

Truthful Mechanism Design for Multi-Dimensional Scheduling via Cycle Monotonicity[☆]

Ron Lavi¹

*Industrial Engineering and Management
The Technion — Israel Institute of Technology*

Chaitanya Swamy²

*Combinatorics and Optimization
University of Waterloo*

Abstract

We consider the makespan-minimization problem on unrelated machines in the context of algorithmic mechanism design. No truthful mechanisms with non-trivial approximation guarantees are known for this *multidimensional* domain. We study a well-motivated special case (also a multidimensional domain), where the processing time of a job on each machine is either “low” or “high”. We give a general technique to convert *any* c -approximation algorithm (in a *black-box* fashion) to a $3c$ -approximation truthful-in-expectation mechanism. Our construction uses *fractional* truthful mechanisms as a building block, and builds upon a technique of Lavi and Swamy (2005). When all jobs have identical low and high values, we devise a *deterministic* 2-approximation truthful mechanism. The chief novelty of our results is that we do not utilize explicit price definitions to prove truthfulness. Instead we design algorithms that satisfy *cycle monotonicity* (Rochet, 1987), a necessary

[☆]A preliminary version of this paper, without the results in Section 6 and the exposition in Section 5.1, appeared in the Proceedings of the 8th Annual ACM Conference on Electronic Commerce, 2007.

Email addresses: ronlavi@ie.technion.ac.il (Ron Lavi),
cswamy@math.uwaterloo.ca (Chaitanya Swamy)

¹Supported by grants from the Israeli Science Foundation and the Binational Science Foundation.

²Supported in part by NSERC grant 32760-06.

and sufficient condition for truthfulness in multidimensional settings; this is the first work that leverages this characterization.

JEL classification numbers: C70, C61, D44, D71.

1. Introduction

Motivation. Mechanism design studies the implementation of social choice functions under the presence of strategic players with privately known types. The focus of classic literature as well as the more recent work on algorithmic mechanism design has mainly been on settings where the social planner or designer wishes to maximize the social welfare (or equivalently, minimize social cost), or on auction settings where revenue-maximization is the main goal. Alternative optimization goals, such as those that incorporate fairness criteria (which have been investigated both in social choice theory and algorithms design), have received much less attention.

In this paper, we consider such an alternative goal in the context of machine scheduling, namely, *makespan minimization*. There are n jobs or tasks that need to be assigned to m machines, where each job has to be assigned to exactly one machine. Assigning a job j to a machine i incurs a load (cost) of $p_{ij} \geq 0$ on machine i , and the load of a machine is the sum of the loads incurred due to the jobs assigned to it; the goal is to schedule the jobs so as to minimize the maximum load of a machine, which is termed the *makespan* of the schedule. Makespan minimization is a common objective in scheduling environments, and has been well studied algorithmically in both the Computer Science and Operations Research communities (see, e.g., the survey of Hall (1996)). Following the work of Nisan and Ronen (2001), we consider each machine to be a *strategic player* or *agent* who privately knows its own processing time for each job, and may misrepresent these values in order to decrease its load (which is its incurred cost). Hence, we approach the problem via mechanism design: the social designer, who holds the set of jobs to be assigned, needs to specify, in addition to a schedule, suitable payments to the players in order to incentivize them to reveal their true processing times. We focus on the solution concept of dominant-strategy implementation (i.e., implementation under dominant strategies), and describe techniques to obtain *incentive-compatible dominant-strategy* mechanisms. Throughout, we will use the more compact phrase *truthful mechanism* to denote a dominant-strategy incentive-compatible mechanism.

The makespan-minimization objective is quite different from the classic goal of *social-welfare maximization* (SWM), where one wants to maximize the total welfare (or minimize the total cost) of all players. Instead, it corresponds to maximizing the minimum welfare and the notion of max-min fairness, and appears to be a much harder problem from the viewpoint of mechanism design. The possibility of constructing a truthful mechanism for makespan minimization is strongly related to assumptions on the players' processing times, in particular, the "dimensionality" of the domain. Nisan and Ronen considered the setting of *unrelated machines* where the p_{ij} values may be arbitrary. This is a *multidimensional domain*, since a player's private type is its entire *vector* of processing times $(p_{ij})_j$. Very few positive results are known for multidimensional domains and non-SWM objectives in general. In particular, the celebrated VCG (Vickrey, 1961; Clarke, 1971; Groves, 1973) family of mechanisms does not apply to the makespan minimization problem, and Nisan and Ronen (2001) showed that employing a VCG mechanism for this problem may result in a makespan that is m times worse than the optimal makespan. To date, no truthful mechanism is known to achieve a better (than $\Omega(m)$) bound for makespan minimization in any multidimensional scheduling domain, regardless of computational considerations.

In this work, we explore the issue of how multidimensionality affects the implementability of the makespan-minimization objective. As is common in Computer Science, we consider the notion of worst-case performance guarantees and say that a mechanism or algorithm is a c -approximation mechanism or algorithm if on *every instance*, it returns a makespan that is at most c times the optimal makespan; c is often called the approximation ratio or performance guarantee of the mechanism or algorithm. Our aim is to devise truthful mechanisms for the makespan-minimization problem that attain small approximation ratios and are computationally efficient. With VCG ruled out, we need to devise new techniques to approach our goal. As mentioned above, even without any computational considerations, no truthful mechanism with an approximation ratio significantly better than m is known to exist for makespan minimization in any multidimensional scheduling domain. On the negative side, Nisan and Ronen showed that no truthful deterministic mechanism can achieve an approximation ratio better than 2, and strengthened this lower bound to m for two specific classes of deterministic mechanisms. These lower bounds have been improved and extended in a series of recent results (Mu'alem and Schapira, 2007; Christodoulou, Koutsoupias, and Vidali, 2007; Christodoulou, Koutsoupias, and Kovács, 2007;

Koutsoupias and Vidali, 2007).

In stark contrast with the above state of affairs, much stronger (and many more) positive results are known for a special case of the unrelated machines problem, namely, the setting of *related machines*. Here, we have $p_{ij} = p_j/s_i$ for every i, j , where p_j is public knowledge, and the speed s_i is the *only* private parameter of machine i . This assumption makes the domain of players' types *single-dimensional*. Truthfulness in such domains is equivalent to a convenient *value-monotonicity* condition (Myerson, 1981; Archer and Tardos, 2001). This provides ample flexibility in mechanism design and appears to make it significantly easier to design truthful mechanisms in such domains. Archer and Tardos (2001) first considered the related-machines setting and gave a polynomial time randomized 3-approximation truthful-in-expectation mechanism. The gap between the single-dimensional and multidimensional domains is perhaps best exemplified by the fact that Archer and Tardos (2001) showed that there exists a truthful mechanism that always outputs an *optimal schedule*. (Recall that in the multidimensional unrelated machines setting, it is *impossible* to obtain a truthful mechanism with approximation ratio better than 2.) Various follow-up results (Archer, 2004; Auletta et al., 2004; Andelman et al., 2007; Kovács, 2005) have given polynomial time mechanisms that have strengthened the notion of truthfulness and/or improved the approximation ratio.

Such difficulties in moving from the single-dimensional to the multidimensional setting also arise in other mechanism design settings (e.g., auctions). Thus, in addition to the specific importance of scheduling in strategic environments, ideas from multidimensional scheduling may also have a bearing in the more general context of truthful mechanism design for multidimensional domains.

Overview of results. We consider the makespan-minimization problem for a special case of unrelated machines, where the processing time of a job is either “low” or “high” on each machine. More precisely, in our setting, $p_{ij} \in \{L_j, H_j\}$ for every i, j , where the L_j, H_j values are publicly known ($L_j \equiv$ “low”, $H_j \equiv$ “high”). We call this model the “job-dependent two-values” case. This generalizes the classic “restricted machines” setting where $p_{ij} \in \{L_j, \infty\}$, which has been well-studied algorithmically. A special case of our model is when $L_j = L$ and $H_j = H$ for all jobs j , which we denote as the “two-values” scheduling model. Both of our domains are multidimensional, since the machines are *unrelated*: one job may be low on one machine and

high on the other, while another job may follow the opposite pattern. The private information of each machine is therefore a vector specifying which jobs are low and high on it. Thus, they retain the core property underlying the difficulty of truthful mechanism design for unrelated machines, and by studying these special settings we hope to gain some insights that will be useful for tackling the general problem.

Our first result (Section 4) is a *general method* to convert *any c -approx. algorithm* for the job-dependent two values setting into a *3c-approximation truthful-in-expectation mechanism*. This is one of the very few known results that use an approximation algorithm in a *black-box* fashion to obtain a truthful mechanism for a multidimensional problem. Our result makes progress towards an important goal in algorithmic mechanism design, namely that of finding ways of translating algorithmic results (i.e., when all information is public knowledge) into the mechanism-design domain (to obtain truthful mechanisms). Our result implies that there *exists* a 3-approximation truthful-in-expectation mechanism for the L_j - H_j setting. Interestingly, the proof of truthfulness is not based on supplying explicit prices, and our construction does not necessarily yield efficiently-computable prices (but the allocation rule is efficiently computable).

Our second result (Section 5) applies to the two-values setting ($L_j = L$, $H_j = H$), for which we both improve the approximation ratio and strengthen the notion of truthfulness. We obtain a *deterministic 2-approx. truthful mechanism* for this problem for which the payments may also be computed efficiently. These are the *first* truthful mechanisms with non-trivial performance guarantees for a multidimensional scheduling domain. Complementing this, we observe that even this seemingly simple setting *does not admit truthful mechanisms that return an optimal schedule* (unlike in the case of related machines). By exploiting the multidimensionality of the domain, we prove that no truthful deterministic mechanism can obtain an approximation ratio better than 1.1 to the makespan (irrespective of computational considerations). In Section 6, we show that the algorithm from Section 5 can also be applied in the L_j - H_j setting when the ratio H_j/L_j is equal for all jobs to obtain a truthful mechanism that returns a *fractional assignment* of makespan at most twice the optimum. By modifying this algorithm suitably, we also obtain a truthful mechanism that, for every equal-ratio instance, returns a fractional assignment of makespan at most the optimal makespan for that instance.

The main technique, and one of the novelties, underlying our construc-

tions and proofs, is that we do not rely on explicit price specifications to prove the truthfulness of our mechanisms. Instead we exploit certain algorithmic monotonicity conditions that characterize truthfulness to first design an *implementable* algorithm, that is, an algorithm for which prices ensuring truthfulness exist, and *then* find these prices (by further delving into the proof of implementability). This kind of analysis has been the method of choice in the design of truthful mechanisms for single-dimensional domains, where value-monotonicity yields a convenient characterization enabling one to concentrate on the algorithmic side of the problem (see, e.g., Archer and Tardos (2001); Briest et al. (2005); Auletta et al. (2004); Andelman et al. (2007); Kovács (2005)). But for multidimensional domains, almost all positive results have relied on *explicit price specifications* to prove truthfulness (an exception is the work on unknown single-minded players in combinatorial auctions Lehmann et al. (2002); Briest et al. (2005)), which yet again shows the gap in our understanding of multidimensional vs. single-dimensional domains.

Our work is the *first* to leverage monotonicity conditions for truthful mechanism design in arbitrary domains. The monotonicity condition we use, which is sometimes called *cycle monotonicity*, was first proposed by Rochet (1987) (see also Gui et al. (2004)). It is a generalization of value-monotonicity and completely characterizes truthfulness in every domain. Our methods and analyses demonstrate the potential benefits of this characterization, and show that *cycle monotonicity can be effectively utilized to devise truthful mechanisms for multidimensional domains*. Consider, for example, our first result showing that any c -approximation algorithm can be “exported” to a $3c$ -approximation truthful-in-expectation mechanism. At the level of generality of an arbitrary approximation algorithm, it seems unlikely that one would be able to come up with prices to prove truthfulness of the constructed mechanism. But, cycle monotonicity *does allow* us to prove such a statement. In fact, some such condition based only on the underlying algorithm (and not on the prices) seems necessary to prove such a general statement.

The method for converting approximation algorithms into truthful mechanisms involves another novel idea. Our randomized mechanism is obtained by first constructing a truthful mechanism that returns a *fractional schedule*. Moving to a fractional domain allows us to “plug-in” truthfulness into the approximation algorithm in a rather simple fashion, while losing a factor of 2 in the approximation ratio. We then use a suitable randomized rounding procedure to convert the fractional assignment into a random integral assign-

ment. For this, we use a recent rounding procedure of Kumar et al. (2005) that is tailored for unrelated-machine scheduling. This preserves truthfulness, but we lose another additive factor equal to the approximation ratio. Our construction uses and extends some observations of Lavi and Swamy (2005), and further demonstrates the benefits of fractional mechanisms in truthful mechanism design.

Related work. Nisan and Ronen (2001) were the first to consider makespan-minimization on unrelated machines, in a mechanism design context. They give an m -approximation truthful mechanism and prove various lower bounds including a lower bound of 2 on the approximation ratio achievable by deterministic truthful mechanisms. Recently, Mu’alem and Schapira (2007) proved a lower bound of 2 for randomized truthful-in-expectation mechanisms, and Christodoulou, Koutsoupias, and Vidali (2007) proved a $(1+\sqrt{2})$ -lower bound for deterministic truthful mechanisms, which has been further strengthened to 2.6 by Koutsoupias and Vidali (2007). Very recently, Christodoulou, Koutsoupias, and Kovács (2007) showed that no *fractional* truthful mechanism can achieve approximation better than $2 - \frac{1}{m}$. Archer and Tardos (2001) first considered the related-machines problem and gave a 3-approximation truthful-in-expectation mechanism. This has been improved to: a 2-approximation randomized mechanism (Archer, 2004); for any fixed $\epsilon > 0$ and fixed number of machines, a $(1 + \epsilon)$ -approximation truthful mechanism (Andelman et al., 2007); and a 3-approximation deterministic mechanism (Kovács, 2005).

The algorithmic problem (i.e., when all information is public knowledge) of makespan minimization on unrelated machines is well understood and various 2-approximation algorithms are known. Lenstra (1990) gave the first such algorithm. Shmoys and Tardos (1993) later gave a 2-approximation algorithm for the *generalized assignment problem*, a generalization where there is a cost c_{ij} for assigning a job j to a machine i , and the goal is to minimize the cost subject to a bound on the makespan. Kumar et al. (2005) gave a randomized rounding algorithm that yields the same bounds. We use their procedure in our randomized mechanism.

The characterization of truthfulness for arbitrary domains in terms of cycle monotonicity seems to have been first observed by Rochet (1987) (see also Gui et al. (2004)). This generalizes the value-monotonicity condition for single-dimensional domains which was proposed by Myerson (1981) and rediscovered by Archer and Tardos (2001). As mentioned earlier, this con-

dition has been exploited numerous times to obtain truthful mechanisms for single-dimensional domains (Archer and Tardos, 2001; Briest et al., 2005; Auletta et al., 2004; Andelman et al., 2007; Kovács, 2005). For convex domains (i.e., each player’s set of private types is convex), Saks and Yu (2005) showed that cycle monotonicity is implied by a simpler condition, called weak monotonicity (Lavi et al., 2003; Bikhchandani et al., 2006). This generalizes the work of Bikhchandani et al. (2006) who showed that weak monotonicity characterizes truthfulness for various auction domains. But even this simpler condition has not found much application in truthful mechanism design for multidimensional problems. McAfee and McMillen (1988) considered convex domains with differentiable social choice functions. Their characterization of incentive-compatibility translates to the condition that the social choice function be a subgradient of a convex function. As Rochet (1987) observed, by a result of Rockafellar (1972), this is equivalent to cycle monotonicity for continuous domains, thus making explicit the connection between the two results.

