Learning from Rational Behavior: 
Predicting Solutions to Unknown Linear Programs

Shahin Jabbari¹, Ryan Rogers², Aaron Roth¹, and Zhiwei Steven Wu¹

¹ Computer and Information Science Department, University of Pennsylvania
{jabbari, aroth, wuzhiwei}@cis.upenn.edu
² Applied Math and Computational Sciences Department, University of Pennsylvania
ryrogers@sas.upenn.edu

Abstract. We define and study the problem of predicting the solution to a linear program, given only partial information about its objective and constraints. This generalizes the problem of learning to predict the purchasing behavior of a rational agent who has an unknown objective function, which has been studied under the name “Learning from Revealed Preferences”. We give mistake bound learning algorithms in two settings: in the first, the objective of the linear program is known to the learner, but there is an arbitrary, fixed set of constraints which are unknown. Each example given to the learner is defined by an additional, known constraint, and the goal of the learner is to predict the optimal solution of the linear program given the union of the known and unknown constraints. This models, among other things, the problem of predicting the behavior of a rational agent whose goals are known, but whose resources are unknown. In the second setting, the objective of the linear program is unknown, and changing in a controlled way. The constraints of the linear program may also change every day, but are known. An example is given by a set of constraints and partial information about the objective, and the task of the learner is again to predict the optimal solution of the partially known linear program.

1 Introduction

We initiate the systematic study of a general class of multi-dimensional prediction problems, where the learner wishes to predict the solution to an unknown linear programming problem, given some partial information about either the set of constraints or the objective. In the special case in which there is a single known constraint that is changing (specified by the example), and the objective is unknown and fixed, this problem has been studied under the name learning from revealed preferences [1,2,3,15]. This case captures the following scenario: a buyer, with an unknown linear utility function over d goods $u : [0, 1]^d \rightarrow \mathbb{R}$ defined as $u(x) = c \cdot x$ faces a purchasing decision every day. Each day $t$, he observes a set of prices $p^t \in \mathbb{R}^d_{\geq 0}$, and buys the bundle of goods that maximizes his unknown utility function, subject to a budget $b$:

$$x(t) = \arg \max_{x \in [0, 1]^d} c \cdot x \quad \text{such that } p^t \cdot x \leq b$$

In this problem, the goal of the learner is to predict the bundle that the buyer will buy, given the prices that he faces. Each example at day $t$ is specified by the vector $p^t \in \mathbb{R}^d_{\geq 0}$ (which fixes the constraint), and the goal is to accurately predict the purchased bundle $x(t) \in [0, 1]^d$ that is the result of optimizing the unknown linear objective.

It is also natural to consider the class of problems in which the learner’s goal is to predict the outcome to a linear program more broadly. For example, suppose the objective $c \cdot x$ is known, but there is an unknown set of constraints $Ax \leq b$. An instance is again specified by a changing known constraint $(p^t, b^t)$, and the goal is to predict:

$$x(t) = \arg \max_{x} c \cdot x \quad \text{such that } Ax \leq b \quad \text{and } p^t \cdot x \leq b^t.$$  (1)
This models the problem of predicting the behavior of an agent whose goals are known, but whose resource constraints are unknown.\(^3\)

Another natural generalization is the problem in which the objective is unknown, and may vary in a specified way across examples, and in which there may also be multiple arbitrary known constraints which vary across examples. Specifically, suppose that there are \(n\) distinct, unknown linear objective functions \(v^1, \ldots, v^n\). An instance is specified by a subset of the unknown objective functions, \(S^t \subseteq [n] := \{1, \ldots, n\}\), and a convex feasible region \(P^t\), and the goal is to predict:

\[
x^{(t)} = \arg\max_x \sum_{i \in S^t} v^i \cdot x \quad \text{such that } x \in P^t.
\]  

(2)

When the changing feasible regions \(P^t\) correspond simply to varying prices as in the revealed preferences problem, this models a setting in which at different times, purchasing decisions are made by different members of an organization, with heterogeneous preferences — but are still bound by an organization-wide budget. The learner’s problem is, given the subset of decision makers at day \(t\) and the prices at day \(t\), to predict which bundle they will purchase. This generalizes some of the preference learning problems recently studied by Blum et al [6]. Of course, in this generality, we may also consider a richer set of changing constraints which represent things beyond prices and budgets.

In all of the settings we study, the problem can be viewed as the task of predicting the behavior of a rational decision maker, who always chooses the action that maximizes her objective function subject to a set of constraints. Some part of her optimization problem is unknown, and the goal is to learn, through observing her behavior, that unknown part of her optimization problem sufficiently so that we may reliably predict her future actions.

### 1.1 Our Results

We study both variants of the problem (specified below) in the strong mistake bound model of learning [13]. In this model, the learner encounters an arbitrary adversarially chosen sequence of examples online and must make a prediction for the optimal solution in each example before seeing future examples. Whenever the learner’s prediction is incorrect, the learner encounters a mistake, and the goal is to prove an upper bound on the number of mistakes the learner can make, in the worst case over the sequence of examples. Mistake bound learnability is stronger than (and implies) PAC learnability.

**Known Objective & Unknown Constraints** We first study this problem under the assumption that there is a uniform upper bound on the number of bits of precision used to specify the constraint defining each example. In this case, we show that there is a learning algorithm with both running time and mistake bound linear in the number of edges of the polytope formed by the unknown constraint matrix \(Ax \leq b\). We note that this is always polynomial in the dimension \(d\) when the number of unknown constraints is at most \(d + O(1)\) (In Appendix D, we show that by allowing the learner to run in time exponential in \(d\), we can give a mistake bound that is always linear in the dimension and the number of rows of \(A\), but we leave as an open question whether or not this mistake bound can be achieved by an efficient algorithm). We then show that our bounded precision assumption is necessary — i.e. we show that when the precision to which constraints are specified need not be uniformly upper bounded, then no algorithm for this problem in dimension \(d \geq 3\) can have a finite mistake bound.

This lower bound motivates us to study a PAC style variant of the problem, where the examples are not chosen in an adversarial manner, but instead are drawn independently at random from an

\(^3\) In fact, the first broad use of linear programming was planning and targeting during WWII. A stylized problem that this scenario models is the following: we wish to predict which cities an enemy will bomb each day. We know the relative military value of each target, which gives us his objective function, but we do not know what resources he has at his disposal. The known, changing constraint each day here might represent weather conditions which can make subsets of the targets difficult to attack.
arbitrary unknown distribution. In this setting, we show that even if the constraints can be specified
to arbitrary (even infinite) precision, there is a learner that requires sample complexity only linear in
the number of edges of the unknown constraint polytope.\footnote{This learner can be implemented efficiently when the constraints are specified with finite precision.}

**Known Constraints & Unknown Objective** For the variant of the problem in which the objective
is unknown and changing and the constraints are known but changing, we give an algorithm that has a
mistake bound and running time that are both polynomial in the dimension $d$. Our algorithm uses the
Ellipsoid algorithm to learn the coefficients of the unknown objective by implementing a separation
oracle that generates separating hyperplanes given examples on which our algorithm made a mistake.

### 1.2 Related Work

Beigman and Vohra [3] were the first to study “revealed preference” problems as learning problems,
and to relate them to multi-dimensional classification problems. They observed that sample complexity
bounds for such problems can be derived by computing the fat shattering dimension of the class of
target utility functions, and showed that the set of Lipschitz-continuous valuation functions had finite
fat-shattering dimension. Zadimoghaddam and Roth [15] gave efficient algorithms with polynomial
sample complexity for PAC learning in the revealed preferences setting over the class of linear (and
piecewise linear) utility functions. Balcan et al. [2] showed a connection between learning from revealed
preferences and the structured prediction problem of learning $d$-dimensional linear classes [7,8,12], and
use an efficient variant of the compression techniques given by Daniely and Shalev-Shwartz [9] to give
efficient PAC algorithms with optimal sample complexity for various classes of economically meaningful
utility functions. Amin et al. [1] study the revealed preferences problem for linear valuation functions
in the mistake bound model of learning, and in the query model in which the learner gets to set prices
and wishes to maximize profit. Roth et al. [14] also study the query model of learning, and give results
for strongly concave objective functions, leveraging a recent algorithm of Belloni et al. [4] for bandit
convex optimization with adversarial noise.

All of the works above focus on the setting of predicting the optimizer of a fixed unknown objective
function, together with a single known, changing constraint representing prices. This is the primary
point of departure for our work — we give algorithms for the more general settings of predicting the
optimizer of a linear program when there may be many unknown constraints, or when the unknown
objective function is changing. Finally, the literature on preference learning (see e.g. [10]) has similar
goals, but is technically quite distinct: the canonical problem in preference learning is learning a
*ranking* on $n$ distinct elements. In contrast, the problem we consider here is to predict the outcome
of a continuous optimization problem as a function of varying constraints.

### 2 Model and Preliminaries

We study an online prediction problem in which we are trying to predict the optimal solution of a
changing linear program (LP) whose parameters are only partially known. More specifically, in each
day $t = 1, 2, \ldots$ an adversary chooses a polytope $P(t)$ defined by a set of linear inequalities (see
Section 2.1 for the formal definition) and coefficients $c(t) \in \mathbb{R}^d$ of a linear objective function. The
learner’s goal is to predict the solution $x(t)$ where

$$ x(t) = \arg \max_{x \in P(t)} c(t) \cdot x. \tag{3} $$

After making the prediction $\hat{x}(t)$, the learner observes the optimal $x(t)$, and in particular, learns
whether she makes a mistake ($\hat{x}(t) \neq x(t)$). We define the mistake bound for a learning algorithm as follows.
Definition 1. Given a linear program with feasible polytope $\mathcal{P}$ and objective function $\mathbf{c}$, let $\sigma(\mathcal{P}, \mathbf{c})$ denote the parameters of the LP that are revealed to the learner (these will be defined in the cases of the problem we study). A learning algorithm $\mathcal{A}$ takes as input the sequence $\{\sigma(\mathcal{P}^{(t)}, \mathbf{c}^{(t)})\}_t$, the known parameters of an adaptively chosen sequence $\{\mathcal{P}^{(t)}, \mathbf{c}^{(t)}\}_t$ of LPs and outputs a sequence of predictions $\{\hat{x}^{(t)}\}_t$. We say that $\mathcal{A}$ has mistake bound $M$ if

$$\max_{(\mathcal{P}^{(t)}, \mathbf{c}^{(t)})} \left\{ \sum_{t=1}^{\infty} 1 \left[ \hat{x}^{(t)} \neq x^{(t)} \right] \right\} \leq M,$$

where $x^{(t)}$ solves (3) each day.

