Learning What's Going on: Reconstructing Preferences and Priorities from Opaque Transactions

AVRIM BLUM, Toyota Technological Institute at Chicago, USA

YISHAY MANSOUR, Tel Aviv University

JAMIE MORGENSTERN, Georgia Institute of Technology

We consider a setting where n buyers, with combinatorial preferences over m items, and a seller, running a priority-based allocation mechanism, repeatedly interact. Our goal, from observing limited information about the results of these interactions, is to reconstruct both the preferences of the buyers and the mechanism of the seller. More specifically, we consider an online setting where at each stage, a subset of the buyers arrive and are allocated items, according to some unknown priority that the seller has among the buyers. Our learning algorithm observes only which buyers arrive and the allocation produced (or some function of the allocation, such as just which buyers received positive utility and which did not), and its goal is to predict the outcome for future subsets of buyers. For this task, the learning algorithm needs to reconstruct both the priority among the buyers and the preferences of each buyer. We derive mistake bound algorithms for additive, unit-demand and single-minded buyers. We also consider the case where buyers' utilities for a fixed bundle can change between stages due to different (observed) prices. Our algorithms are efficient both in computation time and in the maximum number of mistakes (both polynomial in the number of buyers and items).

CCS Concepts: • **Theory of computation** → **Algorithmic game theory and mechanism design**;

Additional Key Words and Phrases: Mechanism design, learning from revealed preferences

ACM Reference format:

Avrim Blum, Yishay Mansour, and Jamie Morgenstern. 2018. Learning What's Going on: Reconstructing Preferences and Priorities from Opaque Transactions. *ACM Trans. Econ. Comput.* 6, 3–4, Article 13 (October 2018), 20 pages.

<https://doi.org/10.1145/3274642>

1 INTRODUCTION

On Monday, a collection of clients request service from a server. The server fulfills the requests of some clients and not those of others. On Tuesday, a different but overlapping collection of clients

Blum was supported in part by the National Science Foundation under grants CCF-1101215, CCF-1116892, CCF-1331175, and IIS-1065251. Mansour was supported in part by The Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), by a grant from the Israel Science Foundation (ISF), by a grant from United States-Israel Binational Science Foundation (BSF), and by a grant from the Israeli Ministry of Science (MoS). Morgenstern was supported in part by the National Science Foundation under grants CCF-1116892, CCF-1331175, CCF-1415460 and by a Simons Award for Graduate Students in Theoretical Computer Science.

Authors' addresses: A. Blum, Toyota Technological Institute at Chicago, Chicago, IL, USA; email: avrim@ttic.edu; Y. Mansour, Tel Aviv University; email: mansour@tau.ac.il; J. Morgenstern, Georgia Institute of Technology; email: jamiemmt.cs@gatech.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.

2167-8375/2018/10-ART13 \$15.00

<https://doi.org/10.1145/3274642>

request service from the server, and a different subset of them are served. Suppose that one can model the server's choice of clients to serve as follows. Each client has a job she wishes to run that is fixed across days, and her job requires a fixed set of computational resources (e.g., requires a certain amount of RAM, some amount of hard drive space, and a dedicated internet port) and cannot be run without *all* of those resources. Represent these requirements as sets of resources $D_i \subseteq \{1, \dots, m\}$ for job i , which can be thought of as a *demand set* for a single-minded job i . Each day, some subset of the clients wish to run their jobs. The server may not have enough resources to run all the requested jobs on any given day, so it decides which jobs to run based on some fixed but unknown *priority* over clients. If the highest-priority client shows up on a given day, then her job is always run; if the second-highest-priority client shows up and there are enough resources remaining to run her job, then her job is run; if the third-highest-priority client shows up and there are enough resources remaining to run her job, then her job is run, and so on. As an outside observer who only sees which clients show up each day, and which jobs are run on those days, can we learn which jobs require which resources or predict for a future subset of clients which jobs will be run?

Alternatively, in the context of computational advertising, consider a publisher that owns a web site and has some collection of advertisers. The advertisers tell the publisher which potential impressions are relevant to them (are they interested in this type of impression), their bid (the value they are willing to pay for a relevant impression), and conflicts (which competing advertisers they refuse to appear concurrently with, for example, competing car makers for a car ad). Each time a user visits a webpage, the publisher's ad server considers the subset of relevant advertisers. It then orders the advertisers (say, by their bid) and greedily assigns an impression to an advertiser if it does not introduce a conflict (otherwise it skips this advertiser). From observing which advertisements are shown and which are not, and knowing which advertisers are relevant, can we learn the conflicts and priority ordering?

In this article, we consider this and several closely related problems. Formally, we assume there are n buyers (clients or advertisers) and a mechanism (the server or ad server) that has a static priority ordering over buyers that is unknown to us. There is a collection of m items, and the buyers each have utility functions over subsets of items (e.g., the examples above correspond to the case of single-minded buyers¹). At each time-step t , some set $S^t \subseteq \{1, \dots, n\}$ of buyers arrive. The mechanism then orders the buyers in S^t by priority and allocates to each its most-preferred bundle from the collection of items not yet given to earlier buyers in the ordering. Finally, we observe some function y^t of the outcome (allocation). We will consider the case that buyers are single minded, and y^t indicates that buyers received positive utility and that did not (as in the examples above), as well as the case that buyers are unit-demand or additive, and y^t indicates the items (if any) that each buyer received. Our goal will be to predict y^t from S^t , and we will present efficient algorithms that can do so while making only a bounded (polynomial in n and m) number of mistakes in total. Formally, we will be working in the *online mistake-bound model*, where on each round t we are given S^t and try to predict y^t . If our prediction is incorrect, then we are shown the true y^t and charged one *mistake*. Our goal is to design polynomial-time algorithms that have a worst-case number of mistakes (known as mistake bounds) that are as small as possible. We remark that we consider priority-based mechanisms, because they are particularly clean and have the desirable property that the computational problem of determining the allocation is easy when all the information is known. Priority-based mechanisms are also a natural generalization of the setting where the priority is fixed and known; our techniques also apply to the problem where

¹In the case of advertisers, each pairwise conflict can be modeled as an abstract item that belongs to the demand-set of both conflicting advertisers.

Type of buyers	Prices (fixed or variable)	Mistake Bound
Single-minded	fixed	$\Theta(n^2)$
Single-item	fixed	$\Theta(nm \log(m))$
Additive	fixed	$O(nm + n^2)$
Unit-demand	fixed	$O(n^2 m \log(m)), \Omega(nm \log(m))$
Additive	variable	$O(n^2 m \log(V))$
Unit-demand	variable	$O(n^2 m \log(V))$

Fig. 1. Summary of our results; V is the maximum value any buyer has for any item, and we assume $p_i^T \in [V]$ for all $i \in [m], t \in [T]$. The lower bounds are information theoretic. Our algorithm for single-minded buyers applies even to the case where observations are only which buyers get their set and which do not rather than the explicit allocation.

the priority varies day to day but is known to the person trying to uncover the preferences of the buyers participating.

Notice that the setting of single-minded buyers can exhibit significant non-monotonicities. For example, consider two clients a and b whose demand sets D_a and D_b do not overlap and with a having higher priority than b . It could be that a 's presence has no effect on b (since their sets do not overlap); it could be that a 's presence *helps* b (if there is a client c present, with priority between a and b , such that $D_a \cap D_c \neq \emptyset$ and $D_c \cap D_b \neq \emptyset$); or it could be that a 's presence *hurts* b (if there are clients d, e present with $D_a \cap D_d \neq \emptyset$, $D_d \cap D_e \neq \emptyset$, and $D_e \cap D_b \neq \emptyset$, with ordering $a > d > e > b$). Thus, this prediction problem requires knowledge of the interaction between demand sets of the entire set of clients present.

To get a feel for the type of results we are aiming for, we describe here a simpler case of this problem and how one can solve it (and, for an overview of the results, see Figure 1). Suppose buyers are additive rather than single minded,² and y^t denotes the *allocation* of items to the agents in S^t . This problem is monotonic in some sense: If $S^{t'} \subseteq S^t$ and $i \in S^{t'}$, then $y_i^{t'} \subseteq y_i^t$ (including more buyers reduces the allocation for i). It is possible to solve this problem by learning two things: First, for each buyer i , the set of items i has ever won, and, second, the relative ordering of the buyers $\succ$.

The algorithm for this simpler case works as follows. Estimate the set for i by the set of all items she has chosen in any round. Begin with some arbitrary estimated ordering $\succ$. To predict the allocation for a set S^t , we order buyers in S^t according to $\succ$, and in that order, give the buyers all of the remaining items she has bought before. If the algorithm makes a mistake, then consider the *first* buyer i according to $\succ$ whose allocation was incorrectly predicted. If item j was purchased by i but we did not predict that, then add j to i 's set of items. If we predicted that j was purchased by i but j was instead purchased by i' , then it must be that $i \succ i'$ but $i' > i$, so we must "demote" i relative to i' .

