Mechanisms with Monitoring for Truthful RAM Allocation*

Annamária Kovács[1], Ulrich Meyer[1], and Carmine Ventre[2]

[1] Goethe University Frankfurt am Main, Germany
[2] Teesside University, UK

Abstract. Novel algorithmic ideas for big data have not been accompanied by advances in the way central memory is allocated to concurrently running programs. Commonly, RAM is poorly managed since the programs' trade offs between speed of execution and RAM consumption are ignored. This trade off is, however, well known to the programmers. We adopt mechanism design tools to truthfully elicit this (multidimensional) information with the aim of designing more clever RAM allocation algorithms. We introduce a novel paradigm wherein programs are bound to overbidding declarations of their running times. We show the limitations of this paradigm in the absence of transfers and prove how to leverage waiting times, as a currency, to obtain optimal money burning mechanisms for the makespan.

1 Introduction

With data volumes growing much faster than typical users' computing infrastructure, the role of efficient algorithms for big data becomes crucial. While it might be tempting to move all data to some huge commercial cloud service, legal and logistic issues will often force users or companies to keep their valuable data locally and either stick to their existing hardware or seek for a moderate cost-effective upgrade. In this situation, users are often rather willing to slowly run their programs concurrently on shared hardware and compete for resources instead of submitting their programs to an offline queuing system where they might experience unpredictable waiting times before their program is quickly executed. As a consequence the accumulated input data will typically not completely fit in the main memory (RAM) of the computer system at hand but has to reside on external storage like hard disks. External-memory (EM) algorithms [12,17] are especially tuned for this setting. They optimize the data access patterns and typically perform the better the more RAM they are assigned. However, there are huge differences: For example during a linear data scan, EM algorithms often can do with only a constant number of pages held in RAM, whereas for EM merge-sort of n items, the number of phases is bounded by $O(\log_x n)$ where x

* Partially supported by the DFG grant ME 2088/3-1, EPSRC grant EP/M018113/1, and by MADALGO – Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

denotes the number of pages that can be kept in RAM simultaneously: obviously, the larger x the faster the sorting, but one essentially has to square x in order to halve the number of sorting phases.

In isolation, a typical EM algorithm prefers to take as much RAM as is available. For example, prior to running a program from the EM algorithms library STXXL [6] the default setting is to reserve a significant fraction of the available RAM for its execution. With several programs competing for a shared RAM the task becomes significantly harder, especially if these programs are not cooperating. Already from the simple example discussed above it is clear that assigning equally sized fixed shares of RAM to all programs will not necessarily optimize overall performance. Therefore, a common operating systems' solution is to apply online paging algorithms like LRU that dynamically decide which pages to keep in RAM. Unfortunately, even for a single program, online paging algorithms do not know about the future request sequences and are therefore prone to wrong decisions in the worst case [7].

We aim to use methods from Algorithmic Mechanism Design (AMD) in order to reasonably solve the RAM assignment problem for concurrently running programs: In principle, the best knowledge about the trade offs between usable RAM size and performance is with the designers/programmers of the individual algorithms. If they extend their programs with an interface in order to bid for individual RAM sizes, an operating system could use these bids within an appropriate mechanism in order to solve the RAM assignment problem for concurrently running programs. The obvious advantage of this setting would be that neither the *users* of programs (who are typically *not the programmers*) nor the operating system need knowledge about the RAM-performance footprints of the individual programs and yet obtain a reasonable RAM sharing. Of course the RAM assignment mechanism must be designed to motivate truthful requests, especially if the users do not have to pay money for the RAM their executed programs occupy. In the absence of money, in fact, selfish programmers could claim to need unreasonably large chunks of central memory for a "fast" execution of their programs.

Our contribution. In this work, we focus on the design of truthful mechanisms in the context of RAM allocation to programs, with the objective to minimize the makespan. Specifically, we concentrate on a feasibility study for the case of a *single* execution interval and ask the question of whether truthfulness can be enforced, and at what cost, in the single-shot case.

Monitoring. In our model, each programmer, also termed *agent* or *bidder* in this setting, controls one program/task, and has as a *type* a decreasing[3] *cost function* mapping an amount of allocated RAM to execution times. She declares to the RAM allocation mechanism a potentially different function as her cost.[4] Ideally, truthfulness of bidders' declarations should be guaranteed without the

[3] For simplicity, throughout the paper we use 'decreasing' with the meaning 'non-increasing', and similarly we use 'increasing' instead of 'non-decreasing'.

[4] When these functions have a "large" representation, oracle queries are used just like in the Combinatorial Auctions (CA) literature [3], see preliminaries for a discussion.

use of monetary transfers since there is no easy way to charge the programmers. However, very little can be done in mechanism design when money is out of the picture, e.g., as noted above, nothing prevents programmers from exaggerating their RAM needs by overbidding their execution times. Therefore, we look at a mechanism design framework wherein a bidder overbidding her execution time (her *cost* in mechanism design terminology), ends up with this augmented execution time (rather than her true execution time).[5] So a bidder's reported cost will be interpreted by the mechanism as a lower bound to her execution time: a bidder will be allowed to be slower than declared but not faster. This assumption was part of the model defined by Nisan and Ronen in [13][6], and has been later dubbed *monitoring* in [14]. We believe these mechanisms to make a feasible assumption that can be implemented in some real-life scenarios (such as the application that motivates us).

