Theory Comput Syst (2014) 54:388–406
DOI 10.1007/s00224-013-9449-0

A Truthful Mechanism for Value-Based Scheduling in Cloud Computing

Navendu Jain · Ishai Menache · Joseph (Seffi) Naor · Jonathan Yaniv

Published online: 8 February 2013

Abstract We introduce a novel pricing and resource allocation approach for batch jobs on cloud systems. In our economic model, users submit jobs with a value function that specifies willingness to pay as a function of job due dates. The cloud provider in response allocates a subset of these jobs, taking into advantage the *flexibility* of allocating resources to jobs in the cloud environment. Focusing on social-welfare as the system objective (especially relevant for private or in-house clouds), we construct a resource allocation algorithm which provides a small approximation factor that approaches 2 as the number of servers increases. An appealing property of our scheme is that jobs are allocated non-preemptively, i.e., jobs run in one shot without interruption. This property has practical significance, as it avoids significant network and storage resources for checkpointing. Based on this algorithm, we then design an *efficient* truthful-in-expectation mechanism, which significantly improves the running

A preliminary version of this work appeared in the proceedings of the 4th Symposium on Algorithmic Game Theory (SAGT 2011).
Part of this work was done while visiting Microsoft Research. This research was supported in part by the Google Inter-university center for Electronic Markets and Auctions. Also supported by ISF grants 1366/07 and 954/11.

N. Jain · I. Menache
Extreme Computing Group, Microsoft Research, Redmond, WA, USA

N. Jain
e-mail: navendu@microsoft.com

I. Menache
e-mail: ishai@microsoft.com

J.(S.) Naor · J. Yaniv (✉)
Computer Science Department, Technion, Haifa, Israel
e-mail: jyaniv@cs.technion.ac.il

J.(S.) Naor
e-mail: naor@cs.technion.ac.il

complexity of black-box reduction mechanisms that can be applied to the problem, thereby facilitating its implementation in real systems.

Keywords Mechanism design · Approximation algorithms · Cloud computing · Resource allocation · Scheduling algorithms

1 Introduction

Cloud computing offers easily accessible computing resources of variable size and capabilities. This paradigm allows applications to rent computing resources and services on-demand, benefiting from the allocation flexibility and the economy of scale of large data centers.

Cloud computing providers, such as Amazon, Google and Microsoft, are offering cloud hosting of user applications under a utility pricing model. The most common purchasing options are pay-as-you-go (or *on-demand*) schemes, in which users pay per-unit resource (e.g., a virtual machine) per-unit time (e.g., per hour). The advantage of this pricing approach is in its simplicity, in the sense that users pay for the resources they get. However, such an approach suffers from two shortcomings. First, the user pays for computation as if it were a tangible commodity, rather than paying for desired performance. To exemplify this point, consider a finance firm which has to process the daily stock exchange data with a deadline of an hour before the next trading day. Such a firm does not care about allocation of servers over time as long as the job is finished by its due date. At the same time, the cloud can deliver higher value to users by knowing the user-centric valuation for the limited resources being contended for. This form of value-based scheduling, however, is not supported by pay-as-you-go pricing. Second, current pricing schemes lack feedback signals that prevent users from submitting unbounded amounts of work. Thus, users are not incentivized to respond to variations in resource demand and supply.

In this work, we propose a novel pricing model for cloud environments, which focuses on *quality* rather than only *quantity*. Specifically, we incorporate the significance of the completion time of a job, rather than the exact number of servers that the job gets at any given time. In our economic model, customers specify the overall amount of resources (server or virtual machine hours) which they require for their job, and how much they are willing to pay for these resources as a function of due date. For example, a particular customer may specify that she needs a total of 1000 server hours, and is willing to pay \$100 if she gets them by 5pm and \$200 if she gets them by 2pm. This framework is especially relevant for batch jobs (e.g., financial analytics, image processing, search index updates) that are carried out until completion. Under our scheme, the cloud determines the *scheduling* of the resources according to given submitted jobs, the users' willingness to pay and its own capacity constraints. This entire approach raises fundamental mechanism design and *incentive compatibility* issues, as users may try to game the system by reporting false value, potentially increasing their utility. Hence, any algorithmic solution has to self enforce the users to report their true values (or willingness to pay) for the different job due dates.

Pricing in shared computing systems such as cloud computing can have diverse objectives, such as maximizing profits, or optimizing system-related metrics (e.g.,

delay or throughput). Our focus in this work is on maximizing the social welfare, i.e., the sum of users' values. This objective is especially relevant for private or in-house clouds, such as a government cloud, or enterprise computing clusters.

1.1 Our Results

We design an efficient truthful-in-expectation mechanism for a new scheduling problem, called the Bounded Flexible Scheduling (BFS) problem, which is directly motivated by the cloud computing paradigm. A cloud containing C servers receives a set of job requests with heterogeneous demand and values per deadline (or due date), where the objective is maximizing the social welfare, i.e., the sum of the values of the scheduled jobs. The scheduling of a job is *flexible*, i.e., it can be allocated a different number of servers per time unit and in a possibly preemptive (non-contiguous) manner, under parallelism thresholds. The parallelism threshold represents the job's limitations on parallelized execution. For every job j, we denote by k_j the maximum number of servers that can be allocated to job j in any given time unit. The maximal parallelism thresholds across jobs, denoted by k, is assumed to be much smaller than the cloud capacity C, as typical in practical settings.

No approximation algorithm is known for the BFS problem. When relaxing the parallelism bounds on jobs, our model coincides with the problem of maximizing the profit of preemptively scheduling jobs on a single server. Lawler [13] gives an optimal solution in pseudo-polynomial time via dynamic programming to this problem, implying an FPTAS for it. However, his algorithm cannot be extended to the case where parallelization is bounded.

Our first result is an LP-based approximation algorithm for BFS that gives an approximation factor of $\alpha \triangleq (1 + \frac{C}{C-k})(1+\varepsilon)$ to the optimal social welfare for every $\varepsilon > 0$. With the gap between k and C being large, the approximation factor approaches 2. The running time of the algorithm, apart from solving the linear program, is polynomial in the number of jobs, the number of time slots and $\frac{1}{\varepsilon}$. The design of the algorithm proceeds through several steps. We first consider the natural LP formulation for the BFS problem. Unfortunately, this LP has a very large integrality gap. Thus, we strengthen this LP by incorporating additional constraints that decrease the integrality gap. We proceed by defining a reallocation algorithm that converts any solution of the LP to a value-equivalent canonical form named MND, in which the number of servers allocated per job does not decrease over the execution period of the job. Our approximation algorithm then decomposes the optimal solution, given in MND form, to a relatively small number of feasible BFS solutions, with their average social welfare being an α-approximation (thus, at least one of them is an α-approximation). An appealing property of our scheme is that jobs are allocated non-preemptively, i.e., jobs run in one shot without interruption. This property has practical significance, as it avoids significant network and storage resources for checkpointing the intermediate state of jobs that are distributed across multiple servers running in parallel.

