, focuses primarily on local policies that sort jobs according to one of their properties, either their processing time or due date. In this paper, we define a notion of simple sorting based scheduling procedures that generalizes these procedures like SPT, LPT, and EDD, all of which are popular local policies for coordination mechanisms. We show that all policies in this class fail to achieve a finite POA which necessitates more involved local policies in order achieve a finite POA. As our main contributions we show that using Moore-Hodgson's algorithm as a natural choice for such a more involved local policy, a POA of $(2m-1)/m$ for identical machines and a POA of 2 for related and unrelated machines can be obtained.

On top, the structural results gained by analyzing the distributed scheduling environment allow us to prove approximation ratios for the classical $\mathcal{NP}$-hard parallel machine scheduling problem with a central planner. We are the first to prove these ratios for two greedy heuristics for the multi-machine problem. While these heuristics have been in use for a long time, only with our new insights are we able to prove approximation ratios for them, turning these algorithms from simple heuristics into actual approximation algorithms. For the problem setting with three identical parallel machines, we prove that these approximation algorithms therefore achieve the currently best known approximation ratio and outperform more complicated approximation algorithms in this setting.

The remainder of this paper is structured as follows. In Section 2, we introduce and define the scheduling problem formally. We derive some general results on NEs and the (A)POA in Section 3, before discussing simple sorting based local policies in Section 4 and Moore-Hodgson's algorithm as a local policy in Section 5. Finally, we discuss approximation ratios in Section 6 and conclude the paper in Section 7.

2. Scheduling environments

In the following, we describe the scheduling environment, both in the central planning as well as in the distributed setting. First, we introduce the notation that is common for both.

Let $J=\{1,\dots,n\}$ be a set of jobs. Each $j\in J$ has a due date d_j. We assume that jobs are numbered in non-decreasing order of their due dates. Furthermore, let $M=\{M_1,\dots,M_m\}$ be a set of m machines. Each machine can process at most one job at a time and each job is processed on a single machine without preemption. We consider three different settings with regard to the machines: machines are identical, related, or unrelated. In the case of identical machines, the processing time of each job $j\in J$ is independent of the machine it is processed on and, thus, is represented by p_j. If machines are related, each job $j\in J$ has a workload w_j and each machine $M_i\in M$ has a speed s_i. The processing time of job j processed on machine M_i is $p_{j,i}=w_j/s_i$. For unrelated machines the processing time of job j processed on machine M_i is given as an arbitrary value $p_{j,i}$.

A schedule is comprised of an assignment of each job $j\in J$ to a machine where it is processed on, as well as a completion time C_j such that no two jobs are processed at the same time on the same machine. Given a schedule, the tardiness of a job j is defined by $T_j=\max\{0,C_j-d_j\}$. A job which is completed after its due date is considered to be tardy, otherwise it is considered to be scheduled on time. For each job j, the binary variable U_j indicates whether the job is tardy, being set to $U_j=1$ if and only if the job is tardy.

For classical scheduling problems with a central planner, Graham et al. (1979) introduced a three-field classification $\alpha|\beta|\gamma$ where α denotes the machine environment, β denotes special job characteristics, and γ defines the objective function. In this classification scheme, identical parallel machines are indicated by $\alpha=P$, related parallel machines by $\alpha=Q$, and unrelated parallel machines by $\alpha=R$. Note that unrelated machines generalize related machines which in turn generalize identical machines. In this paper, we focus on the objective function $\gamma=\max\sum_{j\in J}(1-U_j)$. The objective function counts the number of non-tardy jobs and, thus, we seek to maximize the objective value. For a central planner the problem is then to determine a schedule such that $\sum_{j\in J}(1-U_j)$ is maximized, or $\alpha||\sum(1-U_j)$ with $\alpha\in\{P,Q,R\}$, for short. The β field is empty since we do not consider any special job characteristics. Having in mind that we aim at a maximum number of non-tardy jobs it is easy to see that we cannot benefit from having idle time on machines. Thus, a schedule S can be represented by the sequence of jobs S_i for each machine $M_i\in M$ where each job is in exactly one sequence. A job in S_i is processed on machine i as soon as all preceding jobs in S_i are completed.

In the distributed scheduling settings we consider in this paper, however, no central planner exists. Instead, each job chooses a strategy in the set M of strategies, that is, it decides the machine it is processed on. A strategy profile then is an assignment of jobs to machines. For a strategy profile the actual sequence S_i on machine M_i is determined by the machine's local policy. For the distributed environment, with a slight abuse of notation, we use the $\alpha(\phi)|ut=\beta|\gamma$ classification scheme based on Lee et al. (2012) and Graham et al. (1979). Again, α refers to the machine setting and γ to the central objective of the system. Further, ϕ refers to the local policy of the machines, and β represents the utility function of each job. The utility β of each job in a resulting schedule is a function based on the job's completion time C_j and each job aims at maximizing its utility. In this work, we consider the utility functions $ut_j=-C_j$, $ut_j=-T_j$, and $ut_j=-U_j$.

We assume all machines to apply the same local policy ϕ. In the literature, the most common local policies used within coordination mechanisms include variants of SPT, LPT, or EDD, see, e.g., Hoeksma & Uetz (2011); Immorlica et al. (2009); Lee et al. (2012).

Table 1
POAs and APOAs for $\alpha(\phi)|ut = -\beta|\sum(1-U_j)$ for $\beta \in \{C_j, T_j, U_j\}$.

	POA	Reference	APOA	Reference
$P(MH)$	$\frac{2m-1}{m}$	Theorem 6	$\frac{m-1}{m} \cdot n$	Corollary 1
$P(SSB^\dagger)$	∞	Theorem 1	$\max\{0, n-m\}$	Theorem 2
$\{Q,R\}(MH)$	2	Theorem 7	$\frac{1}{2}n$	Corollary 2
$Q(SSB^\dagger)$	∞	Theorem 1	$n-1$	Theorem 3
$R(SSB^\dagger)$	∞	Theorem 1	$\{n-1, n\}$	Theorem 4

†: SSB local policies - including, but not limited, to SPT, LPT, and EDD.

However, for maximizing the number of on-time jobs, these local policies lead to very bad results, even when considering only a single machine. In fact, we show this for a more general class of local policies.

Definition 1. Let f^p be an injective function $f^p : \mathbb{R} \to \mathbb{N}$ that maps processing times on priority values and does not depend on the jobs' due dates. We call a local policy ϕ of a machine *simple p-sorting based* if it schedules its jobs in non-decreasing order of their priority values implied by f^p with an arbitrary transitive tiebreaker.

Definition 2. Let f^d be an injective function $f^d : \mathbb{R} \to \mathbb{N}$ that maps due dates on priority values and does not depend on the jobs' processing times. We call a local policy ϕ of a machine *simple d-sorting based* if it schedules its jobs in non-decreasing order of their priority values implied by f^d with an arbitrary transitive tiebreaker.

We, furthermore, say that a policy is simple sorting based (SSB) if it is either simple p-sorting based or simple d-sorting based. SSB policies include EDD ($f^d(j) = d_j$), SPT ($f^p(j) = p_j$) or LPT ($f^p(j) = -p_j$). We we will show in Section 4 that for any SSB local policy, the POA is unbounded and hence the welfare loss in the distributed environment can be arbitrarily large. On the other hand, Moore-Hodgson's algorithm (which we describe in detail in Section 5) is not SSB since the position of a job in the schedule depends on the properties of the job as well as the properties of other jobs. In Section 5, we show that far better prices of anarchy can be obtained when using Moore-Hodgson's algorithm as a local policy.

We consider each distributed scheduling setting $\alpha(\phi)|ut = \beta|\sum(1-U_j)$ with $\alpha \in \{P, Q, R\}$, $\phi \in \{SSB, MH\}$, and $\beta \in \{-C_j, -T_j, -U_j\}$. In the remainder, we seek to determine the POA and APOA for those settings. Table 1 summarizes our results. Since the APOA is identical for objective functions $\max\sum(1-U_j)$ and $\min\sum U_j$, the results of Table 1 with regards to the APOA hold for both objective functions. Note that the POA and APOA, respectively, holds for all utility functions discussed above.

3. Nash equilibria and (absolute) prices of anarchy

Throughout the paper, we say that an assignment of jobs to machines is an $(\alpha(\phi), \beta)$-NE for instance I of distributed scheduling setting $\alpha(\phi)|ut = \beta|\sum(1-U_j)$ if the assignment is an NE for I, that is no job can increase its utility by unilaterally deviating from the assignment by switching its machine. We further say that a schedule is an $(\alpha(\phi), \beta)$-schedule if it is the schedule resulting from an $(\alpha(\phi), \beta)$-NE. When discussing NEs we have to specify the utility function of jobs. We first establish a hierarchy between the three functions under consideration.

It can be shown that a pure strategy Nash equilibrium always exists in the scheduling environments we discuss in this paper. We will show this in Section 6 by proving that the result of a greedy scheduling heuristic terminates in a Nash equilibrium for any input.

Lemma 1. *Let I be an instance of $\alpha(\phi)|ut_j = -C_j|\sum(1-U_j)$ and let I' be an instance of $\alpha(\phi)|ut_j = -T_j|\sum(1-U_j)$ with identical sets of machines and jobs. Then, each $(\alpha(\phi), -C_j)$-NE of I is an $(\alpha(\phi), -T_j)$-NE of I'.*

Lemma 2. *Let I be an instance of $\alpha(\phi)|ut_j = -T_j|\sum(1-U_j)$ and let I' be an instance of $\alpha(\phi)|ut_j = -U_j|\sum(1-U_j)$ with identical sets of machines and jobs. Then, each $(\alpha(\phi), -T_j)$-NE of I is an $(\alpha(\phi), -U_j)$-NE of I'.*

We omit a formal proof and give a short elucidation instead. For any schedule in which jobs cannot improve on the utility function $ut = -C_j$, they cannot improve on utility function $ut = -T_j$ either, because tardiness can be decreased only by lowering completion time. Similarly, if a job cannot improve on utility function $ut = -T_j$, it cannot improve on $ut = -U_j$. This has two important consequences for our analysis of (absolute) prices of anarchy.