Monitoring and verification. Motivated by the recent advances in trading verification of bidders' behavior with money in mechanism design for CAs [8,9], in Section 4 we ask whether similar conclusions can be drawn in our scenario. *Verification*, in this context, means that a bidder cannot underbid her execution time, for otherwise the RAM would be preempted and the task would be aborted (see preliminaries for a formal definition). We call a mechanism using monitoring and verification, a *mechanism with strong monitoring*. We prove a quite interesting dichotomy.[7] To the algorithmic characterization of truthfulness, we pair the positive result that the optimum makespan can be computed truthfully when tasks have *known k-step* cost functions (i.e., the values of the cost functions are known to the mechanism but the discontinuity points are not) and the negative result that no approximation of the optimal makespan can be returned by a truthful mechanism when the k-step functions are *unknown* (i.e., value and discontinuity points are both unknown) even for $k = 1$.

Monitoring and transfers. Given the limitations of mechanisms with strong monitoring and no monetary transfers, in Section 5 we turn our attention to mechanisms using (some form of) transfers. Since, as observed above, currency is not available in the setting at hand, we interpret transfers as *waiting times* and focus on *money burning mechanisms* [10]. In details, the output of a mechanism will be a RAM chunk size and a waiting time for each bidder so that the bidder will be able to run her task using the amount of RAM allocated by the mechanism only *after* the waiting time. Since waiting times degrade the performance of the system, the objective function of interest must take transfers in consideration. In our case, the objective is the minimization of the maximum (over all bidders)

[5] This might be implemented by letting the mechanism hold back the results of the computation whenever the program terminates before the reported time.

[6] Specifically, Nisan and Ronen embedded the monitoring assumption in their 'mechanisms with verification', but here we use the term verification in a different sense.

[7] Even though we depart from much of the recent literature (see, e.g., [8,14] and references therein) on mechanisms with verification, which uses no monitoring, we remark that using similar arguments, one can prove the same dichotomy also in that weaker model.

of the *total cost*, where the total cost of a bidder is defined as the *sum* of her execution time and waiting time (transfer). This is a "money burning" version of the makespan objective function. Here we drop the verification assumption but keep the monitoring hypothesis, and call the mechanisms in this section *mechanisms with monitoring*. As a warm-up, we consider the case that RAM chunking is fixed and give a truthful mechanism with monitoring that returns solutions minimizing the makespan (for the fixed chunking), and the total costs of the tasks do not exceed the makespan. This mechanism is thus optimal not only for the classical makespan minimization objective, but also for the money burning objective function. We also show that its transfers (waiting times) are minimal, for the given allocation. We complement this result by showing how to maintain optimality, minimal transfers and truthfulness while computing the RAM chunking that gives the minimum possible makespan.

Following the preliminaries in Section 2, Section 3 provides a graph-theoretic characterization of the algorithms that are truthful(ly implementable) in a mechanism with monitoring. In Sections 4 and 5 we present our results for mechanisms with strong monitoring and no transfers, and for mechanisms with monitoring and transfers, respectively. Some proofs are omitted due to space limitations.

Related work. This study connects to a number of research agendas in (A)MD.

Mechanisms with verification (i.e., strong monitoring in our terminology) have been introduced in [13] for the problem of scheduling unrelated selfish machines. A stream of work has looked instead at verification without monitoring (i.e., assuming only no underbidding) in the presence of money [16,14,11] and without [8,9]. Money burning mechanisms are studied in [10] for single-parameter agents and *utilitarian* money burning objective functions. For multi-unit auctions, [10] shows that the largest relative loss due to money burning is logarithmic in the number of bidders. In contrast, we show that transfers do not add any cost to the *makespan*.

An interesting hybrid between verification and money burning mechanisms is [2], which considers exact but costly verification and seeks to maximize the social welfare minus the verification cost for truthful auctions of indivisible goods.

Mechanisms for selfish jobs are also relevant. [4,1] consider truthful mechanisms for selfish one-parameter tasks and makespan minimization on identical machines, with different definitions for the tasks' completion times, use of money, and definition of verification. Coordination mechanisms [5] deal with selfish tasks but focus on equilibrium approximation guarantee rather than truthfulness.

2 Preliminaries

We have one resource available in m copies and n selfish agents. Each selfish agent has a decreasing *cost function*, also called *type* $t_i : [m]_{>0} \to \mathbb{R}_{>0}$, where $[m]_{>0}$ denotes the set of positive integers not larger than m. For $m' \in [m]_{>0}$, $t_i(m')$ is the cost paid by agent i if she is allowed to use m' copies of the resource. The type t_i is *private knowledge* of agent i. The set of all legal cost functions t_i is called the *domain* of agent i, and is denoted by D_i.