We consider two different instances of the problem described above. First, in Section 3, we study the problem given in (1) in which $\mathbf{c}^{(t)} = \mathbf{c}$ is fixed and known to the learner but the polytope $\mathcal{P}^{(t)} = \mathcal{P} \cap \mathcal{N}^{(t)}$ consists of an unknown fixed polytope $\mathcal{P}$ and a new constraint $\mathcal{N}^{(t)} = \{ \mathbf{x} : \mathbf{p}^{(t)} \cdot \mathbf{x} \leq b^{(t)} \}$ which is revealed to the learner each day (we refer to this as the Known Objective problem). Equivalently in our terminology $\sigma(\mathcal{P}^{(t)}, \mathbf{c}^{(t)}) = (\mathcal{N}^{(t)}, \mathbf{c})$. Then, in Section 4, we study the problem in which the polytope $\mathcal{P}^{(t)}$ is changing and known but the objective function $\mathbf{c}^{(t)} = \sum_{i \in S^{(t)}} \mathbf{v}^i$ is unknown and changing as in (2) where the set $S^{(t)}$ is known (we refer to this as the Known Constraints problem). So $\sigma(\mathcal{P}^{(t)}, \mathbf{c}^{(t)}) = (\mathcal{P}^{(t)}, S^{(t)})$. In both settings we give a learning algorithm with finite mistake bound.

2.1 Preliminaries

In this section we formally define the geometric notions used throughout this paper. A hyperplane and a halfspace in $\mathbb{R}^d$ are the set of points satisfying the linear equation $a_1 x_1 + \ldots + a_d x_d = b$ and the linear inequality $a_1 x_1 + \ldots + a_d x_d \leq b$ for a set of $a_i$s respectively, assuming that not all $a_i$s are simultaneously zero. A set of hyperplanes are linearly independent if the hyperplanes’ gradients (normal vectors to the hyperplanes) are linearly independent. A polytope (denoted by $\mathcal{P} \subseteq \mathbb{R}^d$) is the bounded intersection of finitely many halfspaces, written as $\mathcal{P} = \{ \mathbf{x} : \mathbf{A} \mathbf{x} \leq \mathbf{b} \}$. An edge-space $e$ of a polytope $\mathcal{P}$ is a one dimensional subspace that is the intersection of $d-1$ linearly independent hyperplanes of $\mathcal{P}$, and an edge is the intersection between an edge-space $e$ and the polytope. We denote the set of edges of polytope $\mathcal{P}$ by $E_\mathcal{P}$. A vertex of $\mathcal{P}$ is a point where $d$ linearly independent hyperplanes of $\mathcal{P}$ intersect. We can equivalently write the polytope as the convex hull of its vertices $V$ which we denote by $\text{conv}(V)$. Finally, we define a set of points to be collinear if there exists a line that contains all the points in the set.

Throughout the paper, in order for our prediction problem to be well defined, we make the following assumption about the observed solution $\hat{x}^{(t)}$ in each day. This assumption guarantees that each solution is on a vertex of the polytope $\mathcal{P}^{(t)}$.

Assumption 1 The optimal solution to the LP $\max_{\mathbf{x} \in \mathcal{P}^{(t)}} \mathbf{c}^{(t)} \cdot \mathbf{x}$ is unique for all $t$.

3 The Known Objective Problem

In this section, we focus on the Known Objective Problem where the coefficients of the objective function are fixed and known to the learner. We use $\mathbf{c}$ to denote these coefficients. However, the feasible region $\mathcal{P}^{(t)}$ on day $t$ is unknown and changing. In particular, the feasible region in each day $t$ is the intersection of a fixed and unknown polytope $\mathcal{P} = \{ \mathbf{x} : \mathbf{A} \mathbf{x} \leq \mathbf{b} \}$ and the halfspace $\mathcal{N}^{(t)} = \{ \mathbf{x} : \mathbf{p}^{(t)} \cdot \mathbf{x} \leq b^{(t)} \}$ specified by a changing constraint that is known to the learner i.e., $\mathcal{P}^{(t)} = \mathcal{P} \cap \mathcal{N}^{(t)}$.

Throughout this section we make the following assumptions. First, we assume that w.l.o.g. (up to scaling) the points in $\mathcal{P}$ have $\ell_\infty$-norm bounded by 1.

Assumption 2 The unknown polytope $\mathcal{P} = \{ \mathbf{x} \in \mathbb{R}^d : \mathbf{A} \mathbf{x} \leq \mathbf{b} \}$ lies inside the unit $\ell_\infty$-ball: $\mathcal{P} \subseteq \{ \mathbf{x} : \| \mathbf{x} \|_\infty \leq 1 \}$. 
We also assume that the coordinates of the vertices in $\mathcal{P}$ can be written with finite precision (this is implied if the halfspaces defining $\mathcal{P}$ can be described with finite precision).

**Assumption 3** The coordinates of each vertex of $\mathcal{P}$ are multiples of $1/2^N$ (i.e. they can be written with $N$ bits of precision in binary).

We show in Section 3.3 that Assumption 3 is necessary — without any upper bound on precision, there is no algorithm with a finite mistake bound. Next, we make some non-degeneracy assumptions on polytopes $\mathcal{P}$ and $\mathcal{P}^{(t)}$, respectively.

**Assumption 4** Any subset of $d-1$ rows of $A$ have rank $d-1$ where $A$ is the constraint matrix in $\mathcal{P} = \{x \in \mathbb{R}^d : Ax \leq b\}$.

**Assumption 5** Each vertex of $\mathcal{P}^{(t)}$ is the intersection of exactly $d$-hyperplanes of $\mathcal{P}^{(t)}$.

The rest of this section is organized as follows. First, we present our learning algorithm LearnEdge for the Known Objective Problem in Section 3.1 and analyze its mistake bound in Section 3.2. Then in Section 3.3, we show that the finite precision assumption (Assumption 3) is necessary in order to get a finite mistake bound. Finally, in Section 3.4, we present a learning algorithm LearnHull in a PAC style setting in which the new constraint each day is drawn i.i.d. from an unknown distribution over the constraints, rather than selected adversarially. We show that LearnHull has sample complexity linear in the number of edges of $\mathcal{P}$ even without a finite precision assumption.

### 3.1 LearnEdge Algorithm

In this section we introduce our main algorithm LearnEdge. In our main result in Theorem 6 we show that the number of mistakes of LearnHull depends linearly on the number of edges $E_{\mathcal{P}}$ of $\mathcal{P}$ and the precision parameter $N$ and only logarithmically on the dimension $d$.

**Theorem 6.** The number of mistakes of LearnEdge in the Known Objective Problem is no more than $O(|E_{\mathcal{P}}|N \log(d))$, and it also runs in time $\text{poly}(m,d,|E_{\mathcal{P}}|)$ every day where $m$ is the number of rows of the constraint matrix $A$.

At a high level, LearnEdge maintains a set of prediction information $\mathcal{I}^{(t)}$ about the prediction history up to day $t$, and makes prediction in each day based on $\mathcal{I}^{(t)}$ and a set of prediction rules (P.1 – P.4). If it makes a mistake, LearnEdge updates the information with a set of update rules (U.1 – U.4). The framework of LearnEdge is presented in Algorithm 1. We will now present the details of each component.

#### Prediction Information

It is natural to ask “what information is useful for prediction?” Lemma 1 establishes the importance of the set of edges $E_{\mathcal{P}}$ by showing that all observed solutions will be on an element of $E_{\mathcal{P}}$. We defer the omitted proofs of this section to Appendix A.

**Lemma 1.** On any day $t$, the observed solution $x^{(t)}$ lies on an edge in $E_{\mathcal{P}}$.

In the proof of Lemma 1 we also show that when $x^{(t)}$ does not bind the new constraint, then $x^{(t)}$ is the solution for the underlying LP: $\arg\max_{x \in \mathcal{P}} c \cdot x$.

**Corollary 1.** If $x^{(t)} \in \{x : p^{(t)}x < b^{(t)}\}$ then $x^{(t)} = x^* \equiv \arg\max_{x \in \mathcal{P}} c \cdot x$.

---

5 Lemma 6.2.4 from Grotschel et al. [11] states that if each constraint in $\mathcal{P}$ has encoding length at most $N$ then each vertex of $\mathcal{P}$ has encoding length at most $4d^2N$. Typically the finite precision assumption is made on the constraints of the LP. However, since this assumption implies that the vertices can be described with finite precision, for simplicity, we make our assumption directly on the vertices.
We then proceed to show in Lemma 2 that how an edge-space $e$ of $\mathcal{P}$ can be recovered after seeing three collinear observed solutions.

**Lemma 2.** Let $x, y, z$ be 3 distinct collinear points on edges of $\mathcal{P}$. Then they are all on the same edge of $\mathcal{P}$ and the 1-dimensional subspace containing them is an edge-space of $\mathcal{P}$.