As described, the algorithm can make two types of mistakes (and we can limit the number of each). Consider the first buyer i (according to $\succ$) for whom we make a mistake in predicting her allocation. Suppose she won some item j that we did not predict she would get. The first type of mistake is one where there is some item j where $j \notin \hat{y}_i^t$ but $j \in y_i^t$. In this case, it must be that we were overly conservative in estimating i 's set of items for the following reason. Since she was the first mistake according to $\succ$, we did not predict that someone earlier in $\succ$ won item j . Thus,

²For each buyer i and each item j , either buyer i either wants j or she does not, and buyer i takes all items that she wants that are available when it is her turn. One can think of this as the behavior of additive buyers in the presence of fixed prices.

item j was available in our prediction when we reached buyer i : We did not allocate item j to her, because we had never seen her win item j before. There are at most nm of these sorts of mistakes to make (one per item/buyer pair). The second type of mistake is where we predicted that some item j would be allocated to buyer i but she did not get item j . In this case, we must have an error in our estimate of the ordering $\widehat{\succ}$, since we predicted item j for buyer i , buyer i must have won item j before (and is therefore interested in item j). Then, it must be the case that buyer i is later in the true ordering than in $\widehat{\succ}$: the winner of item j must be earlier than buyer i . Then, we can update $\widehat{\succ}$ by demoting buyer i : If done carefully, as we describe in Section 3, i will never be demoted further than her true position in the ordering, so there can be at most n^2 mistakes of this type.

1.1 Our Results

This article presents several mistake-bound learning algorithms for priority-ordered mechanisms. The crux of these algorithms is to learn the hidden *priority order*, or permutation over buyers, in a way that meshes well with learning the players' preferences at the same time, all in a mistake-bound framework. First, we consider the case without prices (or, equivalently, when prices are fixed across time). In the case of a single item, this problem reduces to learning the priority order over buyers. Previous work describes how to efficiently sample linear extensions of partial orders [5], which can be combined with a simple randomized weighted majority algorithm to learn a permutation with a mistake bound of $\Theta(n \log(n))$ when mistakes are accompanied with some pair i, j that were mis-ordered (see Section 6). When buyers have more general valuations, however, it is not clear how to use this algorithm to learn the priority order over buyers.³ We begin by presenting an algorithm for learning a permutation whose mistake bound is $\Theta(n^2)$ (when a mistake is accompanied by some element of the permutation that needs to be demoted rather than a pair of elements for which the permutation was incorrect). With this algorithm, we build mistake-bound learning algorithms for single-minded, unit-demand, and additive buyers with fixed prices, and for unit-demand and additive buyers with variable, observable prices. The precise form of these bounds is summarized in Figure 1, along with several information-theoretic lower bounds. The results for additive and unit-demand buyers also apply to the case where there are multiple copies of goods. Our results for single-minded buyers hold for a restricted model where our observations (and goal for prediction) are only the set of satisfied agents rather than the allocation. We also show that this restricted observation model is computationally intractable for *unit-demand* bidders under cryptographic assumptions by reducing to the problem of learning Boolean formulas [6].

1.2 Related Work

Our work is related to the literature on learning from revealed preferences [9, 10], which considers the problem of learning about a single buyer from observing her behavior under observed prices. Rationalizable demands (according to some prices) are those that can arise from maximizing some concave, monotone, continuous value function subject to a budget constraint. Beigman and Vohra [3] show that there exist continuous, nondecreasing, and concave utility functions that have unbounded query complexity; they give algorithms with finite sample complexity bounds for learning and predicting from rationalizable demand/price pairs for some special cases. Zadimoghaddam and Roth [11] gave computationally efficient versions of these algorithms for linear and linearly separable utilities. Amin et al. [1] consider both the problem of setting prices (minimizing regret

³In the case of a single item, we learn that the true winner has higher priority than everyone else. In general, mistakes do not give such a simple constraint on the ordering of buyers. When there are multiple items, a buyer may get item a or b , depending on who else shows up: One of these might be due to her preferences, and one might have to do with the availability of one of the items; it is not *a priori* clear which is the case.

w.r.t revenue) and the prediction problem (minimizing classification mistakes w.r.t. exogenous prices) for the online version of this problem. Balcan et al. [2] give tight sample complexity results for predicting linear, separable piecewise linear concave, CES, and Leontif preferences and extend the results to the agnostic setting. Their results extend to indivisible goods and certain nonlinear pricing.

Our work differs from the work on revealed preferences in two key ways. One is that in the settings we consider, each observed transaction involves multiple buyers: As a result, a buyer might not get some item because it was not in her desired bundle, or might not get it because the item (or a complement to it) was taken by some other buyer. The other is that we also, at the same time, are aiming to learn an unknown priority ordering for the seller. The algorithms we derive use as subroutines algorithms that are closely related to those for learning decision lists in a mistake bound setting [4]. We also employ a mistake bound learner for classification by halfspaces from Maass and Turán [7] for efficiently learning a classifier for a fixed unit-demand buyer.

2 MODEL AND PRELIMINARIES

Let B be a set of n buyers and I be a set of m items. Each buyer $i \in B$ has some combinatorial valuation v_i over subsets of items $T \subseteq I$. A *priority-ordered mechanism* $\mathcal{M}$ consists of an ordering over buyers $>$ and allocates items in I as follows. At each time t , an arbitrary subset S^t of the n buyers arrives online. Then, in order according to $>$, each buyer in S^t chooses the bundle from the remaining items that maximizes her value.⁴ So, the buyer i_1 in S^t who is first according to $>$ chooses the bundle $X_{i_1}^t \subseteq I$ of maximum value to her, then the buyer i_2 in S^t who is second according to $>$ chooses the bundle $X_{i_2}^t \subseteq I \setminus X_{i_1}^t$ of maximum value to her, and so on.

The *label* y^t for the example S^t is some function obs of the allocation $(X_1^t, \dots, X_n^t) = \mathcal{M}(S^t)$ that arises from this process. Our goal will be to predict $y^t = \text{obs}(\mathcal{M}(S^t))$ for a new subset S^t given our previous observations. We will focus on $\text{obs} = \text{Id}$ (the identity function; our goal is to predict the allocation) and $\text{obs} = (\mathbb{I}[X_1^t \neq \emptyset], \dots, \mathbb{I}[X_n^t \neq \emptyset])$ (the function that indicates which buyers bought at least one item; our goal is to predict which players have positive utility).

We will be working in the *mistake-bound model*, where our learning algorithm $\mathcal{M}$ will progress as follows. In each round, the algorithm is presented with a subset S^t . The algorithm's current prediction, $\mathcal{M}(S^t)$, will be output. Then, the algorithm observes the true label $y^t = \text{obs}(X_1^t, \dots, X_n^t)$. If $\mathcal{M}(S^t) \neq y^t$ (the label predicted is incorrect), then round t is counted as a *mistake*. The goal in mistake-bound learning is to bound the worst-case total number of mistakes made over an arbitrarily long sequence of examples presented to the algorithm. We will call this worst-case bound the *mistake bound for learning algorithm* $\mathcal{M}$. If one considers a randomized algorithm, then we measure the *expected* number of mistakes over the randomness the algorithm uses.

We also consider an extension of this setting, where each example is a pair (S^t, p^t) of a subset of buyers and a price vector $p^t \in \mathbb{R}^m$. Then, in order according to $>$, each buyer in S^t chooses a bundle from the remaining items maximizing her *utility*. We assume that their utility is *quasi-linear in money*, i.e., that $u_i(X_i^t, p^t) = v_i(X_i^t) - \sum_{j \in X_i^t} p^t(j)$ (where $X_i^t \subseteq I$ represents the bundle player i chose). We call this model the *variable-price model*; the previous model, described without prices, can be thought with a fixed price vector p that does not vary across examples. We also note that, for several of the problems we study, our algorithms extend to the case where for each item $e \in I$, there are k_e *copies* of that item (here, we assume buyers are unit-demand in each item, so they will never purchase more than one copy of a given item).

⁴We assume there are no ties or that ties are broken in a way consistent with some valuation that does not have ties.

3 SINGLE-MINDED BUYERS

Suppose there are n single-minded buyers. That is, each bidder i has a single demand set D_i for which she has value $v_i(D_i) > 0$. At each time t , a subset $S^t \subseteq [n]$ arrives, and each buyer in S^t is offered the items that have not been taken by earlier buyers. If buyer i is offered some set that includes D_i , she will take D_i ; otherwise, she will take nothing (one can imagine ϵ prices giving a disincentive to take excess items). Let W^t denote the set of buyers who won their demand sets in example S^t (so $W^t = \text{obs}(\mathcal{M}(S^t))$). From W^t and S^t alone, we wish to be able to predict the winning set $W^{t'}$ for a new subset $S^{t'}$, where our performance objective is the number of mistakes made.

Suppose for a moment our learning algorithm knew $>$ but not the demand sets D_i . From $>$ alone, we cannot predict sets $W^{t'}$ given $S^{t'}$: We need to understand which buyers have items in common between their demand sets (e.g., if $D_i \cap D_j \neq \emptyset$, then i and j will not simultaneously be in any winning set). For this reason, we call i and j *in conflict* if $D_i \cap D_j \neq \emptyset$. Consider the graph $G(V, E)$ where $V = [n]$ and there is an edge $(i, j) \in E$ iff buyers i and j have a conflict: We call this the *conflict graph*. This concept of a conflict graph arise in the setting of conflicting spectra; see, for example, Ramanathan [8]. For each pair of buyers (i, j) , if $i, j \in W^t$ for some S^t , then the two buyers clearly do not have a conflict. Notice that, even if $i > j$, there will be cases when $j \in W^t$ but $i \notin W^t$ (if i has a conflict, say, with an earlier winning buyer, but j does not). Thus, to predict whether buyer i will win, it is not sufficient to know simply who has precedence over i and who conflicts with i : We need more information about the structure of the conflict graph.