Assuming that each agent has reported or *bid* a (true or false) cost function $b_i \in D_i$, a *mechanism* determines an allocation $(o_1, \ldots, o_n)$ of the m copies of the resource to the n agents. The set $\mathcal{O}$ of all possible allocations contains tuples $(o_1, \ldots, o_n)$ such that $o_i \geq 1$ and $\sum_i o_i \leq m$. Furthermore, depending on (the allocation and) the b_i,'s, it may determine transfers to be paid by the agents. In our model without money, transfers are realized as waiting times, and will be denoted by $w = (w_1, \ldots, w_n)$. (When currency is involved, transfers are usually called payments in literature.) In summary, a *mechanism* is a pair (f, w), where $f : D_1 \times \ldots \times D_n \to \mathcal{O}$ is an algorithm (also termed *social choice function*) that maps agents' costs to a feasible solution in $\mathcal{O}$; and $w : D_1 \times \ldots \times D_n \to \mathbb{R}^n_{\geq 0}$ is a function mapping cost vectors to transfers from each agent i to the mechanism.

A *mechanism without transfers* is simply a social choice function f as above; sometimes, it is convenient to see a mechanism without transfers as a pair (f, w) where w is a constant function of value 0.

Given a vector $\mathbf{b} = (b_i, \mathbf{b}_{-i}) = (b_1, \ldots, b_n)$ of reported cost functions, and $f(\mathbf{b}) = (o_1, \ldots, o_n)$, we let $f_i(\mathbf{b}) = o_i$ denote the number of copies of the resource that the function f assigns to agent i on input $\mathbf{b}$. We assume no externalities, that is, the cost of an agent depends only on her own received number of copies. Therefore, $b_i(f(\mathbf{b})) = b_i(f_i(\mathbf{b}))$. For mechanism (f, w) (with or without transfers) let $cost_i^{(f,w)}(b_i, \mathbf{b}_{-i})$ denote the *total cost* (including transfer w_i) of agent i for the output computed by (f, w) on input $(b_i, \mathbf{b}_{-i})$. Since the types t_i are private knowledge of the agents, they might find it profitable to bid $b_i \neq t_i$. We are interested in mechanisms for which truthtelling is a dominant strategy for each agent.

Definition 1 (Truthful mechanisms). *A mechanism (f, w) (with or without transfers) is* truthful *if for any i, any bids $\mathbf{b}_{-i}$ of the agents other than i, and any $b_i \in D_i$, $cost_i^{(f,w)}(t_i, \mathbf{b}_{-i}) \leq cost_i^{(f,w)}(b_i, \mathbf{b}_{-i})$.*

Observe that the mechanisms we deal with, are not *individually rational*, in that the agents have a positive cost, and therefore negative valuation for any outcome. Also, not giving an agent any portion of the resource, is not a possible output for the mechanism. Formally, we could elaborate on this (i.e., make the mechanism individually rational), by assuming that an agent not performing her task incurs an infinitely high cost.

Commonly, $cost_i$ is defined as a linear combination of the transfer and the agent's *true* cost for the resource allocated by the algorithm. Here, we define a novel mechanism design paradigm, called *mechanisms with monitoring*, wherein this quasi-linear definition is maintained but costs paid by the agents for the allocated resource are more strictly tied to their declarations. Intuitively, monitoring means that those agents who are allocated a portion of the resource for that their reported cost was exaggerated ($b_i > t_i$), have to process their task up to time b_i instead of the true processing time t_i.

Definition 2 (Mechanism with monitoring). *In a* mechanism with monitoring *(f, w), the bid b_i is a* lower bound *on agent i's cost of using $f_i(b_i, \mathbf{b}_{-i})$,*

so an agent is allowed to have a real cost higher than $b_i(f(\mathbf{b}))$ *but not lower. Formally, we have* $cost_i^{(f,w)}(b_i, \mathbf{b}_{-i}) := w_i(\mathbf{b}) + \max\{t_i(f(\mathbf{b})), b_i(f(\mathbf{b}))\}$.

This notion of monitoring is very much related to a concept introduced and termed 'mechanisms with verification' by Nisan and Ronen in [13]. The idea is that if costs are verifiable (e.g., they represent time) and if agents are monitored and claim that the cost using resource $f_i(\mathbf{b})$ is $b_i(f_i(\mathbf{b}))$, then either this is going to be their actual cost (whenever $b_i(f(\mathbf{b})) \geq t_i(f(\mathbf{b}))$), or it can be verified that this declaration is insincere since $b_i(f(\mathbf{b}))$ (which is smaller than $t_i(f(\mathbf{b}))$) is not going to be enough for them to complete their work with the resource (e.g., execute a job). The latter case assumes implicitly that the resource is preempted after $b_i(f(\mathbf{b}))$ time steps at which point the cost of the agent is simply ∞. In other words, agents will never underbid in the model of [13]. Our *mechanism with monitoring* model is much less restrictive and punitive for the agents as we allow them to complete the job, i.e., we do not preempt resources. Moreover, unlike [13], we do not tie transfers with observed costs but only with declarations.

However, for our partially negative results without transfers (Section 4), we allow the mechanisms to even use both monitoring (for overbidding agents) *and* verification (for underbidding agents), i.e., the resource is never provided longer than $b_i(f_i(\mathbf{b}))$ time for processing the task of i. Practically, in this model underbidding is excluded. We call a mechanism with monitoring that also uses this verification for underbidding agents a *mechanism with strong monitoring*.