The approximation algorithm that we develop plays a key role in our construction of an efficient truthful-in-expectation mechanism preserving the α-approximation. To obtain this result, we slightly modify the approximation algorithm to get an exact decomposition of an optimal fractional solution. This decomposition is then used to

simulate (in expectation) a "fractional" VCG mechanism, which we prove to be truthful. The main advantage of our mechanism is that the allocation rule requires only a *single* execution of the approximation algorithm, whereas known black-box reductions that can be applied invoke the approximation algorithm many times, while providing only a polynomial bound on the number of invocations. At the end of Sect. 4, we discuss the process of computing of the charged payments.

1.2 Related Work

We compare our results to known work in algorithmic mechanism design and scheduling. An extensive amount of work has been carried out in these fields, starting with the seminal paper of Nisan and Ronen [14] (see also [15] for a survey book). Of relevance to our work are papers which introduce *black-box* schemes of turning approximation algorithms to incentive compatible mechanisms, while maintaining the approximation ratio of the algorithm. Specifically, Lavi and Swamy [11] show how to construct a truthful-in-expectation mechanism for packing problems that are solved through LP-based approximation algorithms. Dughmi and Roughgarden [8] prove that packing problems that have an FPTAS solution can be turned into a truthful-in-expectation mechanism which is also an FPTAS. Recently, Chawla et al. [7] disproved the existance of a general black-box reduction even when only requiring truthfulness in expectation. Specifically, any such reduction cannot preserve the worst-case approximation ratio of the algorithm within less than a superpolynomial factor. On the other hand, existence of such general reductions have been found for a weaker setting of Bayes-Nash equilibria [5, 9, 10]. We note that there are several papers that combine scheduling and mechanism design (e.g., [1, 12]), mostly focusing on makespan minimization.

Scheduling has been a perpetual field of research in operations research and computer science (see e.g., [3, 4, 6, 13, 16] and the references therein). Of specific relevance to our work are [4, 16], which consider variations of the interval-scheduling problem. These papers utilize a decomposition technique for their solutions, which we extend to a more complex model in which the amount of resources allocated to a job can change over time.

2 Definitions and Notation

In the Bounded Flexible Scheduling **(BFS)** problem, a cloud provider is in charge of a cloud containing a fixed number of C servers. The cloud provider receives requests from n clients, denoted by $\mathcal{J} = \{1, 2, \ldots, n\}$, where each client has a job that needs to be executed. We will often refer to a client either as a player or by the job belonging to her. The cloud provider can choose to reject some of the job requests, for instance if allocating other jobs increases its gain. In this model, the cloud can gain only by fully completing a job. That is, partially completing a job is considered as if the job hasn't been processed and thus the cloud provider does not gain from it. We assume that the time axis is divided into T discrete time slots $\mathcal{T} = \{1, 2, \ldots, T\}$.

Every job j is described by a tuple $\langle D_j, k_j, v_j \rangle$. The first parameter D_j, the *demand* of job j, is the total amount of demand units required to complete the job,

where a demand unit corresponds to a single server being assigned to the job for a single time slot. Parallel execution of a job is allowed, that is, the job can be executed on several servers in parallel. In this model we assume that the additional overhead due to parallelism is negligible. However, parallel execution of a job is limited by a threshold k_j, which is the maximal number of servers that can be simultaneously assigned to job j in a single time slot. As C typically tends to hundreds of thousand of servers, we assume that $k \triangleq \max_j\{k_j\}$ is substantially smaller than the total capacity C, i.e., $k \ll C$, as common in practical settings.

Let $v_j : \mathcal{T} \to \mathbb{R}^{+,0}$ be the *valuation function* of job j. That is, $v_j(t)$ is the value gained by the owner of job j if job j is completed at time t. The goal is to maximize the sum of values of the jobs that are scheduled by the cloud. Naturally, the valuation function $v_j(t)$ can be assumed to be monotonically non-increasing in t. Nevertheless, we will later discuss the case where this assumption is relaxed and v_j is any general function. Three families of valuation functions will be of specific interest to us:

- **Deadline Valuation Functions:** Here, players are interested only in their job being completed until a particular deadline (due date). Formally, $v_j(t)$ is a step function, which is equal to a constant scalar v_j until the deadline d_j, and 0 afterwards.
- **General Monotone Valuation Functions:** The functions $v_j(t)$ are arbitrary monotonically non-increasing functions.
- **General Valuation Functions:** The functions $v_j(t)$ are arbitrary functions.

For simplicity of notation, when discussing the case of general (monotone) valuation functions, we will set $d_j = T$ for every player. Define $\mathcal{T}_j = \{t \in \mathcal{T} : t \leq d_j\}$ as the set of time slots in which job j can be executed and $\mathcal{J}_t = \{j \in \mathcal{J} : t \leq d_j\}$ as the set of jobs that can be executed at time t.

A *mapping* $y_j : \mathcal{T}_j \to [0, k_j]$ is an assignment of servers to job j per time unit, which does not violate the parallelism threshold k_j.[1] A mapping which fully executes job j is called an *allocation*. Formally, an allocation $a_j : \mathcal{T}_j \to [0, k_j]$ is a mapping for job j with $\sum_t a_j(t) = D_j$. Denote by $\mathcal{A}_j$ the set of allocations a_j which fully execute job j and let $\mathcal{A} = \bigcup_{j=1}^{n} \mathcal{A}_j$. Let $s(y_j) = \min\{t : y_j(t) > 0\}$ and $e(y_j) = \max\{t : y_j(t) > 0\}$ denote the start and end times of a mapping y_j, respectively. Specifically, for an allocation a_j, $e(a_j)$ is the time in which job j is completed when the job is allocated according to a_j, and $v_j(e(a_j))$ is the value gained by the owner of job j. We will often use $v_j(a_j)$ instead of $v_j(e(a_j))$ to shorten notations.

3 Approximation Algorithm for BFS

In this section we present an algorithm for BFS that approximates the social welfare, i.e., the sum of values gained by the players. When discussing the approximation algorithm, we assume that players bid truthfully. In Sect. 4, we describe a payment

[1] For tractability, we assume that the assignment y_j is a continuous decision variable. In practice, non-integer allocations will have to be translated to integer ones, for example by processor sharing within each time interval. The precise ways of how to implement such techniques are beyond the scope of the present paper.

scheme that gives players an incentive to bid truthfully. We begin this section by describing an LP relaxation for the case of deadline valuation functions and continue by presenting a canonical solution form in which all mappings are Monotonically Non Decreasing (MND) mappings, defined later. This result is then generalized for general valuation functions (Sect. 3.3). We also show that the case of general valuation functions can be reduced to the case of general monotone valuation functions (Sect. 3.4). Finally, we give a decomposition algorithm (Sect. 3.5) which yields an α-approximation to the optimal social welfare of BFS.