Objectives other than social-welfare maximization and revenue maximization have received very little attention in mechanism design. In the context of combinatorial auctions, the problems of maximizing the minimum value received by a player, and computing an envy-minimizing allocation have been studied briefly. Lavi et al. (2003) showed that the former objective cannot be implemented truthfully; Bezáková and Dani (2005) gave a 0.5-approximation mechanism for two players with additive valuations. Lipton et al. (2005) showed that the latter objective cannot be implemented truthfully. These lower bounds were strengthened in Mu’alem and Schapira (2007).

2. Preliminaries

2.1. The scheduling domain

In our scheduling problem, we are given n jobs and m machines, and each job must be assigned to exactly one machine. In the unrelated-machines setting, each machine i is characterized by a vector of processing times $(p_{ij})_j$, where $p_{ij} \in \mathbb{R}_{\geq 0} \cup \{\infty\}$ denotes i ’s processing time for job j with the value ∞ specifying that i cannot process j . We consider two special cases of this problem:

1. **The job-dependent two-values case**, where $p_{ij} \in \{L_j, H_j\}$ for every i, j , with $L_j \leq H_j$, and the values L_j, H_j are known. This generalizes the

classic scheduling model of *restricted machines* (also called the restricted assignment model), where $H_j = \infty$.

2. **The two-values case**, which is a special case of above where $L_j = L$ and $H_j = H$ for all jobs j , i.e., $p_{ij} \in \{L, H\}$ for every i, j .

We say that a job j is low on machine i if $p_{ij} = L_j$, and high if $p_{ij} = H_j$. We will use the terms schedule and assignment interchangeably. We represent a deterministic schedule by a vector $x = (x_{ij})_{i,j}$, where x_{ij} is 1 if job j is assigned to machine i , thus we have $x_{ij} \in \{0, 1\}$ for every i, j , $\sum_i x_{ij} = 1$ for every job j . We will also consider randomized algorithms and algorithms that return a fractional assignment. In both these settings, we will again specify an assignment by a vector $x = (x_{ij})_{i,j}$ with $\sum_i x_{ij} = 1$, but now $x_{ij} \in [0, 1]$ for every i, j . For a randomized algorithm, x_{ij} is simply the probability that j is assigned to i (thus, x is a convex combination of integer assignments).

We denote the *load* of machine i (under a given assignment) by $l_i = \sum_j x_{ij} p_{ij}$, and the *makespan* of a schedule is defined as the maximum load on any machine, i.e., $\max_i l_i$.

2.2. Mechanism design

We consider the makespan-minimization problem in the above scheduling domains in the context of mechanism design. Mechanism design studies strategic settings where the social designer needs to *ensure* the cooperation of the different entities involved in the algorithmic procedure. Following the work of Nisan and Ronen (2001), we consider the machines to be the *strategic players* or *agents*. The social designer holds the set of jobs that need to be assigned, but does not know the (true) processing times of these jobs on the different machines. Each machine is a selfish entity, that privately knows its own processing time for each job (which constitutes its type). Scheduling a job on a machine incurs a cost to the machine equal to the true processing time of the job on the machine, and a machine may choose to misrepresent its vector of processing times, which are private, in order to decrease its cost.

We consider direct-revelation mechanisms: each machine reports its (possibly false) vector of processing times, the mechanism then computes a schedule and hands out payments to the players (i.e., machines) to compensate them for the cost they incur in processing their assigned jobs. A (direct-revelation) mechanism thus consists of a tuple (x, P) : x specifies the schedule, and $P = \{P_i\}$ specifies the payments handed out to the machines, where both x and the P_i s are functions of the reported processing times $p = (p_{ij})_{i,j}$. The

mechanism's goal is to compute a schedule that has near-optimal makespan with respect to the *true* processing times; a machine i is however only interested in maximizing its own *utility*, $P_i - l_i$, where l_i is its load under the output assignment, and may declare false processing times if this could increase its utility. Therefore, we seek to devise a mechanism that incentivizes the machines/players to truthfully reveal their processing times via the payments. This is made precise using the notion of *dominant-strategy incentive compatibility*, which we also refer to frequently by the more compact term *truthfulness*.

Definition 2.1 (Truthfulness) *A scheduling mechanism is truthful if, for every machine i , every vector of processing times of the other machines, p_{-i} , every true processing-time vector p_i^1 and any other vector p_i^2 of machine i , we have:*

$$P_i^1 - \sum_j x_{ij}^1 p_{ij}^1 \geq P_i^2 - \sum_j x_{ij}^2 p_{ij}^1, \quad (1)$$

where (x^1, P^1) and (x^2, P^2) are respectively the schedule and payments when the other machines declare p_{-i} and machine i declares p_i^1 and p_i^2 , i.e., $x^1 = x(p_i^1, p_{-i})$, $P_i^1 = P_i(p_i^1, p_{-i})$ and $x^2 = x(p_i^2, p_{-i})$, $P_i^2 = P_i(p_i^2, p_{-i})$.

To put it in words, in a dominant-strategy incentive-compatible or truthful mechanism, no machine can improve its utility by declaring a false processing time, no matter what the other machines declare.

We will also consider *fractional mechanisms* that return a fractional assignment, and *randomized mechanisms* where the assignment and the payments may be random variables. The notion of truthfulness for a fractional mechanism is the same as in Definition 2.1, where x^1, x^2 are now fractional assignments. For a randomized mechanism, we will consider the notion of truthfulness in expectation (Archer and Tardos, 2001), which means that a machine (player) maximizes her *expected* utility by declaring her true processing-time vector. Inequality (1) also defines truthfulness-in-expectation for a randomized mechanism, where P_i^1, P_i^2 now denote the *expected* payments made to player i , x^1, x^2 are the fractional assignments denoting the randomized algorithm's schedule (i.e., x_{ij}^k is the probability that j is assigned to i in the schedule output for (p_i^k, p_{-i})).

For our two scheduling domains, the informational assumption is that the values L_j, H_j are publicly known. The private information or *type* of a machine is which jobs have value L_j (or L) and which ones have value H_j

(or H) on it. We emphasize that *both of our domains are multidimensional*, since each machine i 's type is a *vector* saying which jobs are low and high on it.

2.3. Goals and objectives

Our goal is to design truthful mechanisms that assign the jobs to the machines so as to minimize the makespan of the schedule. As we will show, no truthful mechanism can exactly implement the makespan-minimization objective in our multidimensional domains and hence, we consider mechanisms that *approximately* implement this objective. More precisely, our aim is to devise truthful mechanisms such that for *every* (true) processing-time vector (in our scheduling domains) the mechanism returns a schedule of makespan at most c times the optimum, where $c \geq 1$ is some (small) constant. We call such a mechanism a *c-approximation truthful mechanism*, and we (naturally) seek a *c-approximation truthful mechanism* where c is as small as possible.

As is the focus in algorithmic mechanism design, a desirable property that we want our mechanisms to have, is computational efficiency. Intuitively, this means that the mechanism should reach its final outcome (that is, both the schedule and the payments) in a “small” number of steps. Following the standard notion in Computer Science, by “small” we mean polynomial in the size of the instance (measured by the number of bits required to represent it), that is, polynomial in m , n and $\max_j \log H_j$ (where we assume that the L_j s and H_j s are all integers). Observe, for example, that the naive algorithm that exhaustively tries all possible assignments and chooses the one with minimum makespan does not satisfy the polynomial-time criterion since there are m^n possible assignments to enumerate. The requirement of computational efficiency also necessitates a notion of approximate implementation, since it is known that the algorithmic problem (i.e., when all information is public) of makespan minimization is computationally intractable even for the two-values problem (Lenstra, 1990)³; thus, one must settle for some approximation if one desires computational efficiency.

³More precisely, the two-values problem is *NP*-hard. Unless $P=NP$, no polynomial time algorithm can attain approximation ratio better than $4/3$ for the two-values problem, and $3/2$ for the job-dependent two-values problem. This follows from the reductions in Lenstra (1990) (theorems 4 and 5 respectively), which prove that the makespan minimization on unrelated machines is *NP*-hard by constructing instances of these two problems.

3. Cycle monotonicity

Although truthfulness is defined in terms of payments, it turns out that truthfulness actually boils down to a certain algorithmic condition of monotonicity. This seems to have been first observed for multidimensional domains by Rochet (1987), who gave a necessary and sufficient condition that characterizes truthfulness in *every* domain. Rochet’s condition, which is sometimes called *cycle monotonicity*, has been used successfully in algorithmic mechanism design several times, but only for *single-dimensional domains*, where cycle monotonicity reduces to the simpler value-monotonicity condition proposed by Myerson (1981). For multidimensional domains, the monotonicity condition is more involved and there has been no success in employing it in the design of truthful mechanisms. Most positive results for multidimensional domains have relied on *explicit price specifications* in order to prove truthfulness. One of the main contributions of this paper is to demonstrate that cycle monotonicity can indeed be effectively utilized to devise truthful mechanisms in multidimensional settings. We include a brief exposition on cycle monotonicity for completeness. The exposition here is largely based on Gui et al. (2004).

Cycle monotonicity is best described in the abstract social choice setting: there is a finite set A of alternatives, there are m players, and each player has a private type (valuation function) $v : A \mapsto \mathbb{R}$, where $v_i(a)$ should be interpreted as i ’s value for alternative a . In the scheduling domain, A represents all the possible assignments of jobs to machines, and $v_i(a)$ is the negative of i ’s load in the schedule a . Let V_i denote the set of all possible types of player i . A mechanism is a tuple $(f, \{P_i\})$ where $f : V_1 \times \cdots \times V_m \mapsto A$ is the “algorithm” for choosing the alternative, often referred to as the *social choice function* or allocation rule, and $P_i : V_1 \times \cdots \times V_m \mapsto A$ is the price *charged* to player i (in the scheduling setting, the mechanism *pays* the players, which corresponds to negative prices). The mechanism is truthful if for every i , every $v_{-i} \in V_{-i} = \prod_{i' \neq i} V_{i'}$, and any $v_i, v'_i \in V_i$ we have $v_i(a) - P_i(v_i, v_{-i}) \geq v_i(b) - P_i(v'_i, v_{-i})$, where $a = f(v_i, v_{-i})$ and $b = f(v'_i, v_{-i})$. A basic question that arises is given an algorithm $f : V_1 \times \cdots \times V_m \mapsto A$, do there exist prices that will make the resulting mechanism truthful? If there exist such prices, we say that f is *implementable* in dominant strategies, or simply implementable for short.

It is well known (see e.g. Lavi et al. (2003)) that the price P_i can only depend on the alternative chosen and the others’ declarations, that is, we

may write $P_i : V_{-i} \times A \mapsto \mathbb{R}$. Thus, truthfulness implies that for every i , every $v_{-i} \in V_{-i}$, and any $v_i, v'_i \in V_i$ with $f(v_i, v_{-i}) = a$ and $f(v'_i, v_{-i}) = b$, we have $v_i(a) - P_i(a, v_{-i}) \geq v_i(b) - P_i(b, v_{-i})$. Now fix a player i , and fix the declarations v_{-i} of the others. We seek an assignment to the variables $\{P_a\}_{a \in A}$ such that $v_i(a) - v_i(b) \geq P_a - P_b$ for every $a, b \in A$ and $v_i \in V_i$ with $f(v_i, v_{-i}) = a$. (Strictly speaking, we should use $A' = f(V_i, v_{-i})$ instead of A here.) Define $\delta_{a,b} := \inf\{v_i(a) - v_i(b) : v_i \in V_i, f(v_i, v_{-i}) = a\}$. We can now rephrase the above price-assignment problem: we seek an assignment to the variables $\{P_a\}_{a \in A}$ such that

$$P_a - P_b \leq \delta_{a,b} \quad \forall a, b \in A \quad (2)$$

This is easily solved by looking at the *allocation graph* and applying a standard basic result of graph theory.

Definition 3.1 (Gui et al. (2004)) *The allocation graph of f is a directed weighted graph $G = (A, E)$ where $E = A \times A$ and the weight of an edge $b \rightarrow a$ (for any $a, b \in A$) is $\delta_{a,b}$.*

Theorem 3.2 *There exists a feasible assignment to (2) iff the allocation graph has no negative-length cycles. Furthermore, if all cycles are non-negative, a feasible assignment is obtained as follows: fix an arbitrary node $a^* \in A$ and set P_a to be the length of the shortest path from a^* to a .*

Proof : Suppose first that there exists a feasible assignment $\{P_a\}_{a \in A}$ to (2). Consider any cycle $a_1 \leftarrow a_2 \dots \leftarrow a_K \leftarrow a_1$. Adding the inequalities, $P_{a_k} - P_{a_{k+1}} \leq \delta_{a_k, a_{k+1}}$ for $k = 1, \dots, K$, where $K + 1 \equiv 1$, shows that $\sum_{k=1}^K \delta_{a_k, a_{k+1}} \geq 0$, that is, the length of the cycle is nonnegative.

Now suppose that the length of every cycle is nonnegative. Fix some arbitrary $a^* \in A$, and set P_a to be the length of the shortest path from a^* to a . Since there are no negative cycles, the shortest-path distance from a^* to a is well defined and P_a is finite for all $a \in A$. It follows that for any $a, b \in A$, we have $P_a - P_b \leq \delta_{a,b}$ since the shortest-path distance from a^* to a is at most $\delta_{a,b}$ plus the shortest-path distance from a^* to b . ■

The theorem leads to the following definition, which is another way of phrasing the condition that the allocation graph have no negative cycles.

Definition 3.3 (Cycle monotonicity) *A social choice function f satisfies cycle monotonicity if for every player i , every $v_{-i} \in V_{-i}$, every integer K ,*

and every $v_i^1, \dots, v_i^K \in V_i$,

$$\sum_{k=1}^K \left[v_i^k(a_k) - v_i^k(a_{k+1}) \right] \geq 0$$

where $a_k = f(v_i^k, v_{-i})$ for $1 \leq k \leq K$, and $a_{K+1} = a_1$.

Corollary 3.4 *There exist prices P such that the mechanism (f, P) is truthful iff f satisfies cycle monotonicity.*

Recently, Saks and Yu (2005) showed that if each set V_i is convex, then cycle monotonicity reduces to a simpler condition called weak monotonicity (Bikhchandani et al., 2006), which is the condition that the allocation graph contains no negative 2-cycles. Even this simpler condition, however, has not found much use in the design of truthful mechanisms for multidimensional problems.

We now consider our specific scheduling domain. Fix a player i , p_{-i} , and any p_i^1, \dots, p_i^K . Let $x(p_i^k, p_{-i}) = x^k$ for $1 \leq k \leq K$, and let $x^{K+1} = x^1$, $p^{K+1} = p^1$. x^k could be a $\{0, 1\}$ -assignment or a fractional assignment. We have $v_i^k(x^k) = -\sum_j x_{ij}^k p_{ij}^k$, so cycle monotonicity translates to $\sum_{k=1}^K \left[-\sum_j x_{ij}^k p_{ij}^k + \sum_j x_{ij}^{k+1} p_{ij}^k \right] \geq 0$. Rearranging, we get

$$\sum_{k=1}^K \sum_j x_{ij}^{k+1} (p_{ij}^k - p_{ij}^{k+1}) \geq 0. \quad (3)$$

Thus (3) “reduces” our mechanism design problem to a concrete *algorithmic problem*. For most of this paper, we will consequently ignore any strategic considerations and focus on designing an approximation algorithm for minimizing makespan that satisfies (3).