We say that a social choice function f is *implementable with (strong) monitoring* if there exists a suitable transfer function w such that (f, w) is a truthful mechanism with (strong) monitoring. In this case, we say that w (strongly) implements f. Given $\mathbf{b}$, we say that w_i^* *minimally implements* f *at* $\mathbf{b}$ *for agent* i if $w_i^*(\mathbf{b}) = \min_{w \text{ implements } f} w_i(\mathbf{b})$.

We consider mechanisms that run in time polynomial in n and $\log m$; however, the representation of the types might need time exponential in those parameters. We therefore assume, as in the related literature, that types are accessed through value or demand queries [3] depending on the algorithm at hand.

3 Graph-theoretic characterization of truthful mechanisms with monitoring

In this section we show how to adapt the cycle-monotonicity technique to design truthful mechanisms with (strong) monitoring. The proofs are standard, and are omitted in this short version.

The central tools we use are defined next. An edge (a, b) in the defined graphs represents the option of bidding b instead of the true cost function a. The weight $\delta_{a,b}$ of the edge represents the difference of actual costs when bidding b instead of a; thus when negative, its absolute value is a lower bound on the difference of truthful payments.

Definition 3. *Let* f *be a social choice function. For every* i *and* $\mathbf{b}_{-i}$*, the* declaration graph associated to f *has a node for each type in* D_i *and an additional* source node *called* ω*. The set of directed weighted edges is defined*

as follows. For every $a, b \in D_i$, $a \neq b$, *add an edge* (a, b) *of weight* $\delta_{a,b} := \max\{a(f(\mathbf{b}), b(f(\mathbf{b}))\} - a(f(a, \mathbf{b}_{-i}))$*; for any* $a \in D_i$, *add an edge* (ω, a) *of weight* 0.

The verification graph associated to f *is defined similarly but an edge* (a, b) *belongs to the graph only if* $a(f(b, \mathbf{b}_{-i})) \leq b(f(b, \mathbf{b}_{-i}))$.

Note that the *declaration graph* will be useful for proving truthfulness with monitoring, and the *verification graph* can be used in case of strong monitoring. Since in the latter case underbidding is not an option, no edge (a, b) has to be considered if $b(f(b, \mathbf{b}_{-i})) < a(f(b, \mathbf{b}_{-i}))$.

The next theorem states that, in order to check that a social choice function is implementable with (strong) monitoring and with transfers, it suffices to check that all cycles of the associated graph(s) have a nonnegative weight. For implementation without transfers, instead, it suffices to look at the sign of every single edge. The argument is similar to that used for classical mechanisms [15,18] and mechanisms with verification and no monitoring [16,8].

Theorem 1. *A social choice function* f *is implementable with monitoring (resp., strong monitoring) when agents bid from finite domains, if and only if, for all* i *and declarations* $\mathbf{b}_{-i}$*, the declaration (resp., verification) graph associated to* f *does not have negative weight cycles.*

Moreover, f *is implementable with monitoring (resp., strong monitoring) without transfers if and only if, for all* i *and* $\mathbf{b}_{-i}$*, the declaration (resp., verification) graph associated to* f *does not have negative weight edges (the size of the domains does not matter).*

Given our interest in money burning mechanisms, we also prove here what form minimal transfers have.

Theorem 2. *Let* f *be a social choice function* f *implementable with monitoring (resp., strong monitoring). For any* $\mathbf{b} = (b_i, \mathbf{b}_{-i})$ *and any* i*, the transfer function that minimally implements* f *at* $\mathbf{b}$ *for agent* i *is* $w_i^*(\mathbf{b}) = -\mathcal{SP}(\omega, b_i)$*, where* $\mathcal{SP}(\omega, b_i)$ *is the length of the shortest path from* ω *to* b_i *in the declaration (resp., verification) graph associated to* f.

4 Mechanisms with strong monitoring and no transfers

Here we give results on mechanisms with strong monitoring and no transfers.

4.1 Algorithmic characterization

We begin by characterizing the class of algorithms that are truthful with strong monitoring in the case in which transfers are not allowed.

Theorem 3. *An algorithm* f *is truthful with strong monitoring and no transfers if and only if for all* i*,* $\mathbf{b}_{-i}$*, and* $a, b \in D_i$*,* $a(f(b, \mathbf{b}_{-i})) \leq b(f(b, \mathbf{b}_{-i}))$ *implies* $b(f(b, \mathbf{b}_{-i})) \geq a(f(a, \mathbf{b}_{-i}))$.

4.2 Known k-step tasks

We now provide the characterization for a specific family of domains for selfish tasks.

Definition 4. *The task (agent) i has a* known k-step function domain *if, for some* known $c_i^1 \geq \ldots \geq c_i^k$ *and unknown* $r_i^1 \leq \ldots \leq r_i^{k-1} (\leq r_i^k = m)$, *her type satisfies*

$$t_i(m') = \begin{cases} c_i^1 \text{ if } 0 < m' < r_i^1 \\ c_i^j \text{ if } r_i^{j-1} \leq m' < r_i^j,\ 1 < j \leq k \end{cases}.$$

The cost function of such a task is then completely determined by the threshold values r_i^j; i.e., D_i can be assumed to consist of vectors in $[m]_{>0}^k$.