3.1 LP Relaxation of BFS with Deadline Valuation Functions

Consider the following IP. Every variable $y_j(t)$ for $t \in \mathcal{T}_j$ in (IP) denotes the number of servers assigned to j at time t. We use y_j to denote the mapping induced by the variables $\{y_j(t)\}_{t\in\mathcal{T}_j}$ and x_j as a binary variable indicating whether job j has been fully allocated or not at all.

$$\begin{aligned}
\text{(IP)} \quad \max \quad & \sum_{j=1}^{n} v_j x_j \\
\text{s.t.} \quad & \sum_{t\in\mathcal{T}_j} y_j(t) = D_j \cdot x_j \quad \forall j \in \mathcal{J} && (1)\\
& \sum_{j\in\mathcal{J}_t} y_j(t) \le C \quad \forall t \in \mathcal{T} && (2)\\
& 0 \le y_j(t) \le k_j \quad \forall j \in \mathcal{J}, t \in \mathcal{T}_j && (3)\\
& x_j \in \{0,1\} \quad \forall j \in \mathcal{J} && (4)
\end{aligned}$$

Constraints (1), (2) and (3) are job demand, capacity and parallelization constraints.

We first relax the constraints $x_j \in \{0,1\}$ to:

$$0 \le x_j \le 1 \quad \forall j \in \mathcal{J} \tag{5}$$

The integrality gap of the resulting linear program can be as high as $\Omega(n)$. To see this, consider the following instance: There are nC jobs with $k_j = 1$ for every job j which are divided into n sets $S_0, \ldots, S_{n-1}$, each of size C. A job $j \in S_i$ requests $D_j = 2^i$ demand units which need to be completed before time 2^i. Formally, $v_j = 1$ and $d_j = 2^i$. An optimal integral solution can gain at most C, since any completed job must receive a demand unit at time $t = 1$. Yet, the optimal fractional solution gains $C + \frac{C\cdot(n-1)}{2}$ by mapping the jobs as follows (see Fig. 1): For jobs $j \in S_0$ we set $y_j(1) = 1$, thus completing them fully. For jobs $j \in S_i$ with $i \ge 1$, we set $y_j(t) = 1$ for $t \in (2^{i-1}, 2^i]$ and $y_j(t) = 0$ otherwise, and by that completing half of job j before $t = 2^i$.

Thus, we add the following set of constraints to the linear program:

$$y_j(t) \le k_j x_j \quad \forall j \in \mathcal{J}, t \in \mathcal{T}_j \tag{6}$$

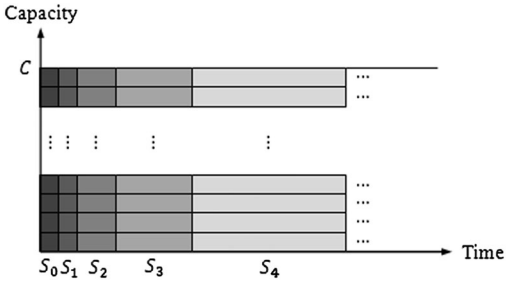

Fig. 1 A fractional solution verifying the integrality gap of the natural relaxation of (IP)

Notice that every feasible solution to BFS satisfies the new set of constraints (6). Hence, the linear program remains a relaxed formulation of BFS, even after the new set of constraints (6). We describe the motivation behind these constraints. Recall that unless a job is fully completed, it is does not contribute any value to the social welfare of a scheduling algorithm for BFS. On the other hand, in order to find an optimal fractional solution, we solve a relaxed scheduling problem, in which partial execution of jobs is permitted and profitable. For a scheduling algorithm to utilize the obtained optimal fractional solution, the algorithm must somehow derive feasible allocations, in which jobs are fully completed, from the optimal fractional solution. Consider some mapping y_j of job j that only provides a fraction $x_j < 1$ of the full resource demand D_j of the job. A natural way of transforming y_j into a feasible allocation would be to multiply each of its entries by $1/x_j$. While this process guarantees that the job is fully completed, we might violate the parallelism constraint of job j. Intuitively, the gap decreasing constraints (6) prevent us from getting bad mappings. For instance, consider the example which realizes the lower bound of $\Omega(n)$ on the integrality gap of the first relaxed formulation. The mappings of jobs $j \in S_i$ for $i \geq 1$ violate constraints of type (6), since for $t \in (2^{i-1}, 2^i]$ we have $y_j(t) = 1$ yet $x_j = \frac{1}{2}$.

Note 1 Henceforth, we refer to **(LP-D)** as the relaxed linear program of **(IP)**, in which (4) is replaced with (5) and (6) is added.

The integrality gap of (LP-D) is at least $\frac{C}{C-k}$. To see this, consider a BFS instance with n jobs, $T = 1$ and $C = kn - \delta$ for some k and some small $\delta > 0$. For every job j set $D_j = k_j = k$, $v_j = 1$ and d_j=1. Any BFS solution can schedule at most $n - 1$ jobs. Yet, an optimal fractional solution to (LP-D) can fully allocate $n - 1$ jobs and fractionally allocate a $(1 - \frac{\delta}{k})$-fraction of the last job, gaining $n - \frac{\delta}{k}$. Thus, the integrality gap is bounded below by:

$$\frac{n - \frac{\delta}{k}}{n - 1} = \frac{kn - \delta}{kn - k} = \frac{C}{C - k + \delta} \tag{7}$$

This bound approaches $\frac{C}{C-k}$ as δ decreases.

In the appendix we present further insights on the choice of (6), by considering an alternative formulation of BFS as a *configuration LP* and showing its equivalence to (LP-D). In fact, (LP-D) can be viewed as an efficient way of implementing the

Reallocate(y)
1. While y contains non-MND mappings
   1.1. Let j be a job generating a maximal(a, b)-violation according to $\preceq$.
   1.2. **ReallocationStep**(y, j, a, b)

ReallocationStep(y, j, a, b)
1. Let j' be a job such that $y_{j'}(a) < y_{j'}(b)$
2. $\mathcal{T}_{\max} = \{t \in [a, b] : y_{j'}(t) = y_{j'}(b)\}$
3. $\delta = \max\{y_{j'}(t) : t \in [a, b] \setminus \mathcal{T}_{\max}\}$
4. $\Delta = \min\{\frac{y_j(a) - y_j(b)}{1+|\mathcal{T}_{\max}|}, \frac{y_{j'}(b) - y_{j'}(a)}{1+|\mathcal{T}_{\max}|}, y_{j'}(b) - \delta\}$
5. Reallocate as follows:
   5.1. $y_{j'}(t) \leftarrow y_{j'}(t) - \Delta$ for every $t \in \mathcal{T}_{\max}$
   5.2. $y_{j'}(a) \leftarrow y_{j'}(a) + \Delta \cdot |T_{\max}|$
   5.3. $y_j(a) \leftarrow y_j(a) - \Delta \cdot |T_{\max}|$
   5.4. $y_j(t) \leftarrow y_j(t) + \Delta$ for every $t \in \mathcal{T}_{\max}$

configuration LP. The remainder of this section is dedicated to constructing an approximation algorithm which gives an upper bound to the integrality gap of:

$$\inf\left\{\left(1 + \frac{C}{C-k}\right)(1+\varepsilon) : \varepsilon > 0\right\} = 1 + \frac{C}{C-k} \tag{8}$$

3.2 The Canonical MND Form

We now present a canonical solution form of solutions for (LP-D), in which all of the mappings are monotonically non decreasing (defined next). This canonical form will allow us to construct an approximation algorithm for BFS with a good approximation factor.