- An upper bound for the (A)POA with utility function $ut = -U_j$ constitutes an upper bound on the (A)POA for the other two utility functions.
- A lower bound on the (A)POA with utility function $ut = -C_j$ constitutes a lower bound on the (A)POA for the other two utility functions.

In this paper, we make use of this observation in order to prove tight bounds on the (A)POA, proving upper bounds for $ut = -U_j$ and lower bounds for $ut = -C_j$. The former, in particular, makes the analysis of (A)POAs a lot easier since we only need to consider two states of jobs in our schedules, whether they are on time or tardy.

Naturally, there are some connections between the POA and the APOA. In Appendix A, we prove the following relationship between (A)POAs for objective function $\sum(1-U_j)$ that hold for arbitrary α, ϕ and β.

Lemma 3. *Consider $\alpha(\phi)|ut = \beta|\sum(1-U_j)$. If the POA is bounded from above by a finite $k \geq 1$, the APOA is bounded from above by $\tilde{k} = \left(1 - \frac{1}{k}\right)n$. Further, if an instance I that proves k as the lower bound of the POA has no tardy job in optimum schedules, this instance also proves $\tilde{k}$ as the lower bound for the APOA.*

4. Local policies based on simple sorting

While applying SSB local policies does often not lead to a big loss in social welfare when minimizing the makespan or the total completion time (Immorlica et al., 2009), the APOAs for settings where the central objective function is based on jobs' due dates tend to be very high (Lee et al., 2012). In this section, we prove that SSB local policies lead to huge losses in welfare for the central objective of maximizing the number of on-time jobs. First, we show that even for a single machine, SSB local policies lead to arbitrarily bad results.

Lemma 4. *For any simple p-sorting based policy ϕ with function f^p and any integer k, there exists an instance with a single machine and k jobs, such that all jobs are scheduled on time in the optimum schedule while only a single job is scheduled on time by ϕ.*

Proof. We consider k distinct processing times $1, \ldots, k$. Let σ be the sequence of these processing times such that $f^p(\sigma_1) < \ldots < f^p(\sigma_k)$ where σ_l, $l = 1, \ldots, k$, is the lth entry of σ. We now construct an instance as follows.

We have job set $J = \{1, \ldots, k\}$ with $p_j = \sigma_j$ for each job $j = 1, \ldots, k$. Note that SSB policy ϕ yields job sequence $1, \ldots, k$, then. We determine due dates of jobs as $d_1 = \sum_{j \in J} p_j$ and $d_j = \sum_{j'=2}^{j} p_{j'}$ for each $j = 2, \ldots, k$. It is not hard to see that, employing ϕ, only job 1 is on time. However, if jobs are in sequence $2, \ldots, k, 1$ all jobs are on time. □

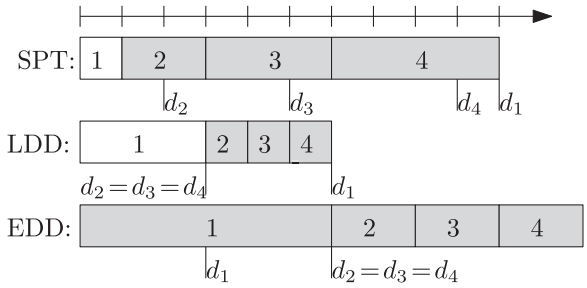

Fig. 1. Examples of SSB schedules in the single machine environment.

Lemma 5. *For any simple d-sorting based policy ϕ with function f^d and any integer k, there exists an instance with a single machine and k jobs, such that $k-1$ more jobs are on time in the optimum schedule than in the schedule obtained by ϕ.*

Proof. We consider due dates $k-1$ and $2k-2$ and distinguish two cases with respect to $f^d(k-1)$ and $f^d(2k-2)$.

If $f^d(k-1) > f^d(2k-2)$, then we construct an instance as follows. We have job set $J = \{1, \ldots, k\}$ with $p_1 = k-1$, $d_1 = 2k-2$ as well as $p_j = 1$ and $d_j = k-1$ for all remaining jobs $j = 2, \ldots, k$. Note that SSB policy ϕ has job 1 as the first jobs. Since the remaining jobs are identical we can assume w.l.o.g. that the job sequence is $1, 2, \ldots, k$. It is not hard to see that, employing ϕ, only job 1 is on time. However, if jobs are in sequence $2, \ldots, k, 1$ all jobs are on time.

If $f^d(k-1) < f^d(2k-2)$, then we construct an instance as follows. We have job set $J = \{1, \ldots, k\}$ with $p_1 = 2k-2$, $d_1 = k-1$ as well as $p_j = 2$ and $d_j = 2k-2$ for all remaining jobs $j = 2, \ldots, k$. Again, w.l.o.g. we can assume that SSB policy ϕ yields job sequence $1, 2, \ldots, k$. It is not hard to see that ϕ leads to all jobs being tardy. However, if jobs are in sequence $2, \ldots, k, 1$ all jobs but job 1 are on time. □

Figure 1 depicts schedules yielded by SSB policies SPT, latest due date (LDD), and EDD for instances as constructed in the proofs of Lemmas 4 and 5.

Since a single machine can schedule its jobs arbitrarily bad using SSB policies, this result immediately leads to an unbounded POA in the distributed scheduling environment.

Theorem 1. *Let ϕ be an SSB local policy. Then, the POA for $\alpha(\phi)|ut_j = \beta|\sum(1-U_j)$ is unbounded for each $\alpha \in \{P, Q, R\}$, each $\beta \in \{-C_j, -T_j, -U_j\}$, and any number of machines.*

Proof. In the following, we consider the case $\alpha = P$ and $\beta = -C_j$ since any lower bound on the POA for this case immediately carries over to the others.

For any SSB local policy ϕ we can easily construct an instance in the following way: Let $J = \{1, \ldots, k\}$ be jobs as constructed in the proofs of Lemmas 4 and 5 as a "bad" instance for ϕ (depending on whether ϕ is based on a function f^p or a function f^d). Copy these jobs m times, resulting in a job set $\bar{J} = \bigcup_{i=1,\ldots,m} J_i$ where $J_i = \{j_i | j \in J\}$ and $p_{j_i} = p_j, d_{j_i} = d_j$ for each $M_i \in M$ and $j \in J$.

If ϕ is simple p-sorting based, assigning all jobs in J_i to machine M_i yields an $(\alpha(\phi), -C_j)$-NE (no job can be processed earlier on any other machine due to the local policies).

If ϕ is simple d-sorting based, we also assign jobs in J_i to machine M_i. Note that all but the first jobs on each machine are identical. Thus, this assignment is an $(\alpha(\phi), -C_j)$-NE if the lth job, $l \geq 2 < k$, on each machine gets preferred to the $(l+1)$st job on each machine. It is not hard to see that we can arrange for that for an arbitrary transitive tiebreaker.

In each of the NEs constructed above, only the first job on each machine might be on time, while the optimum would allow for at least $(k-1)m$ jobs to be scheduled on time. Hence for any $k \in \mathbb{N}$ there exists an instance I with an $(\alpha(\phi), -C_j)$-NE such that $\frac{\gamma(OPT(I))}{\gamma(NE)} \geq \frac{(k-1)m}{m} = k-1$ and, therefore, the POA is unbounded. □

Since the POAs are not bounded, we cannot apply Lemma 3 to obtain the APOAs for these SSB local policies. Lee et al. (2012) already proved an upper bound of $\max\{0, n-m\}$ of the APOA for $P(\phi)|ut = -C_j|\sum U_j$ with an arbitrary local policy ϕ. Moreover, the authors showed this bound to be tight for $\phi = EDD$. In the following, we prove that the APOA is equally bad for any SSB local policy and all utility functions discussed in this paper. First, we extend the result by Lee et al. (2012) for the upper bound. The proof immediately follows from the one of Lee et al. (2012), we recite it in Appendix A in order to keep this paper self-contained.

Lemma 6. *The APOA for $P(\phi)|ut = \beta|\sum U_j$ is bounded from above by $\max\{0, n-m\}$ for any local policy ϕ, each $\beta \in \{-C_j, -T_j, -U_j\}$, and each number m of machines.*

Theorem 2. *The APOA for $P(\phi)|ut = \beta|\sum U_j$ is $\max\{0, n-m\}$ for any SSB local policy ϕ, each $\beta \in \{-C_j, -T_j, -U_j\}$, and any number of machines m.*

Proof. The upper bound is given by Lemma 6. The lower bound can immediately be obtained from the instance with $n = km$ jobs and the $(P(\phi), \beta)$-NE constructed in the proof of Theorem 1 since the optimum schedule has at least $(k-1)m = km - m = n - m$ more jobs on time than the $(P(\phi), \beta)$-NE. □

For related machines, the situation is even more dire. To recognize this, we again fall back on the "bad" instances in the proofs of Lemmas 4 and 5.

Theorem 3. *The APOA for $Q(\phi)|ut = \beta|\sum U_j$ is $n-1$ for any SSB local policy ϕ, each $\beta \in \{-C_j, -T_j, -U_j\}$, and any number of machines m.*

Proof. First, we focus on $\alpha = Q$ and $\beta = -C_j$ for the lower bound. Similar to Theorem 1, an instance can be constructed for any SSB local policy ϕ. Let $J = \{1, \ldots, n\}$ be jobs as constructed in the proofs of Lemmas 4 and 5 as a "bad" instance for ϕ (depending on whether ϕ is based on a function f^p or a function f^d). Let the speed on machine M_1 be 1 and the speeds of all other machines be $1/(\sum_{j \in J} p_j + 1)$.

