Local Computation Mechanism Design

AVINATAN HASSIDIM, Bar Ilan University
YISHAY MANSOUR and SHAI VARDI, Tel Aviv University

We introduce the notion of *local computation mechanism design*—designing game-theoretic mechanisms that run in polylogarithmic time and space. Local computation mechanisms reply to each query in polylogarithmic time and space, and the replies to different queries are consistent with the same global feasible solution. When the mechanism employs payments, the computation of the payments is also done in polylogarithmic time and space. Furthermore, the mechanism needs to maintain incentive compatibility with respect to the allocation and payments.

We present local computation mechanisms for two classical game-theoretical problems: stable matching and job scheduling. For stable matching, some of our techniques may have implications to the global (non-LCA (Local Computation Algorithm)) setting. Specifically, we show that when the men's preference lists are bounded, we can achieve an arbitrarily good approximation to the stable matching within a fixed number of iterations of the Gale-Shapley algorithm.

Categories and Subject Descriptors: J.4 [**Computer Applications**]: Economics

General Terms: Algorithms, Design, Economics

Additional Key Words and Phrases: Local computation algorithms, stable matching, mechanism design

ACM Reference Format:
Avinatan Hassidim, Yishay Mansour, and Shai Vardi. 2016. Local computation mechanism design. ACM Trans. Econom. Comput. 4, 4, Article 21 (August 2016), 24 pages.
DOI: http://dx.doi.org/10.1145/2956584

1. INTRODUCTION

Assume that we would like to design an auction for millions of buyers and items. Alternatively, there is a cloud of hundreds of thousands of computers on which we would like to schedule several millions of jobs. In the not-so-distant past, these ideas would have been unthinkable, but today, technological advances, especially the Internet, have led us to the point where they are not only possible, but necessary. One can easily conceive a cloud computation with thousands of selfish computers, each one wanting to minimize its workload. Alternatively, an ad-auction for millions of businesses competing for advertising on millions of websites does not appear to be a faraway dream. In cases like these, the datasets on which we need to work are so large, that polynomial-time tractability may not be enough. Sometimes, even computing a solution in linear time

This research was supported in part by the Google Inter-university Center for Electronic Markets and Auctions, the Israeli Centers of Research Excellence program, the Israel Science Foundation, the United States-Israel Binational Science Foundation, and the Israeli Ministry of Science.

Authors' addresses: A. Hassidim, Department of Computer Science, Bar Ilan University, Ramat Gan, 5290002, Israel; email: avinatan@cs.biu.ac.il; Y. Mansour, Blavatnik School of Computer Science, Tel Aviv University, Ramat Aviv, 6997801, Israel; email: mansour@tau.ac.il; S. Vardi, Blavatnik School of Computer Science, Tel Aviv University, Ramat Aviv, 6997801, Israel; email: shaivardi@gmail.com. Supported in part by the Google Europe Fellowship in Game Theory.

DOI: http://dx.doi.org/10.1145/2956584

may be infeasible. Often, however, only parts of the solution to a problem are required at each point in time. In such cases, we can use *local computation algorithms* (LCAs).

LCAs, which were introduced by Rubinfeld et al. [2011], consider the scenario in which we need to be able to respond to queries (regarding a feasible solution) quickly, but we never need the entire solution at once. For example, in most auctions, this is a reasonable assumption. When queried, we need to be able to tell each buyer which items she received and how much to pay; for a given item we need to tell the seller to whom and when to ship the item. There is no need to calculate the entire allocation and payment at any specific time or to commit the entire solution to memory. Having an LCA to such an auction would mean that we can reply to queries in polylogarithmic time and only require polylogarithmic space. Furthermore, if all of the items and buyers are queried, combining the results will give us a complete solution that meets our requirements.

The field of *algorithmic mechanism design* is an area at the intersection of economic game theory and algorithm design, whose objective is to design mechanisms in decentralized strategic environments. These mechanisms need to take into account both the algorithmic efficiency considerations and the selfish behavior of the participating agents.

In this article, we propose *local computation mechanism design*, which shares the motivations of both LCAs and algorithmic mechanism design. Our abstract model is the following: We have a large dataset and a set of allowable queries. Our goal is to implement each query in polylogarithmic time and space, while maintaining the incentives of participants. It is worthwhile to give a few illustrative examples:

(1) Consider the problem of assigning doctor interns to hospitals internships, the classical motivation for stable matching. We would like to be able to compute, for each doctor, her assigned hospital, without performing the entire global computation.
(2) Consider a large auction. When an item arrives from the factory to be shipped, we need to know to whom to send it and how much to charge. We never need the complete solution to the auction.
(3) Consider a large distributed data center that has to assign jobs to machines and elicits from each machine its speed. When queried on a job, we would like to reply to which machine it is assigned, and when queried regarding a machine, we would like to reply with the set of jobs that need to run on it. Again, we would like the computation to be local, without constructing a global solution, and still be able to ensure the machines have an incentive to report their speeds truthfully.

A nice property of LCAs is that they are *parallelizable*; this property immediately carries over to Local Computation Mechanisms (LCMs). Therefore, in addition to providing query access to a solution that meets both the combinatorial and game-theoretic requirements of the mechanism, an LCM can be run in parallel on a large number of machines to obtain a complete such solution.

The following are our main contributions. First, we formalize the notion of *local computation mechanism design*. A mechanism is *local* if, for every query, it calculates an allocation (and a payment) in polylogarithmic time and space. Furthermore, the allocation must be consistent with some (single) global solution, and the payment must ensure truthfulness of the agents. Second, we present local computation mechanisms for several interesting problems, where our main result is an LCA for stable matching. Third, we use our techniques to show that in the general case when the men's lists have bounded length (even in cases that do not admit an LCA), we can find arbitrarily good matchings[1] (up to both additive and multiplicative constants) by truncating the Gale-Shapley algorithm to a constant number of rounds.

[1] See Section 3 for a formal discussion.

We provide LCAs for the following problems:

Stable Matching. In the *stable matching* (or *stable marriage*) problem, introduced by Gale and Shapley [1962], we would like to find a *stable* perfect matching[2] between a group of n men and a group of n women. We focus on the model introduced by Immorlica and Mahdian [2005], in which the women can have arbitrary preferences over the men, and the men have preference lists of length k over the women, sampled uniformly at random.

Our main result is a local computation algorithm that matches all but an arbitrarily small fraction of the participants (this is often called an *almost* stable matching; see, for example, Eriksson and Häggström [2008] and Lu and Zheng [2003]). Furthermore, limited to the matched participants, the matching is stable.

Scheduling on Related Machines. In the *makespan minimization* problem, we want to schedule n jobs on m machines so as to minimize the maximal running time (makespan) of the machines. This problem has many variations; we consider the scenario in which m identical jobs need to be allocated among n related machines. The machines are strategic agents, whose private information is their speed. We show the following:

(1) A local mechanism that is truthful in expectation for scheduling on related machines, that provides an $O(\log\log n)$-approximation to the optimal makespan.
(2) A local mechanism that is universally truthful for the restricted case (i.e., when each job can run on one of at most a constant number of predetermined machines), that provides an $O(\log\log n)$-approximation to the optimal makespan.

We also show some subtle and surprising results on the truthfulness of our algorithms.

1.1. Related Work

LCAs. Rubinfeld et al. [2011] showed how to transform distributed algorithms to LCAs, and gave LCAs for several problems, including maximal independent set and hypergraph 2-coloring. Alon et al. [2012] expanded the work of Rubinfeld et al. [2011] and gave better space bounds for maximal independent set and hypergraph 2-coloring, using *query trees*. Query trees were introduced in the local setting by Nguyen and Onak [2008]: a random permutation of the vertices is generated, and a sequential algorithm is simulated on this order. The query tree represents the dependence of each query on the results of previous queries. Nguyen and Onak [2008] showed that if the graph has a bounded degree, the query tree has a constant expected size. Alon et al. [2012] showed that the query tree has polylogarithmic size with high probability, and that the space required by the algorithm can be reduced by using a random seed to generate the ordering. Mansour et al. [2012], showed that the size of the query tree can be bounded, with high probability, by $O(\log n)$, and showed how it is possible to transform many online algorithms to LCAs. Using this technique, they showed LCAs for maximal matching and several machine scheduling problems. Reingold and Vardi [2016] extended these results to a wider family of graphs, and obtained better time and space bounds. Mansour and Vardi [2013] showed an LCA that finds a $(1-\epsilon)$-approximation to the maximum matching.

Mechanism Design. Because we look at two very different game-theoretic settings, we provide a short subsection dedicated to related work pertaining to each topic at the start of the relevant sections.

[2] A stable perfect matching is a perfect matching with no blocking pairs. A blocking pair is a man m and a woman w such that m prefers w to the woman he is matched to, and w prefers m to the man she is matched to.

2. MODEL AND PRELIMINARIES

We assume the standard uniform-cost RAM model, in which the word size is $O(\log n)$ bits, where n is the input size, and it takes $O(1)$ to read and perform simple word operations. We denote the set of integers $\{1, 2, \ldots, n\}$ by $[n]$.

2.1. LCAs

We use the following model of LCAs. A $(t(n), s(n), \delta(n))$-*local computation algorithm* $\mathcal{A}$ for a computational problem is a (possibly randomized) algorithm that receives an input of size n, and a query x. Algorithm $\mathcal{A}$ replies to query x in time $t(n)$ and uses at most $s(n)$ memory, with probability at least $1 - \delta(n)$. Furthermore, the replies to all of the possible queries are consistent with a single feasible solution to the problem. That is, the algorithm *always* replies correctly, but there is a $\delta(n)$ probability that the time and/or space bounds will be violated.

Remark 2.1. The model we use is a generalization of the model introduced by Rubinfeld et al. [2011]. Our model differs from theirs in that their model requires that the LCA *always* obeys the time and space bounds, and returns an error with some probability. It is easy to see that any algorithm that conforms to our model can be modified to conform to the model of Rubinfeld et al. [2011] by forcing it to return an error if the time or space bound is violated (the other direction does not necessarily hold). Note, however, that using this translation, a truthful mechanism in our model would not necessarily translate to a truthful mechanism in their model.