Definition 1 A *monotonically non-decreasing (MND)* mapping (allocation) $y_j : \mathcal{T}_j \to [0, k_j]$ is a mapping (allocation) which is monotonically non-decreasing in the interval $[s(y_j), e(y_j)]$.

MND mappings propose implementational advantages, such as the allocation algorithm being non-preemptive, as well as theoretical advantages which will allow us to construct a good approximation algorithm for BFS. We first present the main result of this subsection:

Theorem 1 *There is a poly(n, T) time algorithm that transforms any feasible solution y of (LP-D) to an equivalent solution that obtains the same social welfare as y, in which all mappings are MND mappings.*

This theorem is a result of the following *reallocation algorithm*. Let y be a feasible solution to (LP-D). To simplify arguments, we add additional "idle" jobs of unit demand and no value, which are allocated whenever there are free servers. This allows us to assume without loss of generality that in every time slot, all C servers

are in use. We present a reallocation algorithm that transforms the mappings in y to MND mappings. The reallocation algorithm will swap between assignments of jobs to servers, without changing the completed fraction of every job (x_j), such that no completion time of a job will be delayed. Since the valuation functions are deadline valuation functions, the social welfare of the resulting solution will be equal to the social welfare matching y. Specifically, an optimal solution to (LP-D) will remain optimal.

We introduce some definitions and notations prior to the description of the reallocation algorithm. Denote by $A_y(t) = \{j : y_j(t) > 0\}$ the set of jobs active at time t in y.

Definition 2 A job $j \in A_y(b)$ *generates an* (a, b)*-violation* if $a < b$ and $y_j(a) > y_j(b)$. Violations are weakly ordered according to a binary relation $\preceq$ over $\mathcal{T} \times \mathcal{T}$, such that:

$$(a, b) \preceq (a', b') \quad \Leftrightarrow \quad b < b' \quad \text{or} \quad (b = b') \wedge (a \leq a') \tag{9}$$

Note that there can be several maximal pairs (a, b) according to $\preceq$.

Given a solution y to (LP-D), our goal is to eliminate all (a, b)-violations in y and consequently remain with only MND mappings, keeping y a feasible solution to (LP-D). The reallocation algorithm works as follows: In every step we try to eliminate one of the maximal (a, b)-violations, according to the order induced by $\preceq$. Let j be the job generating this maximal (a, b)-violation. The main observation is that there must be some job j' with $y_{j'}(a) < y_{j'}(b)$, since in every time slot all C servers are in use. We apply a *reallocation step*, which tries to eliminate this violation by shifting workload of job j from a to later time slots (b in particular), and by doing the opposite to j'. To be precise, we increase y_j in time slots in $\mathcal{T}_{\max}$ (line 2) by a value $\Delta > 0$ (line 4), and increase $y_{j'}(a)$ by the amount we decreased from other variables. We note that if we do not decrease $y_{j'}$ for all time slots in $\mathcal{T}_{\max}$, we will generate $(\tilde{a}, b)$-violations for $a < \tilde{a}$ and therefore the reallocation algorithm may not stop. See Fig. 2 for an example of a reallocation step.

We choose Δ such that after calling the reallocation step either: (1) $y_j(a) = y_j(b)$. (2) $y_{j'}(a) = y_{j'}(b)$. (3) The size of $\mathcal{T}_{\max}$ increases. In the second case, if the (a, b)-violation hasn't been resolved, there must be a different job $j'' \in A_y(b)$ with $y_{j''}(a) < y_{j''}(b)$, and therefore we can call the reallocation step again. In the third case, we simply expand $\mathcal{T}_{\max}$ and recalculate Δ. The reallocation algorithm repeatedly applies the reallocation step, choosing the maximal (a, b)-violation under $\preceq$, until all mappings become MND mappings. The following claim guarantees the correctness of the reallocation algorithm.

Claim *Let y be a feasible solution of* (LP-D) *and let j be a job generating a maximal (a, b)-violation over $\preceq$. Denote by $\tilde{y}$ the vector y after calling ReallocationStep(y, j, a, b) and let $(\tilde{a}, \tilde{b})$ be the maximal violation in $\tilde{y}$ over $\preceq$. Then*:

1. *$\tilde{y}$ is a feasible solution of* (LP-D).
2. *$(\tilde{a}, \tilde{b}) \preceq (a, b)$.*
3. *No new (a, b)-violations are added to $\tilde{y}$.*

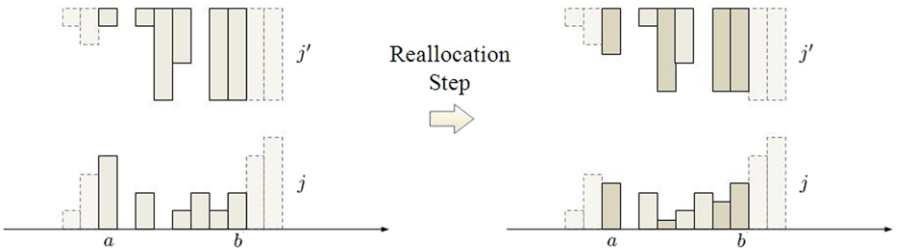

Fig. 2 Resolving an (a, b)-violation generated by j

Proof Denote by $y_j, \tilde{y}_j$ the mappings of j before and after the reallocation step, and denote $y_{j'}, \tilde{y}_{j'}$ similarly. By the choice of (a, b), for every $t \in (a, b]$ there is no (t, b)-violation and thus:

$$\forall t \in (a, b] \quad y_j(t) \leq y_j(b) \tag{10}$$

$$\forall t \in (a, b] \quad y_{j'}(t) \leq y_{j'}(b) \tag{11}$$

By the construction of the reallocation step, since we reduce $y_{j'}$ by Δ for every time slot in $\mathcal{T}_{\max}$ and halt when one of the strong inequalities reach equality or when the size of $\mathcal{T}_{\max}$ increases, we have:

$$\forall t \in [a, b] \quad \tilde{y}_j(t) \leq \tilde{y}_j(a) \tag{12}$$

$$\forall t \in [a, b] \quad \tilde{y}_{j'}(t) \leq \tilde{y}_{j'}(b) \tag{13}$$

since by the choice of Δ, for every $t \in \mathcal{T}_{\max}$ we have $y_j(t) + \Delta \leq y_j(a) - \Delta|\mathcal{T}_{\max}|$ (and similarly for (13)). Notice that no constraint of (LP-D) is violated by $\tilde{y}$: the reallocation algorithm keeps $x_j, x_{j'}$ fixed, therefore no demand constraint (1) is violated; the allocated workload at every time unit remains the same, and the reallocation algorithm only decreases $y_j(a)$ and $y_{j'}(b)$. This proves 1, since the initial solution y is feasible.