A *known k-step task* is a task with a known k-step function domain. Below, with a slight abuse of notation, $a \in D_i$ will both denote the (k-)tuple in D_i and the corresponding cost function. We define the property that characterizes truthful algorithms f in this context. Subsequently, we show that this property is a quite natural one, in the sense that for a large class of objective functions, the optimal allocation fulfils it.

Definition 5. *An algorithm f is* k-step monotone *if for any i, $\mathbf{b}_{-i}$, and $a = (r_i^j)_{j=1}^k, b = (\tilde{r}_i^j)_{j=1}^k \in D_i$, with $r_i^j \leq \tilde{r}_i^j$ for all $1 \leq j < k$, $f_i(a, \mathbf{b}_{-i}) < r_i^j$ implies $f_i(b, \mathbf{b}_{-i}) < \tilde{r}_i^j$.*

Lemma 1. *An algorithm f is truthful with strong monitoring and no transfers for known k-step tasks if and only if it is k-step monotone.*

For a bid vector $\mathbf{b}$ and feasible solution $o = (o_1, \ldots, o_n) \in \mathcal{O}$, let $\mu(\mathbf{b}, o)$ be a function increasing in every single cost $b_i(o_i)$, e.g, for the makespan $\mu(\mathbf{b}, o) = \max_i b_i(o_i)$. Define OPT_μ as the social choice function that on input $\mathbf{b}$ returns a solution minimizing μ using a tie-breaking rule independent of $\mathbf{b}$.

Theorem 4. *For any increasing cost function μ, OPT_μ is k-step monotone.*

4.3 Unknown single-step tasks and limitations of mechanisms without transfers

Definition 6. *The task (agent) i has an* unknown single-step function domain *if her type satisfies*

$$t_i(m') = \begin{cases} h_i \text{ if } m' < r_i \\ l_i \text{ if } m' \geq r_i \end{cases},$$

for some unknown $h_i > l_i$ *and* unknown r_i. *The cost function of such a task is then completely determined by the triple (r_i, h_i, l_i); i.e., D_i can be assumed to consist of vectors in $[m]_{>0} \times \mathbb{R}_{>0}^2$.*

An *unknown single-step task* is a selfish task with an unknown single-step function domain. Given a cost function $a = (a_r, a_h, a_l) \in D_i$, a_r will denote the threshold in $[m]$, and a_h and a_l denote the high and low cost respectively.

Definition 7. *An algorithm f is* unknown single-step monotone *if for any i, $\mathbf{b}_{-i}$, and $a, b \in D_i$, such that $a_r \leq b_r$, $a_h \leq b_h$ and $a_l \leq b_l$, $f_i(a, \mathbf{b}_{-i}) < a_r$ implies $f_i(b, \mathbf{b}_{-i}) < b_r$.*

The property above characterizes truthfulness when costs are not known:

Lemma 2. *An algorithm f is truthful with strong monitoring and no transfers for unknown single-step tasks if and only if it is unknown single-step monotone.*

We now prove that these algorithms cannot return any reasonable approximation of the makespan.

Theorem 5. *For any $\alpha > 0$, there is no algorithm without transfers that is truthful with strong monitoring for unknown single-step tasks, and returns a better than α-approximation of the optimal makespan.*

Proof. Consider an instance with two unknown single-step tasks such that $r_1 = r_2 = r$, $l_1 = l_2 = 1$, $h_1 = \alpha(1+\delta)$ and $h_2 = 1+\delta$ for some $\delta > 0$. Set $r < m < 2r$ so that only one task can get the RAM she needs to be fast. Any better than α-approximate algorithm for the makespan will assign r to task 1 and some $\varepsilon > 0$ to task 2. Consider now a new instance wherein task 2 modifies h_2 as $h'_2 = \alpha^2(1+\delta)$. Since the algorithm is truthful then it must be unknown single-step monotone thus implying that the outcome of the algorithm cannot assign at least r to task 2, thus returning an α-approximation of the makespan (the optimum would indeed allocate r to task 2 and some $\varepsilon > 0$ to task 1).

We next show that by introducing transfers – in terms of waiting time for using the allocated RAM – we can indeed design better mechanisms for tasks with general cost functions.

5 Optimal mechanisms with monitoring using transfers

We begin with a general result. Quite interestingly, the next theorem shows that given monitoring, there is a truthful PTAS for scheduling unrelated machines (at least for finite domains), alternative to the compensation-and-bonus mechanism of [13], that does not need verification to be truthful.

Theorem 6. *For any social choice function f, there exists a transfer function w such that (f, w) is truthful with monitoring when agents bid from finite domains.*

For the applicability of the theorem, we need to bound and discretize the range of the admitted cost functions t_i, so we assume for the rest of the section that the $t_i(m')$ (and the bids $b_i(m')$) are integers from a given Interval $[0, T]$.