2.2. Mechanism Design

We use the standard notation of game-theoretic mechanisms. There is a set I of n rational agents and a set J of m items. In some settings, for example, the stable marriage setting, there are no objects, only rational agents. Each agent $i \in I$ has a valuation function v_i that maps subsets $S \subseteq J$ of the items to nonnegative numbers. The utilities of the agents are quasilinear, namely, when agent i receives subset S of items and pays p, her utility is $u_i(S, p) = v_i(S) - p$. Agents are rational in the sense that they select actions to maximize their utility. We would like to allocate items to agents (or possibly agents to other agents), in order to meet global goal, for example, maximize the sum of the valuations of allocated objects (see, e.g., Nisan et al. [2005]).

A *mechanism with payments* $\mathcal{M} = (\mathcal{A}, \mathcal{P})$ is composed of an allocation function $\mathcal{A}$, which allocates items to agents, and a payment scheme $\mathcal{P}$, which assigns each agent a payment. A mechanism without payments consists only of an allocation function. Agents report their bids to the mechanism. Given the bids $b = (b_1, \ldots, b_n)$, the mechanism allocates the item subset $\mathcal{A}_i(b) \subseteq J$ to agent i, and, if the mechanism is with payments, charges her $\mathcal{P}_i(b)$; the utility of agent i is $u_i(b) = v_i(\mathcal{A}_i(b)) - \mathcal{P}_i(b)$.

A randomized mechanism is *universally truthful* if for every agent i, for every random choice of the mechanism, reporting her true private valuation maximizes her utility. A randomized mechanism is *truthful in expectation*, if for every agent i, reporting her true private valuation maximizes her expected utility. That is, for all agents i, any bids b_{-i} and b_i, $\mathbb{E}[u_i(v_i, b_{-i})] \geq \mathbb{E}[u_i(b_i, b_{-i})]$.

We say that an allocation function $\mathcal{A}$ *admits* a truthful payment scheme if there exists a payment scheme $\mathcal{P}$ such that the mechanism $\mathcal{M} = (\mathcal{A}, \mathcal{P})$ is truthful.

A mechanism $\mathcal{M} = (\mathcal{A}, \mathcal{P})$ fulfills *voluntary participation* if, when an agent bids truthfully, her utility is always on-negative, regardless of the other agents' bids, that is, for all agents i and bids b_{-i}, $u_i(v_i, b_{-i}) \geq 0$.

2.3. Local Computation Mechanisms

Definition 2.2 (*Mechanisms Without Payments*). We say that a mechanism $\mathcal{M}$ is $(t(n), s(n), \delta(n))$-*local* if its allocation function is computed by a $(t(n), s(n), \delta(n))$-local computation algorithm.

Definition 2.3 (*Mechanisms With Payments*). We say that a mechanism $\mathcal{M} = (\mathcal{A}, \mathcal{P})$ is $(t(n), s(n), \delta(n))$-*local* if both the allocation function $\mathcal{A}$ and the payment scheme $\mathcal{P}$ are computed by $(t(n), s(n), \delta(n))$-local computation algorithms.

In other words, given a query x, $\mathcal{A}$ computes an allocation and $\mathcal{P}$ computes a payment, and both run in time $t(n)$ and space $s(n)$ with probability at least $1-\delta(n)$. Furthermore, the replies of $\mathcal{A}$ to all of the queries are consistent with a single feasible allocation.

A *truthful local mechanism* $\mathcal{M} = (\mathcal{A}, \mathcal{P})$ is a local mechanism that is also truthful. Namely, each agent's dominant bid is her true valuation, regardless of the fact that the mechanism is local.

3. STABLE MATCHING

In the *stable matching problem*, we are given a set of men and a set of women. The men have preferences over the women and the women over the men. The goal is to compute a matching H that is stable; that is, there is no man and woman who prefer each other to their partner in H. We formalize this next, following a summary of related work.

3.1. Related Work

Stable matching has been at the center of game-theoretic research since the seminal paper of Gale and Shapley [1962] (see, e.g., Roth [2003] for an introduction and a summary of many important results). Roth and Rothblum [1999] examined the scenario in which the preference lists are of bounded length; in most real-life scenarios, this is indeed the case. For example, a medical student will not submit a preference list for internship over all of the hospitals in the United States, but only a short list. We examine the variant in which each man $m \in M$ is interested in at most k women (and prefers to be unmatched than to be matched to anyone not on their list; cf. Roth and Peranson [1999]). We limit our attention to the setting in which the men's preferences are assumed to be uniformly distributed (cf. Immorlica and Mahdian [2005] and Kojima and Pathak [2009]).

The Gale-Shapley algorithm results in a stable matching regardless of the preferences (see, e.g., Roth and Sotomayor [1990]); however, its running time is $\Omega(n^2)$. Indeed, this is a lower bound on any algorithm that finds a stable matching (under full preference lists) [Ng and Hirschberg 1990]. Furthermore, it is known that a linear number of iterations of the Gale-Shapley algorithm is necessary to attain stability [Gusfield and Irving 1989]. One direction taken to obtain sublinear running time is executing parallel computation on instances with short preference lists. Feder et al. [2000] proposed one such algorithm for stable matching. Unfortunately, it does not appear possible to convert their algorithm to an LCA, as they require m^4 processors, and the running time is $O(\sqrt{m}\log^3 n)$, where m is the sum of the preference list lengths. In some cases, matchings that are "almost" stable may be acceptable (see, e.g., Eriksson and Häggström [2008] and Roth and Xing [1997]). There are several ways of defining what it means for a matching to be *almost stable* (see, e.g., Eriksson and Häggström [2008]). One of the better accepted notions (e.g., Floréen et al. [2010], Ostrovsky and Rosenbaum [2014], and Roth and Xing [1997]) is to count the number of blocking pairs[3]—the fewer the

[3] A *blocking pair* is a man and a woman who both prefer to be paired with each other than with their current partner.

blocking pairs, the more stable the matching. Several experimental works on parallel algorithms for the stable matching problem provide evidence that after a constant number of rounds, the number of blocking pairs can be made arbitrarily small (e.g., Tseng and Lee [1984], Quinn [1985], and Lu and Zheng [2003]). Floréen et al. [2010] showed that in the special case when the lengths of both the men's and women's preference lists are bounded by a constant, there exists a distributed version of the Gale-Shapley algorithm, which can be run for a constant number of rounds and finds an almost stable matching.

The Gale-Shapley algorithm is known to be strategy-proof for the men but not for the women (e.g., Maschler et al. [2013]). Immorlica and Mahdian [2005] showed that in the previous setting (and also for a more general setting), the expected number of people with more than one stable spouse is vanishingly small, and with probability $1 - o(1)$, truth-telling is a dominant strategy if the other players are truthful.

3.2. Model and Main Result

We use a graph-theoretic characterization of the *stable matching* problem (see, e.g., Fleiner [2003] and Floréen et al. [2010]). An instance of the stable marriage problem is represented by a bipartite graph $G = (M \cup W, E)$, where M represents the set of men, and W the set of women. We make the conventional assumption that $|M| = |W| = n$. Each man has a preference list over the women that he is connected to, and each woman has a preference list over the men she is connected to. A matching $H \subseteq E$ is a set of vertex-disjoint edges. An edge e is said to be *matched* if $e \in H$. A vertex v is matched if there is some u such that $e = (u, v)$ is matched. An edge $(u, v) \in E \backslash H$ is *unstable* if it holds that (1) u is unmatched or prefers v over its match in H, *and* (2) v is unmatched or prefers u over its match in H. (An unstable edge is often referred to as a *blocking pair*.) A matching H is *stable* if there are no unstable edges. The stable matching problem where each man has a degree of k and his adjacent edges are chosen uniformly at random is called *k-uniform*. Note that in this case, the women's preference list lengths are binomial random variables whose value is determined by the random choices of the men, and can therefore have any length in $[n]$.

The Gale-Shapley algorithm finds a stable matching in the k-uniform setting (e.g., Gale and Sotomayor [1985]). To ensure the locality of our algorithm, we allow our mechanism to find an almost stable matching. To do this, we allow our mechanism to "disqualify" men, in which case they remain unmatched, but are unable to contest the matching (the number of disqualified men is exactly the number of blocking pairs). We try to keep the number of disqualified men to a minimum. Our main result is the following.

THEOREM 3.1. *Let $A = (M, W, P)$ be a stable matching problem, $|M| = |W| = n$, in the k-uniform setting. For any $\epsilon > 0$, there is an $(O(\log n), O(\log n), 1/n)$-local computation mechanism for A that finds a matching with at most ϵn disqualified men and in which at most $\frac{2n}{k} + \epsilon n$ of the men remain unmatched.*

We begin by describing a nonlocal algorithm, ABRIDGEDGS, and then show how to simulate it locally by a local algorithm, LOCALAGS.

3.3. ABRIDGEDGS

Let ABRIDGEDGS be the Gale-Shapley men's courtship algorithm, where the algorithm is stopped after ℓ rounds, and the men rejected on that round are left unmatched. That is, in each round, each unassigned man approaches to the highest ranked woman that has not (yet) rejected him. Each woman then tentatively accepts the man she prefers out of the men who approached her, and rejects the rest. This continues until the ℓ^{th}

round, and the men who were rejected on the ℓ^{th} round are left unmatched; we say that these men are *disqualified*. Note that the set of disqualified men may be a strict subset of the set of unmatched men: men who were rejected k times before the ℓ^{th} round are unmatched as well. We simulate AbridgedGS on k-uniform stable matching problems to obtain the following LCA.

3.4. LocalAGS—An LCA Implementation of AbridgedGS

Define the *distance* between two people to be the length of the shortest path between them in the graph. Define the d-neighborhood of a person v to be everyone at a distance at most d from v, denoted $N_d(v)$

Assume that we are queried on a specific man, m_1. We simulate AbridgedGS locally as follows: Choose some constant ℓ, whose exact value will be determined later. For each man in the 2ℓ-neighborhood of m_1 (i.e., for all m_i such that $m_i \in N_{2\ell}(m_1)$), we simulate round 1 of AbridgedGS. That is, each one approaches his preferred woman, and is either tentatively accepted or rejected. Then, for each man $m_i \in N_{2\ell-2}(m_1)$, we simulate round 2. And so on, until for $m_i \in N_2(m_1)$ (i.e., m_1 and his closest male neighbors), we simulate round ℓ. We return the woman to whom m_1 is paired, “unassigned” if he was rejected by k women, and “disqualified” if he was rejected by a woman in round ℓ. We denote this algorithm LocalAGS.

In order to prove Theorem 3.1, we need to prove several things: that LocalAGS correctly simulates AbridgedGS (Section 3.5); that its running time and space are bounded by $O(\log n)$ (Section 3.6); and that “not too many” men are left unmatched or disqualified (Section 3.7).