By the maximality of (a, b) and since $y_j(a)$, $y_{j'}(b)$ are upper bounds on the entries of $\tilde{y}_j, \tilde{y}_{j'}$ in $[a, b]$, we couldn't have created any $(\tilde{a}, \tilde{b})$-violation for $b < \tilde{b}$. Consider the second type of violations which might have been caused, i.e., an $(\tilde{a}, b)$-violation for $\tilde{a} > a$. By (13), j' does not generate such a violation. Since $b \in \mathcal{T}_{\max}$ and by (10), the same goes for j, proving 2 and 3. □

The reallocation algorithm converges in $poly(n, T)$ time. In each iteration, we resolve the maximal (a, b)-violation after at most nT calls to the reallocation step. By the claim proved in this section, the maximal violation strictly decreases. Since the maximal number of possible violations is bounded by $O(nT^2)$, the overall running time of the reallocation algorithm is polynomial in n and T.

Note 2 The running time of our algorithms depend on T. Since the output size is $\Theta(nT)$, our algorithms are polynomial in the size of the output.

3.3 From Deadline Valuation Functions to General Valuation Functions

In this section, we extend our linear program relaxation to accommodate general valuation functions. The main difficulty here is how to provide a proper formulation of the objective function, that is, maximizing social welfare. Deadline valuation functions are simple to handle, since the value v_j gained from completing any job j is fixed, regardless of the job completion time. The job completion time $e(y_j)$ is not explicitly required in the formulation: deadlines are implicitly imposed by preventing resources from being allocated to jobs after their deadlines; specifically, the LP relaxation does not hold variables $y_j(t)$ for time slots later than the job deadline d_j. Avoiding the use of $e(y_j)$ in the LP formulation is essential, since it cannot be explicitly used in a linear program: the completion time, which is the maximal t for which $y_j(t) > 0$, cannot be derived by a set of linear constraints. Whereas for deadline valuation functions the use of $e(y_j)$ could be avoided, the more general case requires knowledge of the actual job completion time.

Our solution for general valuation functions makes use of deadline valuation functions, for which we know how to formulate the objective function of maximizing social welfare. The main idea is to decompose every general valuation function $v_j(t)$ into T deadline valuation functions, one per possible completion time. Formally, every player j will be represented by T virtual *sub-players* $j^1, j^2, \ldots, j^T$, where each subplayer j^e represents a copy of player j that has a deadline valuation function with deadline $d_j^e = e$ and corresponding value of $v_j^e = v_j(e)$. The LP relaxation may allocate resources to any subplayer, as long as the total amount of resources allocated to *all* subplayers of player j does not exceed D_j. The parallelism bound of each subplayer remains k_j. The modified linear program **(LP)** for general valuation functions is given below:

$$\text{(LP)} \quad \max \quad \sum_{j=1}^{n} \sum_{e \in \mathcal{T}} v_j^e x_j^e$$

$$\text{s.t.} \quad \sum_{t \le d_j^e} y_j^e(t) = D_j \cdot x_j^e \quad \forall j^e \tag{14}$$

$$\sum_{j^e} y_j^e(t) \le C \quad \forall t \in \mathcal{T} \tag{15}$$

$$\sum_{e \in \mathcal{T}_j} x_j^e \le 1 \quad \forall j \in \mathcal{J} \tag{16}$$

$$y_j^e(t) \le k_j x_j^e \quad \forall j^e, t \le d_j^e \tag{17}$$

$$y_j^e(t) \ge 0 \quad \forall j^e, t \le d_j^e \tag{18}$$

The new constraint (16) guarantees that the total amount of resources allocated to subplayers of each player j does not exceed D_j. We claim that (LP) is a relaxed formulation of BFS with general valuation functions. To see this, consider a feasible solution to BFS where every scheduled job j is allocated according to an allocation

a_j. We can easily translate this solution into a feasible solution to (LP): the allocation a_j corresponds to a subplayer $j^{e(a_j)}$ having a deadline $e(a_j)$. The value gained by this subplayer is exactly $v_j(e(a_j))$, which is exactly the value gained from job j in the feasible solution to BFS.

Consider an optimal fractional solution to (LP). We can assume without loss of generality that all of the resource mappings in this solution are MND. This is true, since we can apply the reallocation algorithm without violating any constraint. As a result, the mapping corresponding to each subplayer in the system will be MND. This will be useful in Sect. 3.5, where we will show how to decompose an optimal fractional solution into a set of feasible BFS solutions.

Before continuing with the construction of the scheduling algorithm, notice that we did not assume at any point of the discussion that the valuation functions were monotonically non-increasing. This is a natural assumption in practice. However, this can potentially invalidate our claim that (LP) is a relaxed formulation of BFS. An optimal fractional solution to (LP) can mach an allocation a_j to a subplayer j^e with $e > e(a_j)$ and $v(e) > v(e(a_j))$ and gain increased value. In the next subsection, we show that we can assume that without loss of generality, all valuation functions are monotonically non-increasing; specifically, a scenario as described in this paragraph is unlikable.

Note 3 The algorithms we construct in the following subsections refer to (LP), the relaxed linear program for general valuation functions after adding constraint (16), instead of (LP-D). Specifically, the decomposition algorithm described in Sect. 3.5 works on a optimal fractional solution for (LP). The linear programming relaxation (LP-D) for deadline valuation functions can be obtained directly from (LP) by viewing each player as having a single subplayer j^{d_j} (making (16) redundant).

3.4 Non-monotone Valuation Functions

Even though in natural settings valuation functions are monotonically non-increasing, as the value gained by users from their job being completed usually decreases over time, we show that this assumption is not mandatory. We reduce the case of general valuation functions to the former setting, where the valuation functions are monotonically non-increasing. Practically speaking, if a job is completed in some time t with the valuation function receiving a higher value at a time later than t, we can simply delay the job until that time. For the sake of completeness, we show that both models are equivalent from a theoretical perspective.

Every non-monotone valuation function v_j will be *lifted* to an equivalent valuation function, breaking non-monotonicity violations of the original valuation function. Formally, we substitute every valuation function v_j with a new valuation function $\bar{v}_j$ defined as follows:

$$\bar{v}_j(t) = \max_{t' \geq t} \left\{ v_j\left(t'\right) \right\} \tag{19}$$

Recall that the value gained by a user allocated according to an allocation a_j is defined by the latest time slot in which j is allocated, that is, $v_j(a_j) = v_j(e(a_j))$. Let $\bar{y}$ be a vector of mappings obtained by solving (LP) with the valuation functions $\bar{v}_j$. We

Coloring Algorithm($\mathcal{S}$)
1. Sort the MND allocations $a \in \mathcal{S}$ according to $e(a)$ in descending order.
2. For every MND allocation a in this order
   2.1 Color a in some color c such that c remains a feasible integral solution.

show how to transform it into a value-equivalent vector under the original valuation functions v_j. As in Sect. 3.2, we simplify arguments by adding an "idle" job which is allocated in all of the free spaces, thus we assume that the capacity is full in every time slot.