4. A general technique to obtain randomized mechanisms

In this section, we consider the case of job-dependent L_j, H_j values (with $L_j \leq H_j$), which generalizes the classical restricted-machines model (where $H_j = \infty$). We show the power of randomization, by providing a *general technique* that converts *any* c -approximation algorithm into a $3c$ -approximation, truthful-in-expectation mechanism. This is one of the few results that shows

how to export approximation algorithms for a multidimensional problem into truthful mechanisms when the algorithm is given as a black box.

Our construction and proof are simple, and based on two ideas. First, as outlined earlier, we prove truthfulness using cycle monotonicity. It seems unlikely that for an allocation rule obtained (using our construction) from an arbitrary approximation algorithm given only as a black box, one would be able to prove implementability by coming up with payments, but cycle monotonicity allows us to prove precisely this. Second, we obtain our randomized mechanism by (a) first moving to a fractional domain, and constructing a *fractional* truthful mechanism that is allowed to return fractional assignments; then (b) using a rounding procedure to express the fractional schedule as a convex combination of integer schedules. This builds upon a theme introduced by Lavi and Swamy (2005), namely that of using fractional mechanisms to obtain truthful-in-expectation mechanisms.

We should point out however that one cannot simply plug in the results of Lavi and Swamy (2005). Their results hold for social-welfare-maximization problems and rely on using VCG to obtain a fractional truthful mechanism. VCG however does not apply to makespan minimization, and in our case even the existence of a near-optimal *fractional* truthful mechanism is not known. We use the following result adapted from Lavi and Swamy (2005).

Lemma 4.1 (Lavi and Swamy (2005)) *Let $M = (x, P)$ be a fractional truthful mechanism. Let \mathcal{R} be a randomized rounding algorithm that given a fractional assignment x , outputs a random assignment $\mathcal{R}(x)$ such that $E[\mathcal{R}(x)_{ij}] = x_{ij}$ for all i, j . Define $X(p)$ to be the random assignment $\mathcal{R}(x(p))$. There exist payments P' such that the mechanism $M' = (X, P')$ is truthful in expectation. Furthermore, if M is individually rational⁴ then M' is individually rational for every realization of coin tosses.*

Let $OPT(p)$ denote the optimal makespan (over integer schedules) for instance p . As our first step, we take a c -approximation algorithm and convert it to a $2c$ -approximation fractional truthful mechanism. This conversion works even when the approximation algorithm returns only a fractional schedule (satisfying certain properties) of makespan at most $c \cdot OPT(p)$ for every instance p . We prove truthfulness by showing that the fractional al-

⁴A mechanism satisfies individual rationality if each player's utility is non-negative if she reveals her true value.

gorithm satisfies cycle monotonicity (3). Although Corollary 3.4 also holds for an infinite alternative set, we note that the alternative-set of our fractional mechanism is *finite* (although the set of all fractional assignments is infinite): its cardinality is at most that of the input domain, which is 2^{mn} in the two-value case. To convert this fractional truthful mechanism into a randomized truthful mechanism we need a randomized rounding procedure satisfying the requirements of Lemma 4.1. Fortunately, such a procedure is implicit in the work of Shmoys and Tardos (1993), and is made explicit by Kumar et al. (2005).

Lemma 4.2 (Shmoys and Tardos (1993); Kumar et al. (2005)) *Given a fractional assignment x and a processing-time vector p , there exists a randomized rounding procedure that yields a (random) assignment X such that,*

1. for any i, j , $\mathbb{E}[X_{ij}] = x_{ij}$.
2. for any i , $\sum_j X_{ij} p_{ij} < \sum_j x_{ij} p_{ij} + \max_{\{j: x_{ij} > 0\}} p_{ij}$ with probability 1.

For completeness, we include the proofs of Lemmas 4.1 and 4.2 in the appendix.

Property 1 will be used to obtain truthfulness in expectation, and property 2 will allow us to prove an approximation guarantee. We first show that any algorithm that returns a fractional assignment having certain properties satisfies cycle monotonicity.

Lemma 4.3 *Let \mathcal{A} be an algorithm that for any input p , outputs a (fractional) assignment x such that, if $p_{ij} = H_j$ then $x_{ij} \leq 1/m$, and if $p_{ij} = L_j$ then $x_{ij} \geq 1/m$. Then \mathcal{A} satisfies cycle monotonicity.*

Proof : Fix a player i , and the vector of processing times of the other players p_{-i} . We need to prove (3), that is, $\sum_{k=1}^K \sum_j x_{ij}^{k+1} (p_{ij}^k - p_{ij}^{k+1}) \geq 0$ for every p_i^1, \dots, p_i^K , where index $k = K + 1$ is taken to be $k = 1$. We will show that for every job j , $\sum_{k=1}^K x_{ij}^{k+1} (p_{ij}^k - p_{ij}^{k+1}) \geq 0$.

If p_{ij}^k is the same for all k (either always L_j or always H_j), then the above inequality clearly holds. Otherwise we can divide the indices $1, \dots, K$, into maximal segments, where a maximal segment is a maximal set of consecutive indices $k', k' + 1, \dots, k'' - 1, k''$ (where $K + \ell \equiv \ell$ for $1 \leq \ell < K$) such that $p_{ij}^{k'} = H_j \geq p_{ij}^{k'+1} \geq \dots \geq p_{ij}^{k''} = L_j$. This follows because there must be some k such that $p_{ij}^k = H_j > p_{ij}^{k-1} = L_j$. We take $k' = k$ and then keep including indices in this segment till we reach a k such that $p_{ij}^k = L_j$ and $p_{ij}^{k+1} = H_j$.

We set $k'' = k$, and then start a new maximal segment with index $k'' + 1$. Note that $k'' \neq k'$ and $k'' + 1 \neq k' - 1$. We now have a subset of indices and we can continue recursively. So all indices are included in some maximal segment. We will show that for every such maximal segment $k', k' + 1, \dots, k''$, $\sum_{k'-1 \leq k < k''} x_{ij}^{k+1} (p_{ij}^k - p_{ij}^{k+1}) \geq 0$. Adding this for each segment yields the desired inequality.

So now focus on a maximal segment $k', k' + 1, \dots, k'' - 1, k''$. Thus, there is some k^* such that for $k' \leq k < k^*$, we have $p_{ij}^k = H_j$, and for $k^* \leq k \leq k''$, we have $p_{ij}^k = L_j$. Now the left hand side of the above inequality for this segment is simply $x_{ij}^{k'}(L_j - H_j) + x_{ij}^{k^*}(H_j - L_j) \geq 0$, since $x_{ij}^{k'} \leq \frac{1}{m} \leq x_{ij}^{k^*}$ as $p_{ij}^{k'} = H_j$ and $p_{ij}^{k^*} = L_j$. \blacksquare

We now describe how to use a c -approximation algorithm to obtain an algorithm satisfying the property in Lemma 4.3. For simplicity, first suppose that the approximation algorithm returns an integral schedule. The idea is to simply “spread” this schedule. We take each job j assigned to a high machine and assign it to an extent $1/m$ on all machines. For each job j assigned to a low machine, say i , we assign $1/m$ -fraction of it to every other machine where it is low, and assign its remaining fraction (which is at least $1/m$) to i . The resulting assignment clearly satisfies the desired properties. Also observe that the load on any machine has at most increased by $\frac{1}{m} \cdot$ (load on other machines) \leq makespan, and hence the makespan has at most doubled. This “spreading out” can also be done if the initial schedule is fractional. Since a job j could be assigned to multiple machines, we consider each machine i for which $x_{ij} > 0$ in turn, and “spread” this x_{ij} -fraction over the machines: if i is a high-machine for j , we assign an x_{ij}/m -fraction of j to all machines; if i is a low-machine for j , we assign j to an extent x_{ij}/m on every other low-machine and the remaining $x_{ij}(1 - \frac{|\{i' \neq i: p_{i'j} = L_j\}|}{m})$ -fraction of it to i . We describe this more precisely below.

Algorithm 1 Let \mathcal{A} be any algorithm that on any input p outputs a possibly fractional assignment x such that $x_{ij} > 0$ implies that $p_{ij} \leq \text{makespan}(x)$. (In particular, note that any algorithm that returns an integral assignment has these properties.) Our algorithm returns the following assignment x^F . For every i, j ,

1. if $p_{ij} = H_j$, set $x_{ij}^F = \sum_{i': p_{i'j} = H_j} x_{i'j}/m$;
2. if $p_{ij} = L_j$, set $x_{ij}^F = x_{ij} + \sum_{i' \neq i: p_{i'j} = L_j} (x_{i'j} - x_{ij})/m + \sum_{i': p_{i'j} = H_j} x_{i'j}/m$.

Theorem 4.4 *Suppose algorithm \mathcal{A} satisfies the conditions in Algorithm 1 and returns a makespan of at most $c \cdot OPT(p)$ for every p . Then, Algorithm 1 is a $2c$ -approximation, cycle-monotone fractional algorithm. Moreover, if $x_{ij}^F > 0$ on input p , then $p_{ij} \leq c \cdot OPT(p)$.*

Proof : First, note that x^F is a valid assignment: for every job j we have

$$\sum_i x_{ij}^F = \sum_i x_{ij} + \sum_{i, i' \neq i: p_{ij} = p_{i'j} = L_j} \frac{x_{i'j} - x_{ij}}{m} = \sum_i x_{ij} = 1.$$

We also have that if $p_{ij} = H_j$ then $x_{ij}^F = \sum_{i': p_{i'j} = H_j} x_{i'j}/m \leq 1/m$. If $p_{ij} = L_j$, then $x_{ij}^F = x_{ij}(1 - \ell/m) + \sum_{i' \neq i} x_{i'j}/m$ where $\ell = |\{i' \neq i : p_{i'j} = L_j\}| \leq m - 1$; so $x_{ij}^F \geq \sum_{i'} x_{i'j}/m = 1/m$. Thus, by Lemma 4.3, \mathcal{A}' satisfies cycle monotonicity.

The total load on any machine i under x^F is at most

$$\sum_{j: p_{ij} = H_j} \sum_{i': p_{i'j} = H_j} H_j \cdot \frac{x_{i'j}}{m} + \sum_{j: p_{ij} = L_j} L_j \left(x_{ij} + \sum_{i' \neq i} \frac{x_{i'j}}{m} \right),$$

which is at most $\sum_j p_{ij} x_{ij} + \sum_{i' \neq i} \sum_j p_{i'j} x_{i'j}/m \leq 2c \cdot OPT(p)$. Finally, suppose $x_{ij}^F > 0$ for some i and j . If $p_{ij} = L_j$, clearly $p_{ij} \leq OPT(p)$; if $p_{ij} = H_j$, then for some i' (possibly i) with $p_{i'j} = H_j$ we have $x_{i'j} > 0$, so by assumption, $p_{i'j} = H_j = p_{ij} \leq c \cdot OPT(p)$. ■

Theorem 4.4 combined with Lemmas 4.1 and 4.2, gives a $3c$ -approximation, truthful-in-expectation mechanism. The entire mechanism is summarized below.

Mechanism $M' = (\mathbf{X}, \mathbf{P}')$ (Given: An algorithm \mathcal{A} satisfying the conditions in Algorithm 1.)

On input p , the mechanism outputs the following random assignment and prices.

Assignment $\mathbf{X}(p)$. Compute the assignment x^F by running Algorithm 1 on the assignment $x = \mathcal{A}(p)$. The random assignment $X(p)$ is the assignment obtained by using Lemma 4.2 to round x^F .

Payments $\{\mathbf{P}'_i(p)\}$. Define \mathcal{A}'' to be the algorithm such that $x^F = \mathcal{A}''(p)$. First compute the payment scheme P so that $M'' = (\mathcal{A}'', P)$ is

a fractional truthful mechanism: the payment $P_i(p)$ to player i for the assignment x^F is computed by using Theorem 3.2. (Note that the weights δ_{ab} for the edges in the allocation graph for player i may be computed trivially in $O(2^{mn})$ time since one can simply go through the 2^{mn} possible two-value inputs.) Now use Lemma 4.1 on the mechanism M'' to obtain the (random) payments $P'_i(p)$.

Theorem 4.5 *Given a c -approximation fractional algorithm \mathcal{A} satisfying the conditions in Algorithm 1, mechanism M' described above is a $3c$ -approximation, truthful-in-expectation mechanism.*

The trivial procedure mentioned in mechanism M' for computing payments is not a polynomial-time procedure. A more efficient procedure will likely need to exploit specific properties of the underlying algorithm \mathcal{A} . We leave the problem of finding an approximation algorithm \mathcal{A} for which the payments P' in mechanism M' are computable in polynomial time as an open problem.

It is known that one can compute in polynomial time the smallest makespan T^* for which there exists a feasible fractional assignment x having the property $x_{ij} > 0 \implies p_{ij} \leq T^*$ (Lenstra, 1990; Shmoys and Tardos, 1993). Observe that $T^* \leq OPT(p)$. Taking \mathcal{A} in mechanism M' above to be this optimal algorithm, we obtain a 3-approximation mechanism.

Corollary 4.6 *There exists a truthful-in-expectation mechanism with approximation ratio 3 for the L_j - H_j setting.*

5. A deterministic mechanism for the two-values case

We now present a deterministic 2-approximation truthful mechanism for the case where $p_{ij} \in \{L, H\}$ for all i, j . As in Section 4, our goal is to obtain an approximation algorithm that satisfies cycle monotonicity. We divide the description and analysis of the mechanism into several parts. We start by giving a brief exposition of network flows, which is the main technical tool that we use in our construction, and discussing its connection to our scheduling problem (Section 5.1). We then describe the algorithm (Section 5.2). In Sections 5.3 and 5.4 respectively, we bound its approximation guarantee and prove that it satisfies cycle monotonicity. In Section 5.5, we make the payments yielding a truthful mechanism explicit and show that they can be computed efficiently. We conclude the section by proving a lower bound on

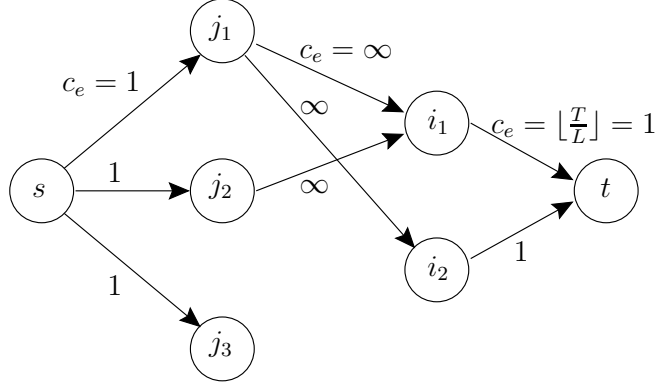


Figure 1: Flow network for (p, T) , where there are 3 jobs and 2 machines, and $T = L$.

the best possible approximation achievable by *any* cycle-monotone algorithm (Section 5.6).

Throughout this section, we will often say that j is assigned to a low-machine to denote that j is assigned to a machine i where $p_{ij} = L$. We will call a job j a low job of machine i if $p_{ij} = L$; the low-load of i is the load on i due to its low jobs, i.e., $\sum_{j:p_{ij}=L} x_{ij}p_{ij}$.

5.1. Network flows

In this section, we give a brief introduction to network flows; the reader is referred to the textbooks Ahuja et al. (1993); Kleinberg and Tardos (2006) for more details. A flow network is defined by a directed graph $G = (V, E)$ with a *capacity* $c_e \geq 0$ associated with every edge $e \in E$, a *source* $s \in V$, and a *sink* $t \in V$. A flow is a function $f : E \rightarrow \mathbb{R}_{\geq 0}$ satisfying the following properties:

1. (*Capacity conditions*) For each edge e , we have $0 \leq f_e \leq c_e$: the flow on e is at most its capacity.
2. (*Conservation conditions*) For every node $v \in V \setminus \{s, t\}$, we have $\sum_{e \text{ into } v} f_e = \sum_{e \text{ out of } v} f_e$: the flow entering v must be equal to the flow leaving v .