3.5. Correctness of LocalAGS

The following claim shows that the steps executed by LocalAGS are sufficient to correctly determine the output of AbridgedGS when queried on m_1.

Claim 3.2. *For any two men, m_i and m_j, whose distance from each other is greater than 2ℓ, m_i's actions cannot affect m_j if Algorithm* AbridgedGS *terminates after ℓ rounds.*

Proof. The proof is by induction. For $\ell = 1$, let w_1 be m_j's first choice. Only men for whom w_1 is their first choice can affect m_j, and these are a subset of the men at distance 2 from m_j. For the inductive step, assume that the claim holds for $\ell - 1$. Assume by contradiction that there is a man m_i whose actions can affect m_j within ℓ rounds, who is at a distance of at least $2\ell + 2$ from m_j. From the inductive claim, none of m_i's actions can affect any of m_j's neighbors within $\ell - 1$ rounds. As their actions in round $\ell - 1$ (or any previous round) will not be affected by m_i, and they are the only ones who can affect m_j in round ℓ, it follows that m_i cannot affect m_j within ℓ rounds. □

3.6. Space and Time Bounds of LocalAGS

The following lemma bounds the running time and space of LocalAGS.

Lemma 3.3. *The running time and space of algorithm* LocalAGS *is $O(\log n)$ per query with probability at least $1 - \frac{1}{n^2}$.*

Because LocalAGS simulates AbridgedGS for a constant number of rounds, the running time and space required per query by LocalAGS is at most the number of rounds multiplied by the size of the neighborhood on which we simulate AbridgedGS. The following claim therefore implies Lemma 3.3:

Recall that $N_i(v)$ is the set of people at distance at most i from v.

CLAIM 3.4. *For sufficiently large n, for any integer $i > 0$, there exists a constant c_i such that* $\Pr[|N_i(v)| \leq c_i \log n] \geq 1 - \frac{1}{n^2}$.

PROOF. Let $\mathcal{N}_v^i$ be the random variable representing the number of vertices in the i-neighborhood of vertex v. As the degree of each woman v is distributed binomially, $\mathcal{N}_v^1 \sim B(n, k/n)$, it holds that $\mathbb{E}[\mathcal{N}_v^1] = k$. We prove by induction that $\Pr[\mathcal{N}_v^i \leq c_i(\log n)] \geq 1 - \frac{i}{n^3}$, where c_i is a constant that depends only on k and i.

For the base, $i = 1$, if v is a man, $\mathcal{N}_v^1 = k$. If v is a woman, we employ the Chernoff bound with $\lambda > 2e - 1$:[4] $\Pr[\mathcal{N}_v^1 > (1+\lambda)k] < 2^{-k\lambda}$. Therefore, for $c_1 = 4$ and $n \geq 2^k$,

$$\Pr[\mathcal{N}_v^1 > c_1 \log n] \leq 2^{-c_1 \log n + k} < \frac{2^k}{n^{c_1}} \leq \frac{1}{n^3}.$$

Assuming that the claim holds for all integers smaller than i, we show that it holds for i. If the outermost vertices of the neighborhood are men, then $\mathcal{N}_v^i \leq k\mathcal{N}_v^{i-1}$ and we can take $c_i = kc_{i-1}$. Otherwise, we use the law of total probability.

$$\begin{aligned}\Pr\left[\mathcal{N}_v^i > c_i \log n\right] =& \Pr\left[\mathcal{N}_v^i > c_i \log n | \mathcal{N}_v^{i-1} \leq c_{i-1} \log n\right] \Pr\left[\mathcal{N}_v^{i-1} \leq c_{i-1} \log n\right] \\ &+ \Pr\left[\mathcal{N}_v^i > c_i \log n | \mathcal{N}_v^{i-1} > c_{i-1} \log n\right] \Pr\left[\mathcal{N}_v^{i-1} > c_{i-1} \log n\right] \\ \leq& \Pr\left[\mathcal{N}_v^i > c_i \log n | \mathcal{N}_v^{i-1} \leq c_{i-1} \log n\right] + \Pr\left[\mathcal{N}_v^{i-1} > c_{i-1} \log n\right] \\ \leq& \Pr\left[\mathcal{N}_v^i > c_i \log n | \mathcal{N}_v^{i-1} \leq c_{i-1} \log n\right] + \frac{i-1}{n^3},\end{aligned}$$

where the last inequality uses the inductive hypothesis. It remains to bound $\Pr[\mathcal{N}_v^i > c_i \log n | \mathcal{N}_v^{i-1} \leq c_{i-1} \log n]$.

The probability that the degree of any woman u is exactly z is at most

$$\Pr[deg(u) = z] \leq \binom{n}{z}\left(\frac{k}{n}\right)^z \leq \left(\frac{ek}{z}\right)^z,$$

using the inequality $\binom{n}{i} \leq \left(\frac{ne}{i}\right)^i$. Hence, for $z \geq e^2k$ we have that $\Pr[deg(u) = z] \leq e^{-z}$. Because for any z, it holds that $\Pr[deg(u) = z] \leq 1 = e^0$, for arbitrary $z \geq 0$, it holds that $\Pr[deg(u) = z] \leq e^{-\widehat{z}}$, where $\widehat{z} = \max\{0, z - e^2k\}$.

We would like to bound the probability that $\mathcal{N}_v^i$ is larger than $c_i \log n$ when $\mathcal{N}_v^{i-1}$ is less than $c_{i-1} \log n$. We define a new random variable $\widehat{\mathcal{N}_v^i}$ as follows. Let $y \leq c_{i-1} \log n$ be the number of nodes at distance $i - 1$ from v and let $z = (z_1, z_2, \ldots, z_y)$ be their degrees. We define the truncated degrees as $\widehat{z} = \{\widehat{z}_1, \widehat{z}_2, \ldots, \widehat{z}_y\}$ such that $\widehat{z}_j = \max\{0, z_j - e^2k\}$ (informally, we ignore the first e^2k neighbors of each vertex). The value of $\widehat{\mathcal{N}_v^i}$ is the sum of the truncated degrees at distance $i - 1$ from v, that is, $\widehat{\mathcal{N}_v^i} = \sum_{i=1}^{y} \widehat{z}_i$. Clearly $\mathcal{N}_v^i \leq \widehat{\mathcal{N}_v^i} + e^2ky \leq \widehat{\mathcal{N}_v^i} + c_{i-1}e^2k \log n$. Therefore, it is sufficient to bound $\widehat{\mathcal{N}_v^i}$.

Let $\widehat{x} = \sum_{i=1}^{y} \widehat{z}_i$. The probability that the truncated degrees of the vertices at distance $i - 1$ are exactly $\widehat{z} = (\widehat{z}_1, \widehat{z}_2, \ldots, \widehat{z}_y)$ is at most $\prod_{i=1}^{y} e^{-\widehat{z}_i} = e^{-\widehat{x}}$. There are $\binom{\widehat{x}+y}{y}$ vectors $\widehat{z}$ that can realize $\widehat{x}$ (the number of ways to partition $\widehat{z}$ into y groups). We bound

[4] Substituting $\lambda \geq 2e - 1$ into the standard Chernoff bound $\Pr[X > (1+\lambda)\mu] \leq (\frac{e^\lambda}{(1+\lambda)^{1+\lambda}})^\mu$ gives this bound.

$\Pr[\widehat{\mathcal{N}}_v^i = \widehat{x} | \mathcal{N}_v^{i-1} \leq y]$, for $\widehat{x} \geq 7y$ as follows:

$$\begin{aligned}
\Pr[\widehat{\mathcal{N}}_v^i &= \widehat{x} | \mathcal{N}_v^{i-1} \leq y] \\
&\leq \binom{\widehat{x} + y}{y} e^{-\widehat{x}} \\
&\leq \left(\frac{e \cdot (\widehat{x} + \widehat{x}/7)}{\widehat{x}/7} \right)^{\widehat{x}/7} e^{-\widehat{x}} \\
&= e^{-(1-(1+\ln(8))/7)\widehat{x}} \\
&\leq e^{-\widehat{x}/2}.
\end{aligned}$$

It follows that

$$\Pr\left[\widehat{\mathcal{N}}_v^i \geq 7y | \mathcal{N}_v^{i-1} \leq y\right] \leq \sum_{\widehat{x}=7y}^{\infty} e^{-\widehat{x}/2} = \frac{e^{-7y/2}}{1 - e^{-1/2}} \leq e^{-y} \leq 1/n^3,$$

which follows since $y \leq c_{i-1} \log n$ and $c_{i-1} \geq 3$. Therefore, for $c_i = (e^2 k + 7)c_{i-1} \leq (16k)^i$, we have

$$\Pr\left[\mathcal{N}_v^i > c_i \log n\right] \leq \frac{1}{n^3} + \frac{i-1}{n^3} = \frac{i}{n^3}. \quad \square$$

Claim 3.4 implies that Algorithm LocalAGS makes $O(\log n)$ queries with probability at least $\frac{1}{n^2}$; Lemma 3.3 follows.

3.7. Bounding the Number of Men Removed

In this section, we prove that "not too many" men remain unmatched. There are two possible reasons for a man to be unmatched by LocalAGS: (1) he had already been rejected k times by round ℓ (hence he never reaches round ℓ), or (2) he was rejected (and hence disqualified) on round ℓ. We up the probability of both (Lemma 3.6 and Corollary 3.11, respectively), and apply a union bound, to obtain the following result.

Lemma 3.5. *For any $\epsilon > 0$, setting $\ell = \frac{2k}{\epsilon}$ in Algorithm* LocalAGS *ensures that at most $\frac{2n}{k} + \epsilon n$ men remain unmatched with probability at least $1 - \frac{1}{n^2}$.*

3.7.1. Removal Due to Short Lists. We bound the number of unassigned women as a result of the fact that the lists are short, noting that the number of unassigned women equals the number of unassigned men. This is given by the following lemma.

Lemma 3.6. *In the k-uniform setting, the Gale-Shapley algorithm results in at most $\frac{2n}{k}$ men being unassigned, with probability at least $1 - \frac{1}{n^2}$.*

Before proving Lemma 3.6, we will require a few preliminaries. We use the principle of deferred decisions: instead of "deciding" on the preference lists in advance, each man chooses the $(i+1)^{th}$ woman on his list only if he is rejected by the i^{th}—this is known to be equivalent to the choices being made in advance (e.g., Knuth [1976]).