Consider some mapping $\bar{y}_j$ (or equivalently, a mapping matching a subplayer j^e) such that $v_j(\bar{y}_j) < \bar{v}_j(\bar{y}_j)$. Let $t = e(\bar{y}_j)$ and let:

$$\bar{t} = \arg\max_{t' \geq t} \{v_j(t')\} \tag{20}$$

Rewriting the assumption, we get $v_j(t) < v_j(\bar{t})$. By the definition of t, $\bar{y}_j(t) > \bar{y}_j(\bar{t}) = 0$. Hence, since the capacity is full in every time slot, there must be some job j' with $\bar{y}_{j'}(t) < \bar{y}_{j'}(\bar{t}) \leq k_{j'}$. Let $\delta > 0$ be a small enough value. We decrease both $\bar{y}_j(t)$ and $\bar{y}_{j'}(\bar{t})$ by δ and increase $\bar{y}_j(\bar{t})$ and $\bar{y}_{j'}(t)$ by δ. Notice that we can choose a small enough value δ such that no parallelization bound is violated and the end time of j' does not change. Reapplying this step gives us the desired reduction.

3.5 Decomposing an Optimal MND Fractional Solution

The approximation algorithm presented in this section constructs a set of feasible solutions to BFS out of a fractional optimal solution to (LP) given in the canonical MND form. The algorithm is similar to the coloring algorithm used in [4, 16] for the weighted job interval scheduling problem. The first step of the algorithm constructs a multiset $\mathcal{S} \subset \bigcup_{j=1}^{n} \mathcal{A}_j$ of allocations out of an optimal fractional solution given in MND form and then divides the allocations in $\mathcal{S}$ into a set of feasible solutions to BFS.

Step I: Construction of $\mathcal{S}$ Let N be a large number to be determined later. Consider a job j which is substituted by a set of subplayers $j^1, j^2, \ldots, j^T$ (or a single subplayer j^{d_j} for the case of deadline valuation functions). Let y be an optimal solution of (LP) after applying the reallocation algorithm. For every subplayer j^e, let a_j^e be the allocation corresponding to y_j^e, defined as follows:

$$a_j^e(t) = \frac{y_j^e(t)}{x_j^e} \quad \forall t \in \mathcal{T}_j \tag{21}$$

Note that a_j^e is an allocation by the definition of x_j^e and by (6). We construct $\mathcal{S}$ as follows: Let $\bar{x}_j^e$ denote the value x_j^e rounded up to the nearest integer multiplication of $\frac{1}{N}$. For every subplayer j^e, we add $N\bar{x}_j^e$ copies of a_j^e to $\mathcal{S}$.

Step II: Coloring Allocations The coloring algorithm (described above) colors copies of MND allocations in $\mathcal{S}$ such that any set of allocations with identical color will induce a feasible integral solution to BFS. Let $1, 2, \ldots, COL$ denote the set of colors used by the coloring algorithm. We use $a \in c$ to represent that an allocation a is colored in color c. Given a color c, let $c(t) = \sum_{a \in c} a(t)$ denote the total load of MND allocations colored in c at time t. The following two claims prove that the number of colors used is relatively small. This allows us to construct an α-approximation algorithm in Theorem 2.

Claim *Consider an iteration after some allocation $a \in \mathcal{S}$ is colored. Then, for every color c, $c(t)$ is monotonically non-decreasing in the range $[1, e(a)]$.*

Proof By induction. Initially, $\forall t, c(t) = 0$ for every color c. Consider an iteration where a is colored. By the induction hypothesis, $c(t)$ is monotonically non decreasing for every color c in the range $[1, e(a)]$ (the induction hypothesis guarantees that $c(t)$ is monotonically non decreasing for a possibly larger range, since we color allocations in decreasing order of $e(\cdot)$). Since a is an MND allocation, coloring a with some color c keeps $c(t)$ non decreasing in $[1, e(a)]$. □

Claim *The coloring algorithm succeeds when*:

$$COL = N \cdot \left(1 + \frac{C}{C-k}\right)\left(1 + \frac{nT}{N}\right) \tag{22}$$

Proof Consider an iteration of the coloring algorithm where some allocation $a \in \mathcal{A}_j$ of a job j is colored. It is enough to show that there is an available color in which we can color the allocation a. A color c may be unavailable due to one of the following reasons: (1) the color c has already been used to color an allocation of j considered earlier throughout the algorithm, i.e., an allocation in $\mathcal{S} \cap \mathcal{A}_j$; (2) coloring a in color c violates the capacity constraint of color c. It remains to bound the number of unavailable colors of both types.

Consider first type (1). Recall that for every allocation corresponding to subplayer j^e we add $N \cdot \bar{x}_j^e$ copies of a_j^e to $\mathcal{A}_j$, and that $\bar{x}_j^e \le x_j^e + 1/N$. Using constraint (16), we get that the total number of colors used to color allocations of j other than a is at most:

$$N \cdot \sum_e \bar{x}_j^e - 1 \le N \cdot \sum_e \left(x_j^e + \frac{1}{N}\right) \le N \cdot \left(1 + \frac{T}{N}\right) \tag{23}$$

where the last inequality follows since each player has T subplayers.

Now, consider a color c belonging to type (2). By the monotonicity of both $c(t)$ and a, we must have $c(e(a)) \ge C - k_j \ge C - k$. The total workload of instances in $\mathcal{S}$ at any time t is at most:

$$N \cdot \sum_{j^e} a_j^e(t) \cdot \bar{x}_j^e \le N \cdot \sum_{j^e} a_j^e(t) \cdot \left(x_j^e + \frac{1}{N}\right) \tag{24}$$

$$\leq N \cdot \sum_{j^e} a_j^e(t) \cdot x_j^e + N \cdot \sum_{j^e} a_j^e(t) \cdot \frac{1}{N} \tag{25}$$

$$\leq CN + knT \tag{26}$$

$$\leq CN \cdot \left(1 + \frac{nT}{N}\right). \tag{27}$$

Inequality (25) follows since the left summation represents the total workload at time t in the optimal fractional solution, which is exactly C; the right summation can be bounded using $a_j^e(t) \leq k$ for every subplayer j^e and time t. Thus, the number of colors in which a cannot be colored due to capacity constraints is at most $\frac{CN}{C-k}(1 + \frac{nT}{N})$. □

Theorem 2 *There is a poly$(n, T, \frac{1}{\varepsilon})$ time approximation algorithm that given an optimal solution to (LP) returns an*

$$\alpha \triangleq \left(1 + \frac{C}{C-k}\right)(1+\varepsilon) \tag{28}$$

approximation to the BFS problem for every $\varepsilon > 0$.