The *value* of a flow is the total outgoing flow from the source s , or equivalently the total flow entering the sink t . A *maximum flow* (or simply a max-flow) is a flow of maximum value.

A core component of our algorithm will be a procedure that takes an integer load threshold T and computes an integer partial assignment x of

jobs to machines such that (a) each job is assigned either to a *low machine* or not at all; (b) the load on any machine is at most T ; and (c) the number of jobs assigned is maximized. Such an assignment can be computed by finding a max-flow in the following flow network. We construct a directed bipartite graph (see Fig. 1) with a node for every job j and every machine i , and an edge (j, i) of infinite (i.e., very large; any number ≥ 1 will suffice) capacity if $p_{ij} = L$. We add a source node s with edges (s, j) having capacity 1, and a sink node t with edges (i, t) having capacity $\lfloor T/L \rfloor$. Figure 1 shows an example with three jobs and two machines, with $p_{i_1j_1} = p_{i_1j_2} = p_{i_2j_1} = L$ and $p_{i_1j_3} = p_{i_2j_2} = p_{i_2j_3} = H$, and $T = L$.

Any integer flow in this network corresponds to an integer partial assignment x , where $x_{ij} = 1$ if and only if there is a flow of 1 on the edge from j to i . Clearly, the load on any machine i due to x is L times the flow on the edge (i, t) and hence is at most T , and the number of jobs assigned by x is equal to the value of the flow. Conversely, any (partial) assignment satisfying (a) and (b) translates to an integer flow in the above network of value equal to the number of jobs assigned by x . Thus, an integer max-flow yields the desired assignment that maximizes the number of assigned jobs. We will therefore use the terms assignment and flow interchangeably. We will refer to the above flow network as the flow network for (p, T) , and an integer max-flow in this network as a max-flow for (p, T) .

The *residual graph* of G with respect to a given flow f is defined as follows: it consists of the same nodes as in G . For every original edge $e = (u, v) \in E$, if $f_e < c_e$ we add edge e to the residual graph with capacity $c_e - f_e$: we call such an edge a *forward edge*; further, if $f_e > 0$ we add edge $e' = (v, u)$, which is called a *backward edge*, with capacity f_e to the residual graph. Figure 2 shows the residual graph of the flow network in Figure 1 with respect the unit-flow $f_{s,j_1} = f_{j_1,i_1} = f_{i_1,t} = 1$ (and $f_e = 0$ for the remaining edges).

A basic important fact about flows is that if a flow is not a max-flow, then there exists a path in the residual graph, called an *augmenting path*, that can be used to increase the value of the flow. We explain this now for our specific flow networks. Let x be an integer flow that is not a max-flow for (p, T) . Then there must be an augmenting path $\mathcal{P} = (s, j_1, i_1, j_2, i_2, \dots, j_K, i_K, t)$ in the residual graph. Here the edges (s, j_1) , (j_ℓ, i_ℓ) for $\ell = 1, \dots, K$, and (i_K, t) are forward edges, and the edges $(i_\ell, j_{\ell+1})$ for $\ell = 1, \dots, K - 1$ are backward edges. This augmenting path denotes that in the current assignment x , j_1 is unassigned, $x_{i_\ell j_\ell} = 0$ as denoted by the forward edges (j_ℓ, i_ℓ) , and jobs j_2, \dots, j_K are assigned to i_1, \dots, i_{K-1} respectively, (i.e., $x_{i_\ell j_{\ell+1}} = 1$) as

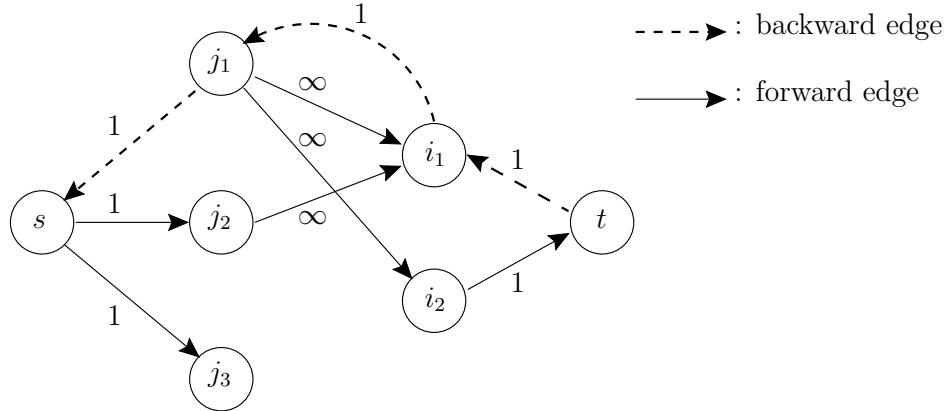


Figure 2: Residual graph of the network in Figure 1 wrt. the flow/assignment $j_1 \mapsto i_1$.

denoted by the backward edges $(i_\ell, j_{\ell+1})$. Augmenting x using \mathcal{P} means that we change the flow/assignment so that each j_ℓ is assigned to i_ℓ in the new assignment. Notice that j_1 is now assigned, so this increases the value of the flow by 1. Observe that any simple augmenting path does not decrease the load of any machine. In the example in Figure 2, a possible (simple) augmenting path is $(s, j_2, i_1, j_1, i_2, t)$. This modifies the original assignment (job j_1 to machine i_1) to the new assignment: j_1 to i_2 and j_2 to i_1 .

The existence of augmenting paths leads to a simple algorithm for computing a maximum flow, called the Ford-Fulkerson algorithm (Ford and Fulkerson (1956); see also Ahuja et al. (1993); Kleinberg and Tardos (2006)). We start off with the zero-flow (i.e., $f_e = 0$ for all edges), and then repeatedly use augmenting paths to increase the value of the flow until no such path exists. This iterative algorithm has one important implication that will be quite useful: there always exists an integer max-flow if all capacities are integers. For example, in the flow network for (p, T) , this follows since we start with an integral flow (the zero-flow), and every time we use an augmenting path to augment an integral flow, we get a new integral flow.

We need to introduce one additional concept that is needed for our algorithm. There could potentially be many max-flows, and we will be interested in the most “balanced” ones, which we formally define as follows. Fix some max-flow. Let $n_{p,T}^i$ be the amount of flow on edge (i, t) (or equivalently the number of jobs assigned to i in the corresponding schedule), and let $n_{p,T}$ be the total value of the max-flow, i.e., $n_{p,T} = \sum_i n_{p,T}^i$. For any $T' \leq T$, define

$n_{p,T}^i|_{T'} = \min(n_{p,T}^i, \lfloor T'/L \rfloor)$, that is, we “truncate” the flow/assignment on i so that the total load on i is at most T' . Define $n_{p,T}|_{T'} = \sum_i n_{p,T}^i|_{T'}$. We define a *prefix-maximal* flow or assignment for T as follows.

Definition 5.1 (Prefix-maximal flow) *A flow for the above network with threshold T is prefix-maximal if for every integer $T' \leq T$, we have $n_{p,T}|_{T'} = n_{p,T'}$.*

That is, in a prefix-maximal flow for (p, T) , if we truncate the flow at some $T' \leq T$, we are left with a max-flow for (p, T') . A prefix-maximal flow for a threshold T always exists, as shown by the following algorithm: compute a max-flow for threshold 1, use simple augmenting paths to augment it to a max-flow for threshold 2, and repeat, each time augmenting the max-flow for the previous threshold t to a max-flow for threshold $t + 1$ using simple augmenting paths. Since, in each such transition, the load on each machine never decreases, this results in a prefix-maximal flow.

5.2. A cycle-monotone approximation algorithm

We now describe our cycle-monotone 2-approximation algorithm. The algorithm consists of three steps. First, we compute a large enough threshold $T^*(p)$, so that after we maximally pack the low jobs up to this threshold, there is enough room left (over all machines) within the threshold to fit in the remaining high jobs (fractionally). We then maximally pack the low jobs up to the threshold $T^*(p)$ by a max-flow computation (as detailed above). Finally, we assign the remaining jobs (which will be high jobs) greedily by assigning them iteratively to the current least-loaded machine.

Algorithm 2 Given a vector of processing times p , construct an assignment of jobs to machines as follows.

1. Compute $T^*(p) = \min \{T \geq H, T \text{ multiple of } L : n_{p,T} \cdot L + (n - n_{p,T}) \cdot H \leq m \cdot T\}$.⁵ Note that $n_{p,T} \cdot L + (n - n_{p,T}) \cdot H - m \cdot T$ is a decreasing function of T , so $T^*(p)$ can be computed in polynomial time via binary search.
2. Compute a prefix-maximal flow for threshold $T^*(p)$ and the corresponding partial assignment (i.e., j is assigned to i iff there is 1 unit of flow on edge (j, i)).

⁵The restriction to multiples of L is not essential, but simplifies certain arguments.

3. Assign the remaining jobs, i.e., the jobs unassigned in the flow-phase, in a greedy manner as follows. Consider these jobs in an arbitrary order and assign each job to the machine with the current lowest load (where the load includes the jobs assigned in the flow-phase).

Our algorithm needs to compute a prefix-maximal assignment for the threshold $T^*(p)$. The proof showing the existence of a prefix-maximal flow yields an algorithm for computing it that uses at most n augmenting paths overall, since there are only n jobs to assign (so the value of the max-flow is at most n , for any threshold T). Thus, this algorithm has polynomial running time.

The main theorem of this section is as follows.

Theorem 5.2 *Algorithm 2 is a polynomial time deterministic 2-approximation algorithm for the two-values scheduling domain, and is implementable. Moreover, one can efficiently compute payments that yield a truthful mechanism.*

In Sections 5.3 and 5.4, we prove that Algorithm 2 is a 2-approximation cycle-monotone algorithm. This will then allow us to compute payments in Section 5.5 and prove Theorem 5.2.

5.3. Proof of approximation

The proof of the approximation guarantee is fairly straightforward. Recall that $OPT(p)$ denotes the optimal makespan over integer schedules for p . It is easy to show that if $OPT(p) < H$ then the makespan returned is at most $OPT(p)$, since both our schedule and the optimal schedule would then assign all jobs to low machines. Suppose $OPT(p) > H$. In Claim 5.4 we argue that $T^*(p) \leq OPT(p) + L$. This follows because if we take $T = L \cdot \lceil \frac{OPT(p)}{L} \rceil \leq OPT(p) + L$ as a candidate threshold then $n_{p,T}$ is at least the number of jobs assigned to low machines in an optimum schedule (since $T \geq OPT(p)$); so the total load after the remaining high-jobs are assigned is at most $m \cdot OPT(p) \leq m \cdot T$ implying that $T^*(p) \leq T$. We next show (Lemma 5.6) that the greedy scheduling of jobs in step 3 results in makespan at most $T^*(p) + H$ (roughly). The proof is a standard list-scheduling argument showing that a larger makespan would imply that the total load after the high-machines are assigned is strictly greater than $m \cdot T^*(p)$, which is a contradiction. These two facts imply the 2-approximation guarantee.

Claim 5.3 *If $OPT(p) < H$, the makespan is at most $OPT(p)$.*

Proof : If $OPT(p) < H$, it must be that the optimal schedule assigns all jobs to low machines, so $n_{p,OPT(p)} = n$. Thus, we have $T^*(p) = L \cdot \lceil \frac{H}{L} \rceil$. Furthermore, since we compute a prefix-maximal flow for threshold $T^*(p)$ we have $n_{p,T^*(p)}|_{OPT(p)} = n_{p,OPT(p)} = n$, which implies that the load on each machine is at most $OPT(p)$. So the makespan is at most (and hence exactly) $OPT(p)$. ■

Claim 5.4 *If $OPT(p) \geq H$, then $T^*(p) \leq L \cdot \lceil \frac{OPT(p)}{L} \rceil \leq OPT(p) + L$.*

Proof : Let $n_{OPT(p)}$ be the number of jobs assigned to low machines in an optimum schedule. The total load on all machines in this optimum schedule is exactly $n_{OPT(p)} \cdot L + (n - n_{OPT(p)}) \cdot H$, and is at most $m \cdot OPT(p)$, since every machine has load at most $OPT(p)$. So taking $T = L \cdot \lceil \frac{OPT(p)}{L} \rceil \geq H$, since $n_{p,T} \geq n_{OPT(p)}$ we have that $n_{p,T} \cdot L + (n - n_{p,T}) \cdot H \leq m \cdot T$. Hence, $T^*(p)$, which is the smallest such T , is at most $L \cdot \lceil \frac{OPT(p)}{L} \rceil$. ■

Claim 5.5 *Each job assigned in step 3 of the algorithm is assigned to a high machine.*

Proof : Suppose j is assigned to machine i in step 3. If $p_{ij} = L$, then we must have $n_{p,T^*(p)}^i = T^*(p)/L$, otherwise we could have assigned j to i in step 2 to obtain a flow of larger value. So at the point just before j is assigned in step 3, the load of each machine must be at least $T^*(p)$. Hence, the total load *after* j is assigned is at least $m \cdot T^*(p) + L > m \cdot T^*(p)$. But the total load is also at most $n_{p,T^*(p)} \cdot L + (n - n_{p,T^*(p)}) \cdot H \leq m \cdot T^*(p)$, yielding a contradiction. ■

Lemma 5.6 *The above algorithm returns a schedule with makespan at most $OPT(p) + \min\{H, OPT(p)\}$.*

Proof : If $OPT(p) < H$, then by Claim 5.3, we are done. So suppose $OPT(p) \geq H$. By Claim 5.4, we know that $T^*(p) \leq OPT(p) + L$. If there are no unassigned jobs after step 2 of the algorithm, then the makespan is at most $T^*(p)$ and we are done. So assume that there are some unassigned jobs after step 2. To complete the proof, we will show that the makespan after step 3 is at most $T + H(1 - \frac{1}{m})$ where $T = \min\{T^*(p), OPT(p)\}$. Suppose the claim is false. Let i be the machine with the maximum load,

so $l_i > T + H(1 - \frac{1}{m})$. Let j be the last job assigned to i in step 3, and consider the point just before it is assigned to i . So $l_i > T - H/m$ at this point. Also since j is assigned to i , by our greedy rule, the load on all the other machines must be at least l_i . So the total load *after* j is assigned, is at least $H + m \cdot l_i > m \cdot T$ (since $p_{ij} = H$ by Claim 5.5). Also, for *any* assignment of jobs to machines in step 3, the total load is at most $n_{p, T^*(p)} \cdot L + (n - n_{p, T^*(p)}) \cdot H$ since there are $n_{p, T^*(p)}$ jobs assigned to low machines. Therefore, we must have $m \cdot T < n_{p, T^*(p)} \cdot L + (n - n_{p, T^*(p)}) \cdot H$. But we will argue that $m \cdot T \geq n_{p, T^*(p)} \cdot L + (n - n_{p, T^*(p)}) \cdot H$, which yields a contradiction.