Assigning all jobs in J to machine M_1 yields a $(Q(\phi), -C_j)$-NE (no job can be processed earlier on any other machine due to the machine speeds). In the optimum schedule, jobs $1, \ldots, n$ are also scheduled on machine M_1 and $n-1$ more jobs are on time than in the $(Q(\phi), -C_j)$-schedule. Hence, the APOA is bounded from below by $n-1$.

For the upper bound, we consider a $(Q(\phi), -U_j)$-schedule and distinguish two cases.

- If at least one job is scheduled on the fastest machine we consider the first among those jobs. If it is tardy, it is tardy in each schedule and, hence, an optimum schedule can have at most $n-1$ more on-time jobs. If it is on time, again, an optimum schedule can have at most $n-1$ more on-time jobs.
- If no job is scheduled on the fastest machine, then any job that would move to this machine would be scheduled as the first job. We now consider the first job j on an arbitrary other machine. If j is tardy, then it would be tardy when moving to the fastest machine, as well, since otherwise we would not have a $(Q(\phi), -U_j)$-schedule. But then it is tardy in each schedule and, hence, an optimum schedule can have at most $n-1$ more on-time jobs. If j is on time on its current machine, again, an optimum schedule can have at most $n-1$ more on-time jobs.

□

Finally, for unrelated machines the APOA may even reach n, depending on the local policy ϕ.

Theorem 4. *The APOAs for $R(\phi)|ut=\beta|\sum U_j$ lie between $n-1$ and n (depending on the local policy ϕ) for each each $\beta \in \{-C_j, -T_j, -U_j\}$, and any number of machines.*

Proof. Clearly, a lower bound for the APOA of $n-1$ for each local policy is implied by Theorem 3. Indeed, this lower bound may be reached. To see this, consider $\phi = LDD$ and an arbitrary $(R(LDD), \beta)$-schedule. Assume that the job with the highest due date is not on time. Since it would be scheduled first on any machine, it cannot be scheduled on time in any schedule. However, if this job is on time, at most $n-1$ more jobs can be on time in an optimum schedule. Hence, the APOA is bounded from above by $n-1$.

For other local policies, however, the APOA reaches n. Consider $\phi = LPT$ and the following instance with $n > m$ jobs. For each job $j = 1, \ldots, m$ and each machine $M_i \in M$ let $p_{j,i} = 2$ if $j = i$ and $p_{j,i} = 1$ if $j \neq i$. Furthermore, let $d_j = 1$ for each job $j = 1, \ldots, m$. For each job $j > m$ let $p_{i,j} = 0$ for each $M_i \in M$ and $d_j = 0$. Assigning each job $j \leq m$ to machine M_j and arbitrarily distributing the remaining jobs yields an $(R(LPT), -C_j)$-NE with no job on time. However, having all zero-length jobs starting at time zero and each job $j = 2, \ldots, m$ assigned to machine M_{j-1} as well as job 1 to machine M_m we obtain a schedule with no tardy job. □

In the next section we show that using Moore-Hodgson's algorithm as local policy leads to way lower welfare losses in the distributed scheduling environment.

5. Moore-Hodgson as a local policy

For a single machine, Moore and Hodgson's algorithm determines a schedule with maximum number of on-time jobs in polynomial time. Hence, it is a prime candidate to be used as a local policy in the distributed scheduling environment. In this section, we will first revisit this algorithm since we heavily rely on it for the remainder of the paper. We refer the reader to Appendix B for the formal and technical proofs of the lemmas.

Algorithm 1 depicts the algorithm as presented in Blazewicz et al. (2019) and Pinedo (2016) with a minor modification which is important in order to reach a unique outcome. The original algorithm is referenced in Moore (1968) as proposed by T.J. Hodgson as a slight modification of the algorithm by Moore. Note that we do not have to distinguish between different machine settings when focusing on a single machine and, therefore, we will use p_j as the actual processing time of job $j \in J$ on the only machine under consideration.

Input: Jobs $j = 1, \ldots, |J|$ in EDD-order
Initiate an empty schedule $S \leftarrow []$ and a set of tardy jobs $\mathcal{T} \leftarrow \emptyset$
for $j = 1, \ldots, |J|$ **do**
  Append j to S
  if j is late **then**
    Remove the job $j' \in S$ with $p_{j'} = \max\{p_{j''} \mid j'' \in S\}$ (in case of ties, the job with the lowest job number) from S and add it to $\mathcal{T}$
  end if
end for
Append all jobs in $\mathcal{T}$ to S in order of jobs indices;

Algorithm 1: Moore-Hodgson's algorithm with tiebreakers.

Starting from an empty schedule S and an empty set $\mathcal{T}$, in each round of the algorithm a job with the smallest due date among jobs that have not been considered yet is selected and appended to S. If j is on time, the algorithm proceeds. Otherwise, a longest job in S is removed from S and added to $\mathcal{T}$. Here is where we deviate slightly from the original variant of the algorithm. We require to remove not an arbitrary longest job in S but the longest job with the smallest job number. When all jobs have been considered, jobs in S are on time and jobs in $\mathcal{T}$ are appended to S in arbitrary order and, thus, tardy.

We say that a set I of jobs is feasible if the jobs in I are non-tardy when scheduled in EDD. In the following we will compare feasible sets of jobs of the same cardinality depending on their EDD sequence of job numbers. Consider job set J and two feasible subsets I and I' of J such that $|I| = |I'|$. Let σ and σ' be the respective EDD sequences. We say that I is lexicographically larger than I' if and only if σ is lexicographically larger than σ' (i.e., at the first slot k at which the elements of σ and σ' do not coincide, $\sigma_k > \sigma'_k$ holds). Note that unless $I = I'$ one set is lexicographically larger than the other.

It is well known that Moore-Hodgson's algorithm minimizes the number of tardy jobs (Moore, 1968). Our modification, obviously, does not change that. In fact, an even stronger result can be obtained for Moore-Hodgson's algorithm and, thus, for Algorithm 1.

Lemma 7. *Moore-Hodgson's algorithm finds a maximum cardinality feasible set of jobs with minimum total processing time.*

With our modification the feasible job set determined by Algorithm 1 is unique in that we do not only obtain a maximum cardinality feasible set of jobs with minimum total processing time, but among sets with this property we achieve the one which is lexicographically largest.

Lemma 8. *Algorithm 1 finds the lexicographically largest among all maximum cardinality feasible sets of jobs with minimum total processing time.*

We provide a short example to highlight the uniqueness of the outcome of Algorithm 1.

Example 1. Consider an instance with six jobs as described in Fig. 2. Figure 2a) shows the iterations of Algorithm 1 and Fig. 2b) its final result. In comparison, Fig. 2c) depicts another schedule with three on-time jobs but larger total processing time and Fig. 2d) shows an other schedule with minimum total processing time that is lexicographically smaller (since it includes job 5 instead of job 6).

We account for the uniqueness of the result of Algorithm 1 by introducing the following notation.

Definition 3. Let J be a set of jobs. A subset $I \subseteq J$ is called *Moore-Hodgson optimal (MH-optimal)* with regards to J, if Algorithm 1 returns a schedule in which jobs I are on time when applied to J. We refer to the unique set that is MH-optimal with regards to J as $MH(J)$. Finally, we say I is *Moore-Hodgson closed (MH-closed)* against a job set J' if $I = MH(I \cup J')$.

In Section 5.1 we will focus on the structure of $MH(J)$ as a result of Algorithm 1 applied as a local policy for a given set J of jobs on this machine. In Section 5.2 we use these insights in order to analyze NEs obtained when Algorithm 1 is employed as the local policy. Finally, in Section 5.3 we determine the (A)POA of $\alpha(MH)|ut=\beta|\sum(1-U_j)$ for all $\alpha \in \{P, Q, R\}$ and $\beta \in \{-C_j, -T_j, -U_j\}$.

5.1. Properties of Moore-Hodgson optimal sets

In the following, we provide several properties of the MH-optimal sets that are produced by Algorithm 1. We are particularly interested in how robust these sets are against changes

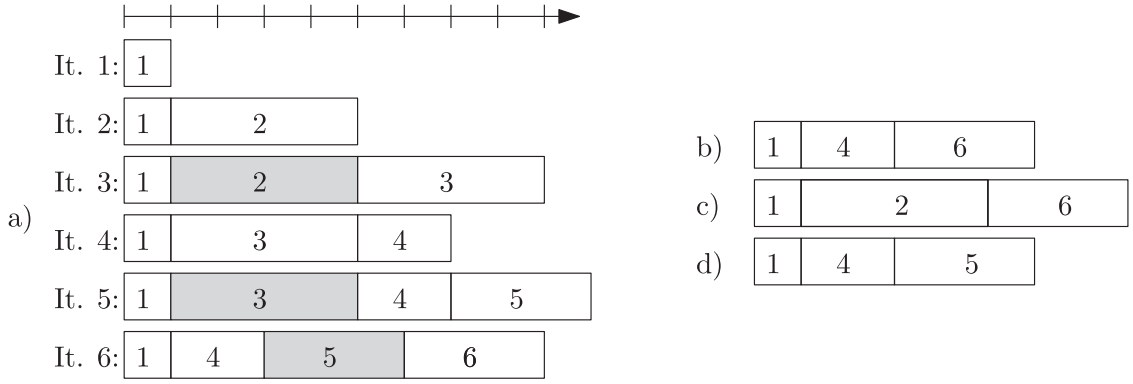

Fig. 2. Schedules with a maximum number of on-time jobs for Example 1. a) Iterations of Algorithm 1, b) Final result of Algorithm 1, c) Subset of jobs with higher processing time, d) Lexicographically smaller subset of jobs with minimum processing time.

of the input of the algorithm. These results are important when we discuss Algorithm 1 as a local policy for parallel machines in Section 5.2 since a job changing its strategy (i.e. switching its machine) is equivalent to adding it to the input of Algorithm 1 on the new machine and removing it from the input on the old machine.

The first Lemma states that if a job is added to the input set but it cannot be scheduled on time, then it has no effect on the set of jobs that are on time.