Proof Let y^* be an optimal solution of (LP) after applying the reallocation algorithm and let OPT^* be the optimal social welfare matching y^*. We construct a multiset $\mathcal{S}$ as described in Step I and decompose $\mathcal{S}$ into COL solutions for BFS according to Step II, with a total value of:

$$N \cdot \sum_{j^e} v_j\left(a_j^e\right) \cdot \bar{x}_j^e \geq N \cdot OPT^* \tag{29}$$

Set $N = \frac{nT}{\varepsilon}$ (and therefore $COL = N\alpha$). The running time of the coloring algorithm is polynomially bounded by n, N, T and thus polynomially bounded by $n, T, \frac{1}{\varepsilon}$. The algorithm will allocate jobs according to the allocations colored by the best color c, in terms of social welfare. Denote by Alg the social welfare gained by this algorithm. Since:

$$Alg \geq \frac{N \cdot OPT^*}{COL} = \frac{OPT^*}{\alpha} \tag{30}$$

We get an α-approximation. We note that for deadline valuation functions, by applying similar arguments to the ones given in the proof of (22), we can show that taking $N = \frac{n}{\varepsilon}$ suffices. □

4 Truthfulness-in-Expectation

Up until now we have assumed that players report their true valuation functions to the cloud provider and that prices are charged accordingly. However, in reality, players may choose to untruthfully report a valuation function b_j which differs from their

true valuation function v_j if they may gain from it. In this section, we construct an efficient mechanism that charges costs from players such that reporting their valuation function untruthfully cannot benefit them. Unlike known black-box reductions for constructing such mechanisms, our construction calls the approximation algorithm only *once*, significantly improving the complexity of the mechanism.

We begin by introducing the common terminology used in mechanism design. Every participating player chooses a type out of a known type space. In our model, players choose a valuation function v_j out of the set of monotonically non-increasing valuation functions (or deadline valuation functions) to represent their true type. Denote by V_j the set of types from which player j can choose and let $V = V_1 \times \cdots \times V_n$. For a vector v, denote by v_{-j} the vector v restricted to entries of players other than j and denote V_{-j} accordingly. Define $\mathcal{O}$ as the set of all possible outcomes of the mechanism and let $v_j(o)$ for $o \in \mathcal{O}$ represents the value gained by player j under outcome o. A *mechanism* $\mathcal{M} = (f, p)$ consists of an allocation rule $f : V \to \mathcal{O}$ and a pricing rule $p_j : V \to \mathbb{R}$ for each player j. Players report a bid type $b_j \in V_j$ to the mechanism, which can be different from their true type v_j. The mechanism, given a reported type vector $b = (b_1, \ldots, b_n)$ computes an outcome $o = f(b)$ and charges $p_j(b)$ from each player. Each player strives to maximize its utility: $u_j(b) = v_j(o_j) - p_j(b)$, where o_j in our model is the allocation according to which job j is allocated, if at all. Mechanisms such as this, where the valuation function does not consist of a single scalar are called *multi-parameter* mechanisms. Our goal is to construct a multi-parameter mechanism where players benefit by declaring their true type. Another desired property is that players do not lose when truthfully reporting their values.

Definition 3 A deterministic mechanism is *truthful* if for any player j, reporting its true type maximizes $u_j(b)$. That is, given any bid $b_j \in V_j$ and any $v_{-j} \in V_{-j}$, we have:

$$u_j\big((v_j, v_{-j})\big) \geq u_j\big((b_j, v_{-j})\big) \tag{31}$$

where $v_j \in V_j$ is the true type of player j. A randomized mechanism is *truthful-in-expectation* if for any player j, reporting its true type maximizes the expected value of $u_j(b)$. That is, (31) holds in expectation.

Definition 4 A mechanism is *individually rational (IR)* if $u_j(v)$ does not receive negative values when player j bids truthfully, for every j and $v_{-j} \in V_{-j}$.

4.1 The Fractional VCG Mechanism

We start by giving a truthful, individually rational mechanism that can return a fractional feasible allocation, that is, allocate fractions of jobs according to (LP). Consider the following *fractional* mechanism:

1. Given reported types $b_j : \mathcal{T} \to \mathbb{R}^{+,0}$, Solve (LP) and get an optimal solution y^*. Let $o \in \mathcal{O}$ be the outcome matching y^* and let OPT^* be the social welfare when jobs are allocated according to y^*.
2. Charge $p_j(b) = h_j(o_{-j}) - \sum_{i \neq j} b_i(o_i)$ from every player j, where h_j is any function independent of o_j.

This is the well known VCG mechanism. Recall that (LP) maximizes the social welfare, i.e., the sum of values gained by all players. Players gain $g_j(v) = OPT^* - h_j(o_{-j})$ by bidding truthfully and therefore the mechanism is optimal, since deviating can only decrease $\sum_i v_i(o)$. Note that by dividing both valuation functions and charged prices by some constant, the fractional VCG mechanism remains truthful. This will be useful later on. Individual rationality of the fractional VCG mechanism is obtained by setting the functions h_j according to Clarke's pivot rule [15].

4.2 A New Efficient Truthful-in-Expectation Mechanism

Lavi and Swamy [11] give a black-box reduction for combinatorial auction packing problems from constructing a truthful-in-expectation mechanism to finding an approximation algorithm that verifies an integrality gap of the "natural" LP for the problem. Their reduction finds an exact decomposition of the optimal fractional solution (scaled down by some constant β) into a distribution over feasible integer solutions. By sampling a solution out of this distribution and charging payments according to the fractional VCG mechanism (scaled down by β), they obtain truthfulness-in-expectation. The downside of the reduction given in [11] is that the approximation algorithm A is used as a separation oracle for an additional linear program used as part of the reduction, making their construction inefficient. We follow along the lines of [11] in order to construct a truthful-in-expectation mechanism for the BFS problem, and show how to achieve the same results as [11] by calling our approximation algorithm once.

Recall that the algorithm from Theorem 2 constructs a set of feasible solutions to BFS out of an optimal solution to LP. Ideally, we would have wanted to replace the exact decomposition found by [11] with the output of our decomposition algorithm (by drawing one of the colors uniformly). However, this does not work since our decomposition is not an exact one, because the values x_j^e have been rounded up to $\bar{x}_j^e$ prior to the construction of $\mathcal{S}$.

To overcome this issue, we use a simple alternative technique to round the entries in x to integer multiplications of $\frac{1}{N}$. We construct a vector $\tilde{x}$ such that $\mathbb{E}[\tilde{x}_j^e] = x_j^e$ for every subplayer j^e, as follows: Assume that $x_j^e = \frac{q}{N} + r$ for $q \in \mathbb{N}$ and $0 \leq r < \frac{1}{N}$. Then, set $\tilde{x}_j^e = \frac{q+1}{N}$ with probability $N \cdot r$ and $\tilde{x}_j^e = \frac{q}{N}$ otherwise. Note that $\mathbb{E}[\tilde{x}_j^e] = x_j^e$ as required. Now, we construct $\mathcal{S}$ out of $\tilde{x}$ and call the coloring algorithm. By uniformly drawing one of the colors c and scheduling jobs according to the allocations colored in c, we obtain an expected welfare of: $\mathbb{E}[\frac{N}{COL}\sum_{j^e} v_j \tilde{x}_j^e] = \frac{OPT^*}{\alpha}$. By charging fractional VCG prices, scaled down by α, we obtain truthfulness-in-expectation. Notice that this mechanism is not individually rational, since unallocated jobs may be charged. Lavi and Swamy [11] solve this problem by showing how to modify the pricing rule so that the mechanism will be individually rational. Notice that the number of colors used by the coloring algorithm must always be *COL*, even though it is an upper bound on the number of colors needed. Otherwise, players might benefit from reporting their valuation functions untruthfully by effecting the number of solutions.