If $T = T^*(p)$, this follows from the definition of $T^*(p)$. If $T = OPT(p)$, then letting $n_{OPT(p)}$ denote the number of jobs assigned to low machines in an optimum schedule, we have $n_{p, T^*(p)} \geq n_{OPT(p)}$. So $n_{p, T^*(p)} \cdot L + (n - n_{p, T^*(p)}) \cdot H \leq n_{OPT(p)} \cdot L + (n - n_{OPT(p)}) \cdot H$. This is exactly the total load in an optimum schedule, which is at most $m \cdot OPT(p)$. ■

5.4. Proof of cycle monotonicity

The cycle-monotonicity proof is somewhat involved, so we begin by giving a high-level overview. We first simplify condition (3) for our two-values $\{L, H\}$ scheduling domain to obtain a condition that will be more convenient to work with. Throughout, we fix a machine i . Considering (3) and fixing an index k , it is clear that each non-zero term in the inner summation for index k is $\pm(H - L)$. Thus, we can replace (3) by an equivalent, more convenient condition stated in terms of the number of jobs assigned to machine i in x^k that switch from high to low, or the other way round, on machine i when one moves from p^k to p^{k+1} . To this end, we define

$$n_H^{k,\ell} = |\{j : x_{ij}^k = 1, p_{ij}^k = L, p_{ij}^\ell = H\}| \quad (4)$$

$$n_L^{k,\ell} = |\{j : x_{ij}^k = 1, p_{ij}^k = H, p_{ij}^\ell = L\}|. \quad (5)$$

Then, $\sum_j x_{ij}^{k+1}(p_{ij}^k - p_{ij}^{k+1}) = (n_H^{k+1,k} - n_L^{k+1,k})(H - L)$. Plugging this into (3) and dividing by $(H - L)$, we get the following.

Proposition 5.7 *Cycle monotonicity in the two-values scheduling domain is equivalent to the condition that, for every player i , every p_{-i} , every integer K , and every p_i^1, \dots, p_i^K ,*

$$\sum_{k=1}^K (n_H^{k+1,k} - n_L^{k+1,k}) \geq 0. \quad (6)$$

To get a better feel for condition (6), consider for simplicity the case where we have only two inputs $p^1 = (p_i^1, p_{-i})$, $p^2 = (p_i^2, p_{-i})$. Suppose first that $p_i^2 \leq p_i^1$, that is, in moving from p_i^1 to p_i^2 , machine i only decreases some jobs from H to L . Then, (6) simplifies to $n_H^{2,1} \geq n_L^{1,2}$; to put it in words, this requires that the number of jobs assigned to i that were decreased (from H to L) should weakly increase when one moves from p^1 to p^2 . In the general case of arbitrary p^1 and p^2 , one can similarly parse condition (6) as requiring that the difference, (number of jobs assigned to i that were decreased – number of jobs assigned to i that were increased), should weakly increase when one transitions from p^1 to p^2 .

We now give some intuition about why Algorithm 2 should satisfy (6). Consider first the special case where all the thresholds $T^*(p_i^k, p_{-i})$ for $k = 1, \dots, K$ are equal. Let T^* be this common threshold. Let p^k denote the input (p_i^k, p_{-i}) . Consider the schedule x^2 computed for p^2 . Note that the number of jobs assigned by x^2 to low machines is precisely n_{p^2, T^*} . We claim that if we assign to each machine i' all the jobs j for which $x_{ij}^2 = 1$ and $p_{i'j}^1 = L$, then we obtain a valid flow for (p^1, T^*) of value $n_{p^2, T^*} - n_H^{2,1} + n_L^{2,1}$. It is clear that the number of low jobs on each machine $i' \neq i$ is $n_{p^2, T^*}^{i'}$, so the low-load on i' is at most T^* . The set of low jobs on machine i can be written as

$$\{j : x_{ij}^2 = 1, p_{ij}^2 = L\} \setminus \{j : x_{ij}^2 = 1, p_{ij}^2 = L, p_{ij}^1 = H\} \cup \{j : x_{ij}^2 = 1, p_{ij}^2 = H, p_{ij}^1 = L\}.$$

Thus, the number of low jobs on i is exactly $n_{p^2, T^*}^i - n_H^{2,1} + n_L^{2,1}$; also, it is not hard to see that the low-load on i is at most T^* . Since n_{p^1, T^*} is the value of the *max-flow* for (p^1, T^*) , it follows that $n_{p^1, T^*} \geq n_{p^2, T^*} - n_H^{2,1} + n_L^{2,1}$. We can repeat the exact same argument for any p^{k+1} and p^k to obtain the inequality $n_{p^k, T^*} \geq n_{p^{k+1}, T^*} - n_H^{k+1, k} + n_L^{k+1, k}$ (where $K + 1 \equiv 1$). Adding all these inequalities gives precisely (6).

Of course, in general, the thresholds for the different p^k 's will be different, so to prove (6) in general, we will need to (prove and) use certain properties of prefix-maximal flows. By delving into the structure of prefix-maximal flows, we show (essentially) that we can replace T^* in the above analysis (where the common threshold T^* allowed us to “compare across” different p^k 's) by the threshold $T^L = T^*(\vec{L}, p_{-i})$, where \vec{L} is the all-low processing time vector. More precisely, we prove (6) by showing that the inequality $n_{p^1, T^L} \geq n_{p^2, T^L} - n_H^{2,1} + n_L^{2,1}$ (inequality (7)) holds for any $p^1 = (p_i^1, p_{-i})$ and $p^2 = (p_i^2, p_{-i})$. The proof of this inequality requires considering a few cases

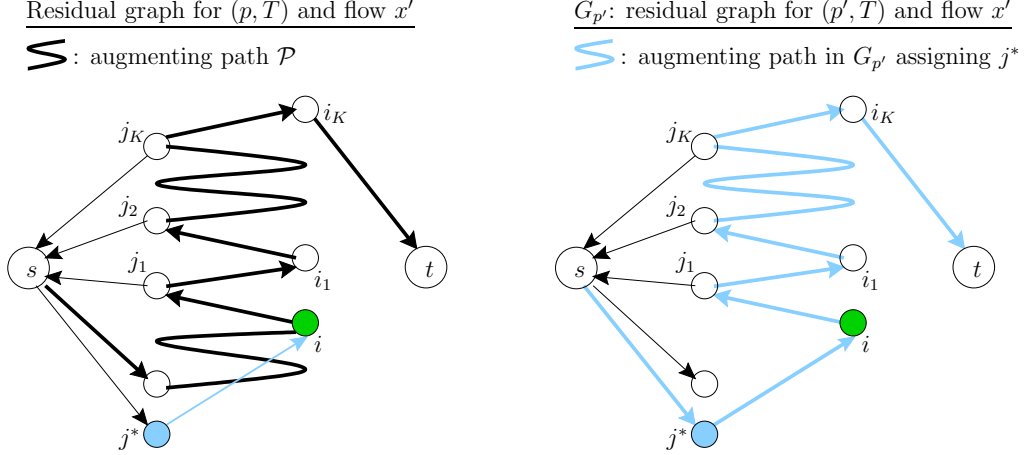


Figure 3: The residual graphs in the proof of Lemma 5.8.

(Lemmas 5.10 and 5.11). A key insight that leads to the proof is exposed in Lemma 5.8 and Corollary 5.9, where we show that if $T^*(p) > T^L$ then it must be that *all* the low-jobs of i are assigned by the T^L -threshold by the assignment computed for p . This allows us to, in some cases, establish the desired inequality by comparing the assignments for p^1 and p^2 truncated at T^L and using a max-flow argument similar to the one used in the common-threshold scenario.

Lemma 5.8 *Consider any two instances $p = (p_i, p_{-i})$ and $p' = (p'_i, p_{-i})$ where $p'_i \geq p_i$, i.e., $p'_{ij} \geq p_{ij} \forall j$. If T is a threshold such that $n_{p,T} > n_{p',T}$, then every maximum flow x' for (p', T) must assign all jobs j such that $p'_{ij} = L$.*

Proof : Let $G_{p'}$ denote the residual graph for (p', T) and flow x' (see Fig. 3). Suppose by contradiction that there exists a job j^* with $p'_{ij^*} = L$ that is unassigned by x' . Since $p'_i \geq p_i$, all edges (j, i) that are present in the network for (p', T) are also present in the network for (p, T) . Thus, x' is a valid flow for (p, T) . But it is not a max-flow, since $n_{p,T} > n_{p',T}$. So there exists an augmenting path \mathcal{P} in the residual graph for (p, T) and flow x' . Observe that node i must be included in \mathcal{P} , otherwise \mathcal{P} would also be an augmenting path in the residual graph $G_{p'}$ contradicting the fact that x' is a max-flow. In particular, this implies that there is a path $\mathcal{P}' \subset \mathcal{P}$ from i to the sink t . Let $\mathcal{P}' = (i, j_1, i_1, \dots, j_K, i_K, t)$. All the edges of \mathcal{P}'

are also present as edges in $G_{p'}$ — all backward edges $(i_\ell, j_{\ell+1})$ are present since such an edge implies that $x'_{i_\ell j_{\ell+1}} = 1$; all forward edges (j_ℓ, i_ℓ) are present since $i_\ell \neq i$ so $p'_{i_\ell j_\ell} = p_{i_\ell j_\ell} = L$, and $x'_{i_\ell j_{\ell+1}} = 0$. But then there is an augmenting path $(s, j^*, i, j_1, i_1, \dots, j_K, i_K, t)$ in $G_{p'}$ (see Fig. 3) which contradicts the maximality of x' . ■

Let \vec{L} denote the all-low processing-time vector. Define $T_i^L(p_{-i}) = T^*(\vec{L}, p_{-i})$, that is, the value of T^* for the input $p = (\vec{L}, p_{-i})$. Since we are focusing on machine i , and p_{-i} is fixed throughout, we abbreviate $T_i^L(p_{-i})$ to T^L . Also, let $p^L = (\vec{L}, p_{-i})$. Note that $T^*(p) \geq T^L$ for every instance $p = (p_i, p_{-i})$.

Corollary 5.9 *Let $p = (p_i, p_{-i})$ be any instance and let x be any prefix-maximal flow for $(p, T^*(p))$. Then, the low-load on machine i is at most T^L .*

Proof : Let $T^* = T^*(p)$. If $T^* = T^L$, then this is clearly true. Otherwise, consider the assignment x' obtained by truncating x at T^L . Since x is prefix-maximal, we know that x' constitutes a max-flow for (p, T^L) . Also, $n_{p, T^L} < n_{p^L, T^L}$ because $T^* > T^L$. So by Lemma 5.8, this truncated flow must assign *all* the low jobs of i . By the definition of a truncated flow, it follows that $x_{ij} \geq x'_{ij}$ for all i, j , so if a job j is assigned by x' (i.e., $\sum_{i'} x'_{ij} = 1$) then its machine-assignment is identical in x' and x . In particular, the assignment of the low jobs of i remains the same in x' and x , and hence, the (low-)load on machine i does not change when we move from x' to x . Thus, the low-load of i (under the assignment x) is at most T^L . ■

Using these properties, we will prove the following key inequality: for any $p^1 = (p_i^1, p_{-i})$ and $p^2 = (p_i^2, p_{-i})$,

$$n_{p^1, T^L} \geq n_{p^2, T^L} - n_H^{2,1} + n_L^{2,1} \quad (7)$$

where $n_H^{2,1}$ and $n_L^{2,1}$ are as defined in (4) and (5), respectively. Notice that this immediately implies cycle monotonicity, since if we take $p^1 = p^k$ and $p^2 = p^{k+1}$, then (7) implies that $n_{p^k, T^L} \geq n_{p^{k+1}, T^L} - n_H^{k+1, k} + n_L^{k+1, k}$; summing this over all $k = 1, \dots, K$ gives (6).

Let $T^1 = T^*(p^1)$ and $T^2 = T^*(p^2)$.

Lemma 5.10 *If $T^*(p^1) > T^L$, then (7) holds.*

Proof : Take the prefix-maximal flow x^2 for (p^2, T^2) , truncate it at T^L , and remove all the jobs from this assignment that are counted in $n_H^{2,1}$, that is, all jobs j such that $x_{ij}^2 = 1$, $p_{ij}^2 = L$, $p_{ij}^1 = H$. Denote this flow by x . Observe that x is a valid flow for (p^1, T^L) , and the value of this flow is *exactly* $n_{p^2, T^2}|_{T^L} - n_H^{2,1} = n_{p^2, T^L} - n_H^{2,1}$. Also none of the jobs that are counted in $n_L^{2,1}$ are assigned by x since each such job j is high on i in p^2 . Since $T^1 > T^L$, we must have $n_{p^1, T^L} < n_{p^L, T^L}$. So if we augment x to a max-flow for (p^1, T^L) , then by Lemma 5.8 (with $p = p^L$ and $p' = p^1$), all the jobs corresponding to $n_L^{2,1}$ must be assigned in this max-flow. Thus, the value of this max-flow is at least (value of x) + $n_L^{2,1}$, that is, $n_{p^1, T^L} \geq n_{p^2, T^L} - n_H^{2,1} + n_L^{2,1}$, as claimed. ■

Lemma 5.11 *Suppose $T^*(p^1) = T^L$. Then (7) holds.*

Proof : Let x^1, x^2 be the complete assignment, that is, the assignment after both steps 2 and 3, computed by our algorithm for p^1, p^2 respectively. Let $S = \{j : x_{ij}^2 = 1 \text{ and } p_{ij}^2 = L\}$ and $S'' = \{j : x_{ij}^2 = 1 \text{ and } p_{ij}^1 = L\}$. Therefore, $|S''| = |S| - n_H^{2,1} + n_L^{2,1}$ and $|S| = n_{p^2, T^2}^i = n_{p^2, T^2}^i|_{T^L}$ (by Corollary 5.9). Let $T'' = |S''| \cdot L$. We consider two cases.

Suppose first that $T'' \leq T^L$. Consider the following flow for (p^1, T^L) : assign to every machine other than i the low-assignment of x^2 truncated at T^L , and assign the jobs in S'' to machine i . This is a valid flow for (p^1, T^L) since the load on i is $T'' \leq T^L$. Its value is equal to $\sum_{i' \neq i} n_{p^2, T^2}^{i'}|_{T^L} + |S''| = n_{p^2, T^2}|_{T^L} - n_H^{2,1} + n_L^{2,1} = n_{p^2, T^L} - n_H^{2,1} + n_L^{2,1}$. The value of the max-flow for (p^1, T^L) is no smaller, and the claim follows.

Now suppose $T'' > T^L$, so $T'' \geq T^L + L$ since T'' and T^L are both multiples of L . Since $|S| \cdot L \leq T^L$ (by Corollary 5.9), it follows that $n_L^{2,1} > n_H^{2,1} \geq 0$. Let $M = n_{p^2, T^2} - n_H^{2,1} + n_L^{2,1} = |S''| + \sum_{i' \neq i} n_{p^2, T^2}^{i'}$. We first show that

$$m \cdot T^L < M \cdot L + (n - M) \cdot H. \quad (8)$$

Let N be the number of jobs assigned to machine i in x^2 . We have $|S''| \cdot L + (N - |S''|) \cdot H \geq |S''| \cdot L > T^L$. Now consider the point in the execution of the algorithm on instance p^2 just before the last high job is assigned to i in Step 3 (there must be such a job since $n_L^{2,1} > 0$). The load on i at this point is $|S| \cdot L + (N - |S| - 1) \cdot H \geq |S''| \cdot L - L - (n_L^{2,1} - 1) \cdot L + (N - |S| - 1) \cdot H$, which is at least $|S''| \cdot L - L \geq T^L$ since $n_L^{2,1} - 1 \leq N - |S| - 1$. By the greedy property, every $i' \neq i$ also has at least this load at this point, so $\sum_j p_{i'j}^2 x_{i'j}^2 \geq T^L$.