Lemma 9. *Consider job set J and job $j \notin J$. If $j \notin MH(J \cup \{j\})$, then $MH(J) = MH(J \cup \{j\})$.*

In particular, jobs which are scheduled late by Algorithm 1 have no affect on on-time jobs. The following two lemmas strengthen this result even further, showing that this holds for all subsets of jobs that are scheduled late by the algorithm.

Lemma 10. *Let $I = MH(J)$. Then, $I = MH(J')$ for each J', $I \subseteq J' \subset J$.*

Lemma 11. *If $I = MH(I \cup \{j\})$ for each $j \in J$ with $I \cap J = \emptyset$, then $I = MH(I \cup A)$ for each $A \subseteq J$.*

Finally, we show that if a job is added to the input and can be scheduled non-tardy along all the jobs in the current MH-optimal set, the new MH-optimal set of the extended input will not contain any other additional jobs.

Lemma 12. *Let $I = MH(J)$ and $j \notin J$. If $I \cup \{j\}$ is feasible, then $I \cup \{j\} = MH(J \cup \{j\})$.*

5.2. Moore-Hodgson-Optimal schedules on parallel machines

We now extend the discussion to the case with multiple parallel machines where the jobs assigned to each machine are sequenced by Algorithm 1. In the following, we lay emphasis on $ut_j = -U_j$. The properties we derive enable us to determine upper bounds for (A)POAs for $\alpha(MH)|ut = -U_j|\sum(1-U_j)$ and, thus, for $\alpha(MH)|ut = \beta|\sum(1-U_j)$ with $\beta \in \{-C_j, -T_j\}$ in Section 5.3. Note that under $ut_j = -U_j$ a job has an incentive to switch machines only if it is tardy.

First, we show that a schedule constitutes an NE if and only if the on-time jobs on all machines are MH-closed against the tardy jobs.

Lemma 13. *Let S be a schedule for an instance I of $\alpha(MH)|ut_j = -U_j|\sum(1-U_j)$ with job set J_i for each machine $M_i \in M$. Let $T = \bigcup_{M_i \in M}(J_i \setminus MH(J_i))$ be the set of tardy jobs. S is an $(\alpha(MH), -U_j)$-schedule for I, if and only if $MH(J_i) = MH(J_i \cup T)$ for each machine $M_i \in M$.*

The next lemma states that there is always an optimal schedule where the set of non-tardy jobs on each machine is MH-closed against the set of tardy jobs.

Lemma 14. *There exists an optimal schedule to $\alpha||\sum U_j$ where $MH(J_i) = MH(J_i \cup T)$ for each machine $M_i \in M$ and the set T of tardy jobs.*

Note that Lemmas 13 and 14 imply that there is an optimal $(\alpha(MH), -U_j)$-schedule. Hence, any loss in welfare stems only from converging to a bad

$(\alpha(MH), -U_j)$-schedule (instead of a good one). The concept of loss of welfare in the best NE is known in the literature as price of stability (Nisan, Roughgarden, Tardos, & Vazirani, 2007). For $\alpha(MH)|ut = -U_j|\sum(1-U_j)$ the price of stability is 1.

Based on the previous lemmas, we can derive the following structural result which serves as the backbone for analyzing upper bounds on the POA as well as the approximation ratio.

Lemma 15. *Let I be an instance of $\alpha(MH)|ut_j = -U_j|\sum(1-U_j)$. Let S be an $(\alpha(MH), -U_j)$-schedule for I with set J_i of jobs assigned to each machine M_i and the set T of tardy jobs. Then, $|MH(T \cup MH(J_i) \setminus A)| \leq |MH(J_i)|$ for each machine $M_i \in M$ and each $A \subseteq MH(J_i)$.*

The lemma bounds the number of jobs which are tardy in the $(\alpha(MH), -U_j)$-schedule but could be scheduled on time if some of the on-time jobs are removed from the schedule.

5.3. Price of anarchy for Moore-Hodgson as a local policy

In this section, we present a complete characterization of the welfare loss in scheduling environments $\alpha(MH)|ut = \beta|\sum(1-U_j)$ for each $\alpha \in \{P, Q, R\}$, each $\beta \in \{-C_j, -T_j, -U_j\}$, and any number of machines.

Theorem 5. *The POA for $P(MH)|ut = -U_j|\sum(1-U_j)$ is bounded from above by $\frac{2m-1}{m}$.*

Proof. Let S be a $(P(MH), -U_j)$-schedule, J_i be the set of jobs on machine M_i for each $i = 1, \ldots, m$, $I_i = MH(J_i)$, and T be the set of tardy jobs. Due to Lemma 13 we have $MH(J_i) = MH(J_i \cup T)$ for each $i = 1, \ldots, m$. Consider an optimal schedule S^O which is an $(P(MH), -U_j)$-schedule (such an optimal schedule exists due to Lemmas 13 and 14).

For each machine M_i we partition the set I_i into disjoint (and possibly empty) subsets $I_{i,i'}$ for $i' = 1, \ldots, m$ and I_i^T such that each $I_{i,i'}$ contains those jobs that are on time on machine $M_{i'}$ in S^O and I_i^T contains those jobs that are late in S^O. We consider jobs in $I_{i,i}$ for each $i = 1, \ldots, m$ to be on the *correct* machine and all jobs in

$I_{i,i'}$ for $i \neq i'$ to be on the *wrong* machine. We denote with $W_i = \bigcup_{i' \neq i} I_{i,i'}$ the set of wrongly scheduled jobs. Concluding, we have $I_i = I_{i,i} \dot\cup W_i \dot\cup I_i^T$.

Since the machines are identical, the numbering of them in S and S^O does not matter. Thus, we can rearrange the machines in such a way that the ratio of correctly scheduled jobs to wrongly scheduled jobs is maximum. W.l.o.g. we assume that this is the case for the initial arrangement. Then, it holds that

$$\sum_{i=1}^{m} |I_{i,i}| \geq \frac{1}{m} \sum_{i=1}^{m} \sum_{i'=1}^{m} |I_{i,i'}|$$

and, hence,

$$\sum_{i=1}^{m} |I_{i,i}| \geq \frac{1}{m-1} \sum_{i=1}^{m} \sum_{i' \neq i=1}^{m} |I_{i,i'}| \tag{1}$$

Denote with $O_i \subseteq T$ the jobs that are scheduled on time on machine M_i in S^O but that are late in S. If we remove all jobs in $A_i := W_i \cup I_i^T$ from machine M_i, we could move all jobs in O_i to M_i and schedule them together with job set $I_{i,i}$ on time. Since $O_i \subseteq T$ and $I_i \setminus A_i = I_{i,i}$ we obtain

$$\begin{aligned} |I_{i,i}| + |O_i| \leq |MH(T \cup I_{i,i})| &= |MH(T \cup I_i \setminus A_i)| \leq |I_i| \\ &= |I_{i,i}| + |W_i| + |I_i^T| \end{aligned}$$

where the second inequality is due to Lemma 15. Note that Lemma 15 applies since S^O is an $(\alpha(MH), -U_j)$-schedule. Hence, $|O_i| \leq |W_i| + |I_i^T|$ and there are at most

$$\sum_{i=1}^{m} (|I_{i,i}| + |W_i|) + \sum_{i=1}^{m} \left(|W_i| + |I_i^T|\right)$$

jobs on time in S^O. Thus,

$$\frac{|S^O|}{|S|} \leq \frac{\sum_{i=1}^{m} (|I_{i,i}|+|W_i|)+\sum_{i=1}^{m} \left(|W_i|+|I_i^T|\right)}{\sum_{M_i \in M} \left(|I_{i,i}|+|W_i|+|I_i^T|\right)} \leq \frac{\sum_{i=1}^{m} (|I_{i,i}|+2|W_i|)}{\sum_{i=1}^{m} (|I_{i,i}|+|W_i|)}$$

because $\sum_{i=1}^{m} \left(|I_{i,i}| + |W_i|\right) \leq \sum_{i=1}^{m} \left(|I_{i,i}| + 2|W_i|\right)$ and, then,

$$\frac{\sum_{i=1}^{m} (|I_{i,i}| + 2|W_i|)}{\sum_{i=1}^{m} (|I_{i,i}| + |W_i|)} \leq \frac{2m-1}{m}$$

due to Equation (1). □

We present a short illustration of how the jobs are partitioned in the proof of Theorem 5.

Example 2. Consider an example with 8 jobs on two machines as depicted in Fig. 3. Schedule S is an $(\alpha(MH), -U_j)$-schedule with 6 jobs scheduled on time while S^O is an optimal $(\alpha(MH), -U_j)$-schedule with 7 on-time jobs. Comparing the two schedules, jobs 1,5,6 and 7 are on the correct machine in S (i.e. in sets $I_{1,1}$ and $I_{2,2}$, respectively) while job 3 is on the wrong machine. Job 4 is scheduled on time in S, but late in S^O, while jobs 2 and 8 are late in S, but scheduled on time in S^O. As shown in Theorem 5, at most one late job can be scheduled on machine M_1 if job 4 is removed. Also, at most one late job can be scheduled on M_2 after job 3 is moved to M_1.

Theorem 6. *The POA for $P(MH)|ut = \beta|\sum(1-U_j)$ is $\frac{2m-1}{m}$ for each $\beta \in \{-C_j, -T_j, -U_j\}$.*

Proof. Theorem 5 states an upper bound for the POA of $\frac{2m-1}{m}$. In the following, we show that this bound is tight for $ut = -C_j$ which carries over to $ut = -T_j$ and $ut = -U_j$ due to Lemmas 1 and 2.