5.1 Optimal mechanism for makespan with fixed memory chunks

Assume that n memory chunks of fixed size have to be allocated one-to-one among n agents, each of whom has a task to process. We identify the memory

chunks with their sizes $m_1 \leq m_2 \leq \ldots \leq m_n$ in increasing order. Let $t_i(m_j)$ denote the (true) processing time of task i using a memory chunk of size m_j.

We consider a greedy allocation rule called *Best-Fit Procedure* that allocates the chunks in increasing order of size, as follows: m_1 is allocated to the task i with the minimum processing time given this amount of memory $t_i(m_1) = \min_k t_k(m_1)$; then iteratively, for every $j = 2, \ldots, n$, m_j is given to the remaining agent with the smallest reported processing time with memory of size m_j.

Best-Fit Allocation Procedure

Input: matrix of processing times $\mathbf{t} = (t_1, t_2, \ldots, t_n)$

1. $N \leftarrow [n]$
2. **for** $j = 1 \ldots n$ **do**
   (a) Let $i = \arg\min_{i \in N} t_i(m_j)$
   (b) Set $f_i^W(\mathbf{t}) = m_j$
   (c) $N \leftarrow N \setminus \{i\}$
3. **Output** $f^W = (f_1^W, ..., f_n^W)$.

We claim first, that without waiting times as transfers, this allocation rule is optimal for the makespan objective (maximum processing time over all tasks). Then we introduce waiting times w_i as payments, so that the resulting mechanism is truthful, and the waiting times do not increase the makespan, so the mechanism is both truthful and achieves optimal makespan.

Lemma 3. *The Best-Fit procedure achieves optimal makespan among all bijective allocations of the n memory chunks to the n agents.*

The proof goes by induction on n, and is based on a standard exchange-argument in a fixed optimal allocation turning it into the Best-Fit allocation while preserving optimality. In particular, if the smallest chunk is allocated to task i in Best-Fit, but to task k Opt, then exchanging the chunks between these two tasks in Opt does not increase the makespan.

Next we show that this allocation rule can be implemented by a truthful mechanism by using waiting times as payments by the agents. Given the allocation f^W, these waiting times are defined to be smallest possible (for each agent) such that in increasing order of chunk size the total costs (processing plus waiting time) of the respective agents become *increasing*.

In the code below we complement the Best-Fit Procedure to a mechanism by setting the waiting times w_i. The mechanism takes as input the matrix $\mathbf{b}$ of reported running times of the agents. Observe that c_j stands for the maximum processing time over chunks 1 to j after allocation step j. For bidder i, who gets chunk m_j, the payment in form of waiting time is $w_i = c_j - b_i(m_j)$.

Best-Fit Mechanism

Input: matrix of reported processing times $\mathbf{b} = (b_1, b_2, \ldots, b_n)$

1. $N \leftarrow [n]$
2. $(c_1, \ldots, c_n) \leftarrow (0, \ldots, 0)$

3. **for** $j = 1 \dots n$ **do**
   (a) Let $i = \arg\min_{i \in N} b_i(m_j)$
   (b) $f_i^W(\mathbf{b}) \leftarrow m_j$
   (c) $N \leftarrow N \setminus \{i\}$
   (d) $c_j \leftarrow \max\{c_{j-1}, b_i(m_j)\}$
   (e) $w_i \leftarrow c_j - b_i(m_j)$
4. **Output** $f^W = (f_1^W, ..., f_n^W)$, and $w = (w_1, w_2, \dots, w_n)$.

Note that the total cost $cost_i(\mathbf{b})$ of the agent who gets m_j, is $\max\{c_j, t_i(m_j)\} \geq \max\{c_{j-1}, t_i(m_j)\}$ (here we use that the cost is always at least the true running time), and it is exactly $c_j = \max\{c_{j-1}, t_i(m_j)\}$ if $b_i(m_j) = t_i(m_j)$.

Theorem 7. *The Best-Fit Mechanism is truthful.*

Proof. Here we provide only a sketch of the proof. For some bidder i, let $b_i \neq t_i$ be an advantageous false bid with the minimum number of indices j such that $b_i(m_j) \neq t_i(m_j)$, and let ℓ be the smallest such index.

There are two nontrivial cases to consider. First, when i receives m_ℓ by bidding $t_i(m_\ell)$ and does not receive m_ℓ when bidding $b_i(m_\ell) > t_i(m_\ell)$. This occurs when there is a bidder k with bid $b_i(m_\ell) \geq b_k(m_\ell) \geq t_i(m_\ell)$, who gets m_ℓ. The total cost of i when bidding t_i would be $\max\{c_{\ell-1}, t_i(m_\ell)\}$. With bid b_i she gets a chunk with higher index, and her cost will be at least $c_\ell = \max\{c_{\ell-1}, b_k(m_\ell)\} \geq \max\{c_{\ell-1}, t_i(m_\ell)\}$ (where c_ℓ is meant with input b_i).