Notice that, given the conflict graph G and the ordering over buyers $>$, we can predict the winners W^t for an arbitrary subset S^t . In particular, buyer i wins exactly when no one before i won with whom i has a conflict. So, we can predict the entire winning set by scanning through S^t in the order $>$ and adding buyer i to W^t if no conflicting buyer is already in W^t . Thus, we are done if we can learn both G and $>$.

We start by presenting a mistake-bound procedure [PermULearn](#) (informally alluded to in the introduction) to learn a permutation $>$ in a model where each time it makes a mistake by guessing $\hat{>}^t$, it is told some item d^t that is incorrectly above some other item r in $\hat{>}^t$ (but is not told r). The procedure employs a data structure $P = (\hat{>}, O)$, where $\hat{>}$ is a permutation and O is a partition of the buyers to levels. Given a permutation σ , let $\text{Loc}(i, \sigma)$ denote the position of i in σ and let $\text{Buyer}(k, S, \sigma)$ denote the name of the k th element in σ restricted to S . The data structure at time t is P^t , and we will associate P^t with its output permutation $\hat{>}^t$, so that $\text{Loc}(i, P^t) = \text{Loc}(i, \hat{>}^t)$. The key interface with P^t will be the function $\text{Demote}(d^t, P^t)$, employed when d^t is returned as a mistake (an item that is incorrectly above some other item r in $\hat{>}^t$). The algorithm is quite similar to that used for learning decision lists in a mistake bound setting [4]. We use the following lemma throughout the rest of our analysis.

LEMMA 3.1. *[PermULearn](#) has a mistake-bound of $O(n^2)$ for learning a permutation $>$ if when $\hat{>}^t \neq >$, it is given some $d^t \in [n]$ such that $\exists r$ such that $d^t \hat{>}^t r$ but $r > d^t$.*

PROOF. Let $\text{Level}(i, O)$ be the level of buyer i in O , i.e., $\text{Level}(i, O) = j$ where $i \in O_j$. Let d^t be the element given (and demoted by [PermULearn](#)) at time t . We show by induction on the algorithm's pushing down elements that no d^t is pushed to a level below her location in $>$, i.e., $\text{Loc}(i, >) \geq \text{Level}(i, O^t)$. If this is the case, then the algorithm is correct: At most n^2 pushes can occur and the limit of these push-downs is some consistent permutation. Prior to any elements being pushed down, all elements are at the first level, hence $\text{Loc}(i, >) \geq \text{Level}(i, O^0)$ initially.

Now, suppose that the inductive hypothesis holds at time $t - 1$, i.e., for any item i we have $\text{Loc}(i, >) \geq \text{Level}(i, O^{t-1})$. Our induction hypothesis implies two things: first, that $\text{Loc}(d^t, >) \geq \text{Level}(d^t, O^{t-1})$ and, second, $\text{Loc}(r, >) \geq \text{Level}(r, O^{t-1})$.

Subroutine PermULearn: Maintains level ordering over $[n]$ and predicts permutation $\succsim^t$, an estimate of $\succ$. Given an error, receives $i \in [n]$ s.t. $\exists r$ s.t. $i \succsim^t r$ but $r \succ i$.

Let $\tilde{\succ}$ be an arbitrary ordering for tie breaking;

* InitPerm, put all items at level 1 *

$O_1 = \{1, \dots, n\} \quad O_{2, \dots, n} = \emptyset \quad \succsim^0 = \tilde{\succ};$

* Permutation(P^t), outputs consistent permutation *

for $l = 1$ to $|O|$ do

$\pi_l = \text{Order } O_l, \text{ level } l \text{ of } O, \text{ according to } \tilde{\succ};$

$\succsim^t = \pi_1 \cdot \pi_2 \cdot \dots \cdot \pi_l$ the concatenation of the levels' permutations;

* Demote(i, P^t), demote bidder i *

Let k be the level in which i resides, e.g., $i \in O_k$;

Let $O'_k = O_k \setminus \{i\}, O'_{k+1} = O_{k+1} \cup \{i\};$

Let $O' = (O_1, \dots, O_{k-1}, O'_k, O'_{k+1}, O_{k+1}, \dots);$

Let $P^{t+1} = (\succsim, O');$

$\succsim^{t+1} = \text{Permutation}(P^{t+1});$

Our assumption states that when d^t is demoted, there is some element r such that $\text{Loc}(d^t, \succ) > \text{Loc}(r, \succ)$ but $\text{Loc}(r, P^t) > \text{Loc}(d^t, P^t) \geq \text{Level}(d^t, O^{t-1})$. The second implication of our induction hypothesis states, prior to the t th demotion, $\text{Loc}(r, \succ) \geq \text{Level}(r, O^{t-1})$. Since $\text{Loc}(r, P^t) > \text{Loc}(d^t, P^t)$, it must be the case that $\text{Level}(r, O^{t-1}) \geq \text{Level}(d^t, O^{t-1})$ (r is only given a later location in P^t than d^t if she is at a weakly larger-numbered level). Thus, $\text{Loc}(d^t, \succ) > \text{Loc}(r, \succ) \geq \text{Level}(r, O^{t-1}) \geq \text{Level}(d^t, O^{t-1})$, so $\text{Loc}(d^t, \succ) > \text{Level}(d^t, O^{t-1})$ and pushing d^t to level $\text{Level}(d^t, O^{t-1}) + 1$ maintains the invariant. So the algorithm is correct, and no element is pushed more than n times, implying a mistake bound of n^2 . $\square$

Thus, [PermULearn](#) is guaranteed to learn a consistent ordering, so long as we never request a demotion of a buyer who should not be demoted. So, when we use this procedure, it suffices to show that we never demote a buyer i unless there is some buyer i' later in P^t but earlier in $\succ$ to guarantee that we learn a consistent ordering with at most $O(n^2)$ mistakes.

Before giving our main result, we first briefly mention that [PermULearn](#) is enough to learn in the simpler case that $\text{obs} = Id$ (we observe the entire allocation, not just who is in the winning set).

COROLLARY 3.2. *There is an efficient algorithm for predicting when $\text{obs} = Id$ (the allocation) for subsets of single-minded bidders with a mistake bound of $O(n^2)$.*

PROOF. Use an instantiation of the permutation data structure P as above. For each bidder i , let $\hat{D}_i = \emptyset$ initially. Whenever we see a bidder i win a nonempty set, set $\hat{D}_i = X_i = D_i$. When a subset S^t arrives, in order according to P , allocate $j \in S^t$ his set $\hat{D}_j$ if it is still available (otherwise, $\hat{X}_j = \emptyset$). When this allocation rule makes a mistake, consider i^t , the first bidder (according to the ordering P) for which our algorithm mis-allocated items. There are two possible mistakes: $\hat{X}_{i^t} = \emptyset$ but $X_{i^t} = D_{i^t}$, or $X_{i^t} = \emptyset$ but $\hat{X}_{i^t} = D_{i^t}$. The first case can occur for two reasons: We have never seen i^t win, in which case we have $\hat{D}_{i^t} = \emptyset$, so we then set $\hat{D}_{i^t} = D_{i^t}$ (there are at most n of these errors), or because there is some i' such that $i^t \succ i'$ but $i' \succsim^t i^t$ (who conflicts with i^t). But this is not possible, or i' would be an earlier mistake. The second kind of mistake can only occur because there is some i' such that $i' \succ i^t$ but $i^t \succsim^t i'$ (i^t is too early in the permutation). Thus, by [Lemma 3.1](#), demoting i^t in these cases is valid and leads to a mistake bound of $O(n^2)$. Thus, in total, there are $O(n^2)$ mistakes made by this algorithm. $\square$

It remains to show how we can use the permutation data structure to solve our original problem, that of learning $>$ alongside the conflict graph for single-minded buyers when we only observe the winning sets W^t . The intuition behind our main algorithm is as follows. We will initialize the permutation data structure and begin by assuming the conflict graph is the complete graph. For a given estimate $\widehat{>}$ and conflict graph $\widehat{G}$, we predict $\widehat{W}^t$ for S^t as follows. The mechanism serves members of S^t in order according to $\widehat{>}$ (e.g., $\text{Loc}(i, P^t) < \text{Loc}(j, P^t)$ will imply i gets served before j), subject to the constraint that if j is in conflict with some earlier buyer who has won, j does not win. Then, there will be two types of mistakes: when W^t includes some pair (i, j) connected by an edge in $\widehat{G}$, and when it does not. In the first case, we can safely remove (i, j) from $\widehat{G}$, and in the second case we will argue that we can safely demote some buyer. We will maintain the invariants alluded to previously: namely, that $E \subseteq \widehat{E}$ (for edges in the conflict/current estimate graph) and that we have never demoted a buyer who did not need to be demoted. Algorithm [SingleMinded](#) formalizes this intuition.