Consider an instance with m machines and $n = 2m-1$ jobs. Jobs $j = 1, \ldots, m$ have processing time of $p_j = 1$ and due date of $d_j = j$. Remaining jobs $j = m+1, \ldots, 2m-1$ have processing time of $p_j = m$ and due date of $d_j = m$. Figure 4 depicts a $(P(MH), -C_j)$-schedule and an optimal schedule, respectively. Scheduling jobs j on machine $M_{((j-1) \bmod m)+1}$ results in a $(P(MH), -C_j)$-schedule where all jobs $j > m$ are late. However, if all jobs $j = 1, \ldots, m$ are scheduled on machine M_1 and the remaining $m-1$ jobs are scheduled individually each on its own machine, all jobs are on time. This proves that the upper bound of $\frac{2m-1}{m}$ on the POA is tight. □

Theorem 7. *The POA for $\alpha(MH)|ut = \beta|\sum(1-U_j)$ is 2 for each $\alpha \in \{Q, R\}$ and each $\beta \in \{-C_j, -T_j, -U_j\}$.*

Proof. For the upper bound, the reasoning in the proof of Theorem 5 holds up to the upper bound of

$$\frac{|S^O|}{|S|} \leq \frac{\sum_{i=1}^{m} (|I_{i,i}|+|W_i|)+\sum_{i=1}^{m} \left(|W_i|+|I_i^T|\right)}{\sum_{i=1}^{m} \left(|I_{i,i}|+|W_i|+|I_i^T|\right)} \leq \frac{\sum_{i=1}^{m} (|I_{i,i}|+2|W_i|)}{\sum_{i=1}^{m} (|I_{i,i}|+|W_i|)}.$$

However, we cannot rely on Equation (1) since machines can no longer be renumbered arbitrarily and, therefore, we conclude

$$\frac{|S^O|}{|S|} \leq \frac{\sum_{i=1}^{m} (|I_{i,i}| + 2|W_i|)}{\sum_{i=1}^{m} (|I_{i,i}| + |W_i|)} \leq 2.$$

A lower bound for $\alpha = Q$ and $\beta = -C_j$ can be constructed as follows. Consider $m = 2$ machines and $n = 2$ jobs with due dates of $d_1 = 2$ and $d_2 = 4$. Jobs 1 and 2 have workloads of $w_1 = 8$ and $w_2 = 4$ and machines M_1 and M_2 have speeds $s_1 = 4$ and $s_2 = 1$. Figure 5 depicts a $(Q(MH), -C_j)$-schedule and an optimal schedule. Both jobs could be scheduled on time in the optimum with job 1 on machine M_1 and job 2 on machine M_2. However, having job 2 preceding job 1 on machine M_1 constitutes a $(Q(MH), -C_j)$-schedule with only job 2 scheduled on time. Obviously, the same instance can be used to derive the lower bound for $\alpha = R$. □

Since all POAs for these distributed scheduling environments are bounded from above, the results in Theorems 6 and 7 can be immediately translated into APOAs for $\alpha(MH)|ut = \beta|\sum(1-U_j)$ due to Lemma 3.

Corollary 1. *The APOA for $P(MH)|ut = \beta|\sum(1-U_j)$ is $\frac{m-1}{2m-1} \cdot n$ for each $\beta \in \{-C_j, -T_j, -U_j\}$.*

Corollary 2. *The APOA for $\alpha(MH)|ut = \beta|\sum(1-U_j)$ is $\frac{1}{2} \cdot n$ for each $\alpha = Q, R$ and $\beta = -C_j, -T_j, -U_j$.*

6. Approximation algorithm

In this section, we apply the insights for distributed scheduling derived in Section 5 to the classical scheduling problem with a central planner in order to construct good optimization procedures. For two identical machines, Leung & Yu (1994) implemented a simple heuristic that achieves a 4/3-approximation. Later, Bar-Noy et al. (2001) designed an algorithm that achieves an $(1 + 1/m)^m/((1 + 1/m)^m - 1)$-approximation factor for m identical parallel machines and a 2-approximation for unrelated parallel machines. Several other authors focused on designing heuristics and evaluated their performance empirically against each other and optimal solutions without providing approximation guarantees (Chen & Powell, 1999; Ho & Chang, 1995; M'Hallah & Bulfin, 2005). In the following, we show that slight modifications of the heuristics proposed by Ho & Chang (1995) indeed are constant-factor approximation algorithms for $\gamma = \max\sum(1-U_j)$ on identical machines. Moreover, we can also derive constant-factor approximation algorithms for $\gamma = \max\sum(1-U_j)$ on related machines or on unrelated machines. We, first, detail the heuristics proposed by Ho & Chang (1995).

In one of the job-focused approaches (named H1S in Ho & Chang (1995)), for each machine a feasible set of jobs is maintained. Jobs in this set are sequenced in EDD for each machine and, thus, in increasing order of job numbers. At the beginning these sets are empty and we consider jobs one by one in SPT order with

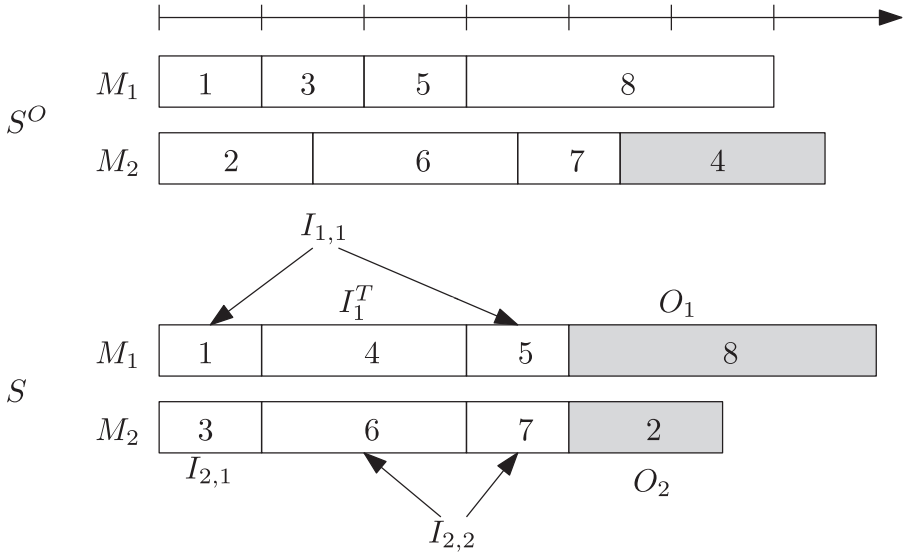

Fig. 3. Illustration of jobs' partition in the proof of Theorem 5.

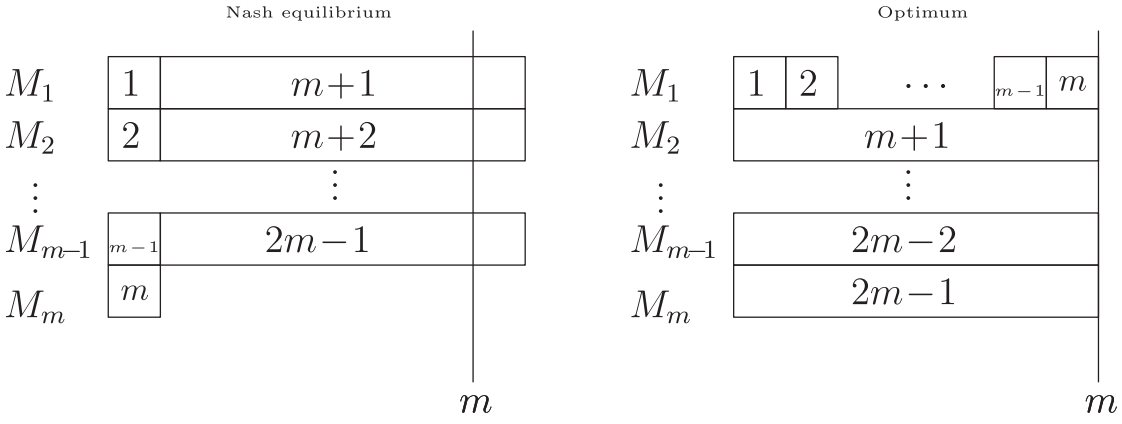

Fig. 4. A lower bound for the POA of $P(MH)|ut = -C_j|\sum(1-U_j)$.

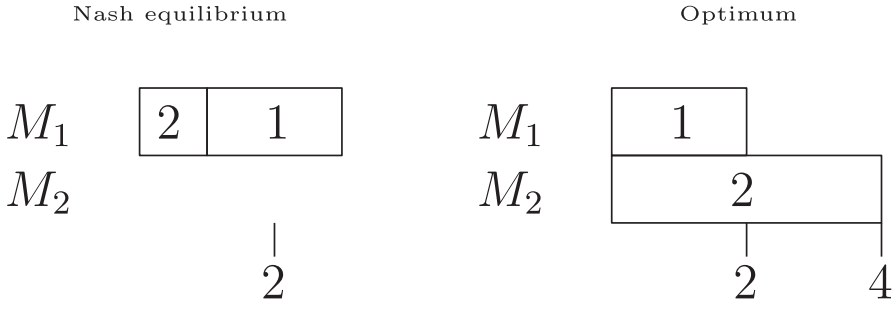

Fig. 5. A lower bound for the POA of $\{Q, R\}(MH)|ut = -C_j|\sum(1-U_j)$.

ties broken by decreasing order of job numbers. This tiebreaker is where we differ slightly from the heuristic proposed by Ho & Chang (1995) as the authors break ties in increasing order of job numbers. We check whether the current job can be added to the feasible set of jobs on any machine and do so – if possible – for an arbitrary machine. If no such machine exists, the job is determined to be late and scheduled at the end of some machine after all other jobs have been considered. Algorithm 2 specifies the job-focused approach.

Algorithm 2 cannot be applied for unrelated machines since jobs do not have a unique workload but processing times depend on jobs as well as on machines instead.