Second, consider the case when i receives m_ℓ by bidding $b_i(m_\ell)$ and does not receive m_ℓ when bidding $t_i(m_\ell) > b_i(m_\ell)$. Again, there must be a bid of some agent k so that $t_i(m_\ell) \geq b_k(m_\ell) \geq b_i(m_\ell)$. Now, if agent i bids b_i, then her cost is $\max\{c_{\ell-1}, t_i(m_\ell)\}$. If she bids $t_i(m_\ell)$ instead of $b_i(m_\ell)$ then she receives a (larger) chunk m_s, for total cost of $\max\{c_{s-1}, t_i(m_s)\}$. However, it can be shown that $\max\{c_{s-1}, t_i(m_s)\} \leq \max\{c_{\ell-1}, t_i(m_\ell)\}$, implying that agent i could change her bid for chunk ℓ from $b_i(m_\ell)$ to $t_i(m_\ell)$.

Finally, we show that the waiting times used as transfers in the Best-Fit mechanism have further appealing features apart from truthfulness. First, these waiting times do not ruin the makespan-minimizing property of the mechanism; second, these waiting times correspond to the transfers that minimally implement the makespan minimizing allocation rule Best-Fit.[8]

Lemma 4. *The Best-Fit mechanism achieves minimum makespan (for any given input* $\mathbf{b}$*).*

Lemma 5. *For fixed memory chunks* $m_1, m_2, \dots, m_n$*, the payments* $w_i = c_j - b_i(m_j)$ *used in the Best-Fit mechanism correspond to the transfer functions that minimally implement the Worst-Fit allocation rule* f^W*.*

[8] For the definition of 'minimally implements', see the Preliminaries.

5.2 Mechanism with memory chunking

In this section we treat the problem of optimally chunking a given total size $m \in \mathbb{N}$ of memory into n chunks $(m_1, m_2, \ldots, m_n)$ (s.t. $\sum_j m_j = m$, and $m_j \in \mathbb{N}$), and then determining a one-to-one allocation of the chunks, with the goal of minimizing the makespan over all chunkings and all bijections $f : [n] \to \{m_1, m_2, \ldots, m_n\}$. We call such a more complex algorithm a *chunking algorithm*, which then can be implemented by a *chunking mechanism*. Unfortunately it turns out that finding the optimal chunking, and applying the Best-Fit mechanism with this given chunking does not yield a truthful chunking mechanism.

Theorem 8. *For any algorithm that takes as input the (reported) cost functions $b_i : [m] \to \mathbb{N}$, then determines an optimal (makespan minimizing) chunking $(m_1, m_2, \ldots, m_n)$, and finally outputs the optimal allocation f^W and transfers w according to the Best-Fit mechanism with input $(m_1, m_2, \ldots, m_n)$ and $\mathbf{b}$, the resulting chunking mechanism is not truthful.*

Proof. Consider the following instance with $n = 3$ tasks, and total memory size $m = 6$. Let the true cost-functions be $t_1(m') = 1$ for all $m' \geq 1$; $t_2(1) = 5$, and $t_2(m') = 3$ for $m' \geq 2$; finally, $t_3(m') = 7$ for $m' < 4$, and $t_3 = 3$ for $m' \geq 4$. The optimal makespan is 5, achieved with the memory chunking $(1, 1, 4)$. In this optimal allocation task 2 has running time $t_2(1) = 5$, and no waiting time. However, if agent 2 bids $b_2(1) = 8$, and $b_2(m') = 3$ for $m' \geq 2$, then $(2, 2, 2)$ becomes the optimal chunking with makespan 7, and task 2 has running time $\max\{t_2(2), b_2(2)\} = 3$ and no waiting time. Thus agent 2 has an incentive to report false running times, so the mechanism is not truthful.

Nevertheless, we know from Theorem 6, that for any fixed optimal allocation algorithm with memory chunking, there do exist transfers that yield a truthful mechanism. Indeed, one such chunking mechanism is the following. Let the chunking algorithm determine an optimal chunking and allocation with makespan M_{opt}. A trivial truthful mechanism charges $w_i = M_{opt} - b_i(m^i)$ to agent i who gets chunk m^i, so that the total cost of each agent i becomes $cost_i = \max\{M_{opt}, t_i(m^i)\}$. (In fact, such mechanism is optimal and truthful also in case of any fixed chunking.) There is a slightly better truthful pricing rule, charging the above prices, *except* for agents who get a memory chunk of minimum size; these agents do not have waiting times. This slight modification of the transfer function may seem to be of little use. Observe though, that charging the makespan as total cost to *every* agent is a highly unrealistic solution, because with this rule the total cost of an agent can become by an arbitrary factor higher than her running time using *any* memory size. In contrast, in the Best-Fit mechanism, the total cost c_j of a truthful agent getting chunk m_j is either her own running time, or the running time of some task getting a smaller chunk, so that agent i would have had a higher running time than c_j with that chunk. That is, for each task her total cost is within the range of running times of this task.

We define a particular chunking mechanism that finds the optimal makespan by binary search, and charges waiting times according to the above rule. Subsequently, we show that the mechanism is truthful, and that the waiting times

correspond to the minimum transfers that implement this particular allocation rule truthfully. Note however, that there might exist different *optimal* allocation rules with smaller truthful payments.