ALGORITHM SingleMinded: MB algorithm for predicting winners; single-minded buyers wrt order $>$

```

P=InitPerm;
Let  $\widehat{G} = ([n], \widehat{E})$  where  $(i, j) \in \widehat{E}$  for all  $i \neq j$ ;
for  $t = 1$  to  $T$  do
    Receive  $S^t$ ;
    Let  $\widehat{W}^t = \emptyset$ ;
    for  $b = 1$  to  $|S^t|$  do
        Let  $i = \text{Buyer}(b, S^t, P)$ ;
        add  $i$  to  $\widehat{W}^t$  if  $\nexists j \in \widehat{W}^t$  such that  $(i, j) \in \widehat{E}$ ;
    Predict  $\widehat{W}^t$ ;
    Learn  $W^t$ ;
    if  $W^t \neq \widehat{W}^t$  then
        if  $\exists i, j \in W^t$  such that  $(i, j) \in \widehat{E}$  then
             $\widehat{E} = \widehat{E} \setminus \{(i, j)\}$ 
        else
            Let  $i^t = \text{Buyer}(1, \widehat{W}^t \setminus W^t, P)$ ;
             $P = \text{Demote}(i^t, P)$ ;

```

THEOREM 3.1. [SingleMinded](#) is a $2n^2$ -mistake bound algorithm for predicting W^t , the winning set for single-minded buyers.

PROOF. Throughout the life of the algorithm, two invariants are maintained. First, the true set of edges in the conflict graph E will always be contained in $\widehat{E}$ the set of conflicts the algorithm tracks. Second, $\widehat{>}$ will only be told to push down a buyer i when there is some j such that $\text{Loc}(i, >) > \text{Loc}(j, >)$ but $\text{Loc}(i, P^t) < \text{Loc}(j, P^t)$.

We proceed to show the first invariant holds. It begins with the complete conflict graph $\widehat{G} = ([n], \widehat{E})$, so the invariant holds at the beginning. Whenever the algorithm deletes an edge (i, j) from $\widehat{G}$, an example has been observed where two buyers are clearly not in conflict (e.g., i and j are both allocated in some example).

Now, we prove the second invariant. This is clearly true when we push some i down the first time; i does not always win when he shows up, implying he is not at the first level according to

$>$. Now, suppose so far this has been the case: No buyer so far has been pushed unless there was proof according to $>$ that he is below someone below him according to $\succ^{t-1}$. Then, when i^t is asked to be pushed down, it is because i^t was not allocated to, even though $\succ^{t-1}$ said he should have been. This is not because of conflicts, by invariant 1, so i^t did not conflict with those above him according to $\succ^{t-1}$. Moreover, since i^t was the first person according to $\succ^{t-1}$ where we made a mistake, i^t must conflict with someone above him according to $>$, implying there is someone below him in $\succ^{t-1}$ who he is below in $>$. Thus, the invariant is maintained after i^t is demoted.

Thus, by Lemma 3.1, using the permutation data structure is appropriate: It is never told to push down some buyer who does not need to be lower according to $>$, so the algorithm is correct. Finally, there can be at most n^2 edges deleted from the conflict graph, and at most n^2 times where some buyer is pushed down in the ordering. Thus, the mistake bound on this algorithm is $2n^2$. $\square$

If we observe allocations, then predicting future allocations trivially reduces to the n^2 bound from the single-item case (it reduces to learning the priority of each buyer and seeing each buyer win one time). If, however, we only observe the subset of buyers who receive their desired bundle, and we aim to construct the smallest consistent model of the items corresponding to the demanded sets, then the problem becomes NP-complete.

OBSERVATION 3.2. *Given a conflict graph G , finding the smallest m for which single-minded bidders over m items suffices to describe the conflicts in G is equivalent to clique edge-cover, and is thus NP-complete. However, there will always exist a consistent set of at most n^2 items: in particular, one item for each edge in G with each player wanting all of its incident edges.*

4 UNIT-DEMAND BUYERS

Suppose now our n buyers are unit-demand, and we wish to predict the allocation rather than just the winning set. When prices are fixed, this problem corresponds to each buyer having an ordering over items $>_i$, as well as the ordering $>$ over buyers. At each time t , a subset S^t arrives, and the players in S^t , in order according to $>$, will each choose their favorite item remaining. For example, the first buyer in S^t will choose his favorite item, the second buyer will choose his favorite that the first buyer did not take, and so on. By a reduction to the single-minded case, we have the following.

THEOREM 4.1. *There is an efficient $O(n^2m^2)$ -mistake bound learning algorithm for predicting the allocation for subsets of buyers, according to an priority-ordered mechanism with fixed prices and unit-demand buyers, when observations are true allocations.*

The idea is to make m “ghost” buyers for each player i and use these to embed the preferences of i into the estimate of the ordering. Informally, each of the ghost buyers will represent i ’s preference for some good j . They will be ordered according to i ’s preferences over goods and will all be in conflict with one another. Furthermore, i_j will have conflicts with i'_j for all other i' (since not both i and i' can be allocated j). We now present the formal proof.

PROOF. Define $B' = \{i_j : i \in B, j \in I\}$, a set of buyers where each one corresponds to an item/buyer pair in the original problem. We construct a $>'$ and conflict graph $G = (B', E)$ such that solving this single-minded problem will encode the solution to our original unit-demand problem. Let $E = \{(i_j, i_{j' \neq j}) \cup \{(i_j, i'_j)\}$ (encoding that no buyer will buy multiple items, and no two buyers can both purchase the same item). Let $i_j >' i_{j'}$ if buyer $j >_i j'$ and $i_j >' i'_j$ if and only iff $i > i'$. Thus, each buyer’s m “ghosts” are in order of her preferences, and the buyers are ordered according to the mechanism’s preference ordering.

Run Algorithm [SingleMinded](#): If buyer i_j is satisfied, then predict $j = \mathcal{M}(S^t)_i$. When $\mathcal{M}(S^t) \neq y^t$, give the algorithm feedback $W^t = \{i_j | y^t_i = \{j\}\}$. Since only one of the m copies of a given buyer

ALGORITHM UnitDemandPrime: Predicts allocation of order-based allocation rule for unit-demand players

<pre> P = InitPerm(n) for i = 1 to m do Let $\hat{\succ}_i$ be an instantiation of the apx. randomized weighted majority alg. (Section 3) for learning $\succ_i$ over [m] For t = 1 to T: Let I = [m], receive S^t for b = 1 to S^t do Let i = Buyer(b, S^t, P) // the predicted choice of item by i Let $\hat{j}_i$ = Buyer(1, I, $\hat{\succ}_i$) Let $\hat{X}_i^t = \{\hat{j}_i\}$ and I = I \ $\{\hat{j}_i\}$ </pre>	<pre> Predict $\hat{X}^t$ Learn X^t if $X^t \neq \hat{X}^t$ then // ind of first error w.r.t. P Let $i^t = \text{FirstMistake}(X^t, \hat{X}^t, P)$ Let $\{j_{i^t}\} = X_{i^t}^t$ Give the constraint $j_{i^t} \succ_{i^t} \hat{j}_{i^t}$ to $\hat{\succ}_{i^t}$ if $\hat{\succ}_{i^t}$ is infeasible then Demote(i^t, P) Reset $\hat{\succ}_{i^t}$ </pre>
--	---

will be allocated to according to $\mathcal{M}$, we will never delete conflict edges between these copies, and will thus never predict two ghosts corresponding to the same true buyer will win simultaneously. Similarly, for any item j , since the item will never simultaneously be given to two buyers, we will never delete conflict edges corresponding to that item. Finally, for a player in position j according to $\succ$, no more than j of his ghost buyers will ever be seen winning. Thus, at most j of his ghosts will need to be rearranged by $\hat{\succ}$. In total, then, there are only $\min(n^2, nm)$ ghost buyers that are relevant. Then, there are at most $O(\min(n^4, n^2 m^2))$ mistakes, by Theorem 6.2. $\square$

We have a slightly tighter bound, which we discuss below.

THEOREM 4.2. *Algorithm UnitDemandPrime has a mistake bound of $\Theta(n^2 m \log(m))$ for predicting allocations for subsets of unit-demand buyers, when the $\mathcal{M}$ is a priority-ordered mechanism with fixed prices when the observation is the true allocation.*

PROOF OF THEOREM 4.2. We describe how to efficiently implement an approximate randomized weighted majority algorithm for learning permutations consistent with a partial order in Section 6. This implies the mistake bound given in the theorem, since one now might update the ordering over buyers $O(n^2)$ times (demotion can happen at most $O(n)$ times per buyer), and for each demotion we restart the $O(m \ln m)$ -mistake bounded algorithm for learning that buyer's preferences.

We now argue about the correctness of the algorithm. We can show that the algorithm maintains the invariant that the level of each bidder is never lower than their true level according to $\succ$, just as we did for PermULearn, by induction. The part of the argument we must complete is to show that a bidder is demoted only when she must have a lower rank in the ordering than her current level.

If i^t is the first mistake, then either i^t is lower in the ordering or our estimate of either the estimate of i^t 's preferences are incorrect. We always assume the latter and update our estimate of i^t 's preferences with the constraint that $j_{i^t} \succ_{i^t} \hat{j}_{i^t}$, until these constraints are no longer consistent for i^t . Once the constraints that updated i^t 's preferences are inconsistent, we have a proof that i^t cannot be at this point in the ordering over buyers. At this point, i^t must be demoted, since our inductive hypothesis implies i^t is only above her true position in the ordering. $\square$

This algorithm and result seem somewhat conservative: They only fix a single mistake when the algorithm might make many. Despite this, in Section 6, we show that this bound is tight up to

constants. Essentially, one cannot in the worst case assume there will be many ways to update for a given mistake. So, more aggressive policies might improve on certain mistakes, but they will not have improved mistake bounds.