Algorithm 1: Learning in Known Objective Problem (LearnEdge)

```plaintext
procedure LearnEdge($\{N^{(t)}, x^{(t)}\}_t$)  
    \begin{align*}
      \text{Initialize } I^{(t)} \text{ to be empty.} & \quad \triangleright \text{Against adaptive adversary} \\
      \text{for } t = 1, 2, \ldots \text{ do} & \quad \triangleright \text{Initialize} \\
      \quad \text{Predict } \hat{x}^{(t)} \text{ according to one of } \mathbf{P.1 - P.4} \text{ based on } N^{(t)} \text{ and } I^{(t)}. & \quad \triangleright \text{Predict} \\
      \quad \text{if } \hat{x}^{(t)} \neq x^{(t)} \text{ then} & \quad \triangleright \text{Predict} \\
      \quad \quad X^{(t+1)} \leftarrow X^{(t)} \cup \{x^{(t)}\}. & \quad \triangleright \text{Predict} \\
      \quad \quad \text{Update } I^{(t+1)} \text{ with } \mathbf{U.1 - U.4} \text{ based on } x^{(t)}. & \quad \triangleright \text{Update} \\
    \end{align*}
end procedure
```

Fig. 1: Regions on an edge-space: feasible region $F_e$ (blue), questionable intervals $Q^0_e$ and $Q^1_e$ (green) with their mid-points $M^0_e$ and $M^1_e$ and infeasible regions $Y^0_e$ and $Y^1_e$ (dashed).

(a) **Feasible Interval** The feasible interval $F_e$ is an interval along $e$ that is identified to be on the boundary of $\mathcal{P}$. More formally, $F_e = \text{Conv}(X^{(t)} \cap e)$.

(b) **Infeasible Region** The infeasible region $Y_e = Y^0_e \cup Y^1_e$ is the union of two disjoint intervals $Y^0_e$ and $Y^1_e$ that are identified to be outside of $\mathcal{P}$. By Assumption 2, we initialize the infeasible region $Y_e$ to $\{x \in e \mid \|x\|_\infty > 1\}$ for all $e$.

(c) **Questionable Region** The questionable region $Q_e = Q^0_e \cup Q^1_e$ on $e$ is the union of two disjoint questionable intervals along $e$. Formally, $Q_e = e \setminus (F_e \cup Y_e)$. The points in $Q_e$ cannot be certified to be either inside or outside of $\mathcal{P}$ by LearnEdge.

(d) **Midpoints in $Q_e$** For each questionable interval $Q^i_e$, let $M^i_e$ denote the midpoint of $Q^i_e$ — the average point of the two endpoints on $Q^i_e$.

We add the superscript $(t)$ to show the dependence of these quantities on days. Furthermore, we eliminate the subscript $e$ when taking the union over all elements in $E^{(t)}$, e.g. $F^{(t)} = \bigcup_{e \in E^{(t)}} F^{(t)}_e$.
So the information $\mathcal{T}(t)$ can be written as follows:

$$\mathcal{T}(t) = \left( X(t), E(t), F(t), Y(t), Q(t), M(t) \right).$$

**Prediction Rules** We now focus on the prediction rules of LearnEdge. At each day $t$, let $\tilde{N}(t) = \{ x \mid p(t) \cdot x = b(t) \}$ be the hyperplane specified by the additional constraint $N(t)$. If $x(t) \notin \tilde{N}(t)$, then $x(t) = x^*$ by Corollary 1. So whenever the algorithm observes such a solution $x^*$, it will store $x^*$ and predict it in the future days when $x^* \in N(t)$. This is case P.1.

So suppose $x^* \notin \tilde{N}(t)$. The analysis of Lemma 1 shows that the observed solution must be in the intersection between $\tilde{N}(t)$ and the edges $E_P$, so $x(t) = \arg\max_{x \in \tilde{N}(t) \cap E_P} c \cdot x$. Hence, LearnEdge can restrict its prediction to the following candidate set:

$$\text{Cand}(t) = \left\{ (E(t) \cup X(t)) \setminus \hat{E}(t) \right\} \cap \tilde{N}(t),$$

where $\hat{E}(t) = \{ e \in E(t) \mid e \subseteq \tilde{N}(t) \}$.

As we show in Lemma 3, $x(t)$ will not be in $\hat{E}(t)$, so it is safe to remove $\hat{E}(t)$ from Cand$(t)$.

**Lemma 3.** Let $e$ be an edge-space of $\mathcal{P}$ such that $e \subseteq \tilde{N}(t)$, then $x(t) \notin e$.

However, Cand$(t)$ can be empty or only contain points in the infeasible regions of the edge-spaces. If so, then there is simply not enough information to predict a feasible point in $\mathcal{P}$. Hence, LearnEdge predicts an arbitrary point outside of Cand$(t)$. This is case P.2.

Otherwise Cand$(t)$ may contain feasible points. LearnEdge predicts from a subset of Cand$(t)$ called the extended feasible region Ext$(t)$ instead of directly predicting from Cand$(t)$. As we will show later this guarantees that LearnEdge makes progress in learning the true feasible region on some edge-space upon making a mistake. Ext$(t)$ is the intersection of $\tilde{N}(t)$ with the union of intervals between the two mid-points $(M^0_e(t))$ and $(M^1_e(t))$ on every edge-space $e \in E(t) \setminus \hat{E}(t)$ and all points in $X(t)$. Formally,

$$\text{Ext}(t) = \left\{ X(t) \cup \bigcup_{e \in E(t) \setminus \hat{E}(t)} \text{conv} \left( (M^0_e(t)), (M^1_e(t)) \right) \right\} \cap \tilde{N}(t).$$

If Ext$(t) \neq \emptyset$, then LearnEdge predicts the point with the highest objective value in it. This corresponds to P.3.

Finally, if Ext$(t) = \emptyset$, then we know $\tilde{N}(t)$ only intersects within the questionable regions of the learned edge-spaces. In this case, LearnEdge predicts the intersection point with the lowest objective value, which corresponds to P.4. Although it might seem counter-intuitive to predict the point with the lowest objective value, this guarantees that LearnEdge makes progress in learning the true feasible region on some edge-space upon making a mistake (see Lemma 6). The formal description of the prediction rules are summarized as follows:

**P.1** First, if $x^*$ is observed and $x^* \in N(t)$, then predict $\hat{x}(t) \leftarrow x^*$;

**P.2** Else if Cand $= \emptyset$ or Cand$(t) \subseteq \bigcup_{e \in E(t)} Y_e(t)$, then predict any point outside Cand$(t)$;

**P.3** Else if Ext$(t) \neq \emptyset$, then predict $\hat{x}(t) = \arg\max_{x \in \text{Ext}(t)} c \cdot x$;

**P.4** Else, predict $\hat{x}(t) = \arg\min_{x \in \text{Cand}(t)} c \cdot x$.

**Update Rules** Next we describe how LearnEdge updates its information. Upon making a mistake, LearnEdge adds $x(t)$ to the set of previously observed solutions $X(t)$ i.e. $X(t+1) \leftarrow X(t) \cup \{x(t)\}$. Then it performs one of the following four mutually exclusive update rules (U.1-U.4) in order.

First, if $x(t) \notin \tilde{N}(t)$, then LearnEdge records $x(t)$ as the unconstrained optimal solution $x^* \equiv \arg\max_{x \in \mathcal{P}} c \cdot x$. This is update U.1.

So in the remaining updates, $x(t) \in \tilde{N}(t)$. If $x(t)$ is not on any edge-space in $E(t)$, LearnEdge will try to learn a new edge-space using the subroutine COLLINE$(X(t), x(t))$ which either produces a
one-dimensional subspace $e$ that contains $x^{(t)}$ (along with two other points in $X^{(t)}$) or returns $\emptyset$ when there is no such element $e$ (see Appendix B). This is update \textbf{U.2}.

If the previous updates were not invoked, then $x^{(t)}$ was on some edge-space $e$. \textbf{LearnEdge} then compares the objective values of $\hat{x}^{(t)}$ and $x^{(t)}$: (a) If $c \cdot \hat{x}^{(t)} > c \cdot x^{(t)}$, then $\hat{x}^{(t)}$ must be infeasible and \textbf{LearnEdge} then updates the questionable and infeasible regions for $e$ via \textbf{U.3}. (b) Otherwise $c \cdot \hat{x}^{(t)} < c \cdot x^{(t)}$ and $x^{(t)}$ was not predicted because it was outside of the extended feasible region of $e$.\footnote{We do not need to consider the case that $c \cdot \hat{x}^{(t)} = c \cdot x^{(t)}$ due to Assumption 1.} In this case \textbf{LearnEdge} updates the questionable region and feasible interval on $e$ via \textbf{U.4}.

In both of \textbf{U.3} and \textbf{U.4}, \textbf{LearnEdge} will shrink some questionable interval $Q_e^t$ substantially. In the case where the updated questionable interval $Q_e^t$ has length less than $2^{-N}$, \textbf{LearnEdge} will call the function \textbf{Eliminate} to eliminate the interval and update the adjacent feasible region $F_e$ and infeasible interval $Y_e^t$ (see Appendix B). We defer the formal descriptions of the updates to Appendix C.

### 3.2 Analysis of \textbf{LearnEdge}

We note that whenever \textbf{LearnEdge} makes a mistake, one of the update rules \textbf{U.1} - \textbf{U.4} is invoked. Hence we can bound the number of mistakes \textbf{LearnEdge} makes in the worst case by bounding the number of times each update rule is invoked.

**Lemma 4.** Update \textbf{U.1} is invoked at most 1 time.

**Lemma 5.** Update \textbf{U.2} is invoked at most 3|E_P| times.\footnote{The dependency on |E_P| can be improved by replacing it with the set of edges of $P$ on which an optimal point is observed. This applies to all the dependencies on |E_P| in our bounds.}

**Lemma 6.** Updates \textbf{U.3} and \textbf{U.4} are invoked at most $O(|E_P|N \log(d))$ times.

**Proof.** Let $Q_e^t$ be the updated questionable interval. We know initially $Q_e$ has length at most $2\sqrt{d}$ by Assumption 2. We show in Lemma 11 in Appendix A that each time an update \textbf{U.3} or \textbf{U.4} is invoked, the length of $Q_e^t$ decreases by at least a half. Then after at most $O(N \log(d))$ updates, the interval will have length less than $2^{-N}$ after which the interval will be updated at most once when \textbf{Eliminate} is invoked to eliminate the interval.