Binary-Chunking Mechanism

Input: reported functions of processing times $\mathbf{b} = (b_1, b_2, \ldots, b_n)$, where $b_i : [m] \to [T]$

1. $M \leftarrow \lfloor T/2 \rfloor$
2. do binary search for the optimum makespan M_{opt}
   (a) **for** $i = 1$ **to** n **do**
   find (with binary search) the minimum demand m^i of agent i in order to finish within M
   (b) **if** $\sum_i m^i > m$ **then** set M higher
   (c) **else** set M lower if possible, otherwise set $M_{opt} = M$
3. **for** $i = 1$ **to** n **do**
   (a) $f_i^C(\mathbf{b}) \leftarrow m^i$
   (b) **if** $m^i = 1$ **then** $w_i \leftarrow 0$
   (c) **else** $w_i \leftarrow M_{opt} - b_i(m^i)$
4. **Output** $f^C = (f_1^C, ..., f_n^C)$, and $w = (w_1, w_2, \ldots, w_n)$.

We note that the binary search for M_{opt} can also be carried out using demand queries. In this case, subsequently the $b_i(m^i)$ have to be queried as well (since $cost_i = M_{opt} - b_i(m^i) + b_i(m^i)$, there is no reason to report these non-truthfully).

Theorem 9. *The Binary-Chunking mechanism is truthful. The same holds for any chunking mechanism with an optimal chunking algorithm, and with the payments of Binary-Chunking.*

Theorem 10. *The waiting times used in the Binary-Chunking mechanism are the minimum transfers that make the allocation rule of Binary-Chunking truthful.*

6 Conclusions

We have started our research from a rather practical problem in the context of concurrent execution of memory-bound programs. Our first solutions presented here, deal with the static case where an appropriate RAM distribution has to be determined once, under the makespan objective.

From a more theoretical point of view, our work introduces an interesting new model of mechanism design wherein studying money burning objective functions is the right research challenge. In fact, we prove that *all* algorithms admit transfers that make them truthful with monitoring and therefore, also in light of the negative results in [10], this paradigm seems to be the right arena to study the optimal trade off between quality of allocation *and* transfers introduced.

We believe that our results pave the way to a number of interesting open questions, the main being the extent to which our positive results can be exported to more general models allowing repeated allocation mechanisms and/or stronger solution concepts (e.g., collusion-resistance for known coalitions). In our setting, the minimization of the sum of the *total* costs of the agents (i.e., the original utilitarian objective for money burning) needs to be explored.

References

1. V. Auletta, R. De Prisco, P. Penna, and G. Persiano. How to route and tax selfish unsplittable traffic. In *SPAA*, pages 196–205, 2004.
2. E. Ben-Porath, E. Dekel, and B.L. Lipman. Optimal allocation with costly verification. *American Economic Review*, 104(12):3779–3813, 2014.
3. L. Blumrosen and N. Nisan. On the computational power of demand queries. *SIAM J. Comput.*, 39(4):1372–1391, 2009.
4. G. Christodoulou, L. Gourvès, and F. Pascual. Scheduling selfish tasks: About the performance of truthful algorithms. In *COCOON*, pages 187–197, 2007.
5. G. Christodoulou, E. Koutsoupias, and A. Nanavati. Coordination mechanisms. In *ICALP*, pages 345–357, 2004.
6. R. Dementiev, L. Kettner, and P. Sanders. STXXL: standard template library for XXL data sets. *Softw., Pract. Exper.*, 38(6):589–637, 2008.
7. A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young. Competitive paging algorithms. *Journal of Algorithms*, 12(4):685–699, 1991.
8. D. Fotakis, P. Krysta, and C. Ventre. Combinatorial auctions without money. In *AAMAS*, pages 1029–1036, 2014.
9. D. Fotakis, P. Krysta, and C. Ventre. The power of verification for greedy mechanism design. In *AAMAS*, pages 307–315, 2015.
10. J. D. Hartline and T. Roughgarden. Optimal mechanism design and money burning. In *STOC*, pages 75–84, 2008.
11. P. Krysta and C. Ventre. Combinatorial auctions with verification are tractable. *Theoretical Computer Science*, 571:21–35, 2015.
12. U. Meyer, P. Sanders, and J. F. Sibeyn, editors. *Algorithms for Memory Hierarchies*, volume 2625 of *LNCS*. Springer, 2003.
13. N. Nisan and A. Ronen. Algorithmic Mechanism Design. *Games and Economic Behavior*, 35:166–196, 2001.
14. P. Penna and C. Ventre. Optimal collusion-resistant mechanisms with verification. *Games and Economic Behavior*, 86:491–509, 2014.
15. J.-C. Rochet. A condition for rationalizability in a quasi-linear context. *Journal of Mathematical Economics*, 16:191–200, 1987.
16. C. Ventre. Truthful optimization using mechanisms with verification. *Theoretical Computer Science*, 518:64–79, 2014.
17. J. S. Vitter. Algorithms and data structures for external memory. *Foundations and Trends in Theoretical Computer Science*, 2(4):305–474, 2006.
18. R.V. Vohra. *Mechanism Design: A Linear Programming Approach*. Cambridge University Press, 2011.