Consider the following stochastic process: In each round t, the (randomized) assignment function f^t is given the matching of the previous round, H^{t-1}, and assigns each man $m \in M$ a woman $w \in W$, such that if m was matched in H^{t-1}, he is assigned the same woman (i.e., if $(m, w) \in H^{t-1}$, then $f^t(m) = w$); if he is unmatched in H^{t-1}, he is assigned a woman uniformly at random. Let $S^t(w)$ be the set of men assigned woman w by f^t, that is, $S^t(w) = \{m : f^t(m) = w\}$. For every woman w such that $S^t(w) \neq \emptyset$, a single $m \in S^t(w)$ is chosen arbitrarily to be w's match in H^t, that is, $(w, m) \in H^t$. The process is initialized with $H^0 = \emptyset$, and iterates for k rounds.

Remark 3.7. Note that a man can choose the same woman more than once. Compare this to the case each man can approach each woman once: If a man approaches a woman he had already approached, she must be matched, and hence in this case, the men have a lower probability of approaching an unassigned woman. Therefore, the number of women that are unassigned at the end of this process is an upper bound to the number of unassigned women in the system where men can only approach each woman once.

Let X_j^t be the indicator variable, which is 1 if woman j is unassigned after round t, that is, there does not exist a man $m \in M$ such that $(m, w) \in H^t$. Let $X^t = \sum_{j=1}^n X_j^t$ be the number of unassigned women after round t.

Proof of Lemma 3.6. As the stochastic process described previously ends at least as early as the Gale-Shapley algorithm with short lists in the k-uniform setting (from Remark 3.7 and the fact that it may be stopped prematurely), it suffices to prove that for any constant t,

$$\Pr\left[X^t > \frac{2n}{t}\right] \le \frac{t}{n^3}.$$

The proof is by induction. The base of the induction, $t = 1$, is immediate. For the inductive step, assume that after round t, $X^t = n/\mu$ (for some $\mu > 0$). In round $t+1$, $\mathbb{E}[X^{t+1}|X^t = \frac{n}{\mu}] = \frac{n}{\mu}(1-1/n)^{n/\mu}$, because each unassigned man approaches any woman with probability $1/n$; hence, the probability that a specific woman is not approached by any man is $(1-1/n)^{n/\mu}$.

For the rest of the proof, assume that $X^t \le \frac{2n}{t}$, and fix X^t to be some such value. We get

$$\mathbb{E}\left[X^{t+1}|X^t \le \frac{2n}{t}\right] \le \frac{2n}{t}(1-1/n)^{2n/t} < \frac{n}{t/2 \cdot e^{2/t}} < \frac{2n}{t+2}, \tag{1}$$

using $e^x > 1 + x$.

It remains to show that X^{t+1} is concentrated around its mean. To do so, we will define a specific martingale and use the following version of Azuma's inequality (see Alon and Spencer [2008]).

Lemma 3.8 [Azuma's Inequality]. *Let* $c = Y_0, \ldots, Y_n$ *be a martingale with* $|Y_{i+1} - Y_i| \le 1$ *for all* $0 \le i \le n$. *Then*

$$\Pr[|Y_n - c| > \lambda\sqrt{n}] < 2e^{-\lambda^2/2}.$$

Order the men arbitrarily, $M = \{1, 2, \ldots n\}$. Let M_i be the set of the first i men in the ordering: $M_i = \{1, 2, \ldots i\}$. Fix some matching H^t. For some realization g of the assignment function f^{t+1}, define the following martingale:

$$Y_i^{t+1}(H^t, g) = \mathbb{E}[X^{t+1}|H^t,\ f^{t+1}(j) = g(j) \text{ for all } j \in M_i].$$

In other words, $Y_i^{t+1}(H^t, g)$ is the expected number of unassigned women at round $t+1$, given that the matching at round t was H^t, where the expectation is taken over all realizations of f^{t+1} that agree with g on the first i men. Note that $Y_0^{t+1}(H^t, g)$ is the expected value of X^{t+1} over all possible realizations of f^{t+1}; that is, the expected number of unmatched women after $t+1$ rounds. $Y_n^{t+1}(H^t, g)$ is simply the number of unmatched women after $t+1$ rounds when the allocation function is g. X^{t+1} satisfies the Lipschitz condition, because if two realizations of f^{t+1}, say f' and f'', only differ on the allocation of a single man, $|X^{t+1}|f' - X^{t+1}|f''| \le 1$ (where $X^{t+1}|f$ denotes the realization of X^{t+1} given that f is the realization of f^{t+1}). Therefore (see Alon and

Spencer [2008]),

$$\left|Y_{i+1}^{t+1}(H^t, g) - Y_i^{t+1}(H^t, g)\right| \le 1.$$

We can therefore apply Lemma 3.8 (Azuma's inequality):

$$\Pr[|X^{t+1} - \mathbb{E}[X^{t+1}]| > \lambda\sqrt{n}] < 2e^{-\lambda^2/2}.$$

Setting $\lambda = \frac{2\sqrt{n}}{(t+1)(t+2)}$, we have that

$$\Pr\left[|X^{t+1} - \mathbb{E}[X^{t+1}]| > \frac{2n}{(t+1)(t+2)}\right] < 2e^{-n/(5t^4)},$$

for $t \ge 2$.

Therefore, since we assume that $X^t \le \frac{2n}{t}$ and hence, by Equation (1), $\mathbb{E}[X^{t+1}] < \frac{2n}{t+2}$, it holds that

$$\Pr\left[X^{t+1} > \frac{2n}{t+1} | X^t \le \frac{2n}{t}\right] < 2e^{-n/(5t^4)} < \frac{1}{n^3}. \tag{2}$$

By the inductive hypothesis

$$\Pr\left[X^t > \frac{2n}{t}\right] \le \frac{t}{n^3}. \tag{3}$$

Therefore, using Equations (2), and (3), we have

$$\begin{aligned}\Pr\left[X^{t+1} > \frac{2n}{t+1}\right] &= \Pr\left[X^{t+1} > \frac{2n}{t+1} | X^t \le \frac{2n}{t}\right]\Pr\left[X^t \le \frac{2n}{t}\right] \\ &\quad + \Pr\left[X^{t+1} > \frac{2n}{t+1} | X^t > \frac{2n}{t}\right]\Pr\left[X^t > \frac{2n}{t}\right] \\ &\le \frac{t}{n^3} + \frac{1}{n^3} \\ &= \frac{t+1}{n^3}. \quad \square\end{aligned}$$

3.7.2. Removal Due to the Number of Rounds Being Limited. Because we stop the LocalAGS algorithm after a constant (ℓ) number of rounds, it is possible that some men who "should have been" matched are disqualified because they were rejected by their i^{th} choice in round ℓ ($i < k$). We show that this number cannot be very large.

Let R_r denote the number of men rejected in round $r \ge 1$.

Observation 3.9. *R_r is monotone decreasing in r.*

Lemma 3.10. *The number of men rejected in round r is at most $\frac{nk}{r}$.*

Proof. As each man can be rejected at most k times, the total number of rejections possible is kn. The number of men who can be rejected in round r is at most

$$\begin{aligned}R_r &\le kn - \sum_{j=1}^{r-1} R_j \\ \Rightarrow R_r &\le kn - (r-1)R_r \quad (4)\\ \Rightarrow R_r &\le n\frac{k}{r},\end{aligned}$$

where inequality (4) is due to monotonicity of R_r, that is, Observation 3.9. □

Corollary 3.11. *For any $\epsilon > 0$, setting $r = \frac{k}{\epsilon}$ ensures that the number of men rejected in round r is at most ϵn.*

4. SOME GENERAL PROPERTIES OF THE GALE-SHAPLEY ALGORITHM

We use the results and ideas of Section 3 to prove some interesting features of the (general) Gale-Shapley stable matching algorithm, when the mens' lists are of length at most k. (These results immediately extend to our local version of the algorithm, LocalAGS.) Note that the proof of Lemma 3.10 makes no assumption on how the men's selection is made, and therefore, Lemma 3.10 implies that as long as each man's list is bounded by k, if we run the Gale-Shapley for ℓ rounds, at most $\frac{nk}{\ell}$ men will be rejected in that round. This immediately gives us an additive approximation bound for the algorithm if we stop after ℓ rounds:

Corollary 4.1 (to Lemma 3.10). *Assume that the output of the Gale-Shapley algorithm on a stable matching problem, where the preference lists of the men are of length at most k, is a matching of size M^*. Then, stopping the Gale-Shapley algorithm after ℓ rounds will result in a matching of size at least $M^* - \frac{nk}{\ell}$.*

We would like to also provide a multiplicative bound. Again, we assume that the mens' list length is bounded by k, but make no other assumptions. For each round i, let M_i be the size of the current matching; let D_i be the number of men who have already approached all k women on their list and have been rejected by all of them; let C_i be the number of men who were rejected by women in round i, but have approached fewer than k women so far; as before, let R_i be the number of men rejected in round i. Denote the size of the matching returned by the Gale-Shapley algorithm (if it were to run to completion) by M^*.

Claim 4.2. $C_{k+1} \le kM^*$.

Proof. Note that $R_i = C_i + D_i - D_{i-1}$. For $i < k$, $D_i = 0$. As M_i is monotonically increasing in i, $\forall i \le k$, $R_i \ge n - M^*$.

$$\sum_{i=1}^{k} R_i \ge kn - kM^*.$$

Hence,

$$C_{k+1} \le kn - \sum_{i=1}^{k} R_i \le kM^*. \quad \square$$

Corollary 4.3. *For every $\epsilon > 0$, there exists a constant $\ell > 0$ such that $C_\ell \le \epsilon M^*$.*

Proof. Denote the maximum number of total rejections possible from round i onwards by L_i. Clearly,

$$L_i \le k(M_i + C_i) \le k(M^* + C_i).$$

For all i such that $C_i \ge \epsilon M^*$, we have

$$L_i \le \left(1 + \frac{1}{\epsilon}\right) kC_i.$$

Therefore, from Claim 4.2,

$$L_{k+1} \le \left(1 + \frac{1}{\epsilon}\right) k^2 M^*.$$

Putting everything together, we have,

$$\begin{aligned} L_{i+1} &\le L_i - C_i \\ \Rightarrow L_{i+1} &\le L_i\left(1 - \frac{1}{k(1+\frac{1}{\epsilon})}\right) \\ \Rightarrow L_{k+i+1} &\le L_{k+1}\left(1 - \frac{1}{k(1+\frac{1}{\epsilon})}\right)^i \\ &\le \left(1+\frac{1}{\epsilon}\right)k^2 M^* \left(1 - \frac{1}{k(1+\frac{1}{\epsilon})}\right)^i \\ &\le 2k^2 M^* e^{-\frac{i}{k(1+\frac{1}{\epsilon})}}. \end{aligned}$$

Taking $i = k(1+\frac{1}{\epsilon})\log\frac{2k^2}{\epsilon}$ gives $C_{k+i+1} \le L_{k+i+1} \le \epsilon M^*$. □

This gives us

THEOREM 4.4. *Consider a stable matching problem. Let the length of each man's list be bounded by k. Denote the size of the stable matching returned by the Gale-Shapley algorithm by M^*. Then, if the process is stopped after $O(\frac{k}{\epsilon}\log\frac{k}{\epsilon})$ rounds, the matching returned is at most a $(1+\epsilon)$-approximation to M^*, and has at most ϵM^* unstable couples.*

As a corollary to Theorem 4.4, when the men's and women's list lengths are both bounded by a constant, there is an LCA that runs in constant time and provides a matching with at most ϵ unstable edges that is a $(1+\epsilon)$ approximation to the matching returned by the Gale-Shapley algorithm.