Therefore, the total number of updates on $Q_e^t$ is bounded by $O(N \log(d))$. Since there are at most $2|E_P|$ questionable intervals, the total number of updates \textbf{U.3} and \textbf{U.4} is bounded by $O(|E_P|N \log(d))$.

Summing up the mistakes bounds in Lemmas 4-6 will result in the mistake bound of \textbf{LearnEdge} as stated in Theorem 6.

### 3.3 On the Necessity of a Precision Bound

We show the necessity of Assumption 3 by showing that our dependence on the precision parameter $N$ is tight. We show that subject to Assumption 3, there exists a polytope $P$ and a sequence of additional constraints $\{N^{(t)}\}_t$ such that any learning algorithm will make $\Omega(N)$ mistakes. This implies that without any upper bound on precision, it is impossible to learn with a finite mistake bound.

**Theorem 7.** For any learning algorithm $A$ in the known objective setting and any $d \geq 3$, there exists a polytope $P$ and a sequence of additional constraints $\{N^{(t)}\}_t$ such that the number of mistakes made by $A$ is at least $\Omega(N)$.\footnote{The condition $d \geq 3$ is necessary in the statement of Theorem 7 since there exists learning algorithms for $d = 1$ and $d = 2$ with mistake bounds independent of $N$.}
3.4 Stochastic Setting

Given the lower bound from Section 3.3, it is natural to ask in what settings we can still learn when we do not have a uniform upper bound on the precision to which constraints can be specified. The lower bound implies we must abandon the adversarial setting, and so we now consider a PAC style variant of our problem. Rather than being chosen by an adversary, the additional constraint at each day $t$ is now drawn i.i.d. from some fixed, unknown distribution $D$ over $\mathbb{R}^d \times \mathbb{R}$ such that each point $(p, b)$ drawn from $D$ corresponds to the halfspace $\mathcal{N} = \{ x : p \cdot x \leq b \}$. We make no assumption on the form of the constraint distribution $D$, and require our bounds to hold in the worst case over all choices of $D$.

We describe LearnHull an algorithm based on the following high level idea: LearnHull keeps track of the convex hull $C^{(t-1)}$ of all the solutions observed up to day $t$. Then it behaves as if this convex hull is the entire feasible region. So at day $t$, given the constraint $\mathcal{N}(t) = \{ x : p^{(t)} \cdot x \leq b^{(t)} \}$, LearnHull predicts $\hat{x}^{(t)}$ where

$$\hat{x}^{(t)} = \arg\max_{x \in C^{(t-1)} \cap \mathcal{N}(t)} c \cdot x.$$  \hspace{1cm} (5)

LearnHull’s hypothetical feasible region is therefore always a subset of the true feasible region – i.e. it can never make a mistake because its prediction was infeasible, but only because its prediction was sub-optimal. Hence, whenever LearnHull makes a mistake, it must have observed a point that expands the convex hull. Hence, whenever it fails to predict $x^{(t)}$, LearnHull will enlarge its feasible region by adding the point $x^{(t)}$ to the convex hull:

$$C^{(t)} \leftarrow \text{conv}(C^{(t-1)} \cup \{ x^{(t)} \}),$$ \hspace{1cm} (6)

otherwise it will simply set $C^{(t)} \leftarrow C^{(t-1)}$. LearnHull is described formally in Algorithm 2.

| Algorithm 2: Stochastic Procedure (LearnHull) |
|---------------------------------------------|
| **procedure** LearnHull($D$) |
| $C^{(0)} \leftarrow \emptyset$. \hspace{1cm} ▷ Initialize |
| Observe $\mathcal{N}^{(t)} \sim D$ and set $x^{(t)}$ as in (5). \hspace{1cm} ▷ Predict |
| Observers $x^{(t)} = \arg\max_{x \in \mathcal{N}^{(t)} \cap \mathcal{N}(t)} c \cdot x$ and update $C^{(t)}$ as in (6). \hspace{1cm} ▷ Update |
| **end procedure** |

We show that the expected number of mistakes of LearnHull over $T$ days is linear in the number of edges of the polytope $\mathcal{P}$ and only logarithmic in $T$. \hspace{1cm} (9)

**Theorem 8.** For any $T > 0$ and any constraint distribution $D$, the expected number of mistakes of LearnHull after $T$ days is bounded by $O(|E_{\mathcal{P}}| \log(T))$.

To prove Theorem 8, first in Lemma 7 we bound the probability that the solution observed at day $t$ falls outside of the convex hull of the previously observed solutions. This is the only event that can cause LearnHull to make a mistake. In Lemma 7, we abstract away the fact that the point observed at each day is the solution to some optimization problem – Lemma 7 holds for any distribution over points restricted to the edges of a polytope.

\hspace{1cm} (9) LearnHull can be implemented efficiently in time poly($T, N, d$) if each coefficient in each of the unknown constraints is represented in $N$ bits. Note that the naive approach of computing the convex hull of the observed solutions does not result in an efficient algorithm. However, given $T' \leq T$ observed solutions so far and a new point a separation oracle can be implemented in time poly($T, N, d$) to determine whether the new point belongs to the convex hull of the observed solutions and if not return a separating hyperplane.
Lemma 7. Let $\mathcal{P} \subseteq \mathbb{R}^d$ be a polytope and $\mathcal{D}$ a distribution over points on the edges of $\mathcal{P}$. Let $X = \{x_1, \ldots, x_{t-1}\}$ be $t-1$ points drawn i.i.d. from $\mathcal{D}$ and $x_t$ an additional point drawn independently from $\mathcal{D}$. Then $\Pr[x_t \not\in \conv(X)] \leq 2|E_{\mathcal{P}}|/t$ where $E_{\mathcal{P}}$ denotes the set of edges in $\mathcal{P}$ and the probability is taken over the draws of all points $x_1, \ldots, x_t$ from $\mathcal{D}$.

Proof. First, since all of the points $x_1, \ldots, x_t$ are drawn i.i.d. from $\mathcal{D}$, we observe by symmetry that the event we are interested in is distributed identically to the following event: draw a set of $t$ points $X' = \{x_1, \ldots, x_t\}$ i.i.d. from $\mathcal{D}$ and select an index $i \in \{1, \ldots, t\}$ uniformly at random and compute the probability that $x_i \not\in \conv(X' \setminus \{x_i\})$. In other words

$$\Pr_{x_1, \ldots, x_t \sim \mathcal{D}}[x_t \not\in \conv(X)] = \Pr_{x_1, \ldots, x_t \sim \mathcal{D}, i \sim \{1, \ldots, t\}}[x_i \not\in \conv(X' \setminus \{x_i\})]. \quad (7)$$

We analyze the quantity on the right hand side of (7) instead, fixing the choices of $x_1, \ldots, x_t$, and analyzing the probability only over the randomness of the choice of index $i$. For each edge $e \in E_{\mathcal{P}}$, let $X'_e = X' \cap e$. Since each edge lies on a one dimensional subspace, there are at most two extreme points $x'_1, x'_2 \in X'_e$ that lie outside of the convex hull of other points i.e. such that $x'_1 \not\in \conv(X' \setminus \{x'_1\})$ and $x'_2 \not\in \conv(X' \setminus \{x'_2\})$. We note that when we choose an index $i$ uniformly at random, the probability that we select a point $x \in X'_i$ is exactly $|X'_i|/t$, and conditioned on selecting a point $x \in X'_i$, the probability that $x$ is an extreme point (i.e. $x \in \{x'_1, x'_2\}$) is at most $2/|X'_i|$. Hence, we can calculate

$$\Pr[x_i \not\in \conv(X' \setminus \{x_i\})] = \sum_{e \in E_{\mathcal{P}}} \Pr[x_i \in X'_e] \cdot \Pr[x_i \not\in \conv(X'_e \setminus \{x_i\}) | x_i \in X'_e]$$

$$\leq \sum_{e \in E_{\mathcal{P}}} \frac{|X'_e|}{t} \cdot \frac{2}{|X'_e|} = \sum_{e \in E_{\mathcal{P}}} \frac{2|E_{\mathcal{P}}|}{t} = \frac{2|E_{\mathcal{P}}|}{t}.$$ 

Finally in Theorem 9 we convert the bound on the expected number of mistakes in Theorem 8 to a high probability bound.

Theorem 9. There exists a deterministic procedure such that after $T = O(|E_{\mathcal{P}}| \log (1/\delta))$ days, the probability (over the randomness of the additional constraint) that the procedure makes a mistake on day $T + 1$ is at most $\delta$ for any $\delta \in (0, 1/2)$.

4 The Known Constraints Problem

We now consider the Known Constraints Problem in which the learner observes the changing constraint polytope $\mathcal{P}^{(t)}$ at each day, but does not know the changing objective function which we assume to be written as $c^{(t)} = \sum_{i \in S^{(t)}} \mathbf{v}^i$, where $\{\mathbf{v}^i\}_{i \in [n]}$ are fixed but unknown. Given $\mathcal{P}^{(t)}$ and the subset $S^{(t)} \subseteq [n]$, the learner must make a prediction $\hat{x}^{(t)}$ on each day.

4.1 An Ellipsoid Based Algorithm

Inspired by Bhaskar et al. [5], we use the Ellipsoid algorithm to learn the coefficients $\{\mathbf{v}^i\}_{i \in [n]}$, and show that the mistake bound of the resulting algorithm is bounded by the (polynomial) running time of Ellipsoid. We use $V \in \mathbb{R}^{d \times n}$ to denote the matrix whose columns are $\mathbf{v}^i$ and make the following assumption about $V$:

Assumption 10 Each entry in $V$ can be represented as a multiple of $1/2^N$. Furthermore, without loss of generality, we assume $\|V\|_F \leq 1$.