In the machine-focused approach, machines are scheduled iteratively. First, Algorithm 1 is applied to the first machine and all jobs. For each subsequent machine, Algorithm 1 is applied to this machine and the set of jobs that have not been scheduled on time on any previous machine. After the last machine is scheduled, all remaining jobs are scheduled tardy on arbitrary machines. Algorithm 3 specifies the machine-focused approach where MH^i refers to Algorithm 1 applied to machine M_i and, thus, considering processing time $p_{j,i}$ for each job $j \in \mathcal{T}$.

In the machine-focused approach of Ho & Chang (1995), named H2, the authors add another re-optimization step in each iteration after applying Moore-Hogdson's algorithm. In the re-optimization,

Initiate empty sets $J_i \leftarrow \emptyset$ for each machine $M_i \in M$ and a set of tardy jobs $\mathcal{T} \leftarrow \emptyset$
σ = List of jobs in increasing workload, ties broken in decreasing job numbers of jobs
for $k = 1, \ldots, |J|$ **do**
  if all jobs in $J_i \cup \{\sigma_k\}$ in EDD sequence are on-time jobs on some machine M_i **then**
    Add σ_k to J_i
  else
    Add σ_k to $\mathcal{T}$
  end if
end for
Append all jobs in $\mathcal{T}$ to machine M_1 in arbitrary order;

Algorithm 2: Job-focused approximation algorithm for identical and related machines.

Initiate empty sets $J_i \leftarrow \emptyset$ for each machine $M_i \in M$ and a set of tardy jobs $\mathcal{T} \leftarrow J$
for i=1,...,m **do**
  $J_i = MH^i(\mathcal{T})$
  Remove J_i from $\mathcal{T}$
end for
Append all jobs in $\mathcal{T}$ to machine M_1 in arbitrary order;

Algorithm 3: Machine-focused approximation algorithm for parallel machines.

on-time jobs are pairwise interchanged with tardy jobs in order to decrease the total slack of jobs (i.e. the difference between a job's completion time and due date). While doing so leads to better results, empirically, we do not need it for our analysis and, hence, drop this step.

In the following, we prove approximation guarantees of Algorithms 2 and 3 when considering objective function $\sum(1-U_j)$. We start with the job-focused approach.

Theorem 8. *Algorithm 2 is a $\frac{2m-1}{m}$-approximation algorithm for $Pm||\sum(1-U_j)$ and a 2-approximation algorithm for $Qm||\sum(1-U_j)$.*

Proof. We, first, show that Algorithm 2 achieves an approximation ratio of $\frac{2m-1}{m}$ and 2 for $Pm||\sum(1-U_j)$ and $Qm||\sum(1-U_j)$, respectively. For this, we prove that the schedule constructed by the algorithm is an $(\alpha(MH), -U_j)$-schedule for $\alpha \in \{P, Q\}$. We denote with J_i^l the job set scheduled on time on machine M_i at the end of iteration l and with $\mathcal{T}^l$ the set of jobs determined to be tardy by Algorithm 2 at the end of iteration l. We show that in each iteration of the algorithm we have $J_i^l = MH(J_i^l \cup \mathcal{T}^l)$ for each machine $M_i \in M$.

Clearly, this holds true after the first iteration $l = 1$. Assume that it holds after step $l-1$ of the algorithm and consider step l (i.e. when job σ_l has to be scheduled). If σ_l is added to J_i^{l-1} and all jobs in $J_i^l = J_i^{l-1} \cup \{\sigma_l\}$ can be scheduled on time, we have $J_i^l = MH(J_i^l \cup \mathcal{T}^l)$ due to Lemma 12. If at least one job in $J_i^{l-1} \cup \{\sigma_l\}$ is tardy, then $\sigma_l \notin MH(J_i^{l-1} \cup \{\sigma_l\})$ since σ_l has maximum processing time among jobs in $J_i^{l-1} \cup \{\sigma_l\}$ and has the smallest job number among jobs with maximum processing time in $J_i^{l-1} \cup \{\sigma_l\}$. Hence, $J_i^l = J_i^{l-1} = MH(J_i^{l-1} \cup \{\sigma_l\})$ due to Lemma 9. Moreover, since $J_i^{l-1} = MH(J_i^{l-1} \cup T^{l-1})$ according to our assumption we have $J_i^{l-1} = MH(J_i^{l-1} \cup \{j\})$ for each $j \in T^{l-1}$ due to Lemma 10 and we can summarize the above to $J_i^{l-1} = MH(J_i^{l-1} \cup \{j\})$ for each $j \in \mathcal{T}^{l-1} \cup \{\sigma_l\}$. Then, it follows from Lemma 11 that $J_i^l = J_i^{l-1} = MH(J_i^{l-1} \cup \mathcal{T}^{l-1} \cup \{\sigma_l\})$. Since $\mathcal{T}^l = \mathcal{T}^{l-1} \cup \{\sigma_l\}$ this proves the induction step and thus the job set J_i^n on each machine M_i is MH-closed against $\mathcal{T}^n$ when Algorithm 2 terminates. Hence, due to Lemma 13, Algorithm 2 yields an $(\alpha(MH), -U_j)$-schedule, $\alpha \in \{P, Q\}$, and due to Theorems 5 and 7 at least $\frac{m}{2m-1}$ and $\frac{1}{2}$ as many jobs are on time as compared to an optimal solution. This yields the claimed approximation ratios.

In order to show that these bounds are tight we refer to the instances used in the proofs of Theorems 6 and 7. Due to the random choice of machines for each job (among those where it can be scheduled on time) Algorithm 2 might yield the $(P(MH), -U_j)$-schedule and $(Q(MH), -U_j)$-schedule in Figs. 4 and 5, respectively. □

While Algorithm 2 cannot be applied for unrelated machines since jobs do not have a unique workload. However, Algorithm 3 is an approximation algorithm even for unrelated parallel machines.

Theorem 9. *Algorithm 3 is a 2-approximation algorithm for $Rm||\sum(1-U_j)$.*

Proof. Obviously, when Algorithm 3 terminates, the set of on-time jobs on each machine is MH-closed against all tardy jobs since Algorithm 1 has been applied. Hence, due to Lemma 13 Algorithm 3 yields a $(R(MH), -U_j)$-schedule and due to Theorem 7 at least $\frac{1}{2}$ as many jobs are scheduled on time as is possible in an optimum schedule.

In order to see that this bound is tight we refer to the instance used in the proof of Theorem 7. Algorithm 2 yields the $(Q(MH), -U_j)$-schedule in Fig. 5. □

These results show that for two as well as three identical parallel machines, Algorithms 2 and 3 achieve better approximation ratios than the algorithm by Bar-Noy et al. (2001). For two machines, our algorithm achieves a ratio of 1.5 (compared to 1.8); for three machines our algorithm achieves a ratio of 1.66 (compared to 1.73). While for two machines, the algorithm by Leung & Yu (1994) yields a better approximation ratio, for three machines our structural results lead to the approximation algorithm with the currently best known ratio.

7. Conclusion and future work

Previously, literature on distributed scheduling mainly focused on utility functions of agents that are based only on their own completion times as well as on local policy that implement SSB procedures. The latter seems natural since in many cases, the local policy resembled the optimal scheduling algorithm of the corresponding single machine problem. In this paper, we extend this line of research to the central utility function of maximizing the number of on-time jobs.

For this central utility function, local policies based on simple-sorting-based procedures can lead to arbitrarily large losses of welfare. However, employing the algorithm of Moore and Hodgson, the optimal scheduling algorithm for the single machine problem, as a local policy yields a reasonably low (A)POA. Considering various machine environments, local policies and utilities of jobs, we performed a thorough analysis of the (A)POA for the distributed scheduling problem of maximizing the number of on-time jobs.

There are multiple ways to extend this line of research of distributed scheduling with due date related objective functions. For instance, the objective function of minimizing the sum of tardiness $\sum T_j$, allows for a pseudo-polynomial algorithm in the single-machine case. Moreover, while some other scheduling problems are already strongly *NP*-hard for single machines, advances in computational power might also allow the use of these algorithms as

local policies. Hence, it is important to also study the (A)POAs for these central objective functions.

Acknowledgements

The authors want to thank Bastian Diekkamp for fruitful discussions on early versions of the proofs. The first author has been supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 425058257.

Appendix A. Properties of Nash Equilibria and (Absolute) Prices of Anarchy

Lemma 3: Consider $\alpha(\phi)|ut=\beta|\sum(1-U_j)$. If the POA is bounded from above by a finite $k \geq 1$, the APOA is bounded from above by $\tilde{k} = \left(1-\frac{1}{k}\right)n$. Further, if an instance I that proves k as the lower bound of the POA has no tardy job in optimum schedules, this instance also proves $\tilde{k}$ as the lower bound for the APOA.

Proof. The upper bound of k on the POA guarantees that for every instance I and any $NE \in N(I)$ it holds that $\frac{\gamma(\mathrm{OPT}(I))}{\gamma(\mathrm{NE})} \leq k$. Therefore,

$$\frac{1}{k}\gamma(\mathrm{OPT}(I)) - \gamma(\mathrm{NE}) \leq 0$$

and

$$\gamma(\mathrm{OPT}(I)) - \gamma(\mathrm{NE}) \leq \left(1-\frac{1}{k}\right)\gamma(\mathrm{OPT}(I)).$$

Since no more than all n jobs can be scheduled on time in the optimum, we obtain

$$\gamma(\mathrm{OPT}(I)) - \gamma(\mathrm{NE}) \leq \left(1-\frac{1}{k}\right)\cdot n.$$

Let I be an instance with $\gamma(\mathrm{OPT}(I)) = n$ for which we find an NE in which only $\frac{n}{k}$ jobs are scheduled on time. This translates into $n - \frac{n}{k} = \left(1-\frac{1}{k}\right)\cdot n$ more jobs scheduled on time in $\mathrm{OPT}(I)$ than in $\mathrm{NE}(I)$ and thus a lower bound of $\left(1-\frac{1}{k}\right)\cdot n$ for the APOA. □

Lemma 6: The APOA for $P(\phi)|ut=\beta|\sum U_j$ is bounded from above by $\max\{0, n-m\}$ for any local policy ϕ, each $\beta \in \{-C_j, -T_j, -U_j\}$, and each number m of machines.