Adding these inequalities for all $i' \neq i$, and the earlier inequality for i , we get that $|S''| \cdot L + (N - |S''|) \cdot H + \sum_{i' \neq i} \sum_j p_{i'j}^2 x_{i'j}^2 > mT^L$. But the left-hand-side is exactly $M \cdot L + (n - M) \cdot H$.

On the other hand, since $T^1 = T^L$, we have

$$m \cdot T^L \geq n_{p^1, T^L} \cdot L + (n - n_{p^1, T^L}) \cdot H. \quad (9)$$

Combining (8) and (9), we get that $n_{p^1, T^L} > M = n_{p^2, T^2} - n_H^{2,1} + n_L^{2,1} \geq n_{p^2, T^L} - n_H^{2,1} + n_L^{2,1}$. ■

Lemma 5.12 *Algorithm 2 satisfies cycle monotonicity.*

Proof: Taking $p^1 = p^k$ and $p^2 = p^{k+1}$ in (7), we get that $n_{p^k, T^L} \geq n_{p^{k+1}, T^L} - n_H^{k+1, k} + n_L^{k+1, k}$. Summing this over all $k = 1, \dots, K$ (where $K + 1 \equiv 1$) yields (6). ■

5.5. Computation of prices

Lemmas 5.6 and 5.12 show that our algorithm is a 2-approximation algorithm that satisfies cycle monotonicity. Thus, by the discussion in Section 3, there exist prices that yield a truthful mechanism. To obtain a *polynomial-time mechanism*, we also need to show how to compute these prices (or payments) in polynomial-time. It is not clear, if the procedure outlined in Section 3 based on computing shortest paths in the allocation graph yields a polynomial time algorithm, since the allocation graph has an exponential number of nodes (one for each output assignment). Instead of analyzing the allocation graph, we will leverage our proof of cycle monotonicity, in particular, inequality (7), and simply spell out the payments.

Recall that the utility of a player is $u_i = P_i - l_i$, where P_i is the payment made to player i . For convenience, we will first specify negative payments (i.e., the P_i s will actually be *prices charged* to the players) and then show that these can be modified so that players have non-negative utilities (if they act truthfully). Let \mathcal{H}^i denote the number of jobs assigned to machine i in step 3. By Corollary 5.5, we know that all these jobs are assigned to high machines (according to the declared p_i s). Let $\mathcal{H}^{-i} = \sum_{i' \neq i} \mathcal{H}^{i'}$ and $n_{p, T}^{-i} = \sum_{i' \neq i} n_{p, T}^{i'}$. The payment P_i to player i is defined as:

$$P_i(p) = -L \cdot n_{p, T^*(p)}^{-i} - H \cdot \mathcal{H}^{-i}(p) - (H - L)(n_{p, T^*(p)} - n_{p, T^L}) \quad (10)$$

Recall that T^L denotes $T_i^L(p_{-i}) = T^*(\vec{L}, p_{-i})$. We can interpret our payments as equating the player's cost to a careful modification of the total load (in the spirit of VCG prices). The first and second terms in (10), when subtracted from i 's load l_i equate i 's cost to the total load. The term $n_{p, T^*(p)} - n_{p, T^L}$ is in fact equal to $n_{p, T^*(p)}^{-i} - n_{p, T^*(p)}^{-i}|_{T^L}$ since the low-load on i is at most T^L (by Corollary 5.9). Thus the last term in equation (10) implies that we treat the low jobs that were assigned beyond the T^L -threshold (to machines other than i) effectively as high jobs for the total utility calculation from i 's point of view. It is not clear how one could have conjured up these payments *a priori* in order to prove the truthfulness of our algorithm. However, by relying on cycle monotonicity, we were not only able to argue the *existence* of payments, but also our proof paved the way for actually inferring these payments. The following lemma explicitly verifies that the payments defined above do indeed give a truthful mechanism.

Lemma 5.13 *Fix a player i and the other players' declarations p_{-i} . Let i 's true type be p_i^1 . Then, under the payments defined in (10), i 's utility when she declares her true type p_i^1 is at least her utility when she declares any other type p_i^2 .*

Proof : Let c_i^1 and c_i^2 denote i 's total cost, defined as the negative of her utility, under the inputs $p^1 = (p_i^1, p_{-i})$ and $p^2 = (p_i^2, p_{-i})$ respectively. Since p_{-i} is fixed, we omit p_{-i} from the expressions below for notational clarity. The true load of i when she declares her true type p_i^1 is $L \cdot n_{p^1, T^*(p^1)}^i + H \cdot \mathcal{H}^i(p^1)$, and therefore

$$\begin{aligned} c_i^1 &= L \cdot n_{p^1, T^*(p^1)} + H \cdot (n - n_{p^1, T^*(p^1)}) + (H - L)(n_{p^1, T^*(p^1)} - n_{p^1, T^L}) \\ &= n \cdot H - (H - L)n_{p^1, T^L}. \end{aligned} \tag{11}$$

On the other hand, i 's true load when she declares p_i^2 is $L \cdot (n_{p^2, T^*(p^2)}^i - n_H^{2,1} + n_L^{2,1}) + H \cdot (\mathcal{H}^i + n_H^{2,1} - n_L^{2,1})$ (since i 's true processing-time vector is p_i^1), and thus

$$c_i^2 = n \cdot H - (H - L)n_{p^2, T^L} + (H - L)n_H^{2,1} - (H - L)n_L^{2,1}.$$

Thus, (7) implies that $c_i^1 \leq c_i^2$. ■

Price specifications are commonly required to satisfy, in addition to truthfulness, *individual rationality*, that is, a player's utility should be non-negative

if she reveals her true value. The payments given by (10) are not individually rational as they actually charge a player a certain amount. However, it is well-known that this problem can be easily solved by adding a large-enough constant to the price definition. In our case, for example, letting \vec{H} denote the vector of all H 's, we can add the term $n \cdot H - (H - L)n_{(\vec{H}, p_{-i}), T^L}$ to (10). Note that this is a constant for player i . Thus, the new payments are

$$Q_i(p) = n \cdot H - L \cdot n_{p, T^*(p)}^{-i} - H \cdot \mathcal{H}^{-i}(p) - (H - L)(n_{p, T^*(p)} - n_{p, T^L} + n_{(\vec{H}, p_{-i}), T^L}).$$

As shown by (11), this will indeed result in a non-negative utility for i (since $n_{(\vec{H}, p_{-i}), T^L} \leq n_{(p_i, p_{-i}), T^L}$ for any type p_i of player i). This modification also ensures the additionally desired normalization property that if a player receives no jobs then she receives zero payment: if player i receives the empty set for some type p_i then she will also receive the empty set for the type \vec{H} (this is easy to verify for our specific algorithm), and for the type \vec{H} , her utility equals zero; thus, by truthfulness this must also be the utility of every other declaration that results in i receiving the empty set. This completes the proof of Theorem 5.2.

5.6. The impossibility of exact implementation

We now show that irrespective of computational considerations, there does not exist a cycle-monotone algorithm for the L - H case with an approximation ratio better than 1.1. Let $H = \alpha \cdot L$ for some $2 < \alpha \leq 2.5$ that we will choose later. There are two machines I, II and seven jobs. Consider the following two scenarios:

Scenario 1. Every job has the same processing time on both machines: jobs 1–5, are L , and jobs 6, 7 are H . Any optimal schedule assigns jobs 1–5 to one machine and jobs 6, 7 to the other, and has makespan $OPT_1 = 5L$. The second-best schedule has makespan at least $Second_1 = H + 3L$.

Scenario 2. If the algorithm chooses an optimal schedule for scenario 1, assume without loss of generality that jobs 6, 7 are assigned to machine II. In scenario 2, machine I has the same processing-time vector. Machine II lowers jobs 6, 7 to L and increases 1–5 to H . An optimal schedule has makespan $2L + H$, where machine II gets jobs 6, 7 and one of the jobs 1–5. The second-best schedule for this scenario has makespan at least $Second_2 = 5L$.

Theorem 5.14 *No deterministic truthful mechanism for the two-value scheduling problem can obtain an approximation ratio better than 1.1.*

Proof : We argue that a cycle-monotone algorithm cannot choose the optimal schedule in both scenarios. Suppose the algorithm chooses an optimum schedule for scenario 1. So machine II is the one on which jobs 6, 7 are scheduled, and whose processing-time vector changes in scenario 2. Now the algorithm cannot choose an optimal schedule for scenario 2, otherwise cycle monotonicity is violated for machine II. Taking p_{II}^1, p_{II}^2 to be machine II's processing-time vectors for scenarios 1, 2 respectively, we get $\sum_j (p_{II,j}^1 - p_{II,j}^2)(x_{II,j}^2 - x_{II,j}^1) = (L - H)(1 - 0) < 0$. Thus, any truthful mechanism must return a sub-optimal makespan in at least one scenario, and therefore its approximation ratio is at least $\min\{\frac{Second_1}{OPT_1}, \frac{Second_2}{OPT_2}\}$, which attains its maximum value of 1.1 when $\alpha = 2.5$ (recall that $\alpha \in (2, 2.5]$). ■

6. Truthful mechanisms for equal-ratio job-dependent two-values

We now consider the $\{L_j, H_j\}$ -case where there is a common ratio $r = \frac{H_j}{L_j}$ for all jobs j . Notice that this setting also captures the restricted-machines scheduling problem, since one can choose r to be a very large number. First, in Section 6.1, we show that the equal-ratio setting reduces to the two-values problem discussed in Section 5. This reduction will immediately show that for any r , Algorithm 2 combined with the prices in Section 5.5 gives a *fractional* truthful mechanism that outputs a fractional assignment of makespan at most $2 \cdot OPT(p)$. (Recall that $OPT(p)$ denotes the optimal makespan over *integer* schedules for input p .)

In Section 6.2, we improve upon the performance guarantee of 2 obtained via the reduction detailed in Section 6.1. We build upon the ideas introduced in Section 5 to design a fractional truthful mechanism (with polynomial-time computable prices) that returns an assignment of makespan at most $OPT(p)$ for every equal-ratio instance p . Unfortunately, neither the assignment x returned by this algorithm nor the one returned by Algorithm 2 need satisfy the condition that $x_{ij} > 0$ implies that p_{ij} is bounded with respect to $makespan(x)$ for arbitrary r . Therefore, we cannot use Lemmas 4.1 and 4.2 to obtain a truthful-in-expectation mechanism with constant approximation ratio. We leave this as an open problem.

6.1. A reduction from the equal-ratio $\{L_j, H_j\}$ -case to the $\{L, H\}$ -case

Recall that the L_j s and H_j s are all integers. We view each job j as consisting of L_j sub-jobs: each of these sub-jobs has size 1 on machine i if

$p_{ij} = L_j$, and size r if $p_{ij} = H_j$. Let \tilde{p} denote this new instance. Notice the following two simple, but important facts:

- Every sub-job has size 1 or r on every machine, so \tilde{p} is an instance of the two-values problem discussed in Section 5.
- Every assignment \tilde{x} for the instance \tilde{p} yields a corresponding fractional assignment x for p , where $x_{ij} = (\sum_{j': \text{sub-job of } j} \tilde{x}_{ij'})/L_j$. So $p_{ij}x_{ij} = \sum_{j': \text{sub-job of } j} \tilde{p}_{ij'}\tilde{x}_{ij'}$, and the load on every machine i under assignment \tilde{x} for instance \tilde{p} is the same as its load under assignment x for instance p . Conversely, every assignment x for the instance p yields an assignment \tilde{x} for \tilde{p} , where we set $\tilde{x}_{ij'} = x_{ij}$ for every sub-job j' of j . So we again have $p_{ij}x_{ij} = \sum_{j': \text{sub-job of } j} \tilde{p}_{ij'}\tilde{x}_{ij'}$.

We set $f(p)$ to be the assignment x obtained as above (by converting the assignment \tilde{x} returned by Algorithm 2 on input \tilde{p}). The second fact implies (by Lemma 5.6) that the makespan of x is at most $2 \cdot OPT(\tilde{p}) \leq 2 \cdot OPT(p)$. We set $P_i(p) = Q_i(\tilde{p})$, where $Q_i(\cdot)$ is the price defined in Section 5.5. So for every machine i , every p_{-i} , every true processing-time vector p_i , and every p'_i , we obtain that

$$\begin{aligned} P_i(p_i, p_{-i}) - l_i(p_i, p_{-i}) &= Q_i(\tilde{p}_i, \tilde{p}_{-i}) - \tilde{l}_i(\tilde{p}_i, \tilde{p}_{-i}) \\ &\geq Q_i(\tilde{p}'_i, \tilde{p}_{-i}) - \tilde{l}_i(\tilde{p}'_i, \tilde{p}_{-i}) = P_i(p'_i, p_{-i}) - l_i(p'_i, p_{-i}) \end{aligned}$$

where l_i and \tilde{l}_i denote the load on machine i according to the true processing-time vectors p_i and \tilde{p}_i respectively. Thus, $M = (f, P)$ is an individually-rational 2-approximation fractional truthful mechanism for the equal-ratio job-dependent two-value setting.

Implementing the above reduction directly yields only a pseudopolynomial time algorithm, that is, an algorithm whose running time is polynomial in $\sum_j H_j$. Instead, we will show that for the input \tilde{p} obtained as above, Steps 1 and 2 of Algorithm 2, that is, the computation of $T^*(\tilde{p})$ and a prefix-maximal flow for $T^*(\tilde{p})$, can be implemented in polynomial time via a suitable max-flow computation. We remark that Step 3 of the algorithm can be implemented in polynomial time. For each job j , we can compute which machine i will have the least load after all remaining sub-jobs of j have been assigned, and thus compute how many sub-jobs of j get assigned to each machine with load at most l_i .

As before, we say that j is a low job of i , or equivalently i is a low-machine for j , if $p_{ij} = L_j$; the low-load of i is $\sum_{j: p_{ij}=L_j} p_{ij}x_{ij}$. Notice that the number

of sub-jobs in \tilde{p} assigned to low-machines in Step 2 of Algorithm 2 is simply the total *work* assigned to low-machines. We compute the maximum amount of work that can be assigned to low machines subject to a load threshold of T by computing a max-flow in the following network, which can be viewed as a compact representation of the flow network (described in Section 5.1) for (\tilde{p}, T) . We create a bipartite graph with nodes for every job j and machine i , a source s and sink t . We add edges (s, j) with capacity L_j , edges (i, t) with capacity T , and edges (j, i) with infinite (any number that is at least $\min(L_j, T)$ will suffice) capacity if $p_{ij} = L_j$. Any flow g in this network corresponds to a *fractional* partial assignment x of makespan at most T , where $x_{ij} = g(j, i)/L_j$.

As in Section 5.1, we define $n_{p,T}$ to be the value of the max-flow (i.e., the total work assigned to all machines) in the above network. Since all capacities are integers there always exists an integer max-flow. Observe that $n_{p,T}$ is precisely the value of the max-flow in the network for (\tilde{p}, T) . Let $n_{p,T}^i$ be the flow on edge (i, t) (i.e., the total work assigned to machine i), so $n_{p,T} = \sum_i n_{p,T}^i$. Define $n_{p,T}^i|_{T'} = \min(n_{p,T}^i, T')$ and $n_{p,T}|_{T'} = \sum_i n_{p,T}^i|_{T'}$. We say that g is *prefix-maximal* if for every integer $T' \leq T$, we have $n_{p,T}|_{T'} = n_{p,T'}$. As before, prefix-maximal flows exist, since one can iteratively augment (via simple augmenting paths) the prefix-maximal flow for $T - 1$ to obtain a prefix-maximal flow for T . But this again only yields a pseudopolynomial time algorithm. We next describe how to compute a prefix-maximal flow for a threshold T efficiently.