Similar to Section 3 we assume finite precision in the sense that we assume the coordinates of $\mathcal{P}^{(t)}$’s vertices can be written with finite precision (this is implied if the halfspaces defining the polytope are described with finite precision [11]).
**Assumption 11** The coordinates of each vertex of the polytope $\mathcal{P}^{(t)}$ are multiples of $1/2^N$ (i.e. they can be written with $N$ bits of precision in binary).

We first observe that the coefficients of the objective function represent a point that is guaranteed to lie in a region $\mathcal{F}$ which may be written as the intersection of possibly infinitely many halfspaces. Given a subset $S \subseteq [n]$, and a polytope $\mathcal{P}$, let $x^{S,\mathcal{P}}$ denote the optimal solution to the instance defined by $S$ and $\mathcal{P}$. Informally, the halfspaces defining $\mathcal{F}$ ensure that for any problem instance defined by arbitrary choices of $S$ and $\mathcal{P}$, the objective value of the optimal solution $x^{S,\mathcal{P}}$ must be at least as high as the objective value of any feasible point in $\mathcal{P}$. Since the convergence rate of the Ellipsoid algorithm depends on the precision to which constraints are specified, we do not in fact consider a hyperplane for every feasible solution, but only for those solutions that are vertices of the feasible polytope $\mathcal{P}$. This is not in fact a relaxation, since LPs always have vertex-optimal solutions.

We denote the set of all vertices of polytope $\mathcal{P}$ by $\text{vert}(\mathcal{P})$, and the set of polytopes $\mathcal{P}$ satisfying Assumption 11 by $\Phi$. We then define $\mathcal{F}$ as follows:

$$
\mathcal{F} = \left\{ W = (w^1, \ldots, w^n) \in \mathbb{R}^{n \times d} : \forall S \subseteq [n], \forall \mathcal{P} \in \Phi, \sum_{i \in S} w^i \cdot (x^{S,\mathcal{P}} - x) \geq 0, \forall x \in \text{vert}(\mathcal{P}) \right\}
$$

The idea behind our LearnEllipsoid algorithm is that we will run a copy of the Ellipsoid algorithm with variables $w \in \mathbb{R}^{d \times n}$, as if we were solving the feasibility LP defined by the constraints defining $\mathcal{F}$. We will always predict according to the centroid of the ellipsoid maintained by the Ellipsoid algorithm (i.e. its candidate solution). Whenever a mistake occurs, we are able to find one of the constraints that define $\mathcal{F}$ such that our prediction violates the constraint – exactly what is needed to take a step in solving the feasibility LP. Since we know $\mathcal{F}$ is nonempty (at least the true objective function $V$ lies within it) we know that the LP we are solving is feasible. Given the polynomial convergence time of Ellipsoid, this gives a polynomial mistake bound for our algorithm.

The ellipsoid algorithm will generate a sequence of ellipsoids with decreasing volume such that each one contains feasible region $\mathcal{F}$. Given the ellipsoid $\mathcal{E}^{(t)}$ at day $t$, LearnEllipsoid uses the centroid of $\mathcal{E}^{(t)}$ as its hypothesis for the objective function $W^{(t)} = ((w^1)^{(t)}, \ldots, (w^n)^{(t)})$. Given the subset $S^{(t)}$ and polytope $\mathcal{P}^{(t)}$, LearnEllipsoid predicts

$$
\hat{x}^{(t)} \in \arg \max_{x \in \mathcal{P}^{(t)}} \left\{ \sum_{i \in S^{(t)}} (w^i)^{(t)} \cdot x \right\}.
$$

LearnEllipsoid then finds the following hyperplane separating the centroid of the current Ellipsoid (the current candidate objective) from $\mathcal{F}$.

$$
\mathcal{H}^{(t)} = \left\{ W = (w^1, \ldots, w^n) \in \mathbb{R}^{n \times d} : \sum_{i \in S^{(t)}} w^i \cdot (x^{(t)} - \hat{x}^{(t)}) > 0 \right\}.
$$

Given the separating hyperplane $\mathcal{H}^{(t)}$ and ellipsoid $\mathcal{E}^{(t)}$, the procedure ELLIPSOID($\mathcal{H}^{(t)}, \mathcal{E}^{(t)}$) computes the minimum-volume ellipsoid $\mathcal{E}^{(t+1)}$ that contains $\mathcal{H}^{(t)} \cap \mathcal{E}^{(t)}$. We then use procedure CENTROID to set $W^{(t+1)}$ to the centroid of $\mathcal{E}^{(t+1)}$. The above procedure is formalized in Algorithm 3.

### 4.2 Analysis of LearnEllipsoid

The update given in (9) leaves the procedure used to solve the LP unspecified. To simplify our analysis, we use a specific LP solver to obtain the prediction $\hat{x}^{(t)}$ in (9) that ensures $\hat{x}^{(t)}$ is a vertex of $\mathcal{P}^{(t)}$.

**Theorem 12** (Theorem 6.4.12 and Remark 6.5.2 in Grotschel et al. [11]). There exists an LP solver that runs in time polynomial in the length of its input and returns an exact solution to (9) that is a vertex of the polytope $\mathcal{P}^{(t)}$. 

**Algorithm 3: Learning with Known Constraints (LearnEllipsoid)**

```plaintext
procedure LearnEllipsoid(A)                         ▷ Against adversary A

I(1) ← (E(1) = \{z ∈ R^n x d : ||z||_F ≤ 2\}, W(1) = CENTROID(E(1))) ▷ Initialize

for t = 1 \ldots do
    Given S(t), P(t), set ̂x(t) as in (9).
    if x(t) ≠ ̂x(t) then  \▷ Predict
        set H(t) as in (10).
        E(t+1) ← ELLIPSOID(H(t), E(t)), W(t+1) = CENTROID(E(t+1)).
        I(t+1) = (E(t+1), W(t+1)).
    else  \▷ Update
        I(t+1) ← I(t).
    end

end procedure
```

In Theorem 13, we show that the number of mistakes made by LearnEllipsoid is at most the number of updates that the Ellipsoid algorithm makes before it finds a point in \( F \) and the number of updates of the Ellipsoid algorithm can be bounded by well-known results from the literature. We defer these details and the proof of Theorem 13 to Appendix E.

**Theorem 13.** The total number of mistakes made by LearnEllipsoid in the Known Constraints Problem is at most \( \text{poly}(n,d,N) \).

**References**

1. Amin, K., Cummings, R., Dworkin, L., Kearns, M., and Roth, A. Online learning and profit maximization from revealed preferences. In Proceedings of the 29th AAAI Conference on Artificial Intelligence (2015), pp. 770–776.
2. Balcan, M., Daniely, A., Mehta, R., Urner, R., and Vazirani, V. Learning economic parameters from revealed preferences. In Proceeding of the 10th International Conference on Web and Internet Economics (2014), pp. 338–353.
3. Beigman, E., and Vohra, R. Learning from revealed preference. In Proceedings of the 7th ACM Conference on Electronic Commerce (2006), pp. 33–42.
4. Belloni, A., Liang, T., Narayanan, H., and Rakhlin, A. Escaping the local minima via simulated annealing: Optimization of approximately convex functions. In Proceeding of the 28th Conference on Learning Theory (2015), pp. 240–265.
5. Bhaskar, U., Ligett, K., Schulman, L., and Swamy, C. Achieving target equilibria in network routing games without knowing the latency functions. In Proceeding of the 55th IEEE Annual Symposium on Foundations of Computer Science (2014), pp. 31–40.
6. Blum, A., Mansour, Y., and Morgenstern, J. Learning what’s going on: reconstructing preferences and priorities from opaque transactions. In Proceedings of the 16th ACM Conference on Economics and Computation (2015), pp. 601–618.
7. Collins, M. Discriminative reranking for natural language parsing. In Proceedings of the 17th International Conference on Machine Learning (2000), Morgan Kaufmann, pp. 175–182.
8. Collins, M. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing (2002), pp. 1–8.
9. Daniely, A., and Shalev-Shwartz, S. Optimal learners for multiclass problems. In Proceedings of the 27th Conference on Learning Theory (2014), pp. 287–316.
10. Fürnkranz, J., and Hüllermeier, E. Preference learning. Springer, 2010.
11. Grötschel, M., Lovász, L., and Schrijver, A. Geometric Algorithms and Combinatorial Optimization, second corrected ed., vol. 2 of Algorithms and Combinatorics. Springer, 1993.
12. Lafferty, J., McCallum, A., and Pereira, F. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In Proceedings of the 18th International Conference on Machine Learning (2001), pp. 282–289.
A Missing Proofs from Section 3

A.1 Section 3.1

Proof of Lemma 1 Let \( x^* \) be the optimal solution of the linear program solved over the unknown polytope \( P \), without the added constraint i.e. \( x^* = \arg \max_{x \in P} c \cdot x \).

1. Suppose that \( x^* \in N(t) \), then clearly \( x(t) = x^* \). By Assumption 1, \( x^* \) lies on a vertex of \( P \) and therefore \( x(t) \) lies on one of the edges of \( P \).