Proof. Due to Lemmas 1 and 2, it suffices to prove the upper bound for $ut=-U_j$. Consider an instance I with jobs J for $P(\phi)|ut=-U_j|\sum U_j$ and let S^O be the optimal schedule and S^N be a $(P(\phi), -U_j)$-schedule of I. We distinguish two cases in the following.

- If there is a machine which has no job in S^N, then each job j is either on time in S^N or has $p_j > d_j$ and, thus, is tardy in both, S^N and S^O. Hence, the number of tardy jobs is the same in S^N and S^O. Note that this implies an APOA of zero if $n < m$.
- If each machine has at least one job in S^N, then let J' be the set of first jobs. Furthermore, let U_j^N and U_j^O be the tardiness variable for job j in schedule S^N and in schedule S^O, respectively. Each job $j \in J'$ is scheduled on time in S^N if it is scheduled on time in S^O since it is the first job on its machine in S^N. Hence, for each $j \in J'$ we have $U_j^N \leq U_j^O$ and thus

$$\sum_{j\in J} U_j^N - \sum_{j\in J} U_j^O = \sum_{j\in J\setminus J'} \left(U_j^N - U_j^O\right) + \sum_{j\in J'} \left(U_j^N - U_j^O\right) \leq n-m.$$

□

Appendix B. Properties of Moore-Hodgson's Algorithm and MH-optimal sets

Lemma 7: Moore-Hodgson's algorithm finds a maximum cardinality feasible set of jobs with minimum total processing time.

Proof. Let us assume there is an instance specified by job set J for which Moore-Hodgson's algorithm determines a maximum cardinality feasible subset I of jobs and there is an other feasible set I' with $|I'| = |I|$ that has smaller total processing time P, that is $\sum_{j\in I'} p_j = P < \sum_{j\in I} p_j$.

We now consider an extended instance $J \cup \{j'\}$ with $d_{j'} = \max\left\{P+1, \max\left\{d_j \mid j \in J\right\}+1\right\}$ and $p_{j'} = d_{j'} - P$. Note that job set $I' \cup \{j'\}$ can be scheduled on time. When Moore-Hodgson's algorithm is applied to $J \cup \{j'\}$, job j' is considered in the last iteration and at the start of the last iteration job set I is in S. Thus, Moore-Hodgson's algorithm cannot achieve $|I' \cup \{j'\}|$ jobs which can be scheduled on time since $\sum_{j\in I} p_j + p_{j'} > P + p_{j'} = d_{j'}$. This contradicts correctness of Moore-Hodgson's algorithm and, thus, I' cannot exist. □

Lemma 8: Algorithm 1 finds the lexicographically largest among all maximum cardinality feasible sets of jobs with minimum total processing time.

Proof. For job set J let $I \subseteq J$ be the result of Algorithm 1. Consider another subset of jobs $I' \subseteq J$ such that $|I| = |I'|$ and both have the same total processing time (which is minimum due to Lemma 7). Assume that I' is lexicographically larger than I and consider the first position k in the corresponding EDD sequences σ and σ' where they do not coincide.

We denote the job in kth position in σ as σ_k. Since I' is lexicographically larger than I we have $\sigma'_k > \sigma_k$. There is at least one job in slots k to $|I'|$ of σ' that is not in σ. Let $j' = \sigma'_{k'}$ be the first such job. Note that $k' \geq k$ due to the choice of k and that jobs $\sigma'_k, \ldots, \sigma'_{k'-1}$ are in σ if $k' > k$. Let, finally, $j = \sigma_k$.

Since $k' \geq k$ we have $j' \geq \sigma'_k > j$ and, hence, $d_j \leq d_{j'}$. Since Algorithm 1 decides j' to be tardy and j on time, we can conclude that $p_{j'} > p_j$. Recall that Algorithm 1 drops the job with the smallest number from S in case of ties with respect to processing times.

We modify σ' by, first, replacing j' by j and restoring EDD afterward. Note that j has not been in σ' before since it has not been in slots $1, \ldots, k$ by choice of k and it did not follow σ'_k since $\sigma'_k > j$. After restoring EDD, j is in the kth position of σ'. Jobs that have been in slots k to $k'-1$ in $\sigma_{k'}$ are delayed. However, since they follow j in σ and j is started at the same time in σ' now these jobs cannot become tardy. No other job is delayed since $p_{j'} > p_j$. Hence, the modified sequence σ' corresponds to a maximum cardinality feasible set of jobs and since $p_{j'} > p_j$ it has smaller total processing time than I and I'. This is a contradiction. □

Figure B.6 sketches σ (top row), σ' before the modifying step (centre row), and σ' after the modifying step (bottom row) as considered in the proof.

Lemma 9: Consider job set J and job $j \notin J$. If $j \notin MH(J \cup \{j\})$, then $MH(J) = MH(J \cup \{j\})$.

Proof. Assume that $j \notin MH(J \cup \{j\})$ and, hence, $MH(J \cup \{j\}) \subseteq J$. Then, both, $MH(J)$ and $MH(J \cup \{j\})$ are maximum cardinality feasible sets of jobs with minimum total processing time for job set J due to Lemma 7. Since Algorithm 1 yields the lexicographically largest among such sets due to Lemma 8 and this is unique we have $MH(J) = MH(J \cup \{j\})$. □

Lemma 10: Let $I = MH(J)$. Then, $I = MH(J')$ for each J', $I \subseteq J' \subset J$.

Proof. Due to Lemma 7, I is a maximum cardinality feasible set of jobs with minimum total processing time for J and, thus, for each

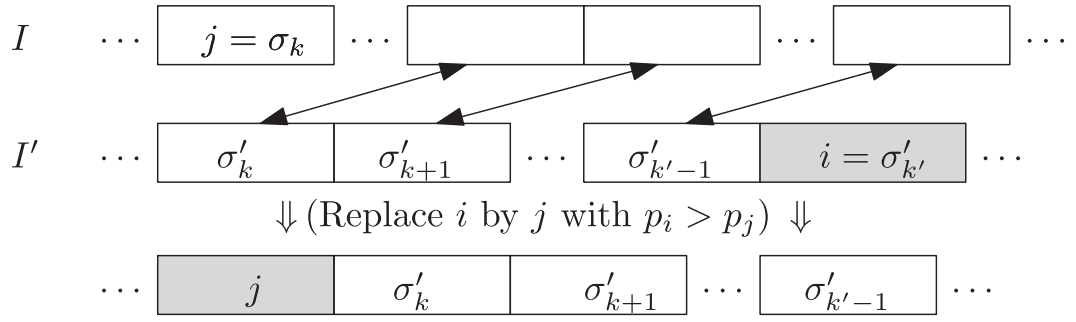

Fig. B.6. Schedules as considered in the proof of Lemma 8.

J', $I \subseteq J' \subset J$. Due to Lemma 8, $MH(J)$ is the lexicographically largest among these sets for J. Since each maximum cardinality feasible set of jobs with minimum total processing time for each J' is such a set for J, $MH(J)$ is the lexicographically largest among these sets for each J'. This set is unique and thus we have $I = MH(J')$ for each J'. □

Lemma 11 If $I = MH(I \cup \{j\})$ for each $j \in J$ with $I \cap J = \emptyset$, then $I = MH(I \cup A)$ for each $A \subseteq J$.

Proof. For a contradiction let $A \subseteq J$ be a smallest cardinality set with $I \neq I' := MH(I \cup A)$. Since $I = MH(I \cup \{j\})$ for each $j \in J$ we have $|A| \geq 2$. Furthermore, let $j', j \in A$ with $j' < j$. Finally, let $(I \cup A)^{<j}$ denote the set of jobs in $I \cup A$ with job number smaller than j.

Let $A' = A \cap I'$ be the set of jobs in A that are scheduled on time by Algorithm 1 applied to $I \cup A$. Then, $MH(I \cup A') = I'$ due to Lemma 10 since $I' \subseteq I \cup A' \subseteq I \cup A$. Hence, $I \neq MH(I \cup A')$ and since A is a smallest cardinality set with this property we have $|A'| = |A|$. Since, also, $A' \subseteq A$ it follows that $A' = A \subseteq I'$ and, in particular, $j', j \in I'$.

However, since A is a smallest cardinality set for which $I \neq MH(I \cup A)$ and $j', j \in A$, we have $j' \notin MH((I \cup A) \setminus \{j\})$ and $j \notin MH((I \cup A) \setminus \{j'\})$. Since

$$|MH(I \cup A)| \leq |MH((I \cup A) \setminus \{j\})| + 1$$

and $j', j \in MH(I \cup A)$ it follows that

$$|MH(I \cup A) \setminus \{j', j\}| \leq |MH((I \cup A) \setminus \{j\}| - 1$$

and, equivilantly,

$$|MH((I \cup A) \setminus \{j\}| \geq |MH(I \cup A) \setminus \{j', j\}| + 1.$$

and, thus, there exists a job $k \in MH((I \cup A) \setminus \{j\})$ with $k \notin MH(I \cup A)$.

In the following, we show that such a job k cannot exist, which leads to a contradiction that $I \neq MH(I \cup A)$ for $A \subseteq J$. Consider Algorithm 1 applied to $(I \cup A) \setminus \{j\}$ and to $I \cup A$. Sequence S is in the same state after all jobs in $(I \cup A \setminus \{j\})^{<j} = (I \cup A)^{<j}$ have been considered when Algorithm 1 is applied to $(I \cup A) \setminus \{j\}$ and when Algorithm 1 is applied to $(I \cup A)$. Since $j' \in MH(I \cup A)$ we have j' in S in both cases. For a job k to be dropped later while j' is not (when Algorithm 1 is applied to $(I \cup A)$) $p_k > p_{j'}$ or $p_k = p_{j'}$ and $k < j'$ must hold. However, then we cannot have $j' \notin MH(I \cup A) \setminus \{j\})$ and $k \in MH(I \cup A) \setminus \{j\})$. Hence, such a job k cannot exist. □

Lemma 12 Let $I = MH(J)$ and $j \notin J$. If $I \cup \{j\}$ is feasible, then $I \cup \{j\} = MH(J \cup \{j\})$.