4.1 $k_{i,t}$ -Demand

The previous results extend to cases where bidder i wishes to buy k_i items, rather than just 1, and chooses the top k_i with respect to her ordering of the remaining available items, even if the k_i s are not known in advance. The algorithm needs to maintain a conservative estimate of each k_i , so the initial estimate should be $\hat{k}_i = 0$ for all i . As before, when we make a mistake, we will only update the *first* buyer (according to our internal ordering over buyers) for whom our algorithm incorrectly predicted T rather than S . There will now be *two* types of mistakes: Where an agent buys a set S that is superset of T , the $\hat{k}_i$ items we predicted (in which case our algorithm should set $\hat{k}_i = |S|$), and when an agent buys a set S when we predicted T and $T \setminus S \neq \emptyset$ (in which case, we demote the *first* item in $T \setminus S$ according to our internal ordering over i 's items). The total mistake bound will increase by $\sum_i k_i$. Similarly, if at each time t , agent i desires $k_{i,t}$ items, and $k_{i,t}$ is part of the *input*, predicting that agent i will buy her $k_{i,t}$ favorite items, with the same update rule, will yield the same mistake bound as in the usual unit-demand case.

4.2 Multiple Copies

Several of these results are easy to extend to the setting where there are multiple copies of each resource, and players are unit-demand for multiple copies of a particular item (e.g., no player wants more than one copy of a given item). If buyers are additive (across bundles of different items), then it suffices to learn their preferences over item *types*. This can be done with no loss, treating any copy of a resource identically internal to the learning algorithm. For prediction, a buyer will take his favorite *bundle* of items, and that bundle can contain an item for which there is at least one copy remaining. This implies a mistake bound for learning the allocations that is *independent* of the number of copies of each item.

For unit-demand buyers, it is not clear how to use the solution from Theorem 4.1, which reduces to the single-minded case (for buyer i to not have item e available, there would need to be k_e “ghosts” that bought item e prior to buyer i , rather than a single conflict). However, Algorithm [UnitDemandPrime](#) can be used directly to learn the permutation over buyers and each buyer i 's preference order over item types.

For single-minded buyers, if we observe the allocations, then we need only uncover the preference ordering (and therefore only need to make $O(n^2)$ mistakes). Then Corollary 3.2 applies directly with no loss in the mistake bound when we have multiple copies. However, if we do not observe the allocation, and we only observe the identities of the people who received their set, the problem is trickier. We can construct a *conflict hypergraph* rather than a conflict graph to capture subsets of buyers who cannot all simultaneously receive their sets: For a given item with k_j copies, no more than k_j buyers who demand that item can be satisfied simultaneously, so any larger subset of buyers who demand that item will correspond to a hyperedge in the conflict hypergraph. The total number of edges in a necessary hypergraph blows up rather quickly: This representation (and thus the number of mistakes) has size $\Theta(n^k)$, where $k = \max_{j \in [m]} k_j$. Based on our previous observation about the complexity of finding a minimal representation (in terms of items) consistent with the perceived conflicts, even when there is only one copy of each item, we suspect this problem may be inherent. We leave the question open of whether one can predict the winning sets of single-minded buyers with a mistake bound and running time that is polynomial in m , n , and k (or even independent of k for the mistake bound).

5 VARIABLE PRICES: ADDITIVE AND UNIT-DEMAND

The previous sections can be thought of simulating a simple mechanism: According to the mechanism, buyers have priorities, and in order of priority, buyers will pick her favorite bundle available. For single-minded buyers, the priorities could be thought of as a sorting of buyers by their bid or some other pecking order. A buyer's preferences could be thought of as an ordering according to value or quasilinear utility according to some fixed prices. We now consider a twist on our original setup: What if, rather than the prices being fixed, each round was fed a price vector p^t along with the subset S^t , assuming that buyers would now take a bundle to maximize their quasilinear utility with respect to these varying prices? These prices might be set by varying exogeneous costs, some outside vendor, or some manager internal to an organization aiming to alter the use of different products. Assume that $p^t \in \{0, 1, \dots, V\}^m$. Simply running our previous mistake bound algorithm for unit-demand, additive, or single-minded buyers is tantalizingly simple. However, as buyer's preferences over bundles changes with prices (and thus rounds), our previous approaches need modification.

The algorithm that solves the fixed price problem for unit-demand buyers can be thought of in a slightly different way, which will be useful for solving the problems with variable prices. An equivalent solution to the problem is to start with a permutation data structure P to learn the ordering over buyers, and for each buyer i , a permutation data structure P_i to learn their preference ordering over items. Whenever a mistake is made, the algorithm blames the subroutine that is learning some buyer's preferences (namely, the earliest buyer for which we made a mistake). If this causes their subroutine to become infeasible, then it must be the case that the buyer needs to be demoted in the larger ordering, so we demote the buyer and restart their subroutine. Then, the total mistake bound for the algorithm will be $n^2 \mathcal{MB}$, where $\mathcal{MB}$ is the mistake bound for the subroutines (because each buyer can be demoted at most n times, and there are at most $\mathcal{MB}$ mistakes for a buyer at each position). This intuition (running a global algorithm with subroutines for each buyer's preferences) is our starting point for constructing mistake-bound learning algorithms for variable prices. We begin by designing an algorithm for additive buyers, which gives some intuition for the case of unit-demand. We assume, for simplicity, there are no ties for a buyer's most-preferred bundle at any set of prices. All results can be extended to allow ties assuming buyers break ties consistently.

Additive. Suppose in each round t , our input is a subset of buyers present S^t and a price vector p^t . The algorithm $\mathcal{A}$ we are trying to simulate is composed of two parts, $\succ$, a priority over buyers, and $v_i(j)$ for each buyer $i \in B$, and $j \in I$, corresponding to the value buyer i has for item j . On a given subset and price vector pair, $\mathcal{A}$ will offer all items to buyer $i \in S^t$ who is first in $\succ$, who will take all items such that $v_i(j) > p^t(j)$. Then, the remaining items are offered to the remaining buyers $i' \in S^t$, in order of $\succ$, who will do the same.

Our algorithm will predict an allocation of item $\widehat{X}_1^t, \dots, \widehat{X}_n^t$. If the allocation is wrong, then the correct allocation $X_1^t, \dots, X_n^t$ is shown to our algorithm. We wish, for arbitrary price vectors and subsets, to minimize the total number of errors made.

The argument of correctness for Algorithm [AdditiveVariable](#) (below) is somewhat more complex than in the previous sections. As before, we need to show that the algorithm never tells P to push down a buyer when it should not. The condition for this is slightly more complicated, however. That is, the infeasibility of the binary search for $v_i(j)$ is proof that this buyer cannot be above all of the buyers below him according to P . It is important that the first error according to $\text{Permutation}(P)$ (and *only* this error) is the one used to update the model: This avoids an earlier error in the ordering P_t (or an error in an earlier binary search $v_i(j)$) placing incorrect constraints

ALGORITHM AdditiveVariable: MB algorithm predicting $X_1^t, \dots, X_n^t$; additive buyers under order $>$

```

 $P = \text{InitPerm}(n);$ 
Let  $\widehat{v}_i(j) = V; \underline{v}_i(j) = 0;$ 
For  $t = 1$  to  $T$ :
    Receive  $S^t, p^t$ 
    Let  $\widehat{X}_i^t = \emptyset; I' = [m]$ 
    Let  $\widehat{v}_i^t(j) = \frac{\widehat{v}_i(j) + \underline{v}_i(j)}{2}$ 
    for  $b = 1$  to  $|S^t|$  do
        //  $b$ th-ranked buyer of  $S^t$  w.r.t  $P$ 
        Let  $i = \text{Buyer}(b, S^t, P)$ 
        Let  $X_{i^t}^t = \{j \in [m] \mid \widehat{v}_i(j) > p^t(j)\}$ 
        Let  $I' = I' \setminus X_{i^t}^t$ 
    Predict  $\widehat{X}_1^t, \dots, \widehat{X}_n^t$ 
    Learn  $X_1^t, \dots, X_n^t$ 

if  $X^t \neq \widehat{X}^t$  then
    // index of first mistake according to  $P$ 
    Let  $i^t = \text{FirstMistake}(X^t, \widehat{X}^t, P);$ 
    if  $\exists j \in X_{i^t}^t \Delta \widehat{X}_{i^t}^t$  such that  $\widehat{v}_{i^t}(j) = \underline{v}_{i^t}(j)$  then
        //  $\exists$  item whose binary search invalid
        Demote( $i^t, P$ ), Let  $\underline{v}_{i^t}(j) = 0$  and  $\widehat{v}_{i^t}(j) = V;$ 
    else
        // Update the binary searches.
        for each  $j \in \widehat{X}_{i^t}^t \setminus X_{i^t}^t$  do
            Set  $\widehat{v}_{i^t}(j) = \widehat{v}_{i^t}(j);$ 
        for each  $j \in X_{i^t}^t \setminus \widehat{X}_{i^t}^t$  do
            Set  $\underline{v}_{i^t}(j) = \widehat{v}_{i^t}(j);$ 

```

on lower buyers. To this end, let $\text{FirstMistake}(X, X', P)$ denote the first i , according to P , for which $X_i \neq X'_i$. Let V be the maximum valuation any buyer has for any item.