COROLLARY 4.5. *If both men and women have lists of length at most k, then for any ϵ there is an $(O(1), O(1), 0)$-LCA for stable matching which returns a matching that is at most a $(1+\epsilon)$-approximation to the matching returned by the Gale-Shapley algorithm, and with at most an ϵ-fraction of the edges being unstable.*

5. LOCAL MACHINE SCHEDULING

5.1. Introduction and Related Work

Consider the following job scheduling problem. n identical jobs arrive online and need to be allocated to m identical machines, with the objective of minimizing the makespan—the maximal load on any machine. Azar et al. [1999] proposed the following algorithm: each job chooses, uniformly at random, d machines, and allocates itself to the least loaded machine from its d choices. They showed that the maximal load is $\Theta(n/m) + (1+o(1))\ln\ln m/\ln d$. A large volume of work has been devoted to variations on this problem, such as having weighted jobs [Talwar and Wieder 2007]; and variations on the algorithm, such as the nonuniform job placement strategies of Vöcking [2003]. Of particular relevance to this work is the case of nonuniform machines: Berenbrink et al. [2014] showed that in this case the maximum load can also be bounded by $\Theta(n/m) + O(\ln\ln m)$.

The classical off-line job scheduling problem has two main variations: (1) related machines, where each job i takes a certain amount time, t_i, to complete, regardless of which machine it is allocated, and (2) unrelated machines, where each job i takes time $t_{i,j}$ to complete on machine j. Both problems are known to be NP-hard. Hochbaum and Shmoys [1988] showed a Polynomial-Time Approximation Scheme (PTAS) for scheduling on related machines. Lenstra et al. [1987], presented a 2-approximation algorithm for scheduling on unrelated machines and showed that the optimal allocation is not approximable to within $\frac{3}{2} - \epsilon$ (unless $P = NP$). The problem of finding a truthful mechanism for scheduling (on unrelated machines) was introduced by Nisan and Ronen [1999], who showed an m-approximation to the problem, and a lower bound of 2. Archer and Tardos [2001] were the first to tackle the related machine case; they showed a randomized 3-approximation polynomial algorithm and a polynomial pricing scheme to derive a mechanism that is truthful in expectation. Since then, much work has gone into finding mechanisms with improved approximation ratios, until Christodoulou and Kovács [2010] settled the problem by showing a deterministic PTAS, and a corresponding mechanism that is deterministically truthful.

5.2. The Model

We consider the following (off-line) job scheduling setting. There is a set $\mathcal{I}$ of m machines (or "bins") and a set $\mathcal{J}$ of n uniform jobs (or "balls"). Each machine $i \in \mathcal{I}$ has an associated capacity c_i (sometimes referred to as its "speed"). We assume that the capacities are positive integers. Given that h_i jobs are allocated to machine i, its *load* is $\ell_i = h_i/c_i$, and h_i is also called the *height* of machine i. The *utility* of machine i is quasilinear, namely, when it has load ℓ_i and receives payment p_i then its utility is $u_i(\ell_i, p_i) = p_i - \ell_i$.

The *makespan* of an allocation is $\max_i\{\ell_i\} = \max_i\{h_i/c_i\}$. In our setting, the players are the machines and their private information is their true capacities. Each machine i submits a *bid* b_i (which represents its capacity). The mechanism designer would like to elicit from the machines the true information about their capacities in order to be able to minimize the makespan of the resulting allocation. We assume that the capacities of the machines cannot depend on the number of machines or jobs in the system (i.e., that the bids of the machines are independent of m or n), and hence are upper bounded by some constant. Although we feel this is a reasonable assumption, in Remark 5.9, we show that in some cases we can relax it.

For any allocation algorithm $\mathcal{A}$, and bid vector b, define $\mathcal{A}(b) = (\mathcal{A}^1(b), \ldots, \mathcal{A}^j(b), \ldots, \mathcal{A}^n(b))$ to be the allocation vector, which, when given b as an input, assigns each job j to a machine $i = \mathcal{A}^j(b)$. When the bids b_{-i} are fixed, we sometimes omit them from the notation for clarity.

Definition 5.1 (*Monotonicity*). A randomized allocation function $\mathcal{A}$ is *monotone in expectation* if for any machine i, and any bids b_{-i}, the expected load of machine i, $\mathbb{E}[\ell_i(b_i, b_{-i})]$, is a nondecreasing function of b_i.

A randomized allocation function $\mathcal{A}$ is *universally monotone* if for any machine i, and any bids b_{-i}, the load of machine i, $\ell_i(b_i, b_{-i})$, is a nondecreasing function of b_i for any realization of the randomization of the allocation function.

Given an allocation function $\mathcal{A}$, we would like to provide a payment scheme $\mathcal{P}$ to ensure that our mechanism $\mathcal{M} = (\mathcal{A}, \mathcal{P})$ is truthful. It is known that a necessary and sufficient condition is that the allocation function $\mathcal{A}$ is monotone ([Myerson 1981]; see also Archer and Tardos [2001]).

THEOREM 5.2 [MYERSON 1981]. *The allocation algorithm $\mathcal{A}$ admits a payment scheme $\mathcal{P}$ such that the mechanism $\mathcal{M} = (\mathcal{A}, \mathcal{P})$ is truthful-in-expectation (universally truthful) if and only if $\mathcal{A}$ is monotone in expectation (universally monotone).*

In this section, we consider two load balancing settings: The *standard* setting (cf. Berenbrink et al. [2014] and Wieder [2007]) is a slight variation on the basic power-of-d choices setting proposed in Azar et al. [1999]. Let $d \geq 2$ be some integer. For each job j, the mechanism chooses a subset $I_j \subseteq \mathcal{I}$, $|I_j| = d$ of machines that the job can be allocated to. The probability that machine $i \in I_j$ is proportional to b_i (specifically, it is $\frac{db_i}{\sum_i b_i}$). In the *restricted* setting (cf. Azar et al. [1995]), each job can be allocated to a subset of at most d machines, where the subsets I_j are given as an input to the allocation algorithm. The restricted setting models the case when the jobs have different requirements, and there is only a small subset of machines that can run each job. We restrict our attention to the case when the number of jobs $n = \Theta(C)$ jobs where C is the total capacity of the machines. This is a standard assumption, as it is considered to be the wors-case scenario (see e.g., Azar et al. [1999], Berenbrink et al. [2014], and Wieder [2007]), and so a solution for this case implies that there is an equally good solution for all other cases as well.

In both of these settings, our results rely on a reduction to an online algorithm for the problem. Mansour et al. [2012] and Reingold and Vardi [2016] showed that it is possible to transform certain online algorithms to LCAs, for a restricted family of graphs (including graphs where the vertex degrees are bounded by a constant or distributed binomially). The idea behind the reduction is simple: generate a (pseudo-) random order on the vertices and simulate the online algorithm on this order. In order to be able to generate this order consistently and "on the fly" whenever the LCA is queried, we need to store a random seed of length $O(\log n)$ (where n is the number of vertices). The pseudorandom order on the vertices guarantees that with high probability, the LCA will need to query at most $O(\log n)$ vertices. This is summarized in the following theorem, which is a specialized version of a result of Reingold and Vardi [2016].

THEOREM 5.3 (CF. MANSOUR ET AL. [2012] AND REINGOLD AND VARDI [2016]). *Consider a job scheduling problem for n jobs and $\Theta(n)$ machines. For each job j, there is a constant-size subset of machines I_j, chosen uniformly at random, and j cannot be allocated to any machine $i \notin I_j$. For any online algorithm $\mathcal{LB}$ to the problem that requires constant time per query, there exists an $(O(\log^2 n), O(\log^2 n), 1/n)$-local computation algorithm that, when queried on a job, allocates it to a machine, such that the resulting allocation is consistent with that of $\mathcal{LB}$.*

5.3. A Truthful in Expectation Mechanism for the Standard Setting

In the standard setting, each machine i has an integer capacity c_i. One way of modeling this is to regard the allocation field as consisting of $\sum_i c_i$ slots of size 1, where machine i "owns" c_i slots. Recall that a machine's *height* is the number of jobs that are allocated to it; the *load* of machine i is its height divided by c_i. The *virtual load* of machine i is its height divided by its bid b_i. Given the bids b of the machines, let $B = \sum_{i=1}^{n} b_i$. An allocation algorithm allocates jobs to slots: when a job j is allocated to a specific slot, the machine that owns the slot receives j. We provide the following simple online allocation algorithm $\mathcal{A}_{SLMS}$, which is modeled on the algorithm presented in Berenbrink et al. [2014].

(1) Choose for job j a subset I_j of d slots out of B, where each slot has equal probability. (I_j may include different slots owned by the same machine.)
(2) Given I_j, job j is allocated to the lowest slot (i.e., the one containing the fewest jobs) in I_j (breaking ties uniformly at random). Slots are treated as being independent of their machines. That is, it is possible that if a job chooses two slots a and b, which belong to machines A and B, a has fewer jobs than b, but B has a higher (virtual) load than A.

Note: Although it may not be possible to compute B locally exactly, it has been shown in that an approximate calculation suffices (e.g., Byers et al. [2003] and Wieder [2007]); therefore, for simplicity, we assume that it is possible to compute B locally.