2. Suppose that \( x^* \notin N(t) \) i.e. \( p(t) \cdot x^* > b(t) \). Then we claim that the optimal solution \( x(t) \) satisfies \( p(t) \cdot x(t) = b(t) \). Suppose to the contrary that \( p(t) \cdot x(t) < b(t) \). Since \( c \cdot x^* \geq c \cdot x(t) \), then for any point \( y \in \text{conv}(x(t), x^*) \),

\[
    c \cdot y = c \cdot (\alpha x(t) + (1-\alpha)x^*) = \alpha(c \cdot x(t)) + (1-\alpha)(c \cdot x^*) \geq c \cdot x(t) \quad \forall \alpha \in [0, 1].
\]

Since \( x(t) \) strictly satisfies the new constraint, there exists some point \( y^* \in \text{conv}(x(t), x^*) \) where \( y^* \neq x(t) \) such that \( y^* \in P(t) \) (i.e. \( y^* \) is also feasible). It follows that \( c \cdot y^* \geq c \cdot x(t) \), which contradicts Assumption 1. Therefore, \( x(t) \) must bind the additional constraint. Furthermore, by non-degeneracy Assumption 5, \( x(t) \) binds exactly \((d-1)\) constraints in \( P \), i.e. \( x(t) \) lies at the intersection of \( d-1 \) hyperplanes of \( P \) which are linearly independent by Assumption 4. Therefore, \( x(t) \) must be on an edge of \( P \).

\[\blacksquare\]

Proof of Lemma 2 Without loss of generality, let us assume \( y \) can be written as convex combination of \( x \) and \( z \) i.e. \( y = \alpha x + (1-\alpha)z \) for some \( \alpha \in (0, 1) \). Let \( B_y = \{j \mid A_j y = b_j\} \) be the set of binding constraints for \( y \), and we know that \( |B_y| \geq d-1 \) by Assumption 5. For any \( j \) in \( B_y \), we consider the following two cases.

1. At least one of \( x \) and \( z \) belongs to the hyperplane \( \{w : A_j w = b_j\} \). Then we claim that all three points bind the same constraint. Assume that \( A_j x = b_j \), then we must have

\[
    A_j z = \frac{A_j(y - \alpha x)}{(1-\alpha)} = \frac{b_j - \alpha b_j}{(1-\alpha)} = b_j.
\]

Similarly, if we assume \( A_j z = b_j \), we will also have \( A_j x = b_j \).

2. None of \( x \) and \( z \) belongs to the hyperplane \( \{w : A_j w = b_j\} \) i.e. \( A_j x < b_j \) and \( A_j z < b_j \) both hold. Then we can write

\[
    b_j = A_j y = \alpha A_j x + (1-\alpha)A_j z < \alpha b_j + (1-\alpha)b_j = b_j,
\]

which is a contradiction.

It follows that for any \( j \in B_y \), we have \( A_j x = A_j y = A_j z = b_j \). Since \( |B_y| \geq d-1 \), we know by Assumption 4 that the set of points that bind any set of \( d-1 \) constraints in \( B_y \) will form an edge-space and further this edge-space will include \( x, y, \) and \( z \).

\[\blacksquare\]

Proof of Lemma 3 First, note that the observed solution \( x(t) \) is a vertex in the polytope \( P(t) = P \cap X(t) \), that is an intersection of \emph{exactly} \( d \) constraints by Assumption 1 and Assumption 5. Second,
note that all points in \( e \) binds at least \( d - 1 \) constraints in \( P \) and since \( e \subseteq \hat{N}(t) \), then all points in \( e \) binds at least \( d \) constraints in \( P(t) \). It follows that any vertex of \( P(t) \) on \( e \) must bind at least \( (d + 1) \) constraints, which rules out the possibility of \( \hat{x}^{(t)} \) being on \( e \).

For completeness, in Lemma 8, we show that we are guaranteed the existence of an edge-space if the update implemented is \( U.3 \) or \( U.4 \).

**Lemma 8.** (1) If update rule \( U.3 \) is to be used, then there exists edge-space \( \hat{c} \in E^{(t)} \) such that \( \hat{x}^{(t)} \in \hat{c} \). (2) If update rule \( U.4 \) is to be used, then there exists edge-space \( c \in E^{(t)} \) such that \( x^{(t)} \in e \).

**Proof.** We first consider the case in which \( c \cdot \hat{x}^{(t)} > c \cdot x^{(t)} \) and \( \hat{x}^{(t)} \in \{x \in X^{(t)} : \forall e \in E^{(t)}, x \notin e \} \). When this is the case we know that \( \hat{x}^{(t)} \) is feasible at day \( t \) and, hence, contradicts \( x^{(t)} \) being optimal at that day because \( c \cdot \hat{x}^{(t)} > c \cdot x^{(t)} \).

We next consider the case in which \( c \cdot \hat{x}^{(t)} < c \cdot x^{(t)} \) and \( x^{(t)} \in \{x \in X^{(t)} : \forall e \in E^{(t)}, x \notin e \} \). We would have used \( P.3 \) to make a prediction because \( \hat{N}(t) \cap \text{Ext}^{(t)} \) is nonempty and includes at least the point \( x^{(t)} \). Note that by \( P.3 \), we have \( \hat{x}^{(t)} = \text{argmax}_{x \in \hat{N}(t) \cap \text{Ext}^{(t)}} c \cdot x \). Since \( x^{(t)} \in \hat{N}(t) \cap \text{Ext}^{(t)} \), we must also have \( c \cdot \hat{x}^{(t)} \geq c \cdot x^{(t)} \), which is a contradiction.

A.2 Section 3.2

**Proof of Lemma 4** As soon as \( \text{LearnEdge} \) invokes update rule \( U.1 \), it records the solution \( x^* = \text{argmax}_{x \in P} c \cdot x \). Then, the prediction rule specified by \( P.1 \) prevents further updates of this type. This is because \( x^* \) continues to remain optimal if it feasible in the more constrained problem (optimizing over the polytope \( P(t) \)).

**Proof of Lemma 5** Rule \( U.2 \) is invoked only when \( x^{(t)} \notin X^{(t)} \) and \( x^{(t)} \notin e \) for any \( e \in E^{(t)} \). So after each invocation, a new point on the edge of \( P \) is observed. Whenever 3 points are observed on the same edge of \( P \), the edge-space is learned by Lemma 2 (since the points are necessarily collinear).

Hence, the total number of times rule \( U.2 \) can be invoked is at most \( 3|E_p| \).

We now introduce Lemmas 9 and 10 that will be used in the proof of Lemma 11 which itself will be useful in the proof of Lemma 6.

**Lemma 9.** If \( U.3 \) is implemented at day \( t \), then \( \hat{x}^{(t)} \notin P \) and \( \hat{x}^{(t)} \in (Q^{(t)}_e) \cap \text{Ext}^{(t)} \) for some \( i = 0 \) or 1 where \( \hat{e} \) is given in \( U.3 \).

**Proof.** Each time the algorithm makes update \( U.3 \) we know that the algorithm’s prediction \( \hat{x}^{(t)} \) was on some edge-space \( \hat{e} \in E^{(t)} \). Therefore, \( \text{LearnEdge} \) did not use \( P.1 \) or \( P.2 \) to predict \( \hat{x}^{(t)} \). So we only need to check \( P.3 \) and \( P.4 \).

- If \( P.3 \) was used, we know that \( \hat{x}^{(t)} \in \hat{N}(t) \) but must violate a constraint of \( P \), due to \( x^{(t)} \) being the observed solution and having lower objective value. This implies that \( \hat{x}^{(t)} \) is in some questionable region, say \((Q^{(t)}_i) \) for \( i = 0 \) or 1 but also in the extended feasible on \( \hat{e} \), i.e. \( \hat{x}^{(t)} \in \text{Ext}^{(t)} \cap (Q^{(t)}_i) \).
- If \( P.4 \) was used, then \( \text{Ext}^{(t)} = \emptyset \). However \( \text{LearnEdge} \) selected \( \hat{x}^{(t)} \) from \( \text{Cand}^{(t)} \neq \emptyset \) with the lowest objective value. Finally, when updating with \( U.3 \) (i) \( \hat{x}^{(t)} \in \text{Cand}^{(t)} \) and (ii) \( c \cdot \hat{x}^{(t)} < c \cdot x^{(t)} \).

So we could not have used \( P.4 \) to predict \( \hat{x}^{(t)} \).

**Lemma 10.** If \( U.4 \) is implemented at day \( t \), then \( x^{(t)} \in (Q^{(t)}_e) \setminus \text{Ext}^{(t)} \) for some \( i = 0 \) or 1 where \( e \) is given in \( U.4 \).

**Proof.** As in Lemma 9, \( \text{LearnEdge} \) did not use \( P.1 \) or \( P.2 \) to predict \( \hat{x}^{(t)} \). So we only need to check \( P.3 \) and \( P.4 \).

- If \( P.3 \) was used, then \( \text{LearnEdge} \) did not guess \( x^{(t)} \) which had the higher objective because it was outside of \( \text{Ext}^{(t)} \) along edge-space \( e \). Since \( x^{(t)} \) is feasible, it must have been on some questionable region on \( e \), say \((Q^{(t)}_i) \) for some \( i = 0 \) or 1. Hence, \( x^{(t)} \in (Q^{(t)}_i) \setminus \text{Ext}^{(t)} \).
If $\textbf{P.4}$ was used, then $\text{Ext}^{(t)} = \emptyset$ and thus $x^{(t)}$ was a candidate solution but outside of the extended feasible interval along edge-space $e$. Further, because $x^{(t)} \in \mathcal{P}$ we know that $x^{(t)}$ must be in some questionable interval along $e$, say $(Q_e^i)^{(t)}$ for some $i = 0$ or $1$. Therefore, $x^{(t)} \in (Q_e^i)^{(t)} \setminus \text{Ext}^{(t)}$. 

**Lemma 11.** Each time $\textbf{U.3}$ or $\textbf{U.4}$ is used, there is a questionable interval on some edge-space whose length is decreased by at least a factor of two.