THEOREM 5.1. *Algorithm [AdditiveVariable](#) is an $O(\log(V)mn^2)$ mistake-bound learning algorithm for the problem of predicting subsets of quasilinear additive buyer's purchases according to $>$ priority with prices $p^t \in \{1, \dots, V\}^m$.*

Lemma 5.1 is the main component of Theorem 5.1; it implies that no agent is demoted unless we are sure her current position is too high and that we always make progress when a mistake is made.

LEMMA 5.1. *Let buyer $i^t \in B$ be the first mistake in round t . Then, there exists some item $j \in I$ for which one of these statements holds:*

- (1) $j \notin \widehat{X}_{i^t}^t$, but $j \in X_{i^t}^t$, and $\widehat{v}_{i^t}(j) < v_{i^t}(j)$
- (2) $j \in \widehat{X}_{i^t}^t$, but $j \notin X_{i^t}^t$, and $\widehat{v}_{i^t}(j) > v_{i^t}(j)$
- (3) $j \in \widehat{X}_{i^t}^t$, but $j \notin X_{i^t}^t$, and there is some i' such that $\text{Loc}(P^t, i^t) < \text{Loc}(P^t, i')$ but $\text{Loc}(P^t, i^t) > \text{Loc}(P^t, i')$.

PROOF OF LEMMA 5.1. A mistake implies there is some item j such that $j \notin \widehat{X}_{i^t}^t$ but $j \in X_{i^t}^t$ or $j \in \widehat{X}_{i^t}^t$ but $j \notin X_{i^t}^t$. Consider the first case. Our algorithm did not give item j to buyer i^t : Moreover, it did not give item j to some buyer i' with higher priority ($\text{Loc}(P^t, i') < \text{Loc}(P^t, i^t)$), since buyer i^t was the *first* mistake, implying our estimate of buyer i^t 's value for item j was too low, which falls into case 1. If, however, our algorithm allocated item j to buyer i^t , but buyer i^t was not awarded item j , then either buyer i^t 's value for item j is less than the price (and our estimate $\widehat{v}_{i^t}(j)$ was too high, implying case 2), or some buyer i' with higher priority w.r.t. $>$ took item j before buyer i^t . In this case, since buyer i^t was the first mistake, this implies $\text{Loc}(P^t, i^t) < \text{Loc}(P^t, i')$ but $\text{Loc}(P^t, i^t) > \text{Loc}(P^t, i')$, implying case 3. $\square$

We now prove Theorem 5.1.

PROOF OF THEOREM 5.1. As in previous proofs, we show our algorithm maintains two invariants:

- (1) For any buyer i , for each item j , at each time t , the binary search for $v_i(j)$ has been given only accurate upper and lower bounds for any permutation $\succ'$ such that i has not been demoted in $\succ'$ from $\widehat{\succ}^t$ (some other buyers may be demoted). So, if buyer i 's precedence does not decrease, any $v_i(j)$ s consistent with $\succ'$ and the observations are consistent with the binary searches and $\widehat{\succ}^t$.
- (2) Any i^t demoted in P_t has some i' s.t. $\text{Loc}(P^t, i^t) < \text{Loc}(P^t, i')$ but $\text{Loc}(\succ, i') < \text{Loc}(\succ, i^t)$.

We begin with the first invariant. It is satisfied prior to any constraints being added to any buyer's binary searches. Now, suppose it is true until time t : All constraints are accurate w.r.t. $\widehat{\succ}^t$ and any $\succ'$ such that i has not been demoted from $\widehat{\succ}^t$ to $\succ'$ but other buyers may have been demoted. If, at time t , i gets another constraint added to her binary searches, then this implies either this constraint is correct or buyer i needs to be demoted, by Lemma 5.1. Thus, if buyer i is not demoted (as is the case for $\succ'$), then this new constraint (and so all the constraints) in her binary searches are valid.

Now, we prove the second invariant. The invariant is true at the beginning of the algorithm prior to any buyer being pushed downwards. Now, consider some time t and assume this invariant holds until time t . The only case of interest is when one of buyer i^t 's binary searches is infeasible and she is demoted.

Due to invariant 1, we know that all the constraints buyer i^t has received until time t are valid at her position in $\widehat{\succ}^t$ (or any earlier position), since buyer i^t 's binary searches are reset whenever buyer i^t is pushed down. Since she is the first mistake, by Lemma 5.1, it is either the case that the current constraint being added is true with respect to her $v_{i^t}(j)$ s and her position (or an earlier position), or she must occur later in the ordering. Thus, all the constraints in her binary searches are correct with respect to her current position (or any earlier one) in the ordering. Since there are some $v_{i^t}(j)$ s that are consistent with the observations, but not the set of constraints, it must be the case that buyer i^t occurs later in the ordering.

Now, we show the mistake bound. If a mistake is made, then some buyer i^t either updates her binary searches or is demoted. At most $\log(V)$ binary search updates can occur for a given item and buyer before the binary search becomes infeasible and she is demoted. Thus, there can be at most $m \log(V)$ mistakes resulting in binary search updates for a buyer before she is demoted. By Lemma 3.1 and invariant 2, no buyer is pushed later in the ordering than she occurs in $\succ$; thus, there are at most n^2 mistakes resulting in demotions. Thus, in total, there are at most $n^2 m \log(V)$ many mistakes. $\square$

Unit-Demand. The case of unit-demand buyers is similar to that of additive buyers, though the buyers will no longer have *separable* preferences over items: Instead, of a set of available items T at prices p^t , buyer i will buy $j = \arg\max_{j \in J} [v_i(j) - p^t(j)]$ to maximize his quasilinear utility (assume that there is some consistent tie-breaking in the event that several items are equally good). So rather than using binary search for each item separately, for each buyer, we will run a mistake bound ellipsoid algorithm; whenever a constraint is added, it will be of the form $v_i(j) - p^t(j) > v_i(j') - p^t(j')$, where the $v_i(j)$ s are variables and the $p^t(j)$ s are constants coming from the online price vectors.

THEOREM 5.2. *Algorithm UnitVariable is an $O(n^2 m \log(V))$ -mistake bound learner for unit-demand buyers with respect to some order $\succ$, assuming prices are integral and at most V .*

The main theorem of this section follows from a similar analysis to that of additive buyers in the previous section, with a twist stemming from the fact that we use the Ellipsoid algorithm as the mistake-bound subroutine (with each mistake serving as its separation oracle) rather than binary

ALGORITHM UnitVariable: MB algorithm predicting $X_1^t, \dots, X_n^t$; unit-demand buyers, order $>$

```

 $P = \text{InitPerm}(n)$ 
Let  $\widehat{R}_i$  be an instance of ellipsoid alg. w. unknowns
 $v_i(j)$  for all  $i \in B, j \in I$ 
For  $t = 1$  to  $T$ :
    Receive  $S^t, p^t$ 
    Let  $\widehat{X}_i^t = \emptyset$  and  $I' = [m]$ 
    for  $b = 1$  to  $|S^t|$  do
        Let  $i = \text{Buyer}(b, S^t, P)$ 
        Let  $\widehat{v}_i \in \mathbb{R}^m$  be the center estimated by  $\widehat{R}_i$ 
        // prediction for  $i$  w.r.t. prices,
        // est. values, rem. items
        Let  $\widehat{j}_i = \arg\max_{j \in I'} \widehat{v}_i(j) - p^t(j)$ 
        if  $\widehat{v}_i(\widehat{j}_i) - p^t(\widehat{j}_i) > 0$  then
            | Let  $X_i^t = \{\widehat{j}_i\}$ 
        else
            | Let  $X_i^t = \emptyset$ 
        | Let  $I' = I' \setminus X_i^t$ 
    Predict  $\widehat{X}_1^t, \dots, \widehat{X}_n^t$ 
    Learn  $X_1^t, \dots, X_n^t$ 

if  $X \neq \widehat{X}$  then
    Let  $i^t = \text{FirstMistake}(X^t, \widehat{X}^t, P)$ 
    if  $X_{i^t}^t = \emptyset$  then
        | Send constraint  $v_{i^t}(\widehat{j}_{i^t}) < p^t(j_{i^t})$ 
        | to  $\widehat{R}_{i^t}$ 
    else
        | Let  $\{j_{i^t}\} = X_{i^t}^t$ ; // The item  $i^t$  actually won
        | if  $\widehat{X}_{i^t}^t = \emptyset$  then
            | | Send constraint  $v_{i^t}(j_{i^t}) > p^t(j_{i^t})$  to  $\widehat{R}_{i^t}$ 
        | else
            | | Let  $\widehat{X}_{i^t}^t = \{\widehat{j}_{i^t}\}$ 
            | | Send constraint  $v_{i^t}(j_{i^t}) - v_{i^t}(\widehat{j}_{i^t}) > p^t(j_{i^t}) - p^t(\widehat{j}_{i^t})$  to  $\widehat{R}_{i^t}$ 
        | if  $\widehat{R}_{i^t}$  is infeasible then
            | | Demote( $i^t, P$ ) and restart  $\widehat{R}_{i^t}$ 

```

search for each item separately. This is similar to the use of the Ellipsoid algorithm by Maass and Turán [7] for learning a linear separator. The proof of Theorem 5.2 can be found in Section A.1.