Proof. For a contradiction we assume that $I' = MH(J \cup \{j\})$ with $I' \neq I \cup \{j\}$. Obviously, $j \in I'$, since otherwise $I' = MH(J)$ due to Lemma 9 which contradicts optimality of Algorithm 1 since $|MH(J)| < |MH(J \cup \{j\})|$. Furthermore, $\sum_{i \in I \cup \{j\}} p_i = \sum_{i \in I'} p_i$ since $\sum_{i \in I \cup \{j\}} p_i \geq \sum_{i \in I'} p_i$ due to Algorithm 1 and $\sum_{i \in I \cup \{j\}} p_i > \sum_{i \in I'} p_i$ would imply that I does not have minimum length among maximum cardinality feasible subsets of J which contradicts $I = MH(J)$, see Lemma 7.

Similar to the proof of Lemma 8, consider the two EDD sequences σ and σ' of $I \cup \{j\}$ and I'. Let k be the first position in which the sequences do not coincide, σ_k the job in the kth position of σ, and σ'_k the job in the kth position of σ'. We then have $\sigma'_k > \sigma_k$ due to Lemma 8. Furthermore, $\sigma'_k = j$ or $\sigma_k = j$ since otherwise $I' \setminus \{j\}$ is lexicographically larger than I and, hence, $I \neq MH(J)$. If $\sigma'_k \neq j$, then $\sigma'_k < j$ since j is in σ' and jobs are in EDD and, hence, $\sigma'_k < \sigma_k = j$ which cannot be the case as seen above. Thus, $\sigma'_k = j$.

Due to the EDD order $\sigma'_{k+1} > j > \sigma_k$. Then, however, $I' \setminus \{j\}$ is once again lexicographically larger than I which contradicts $I = MH(J)$. □

Lemma 13: Let S be a schedule for an instance I of $\alpha(MH)|ut_j = -U_j|\sum(1 - U_j)$ with job set J_i for each machine $M_i \in M$. Let $T = \bigcup_{M_i \in M}(J_i \setminus MH(J_i))$ be the set of tardy jobs. If and only if $MH(J_i) = MH(J_i \cup T)$ for each machine $M_i \in M$, then S is an $(\alpha(MH), -U_j)$-schedule for I.

Proof. We focus on the if-part first. Consider an arbitrary machine $M_i \in M$. Since $MH(J_i) = MH(J_i \cup T)$ it follows from Lemma 10 that $MH(J_i) = MH(J_i \cup \{j\})$ for each $j \in T \setminus J_i$. Thus, no job $j \in T \setminus J_i$ has an incentive to switch to machine M_i. Since this holds for each machine $M_i \in M$ the schedule is an $(\alpha(MH), -U_j)$-schedule.

It remains to show the only-if-part. Since S is an $(\alpha(MH), -U_j)$-schedule for I we have $MH(J_i) = MH(J_i \cup \{j\})$ for each $j \in T \setminus J_i$ and Lemma 11, then, implies that $MH(J_i) = MH(J_i \cup T)$. □

Lemma 14: There exists an optimum schedule to $\alpha||\sum U_j$ where $MH(J_i) = MH(J_i \cup T)$ for each machine $M_i \in M$ and the set T of tardy jobs.

Proof. If there is a machine $M_i \in M$ with $MH(J_i) \neq MH(J_i \cup T)$, then we set $J_i \leftarrow J_i \cup T$ and apply Algorithm 1 to machine i. We repeat this step until $MH(J_i) = MH(J_i \cup T)$ for each machine $M_i \in M$.

Note that the number of on-time jobs cannot increase in such a step since otherwise the origin schedule is not optimum. Nevertheless, since $MH(J_i) \neq MH(J_i \cup T)$ the set of on-time jobs on M_i is altered by this step. The overall procedure cannot cycle since in each step the set of on-time jobs on M_i becomes lexicographically larger or the total processing time of on-time jobs decreases. Thus, eventually, we obtain an optimum schedule to $\alpha||\sum U_j$ where $MH(J_i) = MH(J_i \cup T)$ for each machine $M_i \in M$. □

Lemma 15: Let I be an instance of $\alpha(MH)|ut_j = -U_j|\sum(1 - U_j)$. Let S be a $(\alpha(MH), -U_j)$-schedule for I with set J_i of jobs assigned to each machine M_i and the set T of tardy jobs. Then, $MH(T \cup MH(J_i) \setminus A)| \leq |MH(J_i)|$ for each $A \subseteq MH(J_i)$.

Proof. Due to Lemma 13 we have $MH(J_i) = MH(T \cup J_i)$. Hence, due to Lemma 10 we also have $MH(J_i) = MH(T \cup MH(J_i))$ since $MH(J_i) \subseteq T \cup MH(J_i) \subseteq T \cup J_i$.

Then,

$$|MH(T \cup MH(J_i) \setminus A)| > |MH(J_i)| = |MH(T \cup MH(J_i))|$$

for at least one $A \subseteq MH(J_m)$ contradicts correctness of Algorithm 1. □

References

Abdulkadiroğlu, A., & Sönmez, T. (2003). School choice: A mechanism design approach. *American Economic Review, 93*(3), 729–747.

Ashlagi, I., Tennenholtz, M., & Zohar, A. (2010). Competing schedulers. In *Proceedings of the AAAI conference on artificial intelligence: vol. 24*.

Azar, Y., Fleischer, L., Jain, K., Mirrokni, V., & Svitkina, Z. (2015). Optimal coordination mechanisms for unrelated machine scheduling. *Operations Research, 63*(3), 489–500.

Bar-Noy, A., Guha, S., Naor, J., & Schieber, B. (2001). Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal on Computing, 31*(2), 331–352.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Sterna, M., & Weglarz, J. (2019). *Handbook on scheduling: From theory to applications* (2nd ed.). Springer Science & Business Media.

Caragiannis, I. (2013). Efficient coordination mechanisms for unrelated machine scheduling. *Algorithmica, 66*(3), 512–540.

Chen, B., Potts, C. N., & Woeginger, G. J. (1998). A review of machine scheduling: Complexity, algorithms and approximability. In *Handbook of combinatorial optimization* (pp. 1493–1641). Springer.

Chen, Z.-L., & Powell, W. B. (1999). Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing, 11*(1), 78–94.

Christodoulou, G., Koutsoupias, E., & Nanavati, A. (2009). Coordination mechanisms. *Theoretical Computer Science, 410*(36), 3327–3336.

Cole, R., Correa, J. R., Gkatzelis, V., Mirrokni, V., & Olver, N. (2011). Inner product spaces for minsum coordination mechanisms. In *Proceedings of the forty-third annual ACM symposium on theory of computing* (pp. 539–548). ACM.

Graham, R. L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Technical Journal, 45*(9), 1563–1581.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. R. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Annals of discrete mathematics: vol. 5* (pp. 287–326). Elsevier.

Heydenreich, B., Müller, R., & Uetz, M. (2007). Games and mechanism design in machine scheduling - an introduction. *Production and Operations Management, 16*(4), 437–454.

Heydenreich, B., Müller, R., & Uetz, M. (2010). Mechanism design for decentralized online machine scheduling. *Operations Research, 58*(2), 445–457.

Ho, J. C., & Chang, Y.-L. (1995). Minimizing the number of tardy jobs for m parallel machines. *European Journal of Operational Research, 84*(2), 343–355.

Hoeksma, R., & Uetz, M. (2011). The price of anarchy for minsum related machine scheduling. In *International workshop on approximation and online algorithms* (pp. 261–273). Springer.

Ibarra, O. H., & Kim, C. E. (1977). Heuristic algorithms for scheduling independent tasks on nonidentical processors. *Journal of the ACM (JACM), 24*(2), 280–289.

Immorlica, N., Li, L. E., Mirrokni, V. S., & Schulz, A. S. (2009). Coordination mechanisms for selfish scheduling. *Theoretical Computer Science, 410*(17), 1589–1598.

Koutsoupias, E., & Papadimitriou, C. (1999). Worst-case equilibria. In *Proceedings of the 16th annual conference on theoretical aspects of computer science*. In *STACS'99* (pp. 404–413). Berlin, Heidelberg: Springer-Verlag.

Lee, K., Leung, J. Y.-T., & Pinedo, M. L. (2012). Coordination mechanisms for parallel machine scheduling. *European Journal of Operational Research, 220*(2), 305–313.

Leung, J. Y.-T., & Yu, V. K. (1994). Heuristic for minimizing the number of late jobs on two processors. *International Journal of Foundations of Computer Science, 5*(03n04), 261–279.

M'Hallah, R., & Bulfin, R. (2005). Minimizing the weighted number of tardy jobs on parallel processors. *European Journal of Operational Research, 160*(2), 471–484.

Moore, J. M. (1968). An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science, 15*(1), 102–109.

Nisan, N., & Ronen, A. (2001). Algorithmic mechanism design. *Games and Economic Behavior, 35*(1–2), 166–196.

Nisan, N., Roughgarden, T., Tardos, E., & Vazirani, V. V. (2007). *Algorithmic game theory*. Cambridge University Press.

Pinedo, M. (2016). *Scheduling: Theory, algorithms, and systems*. Springer.

Roughgarden, T. (2005). *Selfish routing and the price of anarchy*: vol. 174. MIT press Cambridge.

Roughgarden, T., Syrgkanis, V., & Tardos, E. (2017). The price of anarchy in auctions. *Journal of Artificial Intelligence Research, 59*, 59–101.

Sahni, S. K. (1976). Algorithms for scheduling independent tasks. *Journal of the ACM (JACM), 23*(1), 116–127.