**Proof.** From Lemma 9 we know that if $\textbf{U.3}$ is used then $\hat{x}^{(t)} \notin \hat{e}$, is infeasible but outside of the known infeasible interval $(Y_e^i)^{(t)}$ and inside of the extended feasible interval along $\hat{e}$. Note that if a point $x$ is infeasible along edge space $\hat{e}$ in the questionable interval $(Q_e^{i})^{(t)}$, then the constraint it violates is also violated by all points in $Y_e^i$. Hence the interval $\text{conv}(x, (Y_e^i)^{(t)})$ contains only infeasible points. By the definition of $(M_e^{i})^{(t)}$ and the fact that $\hat{x}^{(t)}$ is in the extended feasible region on $\hat{e}$, we know that

$$
||Q_e^{i}(t+1)|| = \left|\left| (Q_e^{i})^{(t)} \setminus \text{conv} \left( \hat{x}^{(t)}, (Y_e^i)^{(t)} \right) \right| \right| \leq \left|\left| (Q_e^{i})^{(t)} \setminus \text{conv} \left( (M_e^{i})^{(t)}, (Y_e^i)^{(t)} \right) \right| \right| = \frac{||Q_e^{i}(t)||}{2}.
$$

Further, from convexity we know that if $x^{(t)}$ is feasible on edge-space $e$ at day $t$, then the interval $\text{conv}(x^{(t)}, F_e^{(t)})$ only contains feasible points on $e$. We know that $x^{(t)}$ is feasible and in a questionable interval $(Q_e^{i})^{(t)}$ along edge space $e$ but outside its extended feasible region, by Lemma 10. Thus, by definition of $(M_e^{i})^{(t)}$ we have

$$
||Q_e^{i}(t+1)|| = \left|\left| (Q_e^{i})^{(t)} \setminus \text{conv} \left( x^{(t)}, (F_e^{(t)}) \right) \right| \right| \leq \left|\left| (Q_e^{i})^{(t)} \setminus \text{conv} \left( (M_e^{i})^{(t)}, F_e^{(t)} \right) \right| \right| = \frac{||Q_e^{i}(t)||}{2}.
$$

### A.3 Section 3.3

We prove the impossibility result in Theorem 7 initially for $d = 3$.

**Theorem 14.** If Assumptions 1 and 3 hold, then the number of mistakes of any learning algorithm is at least $\Omega(N)$ for $d = 3$.

**Proof.** The high level idea of the proof is as follows. In each day the adversary can pick two points on the two bold edges in Figure 2 as the optimal points and no matter what the learner predicts, the adversary can return a point that is different than the guess of the learner as the optimal point. If the adversary picks the midpoint of the questionable region in each day, then the size of the questionable region in both of the lines will shrink in half. So this process can be repeated $N$ times where each entry of every vertex can be written with as a multiple of $1/2^N$, by Assumption 3. Finally, we show that at the end of this process, the adversary can return a simple polytope which is consistent with all the observed optimal points so far.

We formalize this high level in procedure $\textbf{ADVERSARY}$ that takes as input any learning algorithm $\mathcal{L}$ and interacts with $\mathcal{L}$ for $N$ days. Each day the adversary presents a constraint. Then no matter what $\mathcal{L}$ predicts, the adversary ensures that $\mathcal{L}$’s prediction is incorrect. After $N$ interactions, the adversary outputs a feasible polytope that is consistent with all of the actions of the adversary.

In procedure $\textbf{ADVERSARY}$, subroutines $\textbf{NAC}$ and $\textbf{AD-2}$ are used to pick a constraint and return an optimal point that causes $\mathcal{L}$ to make a mistake, respectively. We use the notation mid$(R)$ in subroutines $\textbf{NAC}$ and $\textbf{AD-2}$ to denote the middle point of a real interval $R$, top$(R)$ be to the largest point in $R$, and bot$(R)$ to be the smallest value in $R$. Finally, we assume the known objective function is $c = (0, 0, 1)$.

The procedure $\textbf{NAC}$ takes as input two real valued intervals and then outputs two points $r_1$ and $r_2$ as well as the new constraint denoted by the pair $(p, q)$. The two points will be used as input in $\textbf{AD-2}$ along with the learner’s prediction. In procedure $\textbf{AD-2}$ the adversary makes sure that the learner suffers a mistake. On each day, one of the points say $r_2$ produced by $\textbf{NAC}$ has a higher objective than the other one. If the learner chooses $r_2$ then the adversary will simply choose a polytope that makes $r_2$ infeasible so that $r_1$ is actually the optimal point that day. If the learner chooses $r_1$ then the adversary picks $r_2$ as the optimal solution. Note that the three points $r_1$, $r_2$, and $r_3$ computed in $\textbf{NAC}$ all bind
Algorithm 4: Adversary Updates (ADVERSARY)

**Input:** Any learning algorithm $\mathcal{L}$ and bit precision $N$

**Output:** Polytope $\mathcal{P}$ that is consistent with $\mathcal{L}$ making a mistake each day.

```
procedure ADVERSARY($\mathcal{L}, N$)
    Set $R_1^{(0)} = [0, 1], R_2^{(0)} = [1, 2]$.
    for $t = 1, \ldots, N$ do
        $(p^{(t)}, q^{(t)}, r_1^{(t)}, r_2^{(t)}) \leftarrow \text{NAC}(R_1^{(t-1)}, R_2^{(t-1)})$.
        Show constraint $p^{(t)} \cdot x \leq q^{(t)}$ to $\mathcal{L}$.
        Get prediction $\hat{x}^{(t)}$ from $\mathcal{L}$.
        $(\hat{x}^{(t)}, R_1^{(t)}, R_2^{(t)}) \leftarrow \text{AD-2}(R_1^{(t-1)}, R_2^{(t-1)}, r_1^{(t)}, r_2^{(t)}, \hat{x}^{(t)})$.
        Adversary reveals the optimal point $x^{(t)} \neq \hat{x}^{(t)}$ and updates regions $R_1^{(t)}$ and $R_2^{(t)}$.
    end
    $A, b \leftarrow \text{MATRIX}(R_1^N, R_2^N)$  \texttt{▷ Constraint matrix consistent with } $\{x^{(t)} | t \in [N]\}$
    return $A, b$
end procedure
```

the constraint and are not collinear, and thus uniquely define the hyperplane $\{x : p \cdot x = q\}$. Finally, in AD-2 the adversary updates the new feasible region for his use in the next days.

ADVERSARY finishes by actually outputting the polytope that was consistent with the constraints and the optimal solutions he showed at each day. This polytope is defined by constraint matrix $A$ and vector $b$ using the subroutine MATRIX as well as the nonnegativity constraint $x \geq 0$.

To prove that the procedure given in ADVERSARY does in fact make every learner $\mathcal{L}$ make a mistake at every day, we need to show that (i) there exists a simple unknown polytope that is consistent with what the adversary has presented in the previous days. Furthermore, we need to show that (ii) the optimal point returned by the adversary on each day is indeed the optimal point corresponding to the LP with objective $c$ and unknown constraints subject to the additional constraint added on each day.

To show (i) note that point $r_1^{(t)} = (0, 1, \text{mid}(R_1^{(t-1)}))$ is always a feasible point for $t \in [N]$ in the polytope given by $A$ and $b$ and the new constraint added each day will allow $r_1^{(t)}$ to remain feasible.

To show (ii) first note that the new constraint added is always a binding constraint. So by Assumption 5, it is sufficient to check the intersection of the edges of the polytope output by MATRIX and the newly added hyperplane and return the (feasible) point with the highest objective as the optimal point. Second, the following equations define the edges of the polytope which are one dimensional...
subspaces $e^{i,j}$ according to Assumption 4 with $A$ and $b$ being the output of MATRIX.

$$e^{i,j} = \{ \mathbf{x} \in \mathbb{R}^3 : A_i \mathbf{x} = b_i \ \& \ A_j \mathbf{x} = b_j \} \quad i, j \in \{1, 2, 3, 4\}, \quad i \neq j,$$

where $A_i$ is the $i$th row of $A$. Since the first two constraints define two parallel hyperplanes, we only need to consider 5 edges. Let $r_1^{(t)} = (0, 1, \text{mid}(R_1^{(t-1)}))$ and $r_2^{(t)} = (1, \text{mid}(R_2^{(t-1)}), 1 + \epsilon \cdot \text{mid}(R_2^{(t-1)}))$ and show that the new constraint either intersects the edges of the polytope at $r_1^{(t)}$ or $r_2^{(t)}$ or do not intersect with them at all. This will prove that the optimal points shown by the adversary each day is consistent with the unknown polytope.

1. $e^{1,4} = (0, 0, f_1) \cdot s + (0, 1, 0)$ that intersects with the new hyperplane at $r_1^{(t)}$.  
2. $e^{2,3} = (0, f_2, \epsilon \cdot f_2) \cdot s + (1, 0, 1)$ that intersects with the new hyperplane at $r_2^{(t)}$.  
3. $e^{2,4} = (0, 0, 1 + \epsilon \cdot f_2) \cdot s + (1, f_2, 0)$ that does not intersect the new hyperplane unless mid($R_2$) = $f_2$ (which does not happen).  
4. $e^{3,4} = (1, f_2 - 1, 1 + \epsilon \cdot f_2 - f_1) \cdot s + (0, 1, f_1)$ that does not intersect the new hyperplane unless mid($R_2$) = $f_2$ (which does not happen).