LEMMA 5.4. *The randomized allocation algorithm $\mathcal{A}_{SLMS}$ is monotone in expectation.*

PROOF. Let $B = \sum_i b_i$ and $B_{-i} = \sum_{i \neq i} b_i$. Since all the slots are identical, by symmetry the expected number of jobs allocated to each slot is exactly n/B. The expected height of machine i is therefore

$$\mathbb{E}[h_i(b_i)] = \frac{b_i}{B_{-i} + b_i} n,$$

which is monotone increasing in b_i (for $b_i, B_{-i} \geq 0$). □

From Theorem 5.2, we conclude:

LEMMA 5.5. *The randomized allocation function $\mathcal{A}_{SLMS}$ admits a payment scheme $\mathcal{P}_{SLMS}$ such that the mechanism $\mathcal{M}_{SLMS} = (\mathcal{A}_{SLMS}, \mathcal{P}_{SLMS})$ is truthful in expectation.*

It is interesting to note that the preceding algorithm does not admit a universally truthful mechanism. To show this, we prove a slightly stronger claim, which we then adapt to our setting: the GREEDY algorithm – in which each job chooses d machines at random, and is allocated to the least loaded among them (post-placement)[5] , breaking ties arbitrarily – does not admit a universally truthful mechanism.

CLAIM 5.6. *Algorithm* GREEDY *is not universally monotone.*

PROOF. Assume we have four machines: A, B, C, and D, with bids 4, 4, 8, and 1, respectively. The first two jobs choose machines A and D (which we abbreviate to AD), the next two jobs choose BD, and the next six jobs choose CD. After these 10 jobs, the heights of the machines are $(2, 2, 6, 0)$ (recall that the Greedy algorithm allocates according to the *postplacement* load). The 11th job chooses AB, and the 12th job chooses AC. As ties are broken at random, assume machine A receives job 11. Machine C then receives job 12, making the capacities $(3, 2, 7, 0)$.

Now assume machine C bids 9, and the choices of the first 10 jobs and the 12th job remain the same, but because C bid higher, now the 11th job chooses C instead of A (so job 11 chooses BC). Now machine B receives the 11th job and machine A receives the 12th job, making the capacities $(3, 3, 6, 0)$. Machine C received less jobs although it bid more! □

It is easy to adapt the preceding proof to Algorithm $\mathcal{A}_{SLMS}$: instead of choosing machines, each job chooses two slots. So, the first job will choose slot 1 of machine A and the only slot of machine D; the second job will choose slot 2 of machine A and machine D's slot; and so on. This gives the following corollary.

COROLLARY 5.7. *Algorithm $\mathcal{A}_{SLMS}$ is not universally monotone.*

By Theorem 5.3, the allocation function $\mathcal{A}_{SLMS}$ can be transformed to a $(O(\log^2 n), O(\log^2 n), 1/n)$ LCA. For clarity, we overload the notation, letting $\mathcal{A}_{SLMS}$ represent both the online allocation algorithm and its respective LCA, as it is easy to distinguish between them from context. We would now like to show a payment scheme $\mathcal{P}_{SLMS}$ such that the mechanism $\mathcal{M}_{SLMS} = (\mathcal{A}_{SLMS}, \mathcal{P}_{SLMS})$ is a local mechanism. We need to show a payment scheme that can be implemented as an LCA and guarantees truthfulness. We give a deterministic payment scheme that is similar to the payment schemes of Archer et al. [2003] and Berenbrink et al. [2014]. We also comment on the possibility

[5] That is, the load is computed including the allocation of the arriving job. For example, if machine A has capacity 4 and height 2 and machine B has capacity 16 and height 9, the job will go to machine B, as after placing the job, the load on B would 10/16, compared to 3/4 on A.

of a randomized payment scheme when the bids can depend on the total capacity. The randomized payment scheme is similar to that of Babaioff et al. [2010].

LEMMA 5.8. *If the bids of the machines are bounded by a constant, there exists a deterministic local payment scheme $\mathcal{P}_{SLMS}$ such that the mechanism $\mathcal{M}_{SLMS} = (\mathcal{A}_{SLMS}, \mathcal{P}_{SLMS})$ is truthful in expectation.*

PROOF. Archer and Tardos [2001] showed that the following payment scheme makes for a truthful mechanism fulfilling voluntary participation. For bid b_i:

$$p_i(b_i, b_{-i}) = b_i h_i(b_i, b_{-i}) + \sum_{0}^{b_i} h_i(x, b_{-i})dx. \tag{5}$$

As b_i is bounded by a constant, we can execute $\mathcal{A}_{SLMS}$ with all values of $b_i \in [0, b_i]$, to compute p_i. This takes a constant number of executions of $\mathcal{A}_{SLMS}$. □

Remark 5.9. If b_i is not necessarily a constant, but the mechanism has access to the value B_{-i}, there is a randomized payment scheme that we can use. Equation (5) is the expected payment. From symmetry, $\mathbb{E}(h_i(B)) = \frac{b_i}{B}$, hence we can rewrite Equation (5) as

$$p_i(b_i, b_{-i}) = n\frac{b_i^2}{B_{-i} + b_i} + n\sum_{x=0}^{b_i} \frac{x}{B_{-i} + x}.$$

Choose, uniformly at random, $k \in [1, b_i]$, and take the payment to be

$$n\frac{b_i^2}{B} + nb_i \cdot \frac{k}{B_{-i} + k}.$$

This gives the correct expected payment, and takes $O(1)$ time.

Berenbrink et al. [2014], showed that $\mathcal{A}_{SLMS}$ provides an $O(\log\log m)$ approximation to the optimal makespan. Therefore, by Theorem 5.3, the LCA of $\mathcal{A}_{SLMS}$ provides the same approximation ratio. Combining Lemma 5.5, and Lemma 5.8, we state our main result for the standard setting:

THEOREM 5.10. *There exists an $(O(\log^2 n), O(\log^2 n), 1/n)$-local mechanism to scheduling on related machines in the standard setting that is truthful in expectation, and provides an $O(\log\log n)$-approximation to the makespan.*

5.4. A Universally Truthful Mechanism for the Restricted Setting

In the restricted setting, each job can only be allocated to one of a set $I_j \subseteq \mathcal{I}$ of d machines. As opposed to the standard setting, I_j is not selected by the mechanism, but is part of the input. We assume that these sets are selected independently and identically distributed (i.i.d.) from all possible sets, and the probability of machine i to be in I_j is proportional to its capacity c_i. The first assumption is necessary for bounding the running time, and the second to guarantee the approximation ratio. The second requirement can be relaxed slightly (see, e.g., Wieder [2007]) but for clarity of the proofs, we will assume that it holds exactly. Similarly to the previous subsection, we assume that the capacity of each machine is bounded by a constant.

We define the (online) algorithm $\mathcal{A}_{RLMS}$ for assigning jobs to machines as follows. Initially, a permutation π of the machines is selected arbitrarily, for tie-breaking. Job t is assigned to the machine $i \in I_j$ for which the *postplacement load* $lp_i^{t+1}(b_i) = \lfloor \frac{h_i^t(b_i)+1}{b_i} \rfloor$ is smallest, breaking ties according to π. The following claim shows why it is necessary to take the floor of the load, as the simple Greedy algorithm does not admit a universally truthful mechanism in this case.

Claim 5.11. *The (unmodified)* Greedy *algorithm is not universally monotone in the restricted case.*

Proof. Assume we have three machines A, B, C, with bids $(4, 8, 36)$, respectively, and a tie-breaking permutation: $A < B < C$ (jobs always prefer machine A to machines B and C, and machine B to machine C). The allocation at time t is $(1, 3, 18)$. The next job's restricted set contains machines A and B (which we abbreviate to AB), and the following two jobs' sets are BC and AB, respectively. The first job is allocated to A (since the postplacement loads on A and B are $2/4$ and $4/8$, respectively, hence we use the tie-breaking rule). The second job is allocated to B $(4/8 < 19/36)$ and the third job to B $(5/8 > 3/4)$. The heights of the machines are now $(2, 5, 18)$.

Now assume B declares its capacity to be 9, and assume that at time t, there is no difference in the allocation (it is easy to verify that this is indeed possible). The loads at time t in this case are $1/4, 3/9, 18/36$. The jobs' choices are part of the input to the mechanism, so are unaffected by the bids, and remain AB, BC, AB. The first job is allocated to B $(2/4 > 4/9)$, the second job to C $(19/36 < 20/36 = 5/9)$, and the third job to A $(2/4 < 5/9)$. The heights of the machines are now $(2, 4, 19)$. Thus, B receives jobs despite bidding higher. □

Interestingly, although Greedy is not universally monotone, $\mathcal{A}_{RLMS}$ is.

Theorem 5.12. *For any permutation π of the machines and any job arrival order, the allocation function $\mathcal{A}_{RLMS}$ is universally monotone increasing in the machines' bids.*

From the definition of universal monotonicity (Definition 5.1), it suffices to prove the following lemma:

Lemma 5.13. *For any machine i, fixing b_{-i}, for any $b'_i > b_i$, we have that $h_i(\mathcal{A}_{RLMS}(b'_i, b_{-i})) \geq h_i(\mathcal{A}_{RLMS}(b_i, b_{-i}))$.*

To prove Lemma 5.13, define $D^t(k, b'_i, b_i)$ to be the difference in the number of jobs allocated to machine k between $\mathcal{A}_{RLMS}(b'_i)$ and $\mathcal{A}_{RLMS}(b_i)$ up to and including the arrival of job t (which we call *time t*). We abbreviate this to $D^t(k)$ when b'_i and b_i are clear from the context. (If machine k received less jobs, then $D^t(k)$ is negative.) We say that machine k *steals* a job from machine l at time t if $\mathcal{A}^t_{RLMS}(b_i) = l$ and $\mathcal{A}^t_{RLMS}(b'_i) = k$. We will show that the only machine for which $D^t(k)$ can be positive at some time t is machine i, therefore, as $\sum_{j=1}^{n} D^t(j) = 0$, we have that $D^t(i)$ can never be negative.

Proposition 5.14. *For any machine i, fixing b_{-i}, if $b'_i > b_i$, then at all times t, for any machine $k \neq i$, $D^t(k) \leq 0$.*

Informally, Proposition 5.14 says that if bin i claims its capacity is larger than it actually is, no bin except for i can receive more balls. The following corollary follows immediately from Proposition 5.14, and implies Lemma 5.13.