6.1.1. Computing prefix-maximal flows

We reduce the problem of computing a prefix-maximal flow for a threshold T to a *minimum-cost flow problem* with convex cost functions (see, e.g., Ahuja et al. (1993)). The directed graph is the same as the one detailed above. In addition to capacities, edges now also have costs associated with them. All the edges (s, j) and (j, i) have 0 cost, and we give each edge (i, t) the convex cost function $C(x) = x^2$. Given a flow g , its *cost* is given by $\sum_i C(g(i, t)) = \sum_i g(i, t)^2$. We seek an integer max-flow in this flow network whose cost is minimum over all integer max-flows. It is known that such a flow can be computed in time $\text{poly}(\log(T), \sum_j \log L_j)$ (Ahuja et al., 1993; Goldberg and Tarjan, 1990); since T is at most $\sum_j H_j$, this is polynomial in the number of bits required to encode the scheduling instance.

Let g be a min-cost integer max-flow in the above flow network. We show that g is a prefix-maximal flow for threshold T . For an integer $x \geq 1$, let

$\delta(x) = C(x) - C(x - 1) = 2x - 1$. We will use the following property, which is a consequence of the well-known fact that the residual graph of a min-cost flow contains no negative cycles (see, e.g., Ahuja et al. (1993)):

$$\begin{aligned} &\text{for any path } \mathcal{P} = (i_1, j_1, \dots, j_{K-1}, i_K) \text{ in the residual} \\ &\text{graph where } g(i_1, t) > 0 \text{ and } g(i_K, t) < T, \text{ we have} \quad (*) \\ &\quad -\delta(g(i_1, t)) + \delta(g(i_K, t) + 1) \geq 0. \end{aligned}$$

Lemma 6.1 *The min-cost integer max-flow g is prefix-maximal for threshold T .*

Proof : Clearly g is a max-flow for T , so consider any integer $T' < T$ and suppose that $n_{p,T|T'} < n_{p,T'}$. That is, if g' is the flow of g truncated to threshold T' , then g' is not a max-flow for threshold T' . As stated in Section 5.1, this implies that there exists an augmenting path $\mathcal{P} = (s, j_1, i_1, \dots, j_K, i_K, t)$ in the residual graph for (p, T') and flow g' . Thus, we have $g'(i_K, t) = g(i_K, t) \leq T' - 1$. Now consider the residual graph for (p, T) and flow g . Since $g \geq g'$, all the backward edges $(i_\ell, j_{\ell+1})$ for $\ell = 1, \dots, K - 1$ of path \mathcal{P} also appear in this residual graph. Also, the edges (j_ℓ, i_ℓ) for $\ell = 1, \dots, K$ appear as forward edges since these edges have infinite capacity. Thus, there is path $(j_1, i_1, \dots, j_K, i_K, t)$ in this residual graph. Since g is a max-flow for T , the edge (s, j_1) must be saturated by the flow g (otherwise \mathcal{P} would be an augmenting path in the residual graph for (p, T) and flow g). Thus, j_1 is assigned to a greater extent by the flow g than by the flow g' . So there must be some machine i (which could be one of i_1, \dots, i_{K-1}) such that $g(j_1, i) > g'(j_1, i)$. This in turn implies that $g(i, t) > g'(i, t)$, so $g(i, t) > T'$. Thus, we have a path $(i, j_1, i_1, \dots, j_K, i_K)$ in the residual graph for (p, T) and flow g , where $g(i, t) \geq T' + 1$ and $g(i_K, t) \leq T' - 1$. This gives a contradiction to $(*)$ since $-\delta(g(i, t)) + \delta(g(i_K, t) + 1) \leq -2 < 0$. ■

We obtain the following theorem.

Theorem 6.2 *The mechanism $M = (f, P)$ obtained by the above reduction is a fractional truthful mechanism for the the equal-ratio job-dependent two-values domain, that on each such instance p returns a (fractional) schedule with makespan at most $2 \cdot OPT(p)$.*

6.2. A fractional optimal mechanism for any ratio r

We now present a fractional truthful mechanism for the equal-ratio problem that improves upon the 2-approximation guarantee obtained by the re-

duction in Section 6.1. We describe a cycle-monotone algorithm (with efficiently computable prices) that returns a fractional assignment with makespan at most $OPT(p)$ for every equal-ratio instance p . As mentioned earlier, neither this algorithm nor the earlier 2-approximation algorithm need have the property that if $x_{ij} > 0$, then p_{ij} is bounded with respect to $\text{makespan}(x)$.

We only need to change Step 3 of Algorithm 2. We replace the earlier greedy rule by a more efficient greedy rule for packing the unassigned fraction of jobs, which will allow us to argue that the fractional assignment has makespan at most $OPT(p)$. Algorithm 3 summarizes these steps.

Algorithm 3 Given a vector of processing times p , construct an assignment of jobs to machines as follows. (Observe that in the instance \tilde{p} obtained from p , we have $L = 1$ and $H = r$.)

1. Compute $T^*(p) = \min \{T \geq r, T \text{ integer} : n_{p,T} + (\sum_j L_j - n_{p,T}) \cdot r \leq m \cdot T\}$ in polynomial time via binary search.
2. Compute a prefix-maximal flow for threshold $T^*(p)$ (as in Section 6.1.1).
3. Complete the partial assignment obtained in step 2 by assigning the remaining portions of jobs in a greedy manner. Conceptually, the easiest way to think of this step is to view the unassigned load of job j as being composed of infinitely many units of infinitesimal work, and schedule these units greedily as in step 3 of Algorithm 2.

Consider the jobs in arbitrary order. Let j be a job that is not completely assigned, and $y_j = 1 - \sum_i x_{ij}$ be its unassigned fraction. Compute t_j such that $\sum_{i:l_i < t_j} (t_j - l_i) = H_j y_j$, where l_i is the load on machine i . For every machine i with $l_i < t_j$, we set $x_{ij} \leftarrow x_{ij} + (t_j - l_i)/H_j$. We argue in Lemma 6.3 that $t_j \leq T^*(p)$, so $p_{ij} = H_j$ for every machine i with $l_i < t_j$; thus, the load on every machine i with $l_i < t_j$ increases to t_j (which now becomes the least load).

We repeat this until all jobs are completely assigned. We call each iteration of the above loop, a phase.

6.2.1. Analysis

The analysis follows the same outline as in Sections 5.3 and 5.4. We show in Lemma 6.3 that Algorithm 3 returns an assignment of makespan at most $OPT(p)$, and in Lemma 6.8 that it satisfies cycle monotonicity. The proof of cycle monotonicity dovetails the proof in Section 5.4, but as suggested by the reduction in Section 6.1, whereas earlier, we considered the number of

assigned jobs at various places, the analysis here will be based on the total load/work that is assigned to the machines. As in Section 5.5, this proof will also yield polynomial-time computable prices and hence give a fractional truthful mechanism (Theorem 6.10).

Claims 5.3 and 5.4 continue to hold (with $L = 1$, $H = r$). Therefore, since $OPT(p)$ is an integer (since all the L_j s and H_j s are integers), we get that $T^*(p) \leq OPT(p)$. We now show that the unassigned fractions of jobs are packed in Step 3 within the $T^*(p)$ threshold, showing that the makespan is at most $OPT(p)$.

Lemma 6.3 *Algorithm 3 computes a fractional assignment of makespan at most $OPT(p)$.*

Proof : The makespan at the end of Step 2 is at most $T^*(p)$, which is at most $OPT(p)$. We argue by induction that the makespan remains at most $T^*(p)$ in any phase of Step 3, which will prove the lemma. Consider any phase of Step 3, and let j be the job assigned in this phase. On every machine i with load $l_i < T^*(p)$, we must have $p_{ij} = H_j$, otherwise we could have assigned some portion of j to i and increased the value of the flow computed in Step 2, contradicting the fact that the flow computed is a max-flow. Consider the quantity t_j computed for job j . If $t_j > T^*(p)$, then after j is assigned, the total load is strictly greater than $m \cdot T^*(p)$. But this gives a contradiction since for *any* assignment, the total load is at most $n_{p,T^*(p)} + (\sum_j L_j - n_{p,T^*(p)}) \cdot r \leq m \cdot T^*(p)$. Therefore, $t_j \leq T^*(p)$. Hence, the load on any machine i at the end of this phase is $\max(l_i, t_j) \leq T^*(p)$. ■

We also state the following Corollary that will be useful later.

Corollary 6.4 *Let j be a job assigned (partially) in a phase of Step 3 to a machine i . Then, i is a least-loaded machine at the end of Step 3.*

Proof : At the end of the phase where j is assigned, the load on i must be t_j (since $p_{ij} = H_j$), which is also the least load on any machine. Inductively, due to our packing rule, i continues to be a least-loaded machine in every subsequent phase. ■

We now show that the algorithm satisfies cycle monotonicity. We first simplify the cycle-monotonicity condition (3) slightly as follows: Analogous

to (4) and (5), we now define

$$n_H^{k,\ell} = \sum_{\substack{j:x_{ij}^k > 0, \\ p_{ij}^k = L_j, p_{ij}^\ell = H_j}} L_j x_{ij}^k \quad \text{and} \quad n_L^{k,\ell} = \sum_{\substack{j:x_{ij}^k > 0, \\ p_{ij}^k = H_j, p_{ij}^\ell = L_j}} L_j x_{ij}^k.$$

Notice that these definitions correspond to (4) and (5) respectively, when applied to the instance \tilde{p} obtained from p . With these definitions, (3) translates to

$$\sum_{k=1}^K (n_H^{k+1,k} - n_L^{k+1,k}) \geq 0. \quad (12)$$

Let \vec{L} denote the all- L_j processing-time vector. Define $T_i^L(p_{-i}) = T^*(\vec{L}, p_{-i})$. Since we are focusing on machine i , and p_{-i} is fixed throughout, we abbreviate $T_i^L(p_{-i})$ to T^L . Also, let $p^L = (\vec{L}, p_{-i})$. Clearly $T^*(p) \geq T^L$ for every instance $p = (p_i, p_{-i})$. Applying Lemmas 5.8 and Corollary 5.9 to the instance \tilde{p} and translating back to the instance p gives the following.

Lemma 6.5 *Consider any two instances $p = (p_i, p_{-i})$ and $p' = (p'_i, p_{-i})$ such that $p'_{ij} \geq p_{ij} \forall j$. If T is a threshold such that $n_{p,T} > n_{p',T}$, then every maximum flow x' for (p', T) must completely assign all jobs j such that $p'_{ij} = L_j$.*

Corollary 6.6 *Let $p = (p_i, p_{-i})$ be any instance and let x be any prefix-maximal flow for $(p, T^*(p))$. Then, the low-load on machine i is at most T^L .*

As before, we show (12) by showing that for any $p^1 = (p_i^1, p_{-i})$ and $p^2 = (p_i^2, p_{-i})$, we have $n_{p^1, T^L} \geq n_{p^2, T^L} - n_H^{2,1} + n_L^{2,1}$. The only portion of the proof in Section 5.4 that will change is the second-half of the proof of Lemma 5.11, which is the only place where we use information from Step 3 of the algorithm.

Lemma 6.7 $n_{p^1, T^L} \geq n_{p^2, T^L} - n_H^{2,1} + n_L^{2,1}$.

Proof : Let $T^1 = T^*(p^1)$ and $T^2 = T^*(p^2)$. If $T^1 > T^L$, then the proof follows from Lemma 6.5 as in the proof of Lemma 5.10. So let $T^1 = T^L$. We mimic the proof of Lemma 5.11. Let x^1, x^2 be the complete assignment, i.e., the assignment after both steps 2 and 3, returned by Algorithm 3 for

p^1, p^2 respectively. Let $S = \{j : x_{ij}^2 > 0 \text{ and } p_{ij}^2 = L_j\}$ and $S'' = \{j : x_{ij}^2 > 0 \text{ and } p_{ij}^1 = L_j\}$. Let $T'' = \sum_{j \in S''} L_j x_{ij}^2$. Therefore, $T'' = \sum_{j \in S} L_j x_{ij}^2 - n_H^{2,1} + n_L^{2,1}$ and $\sum_{j \in S} L_j x_{ij}^2 = n_{p^2, T^2}^i = n_{p^2, T^2}^i |_{T^L}$ (by Corollary 6.6). The case where $T'' \leq T^L$ follows exactly as in Lemma 5.11, so we are left with the case $T'' > T^L$.

Since $\sum_{j \in S} L_j x_{ij}^2 \leq T^L$ (by Corollary 6.6), it follows that $n_L^{2,1} > n_H^{2,1} \geq 0$. Let $M = n_{p^2, T^2} - n_H^{2,1} + n_L^{2,1} = T'' + \sum_{i' \neq i} n_{p^2, T^2}^{i'}$. We first show that $m \cdot T'' \leq M + (\sum_j L_j - M) \cdot r$. Clearly, $T'' + (\sum_j L_j x_{ij}^2 - T'') \cdot r \geq T''$. Consider Step 3 of the execution of Algorithm 3 on instance p^2 . Notice that there exists some job j such that $x_{ij}^2 > 0$ and $p_{ij}^2 = H_j$ (because $n_L^{2,1} > 0$), that is, j is assigned partially to i in Step 3 of the algorithm. By Corollary 6.4, i has the least load at the end of Step 3 (and the algorithm). Thus, we get that for every machine $i' \neq i$,

$$n_{p^2, T^2}^{i'} + \left(\sum_j L_j x_{ij}^2 - n_{p^2, T^2}^{i'} \right) \cdot r = \sum_j p_{ij}^2 x_{ij}^2 \geq \sum_j p_{ij}^2 x_{ij}^2 \geq \sum_j L_j x_{ij}^2 \geq T''.$$

Adding these inequalities for all $i' \neq i$, and the earlier inequality for i , we obtain that $M + (\sum_j L_j - M) \cdot r \geq m \cdot T''$.

On the other hand, since $T^1 = T^L$, we have $m \cdot T'' \geq m \cdot T^L \geq n_{p^1, T^L} + (\sum_j L_j - n_{p^1, T^L}) \cdot r$. Combining this with the above upper bound on $m \cdot T''$, we get that $n_{p^1, T^L} > M = n_{p^2, T^2} - n_H^{2,1} + n_L^{2,1} \geq n_{p^2, T^L} - n_H^{2,1} + n_L^{2,1}$. ■

Lemma 6.8 *Algorithm 3 satisfies cycle monotonicity.*

Proof : Applying the above lemma with $p^1 = p^k$ and $p^2 = p^{k+1}$, we get that $n_{p^k, T^L} \geq n_{p^{k+1}, T^L} - n_H^{k+1, k} + n_L^{k+1, k}$. Summing this over all $k = 1, \dots, K$ (where $K + 1 \equiv 1$) yields (12). ■

We now use Lemma 6.7 to derive payments that when combined with Algorithm 3 will yield an individually-rational truthful mechanism. Let \vec{H} denote the all- H_j processing-time vector. Let $n_{p, T}^{-i} = \sum_{i' \neq i} n_{p, T}^{i'}$. The payment P_i to player i given assignment x is defined as:

$$P_i(p) = \sum_j H_j - \sum_{i' \neq i} p_{i'j} x_{i'j} - (r - 1)(n_{p, T^*(p)} - n_{p, T^L} + n_{(\vec{H}, p_{-i}), T^L}). \quad (13)$$

We now verify that these payments do indeed give an individually-rational truthful mechanism. Recall that the utility of a machine i is her payment minus her load.