6 LOWER BOUNDS AND HARDNESS

Given the results in the previous section, it is natural to ask what mistake bounds cannot be achieved for these problems. This section addresses this question: We show information-theoretic lower bounds on the number of mistakes for several of these problems. For the case of single-minded bidders, our lower bound is matching the mistake bound we prove for an efficient algorithm. For unit-demand bidders with fixed prices, our lower bound is within a factor of n of the upper bound we have shown in the previous section using an efficient algorithm. We mention a computationally inefficient algorithm that achieves the information-theoretic optimal mistake bound and leave it as an open question whether a polynomial-time algorithm can achieve the same mistake bound. Finally, we show cryptographic hardness of a related problem. Suppose bidders are unit-demand, and for each bidder i has a set of satisfying items, and it is our goal to predict the set of agents who received one of their satisfying items. When our algorithm makes a mistake, it is presented with the set of satisfied agents rather than the allocation. We show, by the reductions of Kearns and Valiant [6], that this problem is at least as hard as several encryption problems that are generally accepted as computationally hard problems, even in the average case. We now state our information-theoretic lower bound for predicting the winning bidders for single-minded bidders.

THEOREM 6.1. *Suppose bidders are single minded, and we predict and observe only which subset $W^t \subseteq S^t$ wins in round t . Then, when $m \geq n^2$, any online algorithm can be forced to make $\Omega(n^2)$ mistakes.*

PROOF. Suppose there are $\frac{n(n-1)}{2}$ items (one for each pair of buyers). In each round t , the adversary presents the algorithm with S^t containing a pair of buyers that has never been presented

before. The algorithm needs to predict whether one or both of the buyers will be satisfied (guessing whether both buyers are both interested in their “shared” item or not). Regardless of the algorithm’s choice, the adversary will say that was a mistake: This yields a consistent set of conflicts and will force the algorithm to make $\Omega(n^2)$ mistakes. $\square$

However, there may be room for improving our unit-demand results. As a warm-up, we first present an upper and lower bound for the single item case (where we *do* have an efficient algorithm for the matching the upper bound).

THEOREM 6.2. *The problem of learning the allocation made by a priority-ordered mechanism with fixed prices and a single item has a mistake bound $M = \Theta(n \log(n))$, and there is an algorithm with this mistake bound that runs in polynomial time.*

PROOF. For the lower bound, an adversary can present subsets of size 2 and essentially simulate merge-sort. To start, for $i = 1, \dots, n/2$, the adversary presents subset $\{2i - 1, 2i\}$, and tells the algorithm it has made a mistake (regardless of its prediction), causing $n/2$ mistakes. In general, given n/L sorted lists of size L , the adversary pairs the lists together and then for each pair runs through the merging process (presenting the subset consisting of the top element in each list, telling the algorithm it has made a mistake whatever its prediction is, and popping off the true largest element). This maintains consistency with an overall ordering and creates at least L mistakes per pair, or again $n/2$ mistakes total for the round. There are $\log(n)$ rounds, leading to an overall lower bound of $\Omega(n \log n)$.

We can construct a computationally efficient algorithm that matches this information-theoretic lower bound using two ideas. First, each mistake gives us a new pair of agents (i, j) for which we learn $i > j$ but $\text{Loc}(i, P^t) < \text{Loc}(j, P^t)$ (the true winner i has higher priority than every other $i' \in S^t$, and in particular, the estimated winner j). Second, Karzanov and Khachiyan [5] given an efficient sampling algorithm that samples uniformly a consistent linear extension of a partial order.

Then, consider the following prediction algorithm. Consider a new subset S^t . Take a single sample $>_\gamma$ using the algorithm of Karzanov and Khachiyan [5], and predict the winner is $j^t = \text{Buyer}(1, S^t, >_\gamma)$. If a mistake is made, and i^t is the winner, then add the set of constraints $i^t > j$ for all $j \in S^t$ to the partial order. We claim that each constraint added to the partial order over the life of the algorithm is correct: A mistake is proof of the constraint we add.

We now measure the expected number of mistakes we have, using an analysis similar to analyzing randomized weighted majority. Call the number of remaining consistent orderings in round t w_t , and f_t the probability we make a mistake in round t . Let R be the initial number of consistent orderings. Let M represent the expected number of mistakes made by our algorithm. Notice that $w_T = R \cdot (1 - f_1)(1 - f_2) \dots$, so

$$\ln(w_T) = \ln R + \sum_t \ln(1 - f_t) \leq \ln R - \sum_t f_t = \ln R - M,$$

since $\ln(1 - c) \leq -c$. Since some ordering is consistent will all of these constraints, we know that $w_T \geq 1$, and so $M \leq \ln R$. Thus, since there are $O(n!)$ orderings, this upper bounds M by $\Theta(n \log(n))$. $\square$

The case of unit-demand buyers also has matching information-theoretic bounds.

THEOREM 6.3. *The problem of learning the allocation made by a priority-ordered mechanism with fixed prices with over unit-demand buyers has a mistake bound of $\Theta(mn \log(m))$ (assuming $m = \Omega(\log(n))$).*

PROOF. The lower bound for this problem is similar to the previous argument. The generalization uses n buyers, the first m of which are “dummy” buyers and have favorite items $a_1, \dots, a_m$.

We can use these first buyers to control which items are available for the true buyers. Then, each example S^t will contain $m - 2$ “dummy” buyers (who take all but just 2 items a_f, a_g) and one true buyer i . Then, the algorithm needs to decide which of a_f or a_g the true buyer will select. This will be repeated for each pair of items and each non-dummy buyer. Thus, the algorithm is solving $n - m$ separate instances of sorting m items (for each buyer), and so the problem has a lower bound of $\Omega(mn \log(m))$ mistakes. $\square$

Without computational constraints, we can construct an algorithm with a matching mistake bound for unit-demand bidders. The algorithm will maintain a list of consistent permutations over buyers (and, for each of those permutations over buyers, the consistent permutations for each buyer over items). Given a new subset S^t , the algorithm predicts the allocation most consistent permutations agree on. Since there are $n! \cdot (m!)^n$ many initial hypotheses (an ordering over buyers and, for each buyer, an ordering over items), this weighted majority algorithm will make $O(n \log(n) + nm \log(m))$ mistakes. It is not clear how to make this algorithm computationally efficient without increasing the mistake bound: Unlike the single-item case, we have no clear culprit to our mistake. In the single-item case, we can add another constraint to our partial order, generating a refined partial order. In the unit-demand case, a mistake could be made either because the understanding of some individual's preferences are wrong or because our ordering over buyers was wrong.

Unit-demand Bidders with Acceptable Items Is Hard. Finally, we show hardness for a variant of the single-minded model (where we only observe which agents received their demanded sets, not the allocation itself). Suppose rather than agent i being single minded with demand set D_i , agent i is unit-demand, with preference order $\succ_i$ and an *acceptable set* D_i . If agent i is offered a set of items I , then they will pick the item $j \in D_i \cap I$ (if the intersection is nonempty) that is best according to $\succ_i$. If agent i receives any item $j \in D_i$, then we will say agent i is satisfied. Our goal is to predict the satisfied set W^t from S^t , the set of agents who arrive at time t . If a mistake is made, then the algorithm is presented with the correct satisfied set (rather than the actual allocation). We call this problem the *unit-demand satisfaction prediction problem*.

THEOREM 6.4. *Weakly learning general Boolean formulae of size n is polynomial-time reducible to the unit-demand satisfaction problem.*

Informally, one can encode a Boolean formula (in its binary tree representation with and/or gates as internal nodes, and the leaves are inputs to the formula, the root's value is the value of the formula evaluated on its leaves) into an instance of the problem of predicting the set of satisfied unit-demand buyers. The intuitive correspondence is that each node will correspond to an item: That node will evaluate to true if and only if the corresponding item is not chosen by some agent.

Each leaf node in a Boolean formula will have a corresponding agent and item (that agent wants exactly that item). Each AND gate will introduce four bidders and four items. Two of the buyers each prefers one of two “input” items to the AND gate, then one of the non-input items associated with the gate. The other two buyers are each satisfied by only one of the non-input items corresponding to the gate; the preference ordering gives higher priority to the first two buyers. These latter two buyers will be satisfied if and only if both inputs to the AND gate were true. Each OR gate introduces two additional bidders and three items; the first agent will choose either of the input items, then the third non-input item; the other bidder is satisfied only by the non-input item. If at least one of the input items is available, then both agents will be satisfied.

PROOF OF THEOREM 6.4. Consider a Boolean formula B , and a representation of B as a binary tree T whose internal nodes are and/or gates: The inputs are the leaves of the tree and the evaluation of

the tree is the value of the root with respect to the leaves' inputs. We want to show that the evaluation of the root of this tree is at least as hard as predicting the satisfied set of unit-demand bidders for some corresponding set of agents and preferences. The intuitive correspondence is that each node will correspond to an item: That node will evaluate to true if and only if the corresponding item is not chosen by some agent.