Theorem 3 *There is a truthful-in-expectation, individually rational mechanism for BFS that provides an expected α-approximation of the optimal social welfare.*

Finally, we discuss the process of computing the payments $p_j(b)$. Note that to directly calculate the payments charged by VCG, one must solve a linear program for every player j. It is possible to construct a much more efficient mechanism: Babaioff et al. [2] describe an implicit pricing scheme that requires only a single invocation of the approximation algorithm to construct both an allocation rule and pricing rules of a truthful-in-expectation mechanism. This result can be plugged into our mechanism, thus decreasing the number of calls to our approximation algorithm to one. However, their scheme induces a mechanism that is only individually rational in expectation (specifically, it may charge negative prices) and causes a multiplicative (constant) loss to social welfare. To conclude, a truthful-in-expectation mechanism requiring a single call to the approximation algorithm exists, however it not being rational makes it unpractical.

Appendix: The Configuration LP: A Different View of the LP Formulation

We present further insights on the choice of (6). An alternative formulation of BFS is its representation as a configuration LP. For every job j and every allocation $a_j \in \mathcal{A}_j$ we have a variable $z_j(a_j)$ representing whether job j has been fully allocated according to a_j or not. The configuration LP is defined as follows:

$$\begin{aligned}
\max \quad & \sum_{j=1}^{n} \sum_{a_j \in \mathcal{A}_j} v_j \cdot z_j(a_j) \quad \text{(CONF-LP-D)} \\
\text{s.t.} \quad & \sum_{a_j \in \mathcal{A}_j} z_j(a_j) \leq 1 \quad \forall j \in \mathcal{J} \\
& \sum_{j:t \leq d_j} \sum_{a_j \in \mathcal{A}_j} a_j(t) \cdot z_j(a_j) \leq C \quad \forall t \in \mathcal{T} \\
& z_j(a_j) \geq 0 \quad \forall j \in \mathcal{J}, a_j \in \mathcal{A}_j
\end{aligned}$$

The first constraints allow us to choose at most one allocation per job and the second constraints are capacity constraints. Note that since allocations are defined over the reals, the number of allocations in a set $\mathcal{A}_j$ is uncountable. However, the following proposition shows that (LP-D), is actually an efficient representation of (CONF-LP-D).

Proposition 1 The optimal values of fractional social welfare of (*LP-D*) and (*CONF-LP-D*) are equal.

Proof Consider a solution y of the relaxed linear program and recall that $x_j = \frac{1}{D_j} \sum_t y_j(t)$. For every job j we construct an allocation a_j matching the values $\{y_j(t)\}$ by setting: $a_j(t) = \frac{y_j(t)}{x_j}$. This gives us a feasible allocation, since $\sum_t a_j(t) = D_j$ and $a_j(t) \leq k_j$ for every $t \in \mathcal{T}_j$ by (6). Now, set $z_j(a_j) = x_j$ and $z_j(a) = 0$ for

every $a \in \mathcal{A}_j \setminus \{a_j\}$. It is easy to see that z is a feasible solution of the configuration LP.

In the opposite direction, consider a solution z of the configuration LP. Define for every $t \in \mathcal{T}_j$:

$$a_j(t) = \frac{\sum_{a \in \mathcal{A}_j} z_j(a) \cdot a(t)}{x_j}$$

where $x_j = \sum_{a \in \mathcal{A}_j} z_j(a)$. Notice that a_j is a feasible allocation, since $a_j(t) \leq k_j$ for every $t \in \mathcal{T}_j$ and since:

$$\sum_{t \in \mathcal{T}_j} a_j(t) = \frac{\sum_{t \in \mathcal{T}_j} \sum_{a \in \mathcal{A}_j} z_j(a) \cdot a(t)}{x_j}$$

$$= \frac{\sum_{a \in \mathcal{A}_j} z_j(a) \cdot \sum_{t \in \mathcal{T}_j} a(t)}{x_j} = D_j$$

By the definition of a_j, the total capacity taken by a_j is identical to the capacity taken by allocations according to z. Moreover, the contribution of a_j to the objective function is the sum of contributions by allocations in $\mathcal{A}_j$. To conclude, we translate a_j to its matching vector y_j by setting $y_j(t) = a_j(t) \cdot x_j$ for every $t \in \mathcal{T}_j$. □

References

1. Archer, A., Tardos, É.: Truthful mechanisms for one-parameter agents. In: FOCS, pp. 482–491 (2001)
2. Babaioff, M., Kleinberg, R., Slivkins, A.: Truthful mechanisms with implicit payment computation. In: EC, pp. 43–52 (2010)
3. Bar-Noy, A., Bar-Yehuda, R., Freund, A., Naor, J., Schieber, B.: A unified approach to approximating resource allocation and scheduling. J. ACM **48**, 1069–1090 (2001)
4. Bar-Noy, A., Guha, S., Naor, J., Schieber, B.: Approximating the throughput of multiple machines in real-time scheduling. SIAM J. Comput. **31**(2), 331–352 (2001)
5. Bei, X., Huang, Z.: Bayesian incentive compatibility via fractional assignments. In: SODA (2011)
6. Brucker, P.: Scheduling Algorithms, 4th edn. Springer, Berlin (2004)
7. Chawla, S., Immorlica, N., Lucier, B.: On the impossibility of black-box transformations in mechanism design. CoRR **abs/1109.2067** (2011)
8. Dughmi, S., Roughgarden, T.: Black-box randomized reductions in algorithmic mechanism design. In: FOCS, pp. 775–784 (2010)
9. Hartline, J.D., Lucier, B.: Bayesian algorithmic mechanism design. In: STOC, pp. 301–310 (2010)
10. Hartline, J.D., Kleinberg, R., Malekian, A.: Bayesian incentive compatibility via matchings. In: SODA (2011)
11. Lavi, R., Swamy, C.: Truthful and near-optimal mechanism design via linear programming. In: FOCS, pp. 595–604 (2005)
12. Lavi, R., Swamy, C.: Truthful mechanism design for multi-dimensional scheduling via cycle monotonicity. In: EC (2007)
13. Lawler, E.L.: A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. Ann. Oper. Res. **26**, 125–133 (1991)
14. Nisan, N., Ronen, A.: Algorithmic mechanism design. In: STOC (1999)
15. Nisan, N., Roughgarden, T., Tardos, É., Vazirani, V.V.: Algorithmic Game Theory. Cambridge University Press, Cambridge (2007)
16. Phillips, C.A., Uma, R.N., Wein, J.: Off-line admission control for general scheduling problems. In: SODA, pp. 879–888 (2000)