Corollary 5.15. *For any machine i, fixing b_{-i}, if $b'_i > b_i$ then at all times t, $D^t(i) \geq 0$.*

Before proving Proposition 5.14, we first will make the following simple observation:

Observation 5.16. *For any machine k, if $D^t(k) \leq 0$, then $lp^t_k(b'_i) \leq lp^t_k(b_i)$.*

Proof. For $k \neq i$, as k's bid is the same in both allocations, if it received less jobs by time t in $\mathcal{A}_{RLMS}(b_i)$, then the observation follows. If $k = i$, the observation follows since $b'_i > b_i$. □

We now prove Proposition 5.14.

Proof of Proposition 5.14. The proof is by induction on t. At time $t = 0$, $D^0(k) = 0$ for every k.

Assume the proposition is true for times $t = 0, 1, \ldots, \tau - 1$. We show it holds for $t = \tau$, by contradiction. Assume that we have a machine $k \neq i$ such that $D^{\tau}(k) > 0$. At time $\tau - 1$, for all $k \neq i$, by the induction hypothesis, it holds that $D^{\tau-1}(k) \leq 0$. The only way that $D^{\tau}(k) > 0$ is if machine k has $D^{\tau-1}(k) = 0$ and at time τ steals a job. Assume first that machine k steals a job from machine $l \neq i$. This means that in $\mathcal{A}_{RLMS}(b_i)$, machine l received job τ, therefore

$$lp_l^{\tau}(b_i) \leq lp_k^{\tau}(b_i). \tag{6}$$

By Observation 5.16, $lp_l^{\tau}(b_i') \leq lp_l^{\tau}(b_i)$, and so

$$lp_l^{\tau}(b_i') \leq lp_l^{\tau}(b_i) \leq lp_k^{\tau}(b_i) = lp_k^{\tau}(b_i').$$

If machine k steals job τ from machine l, then $lp_k^{\tau}(b_i') \leq lp_l^{\tau}(b_i')$. This is a contradiction to Equation (6) because there cannot be an equality both here and in Equation (6), as the tie-breaking permutation π is fixed. More precisely, if $lp_l^{\tau}(b_i') = lp_l^{\tau}(b_i) = lp_k^{\tau}(b_i) = lp_k^{\tau}(b_i')$, then job τ will be allocated to the same machine in b_i and b_i', according to the permutation π.

Therefore, machine k must steal job τ from machine i, which gives us

$$lp_i^{\tau}(b_i) \leq lp_k^{\tau}(b_i) = lp_k^{\tau}(b_i') \leq lp_i^{\tau}(b_i'). \tag{7}$$

The first inequality is due to the fact that machine i receives job τ in $\mathcal{A}_{RLMS}(b_i)$. The equality is due to the fact that $D^{\tau-1}(k) = 0$, and the second inequality is because machine k receives job τ in $\mathcal{A}_{RLMS}(b_i')$. And so,

$$lp_i^{\tau}(b_i) < lp_i^{\tau}(b_i'), \tag{8}$$

because one of the inequalities in Equation (7) must be strict, as the tie-breaking permutation π is fixed.

Assume that the last time before τ that machine i stole a job is time ρ, and label by z the machine that i stole from at that time. We have

$$lp_i^{\rho}(b_i') \leq lp_z^{\rho}(b_i') \leq lp_z^{\rho}(b_i) \leq lp_i^{\rho}(b_i).$$

The first inequality is because machine i received job ρ in $\mathcal{A}_{RLMS}(b_i')$. The middle inequality is because $D^{\rho}(z) \leq 0$. The last inequality is because machine z received job ρ in $\mathcal{A}_{RLMS}(b_i)$. Again, at least one inequality must be strict, giving

$$lp_i^{\rho}(b_i') < lp_i^{\rho}(b_i),$$

which implies, for all $\alpha \geq 0$,

$$\left\lfloor \frac{h_i^{\rho}(b_i') + \alpha + 1}{b_i'} \right\rfloor \leq \left\lfloor \frac{h_i^{\rho}(b_i) + \alpha}{b_i} \right\rfloor, \tag{9}$$

since $b_i' > b_i \geq 1$.

Because job ρ was the last job that machine i stole, it received at least as many jobs between ρ and τ in $\mathcal{A}_{RLMS}(b_i)$ as in $\mathcal{A}_{RLMS}(b_i')$. Label the number of jobs i received between ρ and τ (including ρ but excluding τ) in $\mathcal{A}_{RLMS}(b_i)$ by β and in $\mathcal{A}_{RLMS}(b_i')$ by β^*.

Observation 5.17. $\beta^* \leq \beta + 1$.

Proof. Machine i received at least as many jobs in $\mathcal{A}_{RLMS}(b_i)$ as in $\mathcal{A}_{RLMS}(b_i')$ after ρ. This must be true because ρ was the last time machine i stole a job. However, machine

i received the job at time ρ in $\mathcal{A}_{RLMS}(b_i')$ but not in $\mathcal{A}_{RLMS}(b_i)$, and so we cannot claim that $\beta^* \leq \beta$, but only that $\beta^* \leq \beta + 1$. □

Proof of Proposition 5.14 continued. From the definition of lp and Equation (9), we get

$$\begin{aligned} lp_i^\tau(b_i') &= \left\lfloor \frac{h_i^\tau(b_i') + 1}{b_i'} \right\rfloor \\ &= \left\lfloor \frac{h_i^\rho(b_i') + \beta^* + 1}{b_i'} \right\rfloor && (10) \\ &\leq \left\lfloor \frac{h_i^\rho(b_i') + \beta + 2}{b_i'} \right\rfloor && (11) \\ &\leq \left\lfloor \frac{h_i^\rho(b_i) + \beta + 1}{b_i} \right\rfloor && (12) \\ &= lp_i^\tau(b_i). && (13) \end{aligned}$$

Equality (10) stems from the definition of β^*, inequality (11) is due to Observation 5.17, inequality (12) is due to Equation (9), and equality (13) is from the definition of β.

This is in contradiction to Equation (8), and therefore $D^\tau(k) \leq 0$. This concludes the proof of the proposition. □

Given that $\mathcal{A}_{RLMS}$ is universally monotone, we can once again use the payment scheme of Archer and Tardos [2001] to obtain the following lemma.

Lemma 5.18. *There exists a local payment scheme $\mathcal{P}_{RLMS}$ such that the mechanism $\mathcal{P}_{RLMS} = (\mathcal{A}_{RLMS}, \mathcal{P}_{RLMS})$ is universally truthful.*

It remains to bound the approximation ratio of our algorithm.

Lemma 5.19. *The allocation algorithm $\mathcal{A}_{RLMS}$ provides an $O(\log\log n)$-approximation to the optimal allocation.*

The proof is similar to the proof for the unmodified Greedy algorithm in the case of nonuniform bins of Berenbrink et al. [2014]. We provide it in Appendix A for completeness.

Putting everything together gives our main result for this subsection.

Theorem 5.20. *There exists an $(O(\log^2 n), O(\log^2 n), 1/n)$-local mechanism for scheduling on related machines in the restricted setting that is universally truthful and gives an $O(\log\log n)$-approximation to the makespan.*

APPENDIX

A. PROOF OF LEMMA 5.19

Lemma 5.19. *The allocation algorithm $\mathcal{A}_{RLMS}$ provides an $O(\log\log n)$-approximation to the optimal allocation.*

We prove the theorem for the case $d = 2$ (each job can be assigned to one of two machines). The proof is easily extendable to $d > 2$. For the proof (not the algorithm), we regard each machine i of capacity c_i as having c_i slots of capacity 1. Before presenting the proof we need several definitions.

The *load vector* of an allocation of jobs to m machines is $L = (\ell_1, \ldots, \ell_m)$, where ℓ_i is the load of machine i. The *normalized load vector* $\bar{L}$ consists of the members of L in nonincreasing order (ties are broken arbitrarily). For the case of nonuniform machines of capacities $c_1, \ldots c_m$, and total capacity $C = \sum_{i=1}^{m} c_i$, we define the *slot-load vector* $S = (h_{1,1}, \ldots h_{1,c_1}, h_{2,1}, \ldots h_{2,c_2}, \ldots h_{n,1}, \ldots h_{n,c_n})$, where if machine i is allocated r jobs, the first $r \bmod c$ slots will have $\lceil r/c \rceil$ jobs, and the remaining slots will have $\lfloor r/c \rfloor$ jobs. If a machine has an uneven allocation of jobs, we call the slots with more jobs *heavy*, and the slots with less jobs *light*. If all of the slots of the machine have an identical number of jobs assigned to them, we call all the slots *light*. When we allocate a job to a machine, we add it to one of the light slots, arbitrarily. The *normalized slot load vector* $\bar{S}$ is S sorted in nonincreasing order (slots of the same machine may be separated in $\bar{S}$). We add a subscript t to these vectors, that is, L_t, $\bar{L}_t$, S_t, and $\bar{S}_t$ to indicate the vector after the allocation of the t-th job.

Definition A.1 (*Majorization,* $\succeq$). We say that a vector $P = (p_1, \ldots, p_a)$ *majorizes* vector $Q = (q_1, \ldots, q_b)$ (denoted $P \succeq Q$) if and only if for all $1 \leq k \leq \min(a, b)$,

$$\sum_{i=1}^{k} \bar{p}_i \geq \sum_{i=1}^{k} \bar{q}_i,$$

where $\bar{p}_i$ and $\bar{q}_i$ are the i-th entries of the normalized vectors $\bar{P}$ and $\bar{Q}$.

For $n \in \mathbb{N}$, let $[n]$ denote $\{1, \ldots, n\}$.

Definition A.2 (*System Majorization*). Let A and B be two processes allocating n jobs to machines with total capacity C. Let $\tau = (\tau_1 \ldots \tau_{2n})$, $\tau_i \in [C]$ be a vector representing the (slot) choices of the n jobs (τ_{2i-1} and τ_{2i} are the choices of the i-th job). Let $S^A(\tau)$ and $S^B(\tau)$ be the slot load vectors using A and B, respectively, with the random choices specified by τ. Then we say

(1) *A majorizes B* (denoted by the overloaded notation $A \succeq B$) if there is a bijection $f : [C]^{2n} \to [C]^{2n}$ such that for all possible random choices $\tau \in [C]^{2n}$, we have

$$L^A(\tau) \succeq L^B(f(\tau)).$$

(2) The maximum load of A *majorizes* the maximum load of B (denoted by $A \succeq_n B$) if there is a bijection $f : [C]^{2n} \to [C]^{2n}$ such that for all possible random choices $\tau \in [C]^{2n}$, it holds that

$$\ell_1^A(\tau) \geq \ell_1^B(f(\tau)),$$

where $\ell_1^A(\tau)$ and $\ell_1^B(f(\tau))$ are the loads of the most loaded bins in A and B, respectively, with the random choices specified by τ and $f(\tau)$, respectively.