Lemma 6.9 Fix a player i and the other players' declarations p_{-i} . Let i 's true type be p_i^1 . Then, under the payments defined in (13), i 's utility when she declares her true type p_i^1 is nonnegative, and at least her utility when she declares any other type p_i^2 .

Proof : Let u_i^1, u_i^2 denote i 's utility, and x^1 and x^2 be the assignments returned, when i declares p^1 , and p^2 , respectively (and the others declare p_{-i}). Since p_{-i} is fixed, we omit p_{-i} from the expressions below for notational clarity. Also let p_i^H denote (\vec{H}, p_{-i}) . The utility of i when she declares her true type p_i^1 is

$$\begin{aligned} u_i^1 &= \sum_j H_j - \sum_{i',j} p_{i'j}^1 x_{i'j}^1 - (r-1)(n_{p^1, T^*(p^1)} - n_{p^1, TL} + n_{p_i^H, TL}) \\ &= \sum_j H_j - n_{p^1, T^*(p^1)} - r \left(\sum_j L_j - n_{p^1, T^*(p^1)} \right) \\ &\quad - (r-1)(n_{p^1, T^*(p^1)} - n_{p^1, TL} + n_{p_i^H, TL}) \\ &= (r-1)(n_{p^1, TL} - n_{p_i^H, TL}) \geq 0. \end{aligned}$$

When i declares p_i^2 , her true load is $\sum_j p_{ij}^1 x_{ij}^2 = n_{p^2, T^*(p^2)}^i + (r-1)n_H^{2,1} - (r-1)n_L^{2,1} + r(\sum_j L_j x_{ij}^2 - n_{p^2, T^*(p^2)}^i)$. Therefore, her utility is

$$\begin{aligned} u_i^2 &= \sum_j H_j - n_{p^2, T^*(p^2)} - r \left(\sum_j L_j - n_{p^2, T^*(p^2)} \right) - (r-1)n_H^{2,1} \\ &\quad + (r-1)n_L^{2,1} - (r-1)(n_{p^2, T^*(p^2)} - n_{p^2, TL} + n_{p_i^H, TL}) \end{aligned}$$

which simplifies to $u_i^2 = (r-1)(n_{p^2, TL} - n_H^{2,1} + n_L^{2,1} - n_{p_i^H, TL})$. Thus, by Lemma 6.7, it follows that $u_i^1 \geq u_i^2$. ■

Theorem 6.10 Algorithm 3 combined with the payments specified in (13) gives a polynomial time deterministic fractional truthful mechanism for the equal-ratio job-dependent two-values scheduling domain, that on every instance p returns a (fractional) schedule with makespan at most $OPT(p)$.

Acknowledgments

We thank Elias Koutsoupias for his help in refining the analysis of the lower bound in Section 5.6. We warmly thank the reviewers of the EC'07-version of this paper, the three referees for Games and Economic Behavior,

and the two associate editors, for their various helpful comments and suggestions.

References

- Ahuja, R., Magnanti, T., Orlin, J., 1993. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.
- Andelman, N., Azar, Y., Sorani, M., 2007. Truthful approximation mechanisms for scheduling selfish related machines. *Theory of Computing Systems*, 40(4), 423–436.
- Archer, A., 2004. *Mechanisms for Discrete Optimization with Rational Agents*. PhD thesis, Cornell University.
- Archer, A., Tardos, É., 2001. Truthful mechanisms for one-parameter agents. In *Proc. 42nd FOCS*, pp. 482–491.
- Auletta, V., De-Prisco, R., Penna, P., Persiano, G., 2004. Deterministic truthful approximation mechanisms for scheduling related machines. In *Proc. 21st STACS*, pp. 608–619.
- Bezáková, I., Dani, V., 2005. Allocating indivisible goods. *ACM SIGecom Exchanges*, 5(3), 11 – 18.
- Bikhchandani, S., Chatterjee, S., Lavi, R., Mu’alem, A., Nisan, N., Sen, A., 2006. Weak monotonicity characterizes deterministic dominant-strategy implementation. *Econometrica*, 74, 1109 – 1132.
- Briest, P., Krysta, P., Vöcking, B., 2005. Approximation techniques for utilitarian mechanism design. In *Proc. 37th STOC*, pp. 39 – 48.
- Christodoulou, G., Koutsoupias, E., Kovács, A., 2007. Mechanism design for fractional scheduling on unrelated machines. In *Proc. 34th ICALP*, pp. 40 – 52.
- Christodoulou, G., Koutsoupias, E., Vidali, A., 2007. A lower bound for scheduling mechanisms. In *Proc. 18th SODA*, pp. 1163 – 1170.
- Clarke, E., 1971. Multipart pricing of public goods. *Public Choice*, 8, 17–33.

- Ford, L., Fulkerson, R., 1956. Maximal flow through a network. *Canadian Journal of Mathematics*, 8, 399 – 404.
- Goldberg, A., Tarjan, R., 1990. Solving minimum-cost flow problems by successive approximation. *Mathematics of Operations Research*, 15, 430 – 466.
- Groves, T., 1973. Incentives in teams. *Econometrica*, 41, 617 – 631.
- Gui, H., Muller, R., Vohra, R. 2004. Characterizing dominant strategy mechanisms with multi-dimensional types. Working paper.
- Hall, L., 1996. Approximation algorithms for scheduling. In: Hochbaum, D. (Ed), *Approximation Algorithms for NP-Hard Problems*. PWS Publishing, MA.
- Kleinberg, J., Tardos, É., 2006. *Algorithm Design*. Addison Wesley.
- Koutsoupias, E., Vidali, A., 2007. A $1+\phi$ lower bound for truthful scheduling. In *Proc. 32nd MFCS*, pp. 454 – 464.
- Kovács, A., 2005. Fast monotone 3-approximation algorithm for scheduling related machines. In *Proc. 13th ESA*, pp. 616–627.
- Kumar, V., Marathe, M., Parthasarathy, S., Srinivasan, A., 2005. Approximation algorithms for scheduling on multiple machines. In *Proc. 46th FOCS*, pp. 254 – 263.
- Lavi, R., Mu’alem, A., Nisan, N., 2003. Towards a characterization of truthful combinatorial auctions. In *Proc. 44th FOCS*, pp. 574–583.
- Lavi, R., Swamy, C., 2005. Truthful and near-optimal mechanism design via linear programming. In *Proc. 46th FOCS*, pp. 595–604.
- Lehmann, D., O’Callaghan, L., Shoham, Y., 2002. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM*, 49, 577 – 602.
- Lenstra, J., Shmoys, D., Tardos, É., 1990. Approximation algorithms for scheduling unrelated parallel machines. *Math. Prog.*, 46, 259–271.

- Lipton, R., Markakis, E., Mossel, E., Saberi, A., 2004. On approximately fair allocations of indivisible goods. In *Proc. 5th EC*, pp. 125–131.
- McAfee, R., McMillan, J., 1988. Multidimensional incentive compatibility and mechanism design. *Journal of Economic Theory*, 46, 335 – 354.
- Mu’alem, A., Schapira, M., 2007. Setting lower bounds on truthfulness. In *Proc. 18th SODA*, pp. 1143 – 1152.
- Myerson, R., 1981. Optimal auction design. *Mathematics of Operations Research*, 6, 58 – 73.
- Nisan, N., Ronen, A., 2001. Algorithmic mechanism design. *Games and Econ. Behavior*, 35, 166 – 196.
- Rochet, J., 1987. A necessary and sufficient condition for rationalizability in a quasilinear context. *Journal of Mathematical Economics*, 16, 191–200.
- Rockafellar, R., 1972. *Convex Analysis*. Princeton University Press, NJ.
- Saks, M., Yu, L., 2005. Weak monotonicity suffices for truthfulness on convex domains. In *Proc. 6th EC*, pp. 286 – 293.
- Schrijver, A., 1998. *Theory of Linear and Integer Programming*. John Wiley & sons.
- Shmoys, D., Tardos, É., 1993. An approximation algorithm for the generalized assignment problem. *Mathematical Programming*, 62, 461–474.
- Vickrey, W., 1961. Counterspeculations, auctions, and competitive sealed tenders. *Journal of Finance*, 16, 8–37.

A. Proofs from Section 4

Proof of Lemma 4.1 : The proof is borrowed from Lavi and Swamy (2005), where a somewhat more general statement was proved.

We are given that $X(p)$ is the random assignment $\mathcal{R}(x(p))$ that satisfies $E[X(p)_{ij}] = x(p)_{ij}$ for all i, j . (Recall that $M = (x, P)$ is the fractional truthful mechanism, so $x(p)$ is the assignment, and $P_i(p)$ is the payment made to player i on input p .) We need to define the (possibly random)

payments $P'_i(p)$ to each player i such that $M' = (X, P')$ is a truthful-in-expectation mechanism. For an input p , machine i and assignment x , define $p_i(x) = \sum_j p_{ij}x_{ij}$. Observe that for any inputs p^1 and p^2 , we have

$$\mathbb{E}[p_i^1(X(p^2))] = p_i^1(\mathbb{E}[X(p^2)]) = p_i^1(x(p^2)),$$

where the first equality follows from the linearity of $p_i^1(\cdot)$, and the second since $\mathbb{E}[X(p^2)_{ij}] = x(p^2)_{ij}$ for every i, j . Thus, if we set the payments $P'_i(\cdot)$ so that $\mathbb{E}[P'_i(p)] = P_i(p)$ for every input p , the expected utility of player i when her true input is p^1 and reported input is p^2 will be *exactly* the utility she receives under the mechanism M when her true input is p^1 and reported input is p^2 . It follows that since M is truthful, M' is truthful in expectation.

The condition $\mathbb{E}[P'_i(p)] = P_i(p)$ leaves plenty of flexibility in devising the payments (e.g., one could simply set $P'_i(p) = P_i(p)$). We want a payment-scheme that “transfers” individual rationality from M to M' . We do so by devising the following payments. Let $\{y^{(1)}(p), \dots, y^{(k)}(p)\}$ be the support of the random variable $X(p)$, that is, the $y^{(\ell)}(p)$'s are all possible values (i.e., integer assignments) that the random variable $X(p)$ takes. We set $P'_i(p)$ to be the following random variable:

$$P'_i(p) = \begin{cases} p_i(y^{(\ell)}(p)) \cdot \frac{P_i(p)}{p_i(x(p))} & \text{if } p_i(x(p)) > 0 \text{ and } X(p) = y^{(\ell)}(p); \\ P_i(p) & \text{if } p_i(x(p)) = 0. \end{cases}$$

It is clear that $\mathbb{E}[P'_i(p)] = P_i(p)$. If M is individually rational, we have that $P_i(p) \geq p_i(x(p))$ for every input p . This implies that $P'_i(p) \geq p_i(X(p))$ under every coin toss, so M' is individually rational for every coin toss. ■

Proof of Lemma 4.2 : This lemma follows from a theorem (Theorem 11) proved in Kumar et al. (2005), and is implicit in the work of Shmoys and Tardos (1993). The rounding procedure we describe is simply a randomized version of the rounding algorithm in Shmoys and Tardos (1993). Our exposition very closely follows the exposition in Shmoys and Tardos (1993); Hall (1996) (see Section 1.4.2 in Hall (1996)).

The idea behind the rounding procedure is as follows. Let $n_i = \lceil \sum_j x_{ij} \rceil$. Given the fractional assignment x , we will create a bipartite graph $G = (V, W, E)$. W is a set of job nodes containing a node w_j for each job j , and V is a set of machine nodes; for each machine i , we create n_i nodes $v_{i,c}$, where $c = 1, \dots, n_i$. We sketch the way in which the edges are created; more

details may be found in Shmoys and Tardos (1993); Hall (1996). For each machine i , we order the jobs with $x_{ij} > 0$ in decreasing order of their p_{ij} s. We first consider “copy” $v_{i,1}$. Considering jobs in this order, we create an edge $(v_{i,1}, w_j)$ for each job j until the total x_{ij} -weight of these jobs becomes at least 1. We then move on to the next copy $v_{i,2}$, consider the remaining jobs (again in order), and create an edge $(v_{i,2}, w_j)$ for each such job until the total x_{ij} -weight of these jobs exceeds 1, and so on and so forth. One point of detail here is the following: it may happen that for a copy $v_{i,c}$, the total x_{ij} -weight of the jobs with edges to $v_{i,c}$ exceeds 1. If this happens, then for the last job k (in the ordering) with an edge $(v_{i,c}, w_k)$, when we move on to the next copy $v_{i,c+1}$ we still consider k as a “remaining” job, but with an x_{ik} -weight equal to (the original) x_{ik} decreased by the amount required to make the total x -weight for copy $v_{i,c}$ *exactly* 1; thus job k is “split” between copies $v_{i,c}$ and $v_{i,c+1}$, and will be the first job considered for $v_{i,c+1}$. We summarize some properties of this bipartite graph.

- (a) G contains an edge $(v_{i,c}, w_j)$ for some copy $v_{i,c}$ iff $x_{ij} > 0$. For every i, j with $x_{ij} > 0$, the edges incident to w_j comprise either (i) a single edge of the type $(v_{i,c}, w_j)$; or (ii) two edges of the type $(v_{i,c}, w_j)$, $(v_{i,c+1}, w_j)$.
- (b) For each machine i and edges $(v_{i,c}, w_j)$, $(v_{i,c'}, w_k)$, where $c < c'$, we have $p_{ij} \geq p_{ik}$.
- (c) Any matching in G that matches all the job nodes corresponds to an integer assignment, where a job j is assigned to machine i iff there is an edge of type $(v_{i,c}, w_j)$ in the matching. Moreover, this assignment satisfies property 2, that is, the load on each machine i is at most $\sum_j x_{ij} p_{ij} + \max_{\{j: x_{ij} > 0\}} p_{ij}$.
- (d) The assignment x translates to a *fractional matching* in this graph that completely matches all the job nodes and all nodes $v_{i,c}$, $c = 1, \dots, n_i - 1$. The total weight assigned to edges of the type $(v_{i,c}, w_j)$ by this fractional matching is exactly x_{ij} .

The details about the bipartite-graph construction may be found in Hall (1996). Notice that property (c) follows from property (b): let M^F be the fractional matching obtained from x , which assigns weight $M_{v_{i,c}, w_j}^F$ to edge $(v_{i,c}, w_j)$. If y is the assignment corresponding to an integer matching M

that matches all job nodes, then the load on machine i is at most

$$\begin{aligned} \max_{\{j:x_{ij}>0\}} p_{ij} + \sum_{c=2}^{n_i} \sum_{j:(v_{i,c},w_j)\in M} p_{ij} &\leq \max_{\{j:x_{ij}>0\}} p_{ij} + \sum_{c=2}^{n_i} \sum_{j:(v_{i,c-1},w_j)\in E} M_{v_{i,c},w_j}^F p_{ij} \\ &\leq \max_{\{j:x_{ij}>0\}} p_{ij} + \sum_{j:x_{ij}>0} x_{ij} p_{ij}. \end{aligned}$$

It is well known that a fractional matching in a bipartite graph can be expressed as a convex combination of integer matchings; for example, this follows since the matching polytope for bipartite graphs has integer extreme points (see e.g., Schrijver (1998)). Moreover, such a convex combination can be found efficiently (for example, by repeated max-flow computations). We can thus decompose (the matching yielded by x into a convex combination of integer matchings. Notice that each such matching must match all the job nodes (since the matching yielded by x completely matches all job nodes), and thus corresponds to an integer assignment satisfying property 2. This convex combination may be equivalently viewed as a probability distribution over integer assignments, which yields the desired randomized rounding procedure. \blacksquare