We now proceed with the construction. For each leaf ℓ node in T , let there be an agent i_ℓ whose demand set $D_{i_\ell} = \{j_\ell\}$ is exactly the item corresponding to ℓ . Thus, we can set each of the leaves to true or false (if a leaf agent appears, that leaf is false, otherwise the leaf is true). For the root r of T , there is an agent i_r whose only satisfying item is j_r : She is happy if and only if the root is available (evaluates to true). Next, we show how to simulate AND/OR gates within the tree.

For each AND(u_1, u_2) gate g in T , let there be an agent i_{g_1} who finds j_{u_1}, j_g acceptable with preference order $j_{u_1} \succ_{i_{g_1}} j_g$, and an agent i_{g_2} who finds items j_{u_2}, j_g, j_x acceptable whose ordering is $j_{u_2} \succ_{i_{g_2}} j_g$. Then, if u_1 or u_2 are taken by other agents, either agent i_{g_1} or i_{g_2} will select item j_g , making it unavailable (which causes the node to evaluate to false). If both j_{u_1}, j_{u_2} are available, then the agents will take those items and leave j_g available, causing the node to evaluate to true.

For each OR(u_1, u_2) gate g in T , let there be an agent i_g whose satisfying items are j_{u_1}, j_{u_2} , and j_g , and $j_{u_1} \succ_{i_g} j_{u_2} \succ_{i_g} j_g$. Thus, if either of the items corresponding to u_1 or u_2 is available, agent i_g will select one of them and item j_g will remain available, causing the node to evaluate to true. However, if both u_1, u_2 are unavailable (set to false), then agent i_g will take item j_g , causing the node to evaluate to false.

Thus, to predict whether i_r is in the satisfied set, one must evaluate the binary tree correctly. So, the problem of predicting the winning set of unit-demand bidders can solve the problem of weakly learning general Boolean formulae. $\square$

Thus, by the reduction of Kearns and Valiant [6], there is a polynomial $p(n)$ such that inverting the RSA encryption function, recognizing quadratic residues, and factoring Blum integers are probabilistically polynomial-time reducible to the unit-demand satisfaction problem.

7 DISCUSSION

In this article, we present algorithms that, based only on observations of opaque transactions (observing just who wins and who does not in the case of single-minded buyers, or observing the allocations produced in the case of additive or unit-demand buyers), can reconstruct both the preferences of buyers and the mechanism used by the seller sufficiently well to predict the outcomes of new transactions. We focus on priority-based *ordered priority mechanisms* on the side of the seller and commonly-studied classes of valuation functions for the buyers. It would be interesting to consider this problem in the context of other mechanisms and other observation models as well. Note that for mechanisms such as VCG (producing a social-welfare-maximizing allocation) certain complications arise: For instance, even in the case that all buyer valuations are known, finding the allocation produced can be NP-complete if buyers are single minded. Therefore, if one wishes to solve the prediction problem with efficient algorithms, it is necessary to consider settings where the problem of computing the allocation with full knowledge of the mechanism and preferences can be done in polynomial time.

A APPENDIX

A.1 Variable Prices for Unit-Demand Bidders

LEMMA A.1. *Using the ellipsoid algorithm to learn the collection $\{v_i(j)\}_j$ has an expected mistake bound of $O(m \log V)$ so long as each mistake returns a constraint such that its current hypothesis $\hat{v}_i$ is no longer feasible, assuming $v_i \in V^m$.*

PROOF. We will have m variables corresponding to the valuations of buyer i to each of the m items, and will run a separate instance of the Ellipsoid algorithm for each buyer. The Ellipsoid algorithm maintains an ellipsoid that contains the feasible region (the possible valuations consistent with observations so far, representable in $\mathbb{R}^m$) and proposes as its current hypothesis the center of that ellipsoid.

We use this center as a proposed valuation for buyer i until we make an error involving her. Once we make an error we identify a violated linear constraint: namely, that $v_i(j) - p_j > v_i(j') - p(j')$ for the j' we predicted when i bought j , and we return it to the Ellipsoid algorithm, which then updates its ellipsoid and hypothesis.

In each iteration (mistake of the algorithm) the volume shrinks multiplicatively by a fraction of $1 - \frac{1}{e}$. The initial feasible region fits inside $B(0, V\sqrt{m})$. Moreover, we can halt when our ellipsoid fits in $B(0, 1/2\sqrt{m})$, since at that point all valuations within the Ellipsoid will make the same purchasing prediction for every pricing (since $v_{i,j} - v'_{i,j} \leq \frac{1}{2}$ for all j and the prices are integral). Thus, there will be $O(m \log(V))$ possible mistakes made before the ellipsoid feasibility oracle reports that the current set of constraints is infeasible. $\square$

We now state the analog to Lemma 5.1 for the unit-demand case.

LEMMA A.2. *Suppose the unit-demand algorithm makes a mistake at time t . Let i^t be the first mistake (according to $\hat{>}^t$). Let $\hat{j}_{i^t}$ be the item we predicted i^t to win (if any) and j_{i^t} the item i^t won (if any). Then one of these holds:*

- (1) $v_{i^t}(\hat{j}_{i^t}) - p^t(\hat{j}_{i^t}) < 0 < \widehat{v}_{i^t}(\hat{j}_{i^t}) - p^t(\hat{j}_{i^t})$, or $v_{i^t}(\hat{j}_{i^t}) < \widehat{v}_{i^t}(\hat{j}_{i^t})$
- (2) $v_{i^t}(\hat{j}_{i^t}) - p^t(\hat{j}_{i^t}) < v_{i^t}(j_{i^t}) - p^t(j_{i^t})$
- (3) $v_{i^t}(j_{i^t}) > \widehat{v}_{i^t}(j_{i^t})$
- (4) $\hat{j}_{i^t}$ was not available ($\exists i' \text{ s.t. } \text{Loc}(P^t, i^t) < \text{Loc}(P^t, i') \text{ but } \text{Loc}(\widehat{>}^t, i^t) > \text{Loc}(\widehat{>}^t, i')$).

PROOF. Consider a mistake on buyer i^t . Either it is the case that (a) i^t bought nothing and we predicted she bought something, (b) she bought something and we predicted nothing, or (c) we predicted the wrong item.

Case (a) occurs only when $\hat{j}$ was no longer available (i^t needs to be demoted, case 4) or $\hat{j}$ was too expensive, $v_{i^t}(\hat{j}) - p^t(\hat{j}) < 0$ (case 1). Case (b) can only occur because our estimate of her value of an item was too small (case 3), since she is the first mistake it cannot be because we predicted that someone earlier took j . Case (c) occurs when either $\hat{j}$ was not available (and i needs a demotion, case 4) or our estimate of utility was wrong (case 2). $\square$

PROOF OF THEOREM 3.2. We claim the same two invariants are true of Algorithm UnitVariable as were true of Algorithm AdditiveVariable, since Lemma A.2 provides the analogous guarantees (namely, that when we make a mistake, we either get to add a constraint to some ellipsoid algorithm, or we get to demote some buyer). Thus, the algorithm is correct. Each instantiation of the ellipsoid algorithm makes at most $m \log(V)$ mistakes before we demote and restart, and there are at most n^2 demotions total. Thus, at most $n^2 m \log(V)$ mistakes in total are made. $\square$

REFERENCES

- [1] Kareem Amin, Rachel Cummings, Lili Dworkin, Michael Kearns, and Aaron Roth. 2014. Online learning and profit maximization from revealed preferences. *arXiv* (2014).
- [2] Maria-Florina Balcan, Amit Daniely, Ruta Mehta, Ruth Uner, and Vijay V. Vazirani. 2014. Learning economic parameters from revealed preferences. *arXiv abs/1407.7937* (2014).
- [3] Eyal Beigman and Rakesh Vohra. 2006. Learning from revealed preference. In *Proceedings of the 7th ACM Conference on Electronic Commerce*. ACM, 36–42.

- [4] D. Helmbold, R. Sloan, and M. K. Warmuth. 1990. Learning nested differences of intersection closed concept classes. *Mach. Learn.* 5, 2 (1990), 165–196. Special Issue on Computational Learning Theory; first appeared in 2nd COLT conference (1989).
- [5] Alexander Karzanov and Leonid Khachiyan. 1991. On the conductance of order Markov chains. *Order* 8, 1 (1991), 7–15.
- [6] Michael Kearns and Leslie Valiant. 1994. Cryptographic limitations on learning Boolean formulae and finite automata. *J. ACM (JACM)* 41, 1 (1994), 67–95.
- [7] Wolfgang Maass and György Turán. 1990. *How Fast Can a Threshold Gate Learn?* International Computer Science Institute.
- [8] Subramanian Ramanathan. 1999. A unified framework and algorithm for channel assignment in wireless networks. *Wireless Netw.* 5, 2 (1999), 81–94.
- [9] Paul A. Samuelson. 1938. A note on the pure theory of consumer's behaviour. *Economica* (1938), 61–71.
- [10] Hal R. Varian. 2006. Revealed preference. *Samuelsonian Economics and the Twenty-first Century* (2006), 99–115.
- [11] Morteza Zadimoghaddam and Aaron Roth. 2012. Efficiently learning from revealed preference. In *Internet and Network Economics*. Springer, Berlin, 114–127.

Received January 2016; revised April 2018; accepted May 2018