It is immediate that the following holds.

Observation A.3. $A \succeq B \Rightarrow A \succeq_n B$.

We now turn to the proof of Lemma 5.19.

First, notice that if we have a system of m identical machines, each of capacity 1, both the unmodified Greedy algorithm and the allocation algorithm $\mathcal{A}_{RLMS}$ will behave in exactly the same way—the load and the $\lfloor \text{load} \rfloor$ are the same if the capacity is 1. From Azar et al. [1999], we know that the maximal load on any machine when allocating $n = m$ jobs (to m machines with capacity 1) with the Greedy algorithm, is

$\Theta(\log\log n)$. Therefore, the maximal load when allocating $n = m$ jobs with $\mathcal{A}_{RLMS}$ is also $\Theta(\log\log n)$ in this setting. We would like to show that the maximal load of a system with nonuniform machines of total capacity C is majorized by the maximal load of a system with C machines of capacity 1, when the allocating algorithm is $\mathcal{A}_{RLMS}$. We will show that the first system majorizes the second, and deduce the required result from Observation A.3.

We restate Claim 2.4 of Wieder [2007]:

CLAIM A.4 ([WIEDER 2007]). *Let P and Q be two normalized integer vectors such that $P \succeq Q$. If $i \leq j$, then $P + e_i \succeq Q + e_j$ where e_i is the i-th unit vector and $P + e_i$ and $Q + e_j$ are normalized.*

LEMMA A.5. *For allocation algorithm $\mathcal{A}_{RLMS}$, let A be a system with nonuniform machines of total capacity C, and B be a system with C uniform machines of capacity* 1 *each. Then $B \succeq A$.*

PROOF. We use the slot load vectors of systems A and B (in B the load vector and slot load vector are identical), and show that $S^B(f(\tau)) \succeq S^A(\tau)$. The bijection is such that the jobs in both processes choose the same $k_1 < k_2 \in \{1, \ldots, C\}$ in the normalized slot load vectors, and the choice corresponds to machines k_1, k_2 in B and the machines associated with those specific slots in system A. We use induction: for $t = 0$, the claim is trivially true.

From the inductive hypothesis, before the allocation of the t-th job, $S^B_{t-1}(f(\tau)) \succeq S^A_{t-1}(\tau)$. In system B, the t-th job goes to machine k_2. In system A, if the $\lfloor$load$\rfloor$ of the machine of k_1 is greater than that of the machine of k_2, the job goes to k_2 if k_2 is a light slot, or to a slot to the right of k_2 (a lighter slot of the same machine), if k_2 is a heavy slot. If the $\lfloor$loads$\rfloor$ of the machines of k_1 and k_2 are the same, again, the job goes to k_2 if k_2 is a light slot, or to a slot to the right of k_2 (again, a lighter slot of the same machine), if k_2 is a heavy slot. In all cases, by Claim A.4, it follows that $S^B(f(\tau)) \succeq S^A(\tau)$. □

ACKNOWLEDGMENTS

We would like to thank Amos Fiat, Alon Naor, Amit Weinstein, and the anonymous reviewers for their useful input.

REFERENCES

Noga Alon, Ronitt Rubinfeld, Shai Vardi, and Ning Xie. 2012. Space-efficient local computation algorithms. In *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1132–1139.

Noga Alon and Joel Spencer. 2008. *The Probabilistic Method* (3rd ed.). John Wiley.

Aaron Archer, Christos H. Papadimitriou, Kunal Talwar, and Éva Tardos. 2003. An approximate truthful mechanism for combinatorial auctions with single parameter agents. *Internet Mathematics* 1, 2 (2003), 129–150. DOI:http://dx.doi.org/10.1080/15427951.2004.10129086

Aaron Archer and Éva Tardos. 2001. Truthful mechanisms for one-parameter agents. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 482–491.

Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal. 1999. Balanced allocations. *SIAM J. Comput.* 29(1) (1999), 180–200.

Yossi Azar, Joseph Naor, and Raphael Rom. 1995. The competitiveness of on-line assignments. *J. Algorithms* 18, 2 (1995), 221–237.

Moshe Babaioff, Robert D. Kleinberg, and Aleksandrs Slivkins. 2010. Truthful mechanisms with implicit payment computation. In *Proceedings of the ACM Conference on Electronic Commerce*. 43–52.

Petra Berenbrink, André Brinkmann, Tom Friedetzky, and Lars Nagel. 2014. Balls into non-uniform bins. *J. Parallel Distrib. Comput.* 74, 2 (2014), 2065–2076.

John W. Byers, Jeffrey Considine, and Michael Mitzenmacher. 2003. Simple load balancing for distributed hash tables. In *IPTPS*. 80–87.

George Christodoulou and Annamária Kovács. 2010. A deterministic truthful PTAS for scheduling related machines. In *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 1005–1016.

Kimmo Eriksson and Olle Häggström. 2008. Instability of matchings in decentralized markets with various preference structures. *Int. J. Game Theory* 36, 3-4 (2008), 409–420.

Tomás Feder, Nimrod Megiddo, and Serge A. Plotkin. 2000. A sublinear parallel algorithm for stable matching. *Theor. Comput. Sci.* 233, 1–2 (2000), 297–308.

Tamás Fleiner. 2003. A fixed-point approach to stable matchings and some applications. *Math. Oper. Res.* 28, 1 (Feb. 2003), 103–126.

Patrik Floréen, Petteri Kaski, Valentin Polishchuk, and Jukka Suomela. 2010. Almost stable matchings by truncating the Gale-Shapley algorithm. *Algorithmica* 58, 1 (2010), 102–118.

David Gale and Lloyd S. Shapley. 1962. College admissions and the stability of marriage. *Amer. Math. Monthly* 69 (1962), 9–14.

David Gale and Marilda Sotomayor. 1985. Some remarks on the stable matching problem. *Discr. Appl. Math.* 11, 3 (1985), 223–232.

Dan Gusfield and Robert W. Irving. 1989. *The Stable Marriage Problem—Structure and Algorithms*. MIT Press.

Dorit S. Hochbaum and David B. Shmoys. 1988. A polynomial approximation scheme for machine scheduling on uniform processors: Using the dual approximation approach. *SIAM J. Comput.* 17(3) (1988), 539–551.

Nicole Immorlica and Mohammad Mahdian. 2005. Marriage, honesty, and stability. In *SODA*. 53–62.

Donald E. Knuth. 1976. Mariages stables. *Les Presses de l'Université de Montréal* (1976).

Fuhito Kojima and Parag A. Pathak. 2009. Incentives and stability in large two-sided matching markets. *Am. Econ. Rev.* 99, 3 (2009), 608–627.

Jan Karel Lenstra, David B. Shmoys, and Éva Tardos. 1987. Approximation algorithms for scheduling unrelated parallel machines. In *Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 217–224.

Enyue Lu and S. Q. Zheng. 2003. A parallel iterative improvement stable matching algorithm. In *HiPC*. 55–65.

Yishay Mansour, Aviad Rubinstein, Shai Vardi, and Ning Xie. 2012. Converting online algorithms to local computation algorithms. In *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP)*. 653–664.

Yishay Mansour and Shai Vardi. 2013. A local computation approximation scheme to maximum matching. In *APPROX-RANDOM*. 260–273.

Michael Maschler, Eilon Solan, and Shmuel Zamir. 2013. *Game Theory*. Cambridge Press.

Roger B. Myerson. 1981. Optimal auction design. *Math. Oper. Res.* 6 (1981), 58–74.

Cheng Ng and Daniel S. Hirschberg. 1990. Lower bounds for the stable marriage problem and its variants. *SIAM J. Comput.* 19, 1 (1990), 71–77.

Huy N. Nguyen and Krzystof Onak. 2008. Constant-time approximation algorithms via local improvements. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 327–336.

Noam Nisan and Amir Ronen. 1999. Algorithmic mechanism design (extended abstract). In *Proceedings of the 31st Annual ACM Symposium on the Theory of Computing (STOC)*. 129–140.

N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani (Eds.). 2005. *Algorithmic Game Theory*. Cambridge University Press.

Rafail Ostrovsky and Will Rosenbaum. 2014. On the communication complexity of finding an (approximate) stable marriage. *CoRR* abs/1406.1273 (2014). http://arxiv.org/abs/1406.1273.

Michael J. Quinn. 1985. A note on two parallel algorithms to solve the stable marriage problem. *BIT Num. Mathe.* 25(3) (1985), 473–476.

Omer Reingold and Shai Vardi. 2016. New techniques and tighter bounds for local computation algorithms. *Journal of Computer and System Sciences* 82, 7 (2016), 1180–1200.

Alvin E. Roth. 2003. The origins, history, and design of the resident match. *J. Am. Med. Assoc.* 289, 7 (2003), 909–912.

Alvin E. Roth and Elliott Peranson. 1999. The redesign of the matching market for American physicians: Some engineering aspects of economic design. *Am. Econ. Rev.* 89 (1999), 748–780.

Alvin E. Roth and Uriel G. Rothblum. 1999. Truncation strategies in matching markets—In search of advice for participants. *Econometrica* 67, 1 (1999), 21–43.

Alvin E. Roth and Marilda Sotomayor. 1990. *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Cambridge University Press.

Alvin E. Roth and Xiaolin Xing. 1997. Turnaround time and bottlenecks in market clearing: Decentralized matching in the market for clinical psychologists. *J. Pol. Econ.* 105 (1997), 284–329.

Ronitt Rubinfeld, Gil Tamir, Shai Vardi, and Ning Xie. 2011. Fast local computation algorithms. In *Proceedings of the 2nd Symposium on Innovations in Computer Science (ICS)*. 223–238.

Kunal Talwar and Udi Wieder. 2007. Balanced allocations: The weighted case. In *Proceedings of the 39th Annual ACM Symposium on the Theory of Computing (STOC)*. 256–265.

Shian-Shyong Tseng and Richard C. T. Lee. 1984. A parallel algorithm to solve the stable marriage problem. *BIT* 24, 3 (1984), 308–316.

Berthold Vöcking. 2003. How asymmetry helps load balancing. *J. ACM* 50, 4 (2003), 568–589.

Udi Wieder. 2007. Balanced allocations with heterogenous balls. In *Proceedings of the 19th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*. 188–193.

Received December 2014; revised April 2015; accepted June 2015