Algorithm 5: New Adversarial Constraint (NAC)

\begin{algorithm}
\begin{algorithmic}
\Procedure{NAC}{R_1, R_2}
\State $\epsilon \leftarrow 0.01$.
\State \textbf{Set} $r_1 \leftarrow (0, 1, \text{mid}(R_1))$
\State $r_2 \leftarrow (1, \text{mid}(R_2), 1 + \epsilon \cdot \text{mid}(R_2))$
\State $r_3 \leftarrow (1, \text{mid}(R_2), 0)$.
\State \textbf{Set} $p = (1 - \text{mid}(R_2), 1, 0)$ and $q = 1$
\State \Return $(p, q)$ and $r_1, r_2$.
\EndProcedure
\end{algorithmic}
\end{algorithm}

Algorithm 6: Adaptive Adversary (AD-2)

\begin{algorithm}
\begin{algorithmic}
\Procedure{AD-2}{R_1, R_2, r_1, r_2, x}
\If{$x == r_2$}
\State $x \leftarrow r_1, \ R_2 \leftarrow [\text{bot}(R_2), \text{mid}(R_2)].$ \Comment{$r_1$ and $r_2$ as in Algorithm 5}
\Else
\State $x \leftarrow r_2, \ R_2 \leftarrow \text{mid}(R_2), \text{top}(R_2)$. \Comment{$R_1 \leftarrow \text{mid}(R_1), \text{top}(R_1)$.
\Endif
\Return $x, R_1, R_2$.
\EndProcedure
\end{algorithmic}
\end{algorithm}

Algorithm 7: Matrix consistent with adversaries reports (MATRIX)

\begin{algorithm}
\begin{algorithmic}
\Procedure{MATRIX}{R_1, R_2}
\State Set $f_1 = (\text{top}(R_1) + \text{bot}(R_1)) / 2$ and $f_2 = (\text{top}(R_2) + \text{bot}(R_2)) / 2 \text{ and } \epsilon > 0$
\State $A \leftarrow \begin{pmatrix} -1 & 0 & 0 \\ 1 & 0 & 0 \\ f_1 - 1 - \epsilon & -\epsilon & 1 \\ -(f_2 - 1) \cdot f_1 & f_1 & 0 \end{pmatrix}$ and $b \leftarrow \begin{pmatrix} 0 \\ 1 \\ f_1 - \epsilon \end{pmatrix}.$
\State \Return $A$ and $b$.
\EndProcedure
\end{algorithmic}
\end{algorithm}
5. \(e^{1,3} = (0, -1, f_1) \cdot s + (0, 1, f_1)\) never intersects the hyperplane.

And this concludes the proof.

We can then prove Theorem 7 even for \(d > 3\).

**Proof of Theorem 7** We modify the proof of Theorem 14 to \(d > 3\) by adding dummy variables. These dummy variables are denoted by \(x_{4:d}\). Furthermore, we add dummy constraints \(x_i \geq 0\) for all the dummy variables. We modify the objective function in the proof of Theorem 14 to be \(c = (0, 0, 1, -1, \ldots, -1)\). This will cause all the newly added variables to have no effect on the optimization (they should be set to 0 in the optimal solution) and, hence, the result from Theorem 14 extends to the case when \(d > 3\).

**A.4 Section 3.4**

**Proof of Theorem 8** First, we show that LearnHull makes a mistake only if the true optimal point \(x^{(t)}\) lies outside of the convex hull \(C^{(t-1)}\) formed by the previous observed optimal points \(\{x^{(1)}, \ldots, x^{(t-1)}\}\). Suppose that at day \(t\), the algorithm predicts the point \(\hat{x}^{(t)}\) instead of the optimal point \(x^{(t)}\). Since each point in \(C^{(t-1)}\) is feasible and \(\hat{x}^{(t)}\) is the point with the highest objective value among the points in \(\{x \in C^{(t-1)} \mid p' \cdot x \leq b'\}\), then it must be that \(x^{(t)} \notin C^{(t-1)}\) because otherwise \(c \cdot x^{(t)} > c \cdot \hat{x}^{(t)}\). By Lemma 7, we also know that the probability that \(x^{(t)}\) lies outside of \(C^{(t-1)}\) is no more than \(2|E_T|/t\) in expectation, which also upper bounds the probability of LearnHull making a mistake at day \(t\). Therefore, the expected number of mistakes made by LearnHull over \(T\) days is bounded by the sum of probabilities of making a mistake in each day which is \(\sum_{t=1}^{T} 2|E_T|/t < 2|E_T|/3\).

**Proof of Theorem 9** The deterministic procedure runs \([18 \log(1/\delta)]\) independent instances of the LearnHull each using independently drawn examples. The independent instances are aggregated into a single prediction rule by predicting using the modal prediction (if one exists), and otherwise predicting arbitrarily. Hence, the aggregate prediction is correct whenever at least half of the instances of LearnHull are correct.

We show that if each instance of the LearnHull is run for \(8|E_T|\) days, then the probability that more than half of the instances of LearnHull make a mistake on a newly drawn constraint at day \(T + 1\) is at most \(\delta\). The result is that with probability at least \(1 - \delta\) the majority of instances of LearnHull predict the correct optimal point, and hence the aggregate prediction is also correct.

Let \(Z_i\) be the random variable that denotes the probability that the \(i\)th instance of the LearnHull algorithm makes a mistake on a fresh example, after it has been trained for \(8|E_T|\) days. By Theorem 8, we know \(E[Z_i] \leq 1/4\) for all \(i\). Now by Markov’s inequality,

\[
\Pr\left[Z_i \geq \frac{3}{4}\right] = \Pr\left[Z_i \geq 3 \cdot E[Z_i]\right] \leq \frac{1}{3},
\]

for all \(i\). Hence, the expected number of instances that make a mistake is at most \(1/3\). Finally, since each instance is trained on independent examples, a Chernoff bound implies that the probability that at least half of the instances of LearnHull make a mistake is bounded by \(\delta\).

**B Subroutines for LearnEdge**

We first describe a procedure to detect collinear points. The Procedure COLLINE in Algorithm 9 takes a set of points \(X\) and a new point \(z\) as inputs and checks whether \(z\) and any two points in \(X\) form a
line. If so, the procedure returns that line. Otherwise, it returns the empty set. It is easy to see that the running time of \textsc{Colline}(X, z) is \( O(d|X|^2) \). We simply check if there is any pair \( x, y \in X \) such that \((z - y) = \lambda \cdot (x - z)\) for \( \lambda \in \mathbb{R} \), in which case we output the line that \( x, y \), and \( z \) is on.

### Algorithm 8: \textsc{Colline}

```plaintext
procedure \textsc{Colline}(X, z)
    \( e \leftarrow \emptyset \) and \( \lambda = \perp \).
    for \( x, y \in X \) do
        Set \( w^1 = (x - y) \) and \( w^2 = (x - z) \).
        for \( j \in [d] \) do
            if \( w^1_j = 0 \) and \( w^2_j \neq 0 \) or \( w^1_j \neq 0 \) and \( w^2_j = 0 \) then
                EXIT LOOP;
            else if \( \lambda == \perp \) then
                Set \( \lambda \leftarrow w^1_j/w^2_j \).
            else if \( \lambda \neq w^1_j/w^2_j \) then
                EXIT LOOP;
            else
                \( e \leftarrow \{(x - z) \cdot t + z : t \in \mathbb{R}\} \).
        return \( e \).
end procedure
```

Next, we describe a subroutine that identifies a vertex of the underlying polytope \( P \) given a questionable interval \( Q^e \) of length less than \( 1/2^N \). The procedure will then update the feasible and infeasible region on that edge-space \( e \). In the following, let \( Z \) denote the set of points with coordinates being multiples of \( 1/2^N \).

### Algorithm 9: \textsc{Eliminate}

```plaintext
procedure \textsc{Eliminate}((\( Q^e \))^{(t)}, F^{(t)}_e, (Y^e_i)^{(t)})
    Let \( a, b \) be the endpoints of \((Q^e_i)^{(t)}\) where \( a \) is closer to \( F^{(t)}_e \) and \( b \) is closer to \((Y^e_i)^{(t)}\).
    Let \( v \) be the point in \((Q^e_i)^{(t)} \cup \{a, b\}\) such that \( v \in Z \).
    Let \((Q^e_i)^{(t+1)} \leftarrow \emptyset\), \( F^{(t+1)}_e \leftarrow \text{conv}(F^{(t)}_e \cup \{v\}) \) and \((Y^e_i)^{(t+1)} \leftarrow \text{conv}((Y^e_i)^{(t)} \cup \{v\})\).
end procedure
```

**Lemma 12.** Let \( Q^e_i \) be a questionable interval of length less than \( 1/2^N \), and let \( a \) and \( b \) be the two endpoints of the interval. Then \( Q^e_i \cup \{a, b\} \) contains exactly one point \( v \) in \( Z \), and \( v \) is a vertex of \( P \).

**Proof.** Note that \( Q^e_i \cup \{a, b\} \) must contains a vertex in \( P \), because there exists a vertex that separates the disjoint intervals \( F^e \) and \( Y^e_i \), and all points between these intervals are exactly the points in \( Q^e_i \cup \{a, b\} \).

Note that the length of the interval \( Q^e_i \) is less than \( 1/2^N \), so it contains at most one point in the finite precision set \( Z \). So this point must be \( v \).

Hence \textsc{Eliminate} correctly eliminates the questionable interval and updates \( F^e \) and \( Y^e_i \).

### C Formal Description of the Update Rules in LearnEdge

We present the formal descriptions of the updates of \textsc{LearnEdge} in this section. Each update rule provide an update for only a portion of the information \( I(t) \) at day \( t \) given in (4). For the items not mentioned in an update rule, we simply copy it over into \( I(t+1) \).
U.1 If $x^{(t)} \notin \overline{N}(t)$, then set $x^* \leftarrow x^{(t)}$.

U.2 Else if $x^{(t)} \notin e$ for any $e \in E(t)$ and $x^{(t)} \notin X(t)$, then update the edge set

$$E^{(t+1)} \leftarrow E^{(t)} \cup \text{COLLINE}(X^{(t)}, x^{(t)})$$

U.3 Else if $c \cdot \hat{x}^{(t)} > c \cdot x^{(t)}$, then there exists $\hat{e} \in E^{(t)}$ such that $\hat{x}^{(t)} \in \hat{e}$ and the algorithm updates the questionable interval along this edge-space. Let $(Q_\hat{e}^i)^{(t)}$ be the questionable interval containing $\hat{x}^{(t)}$.
- If the length of $(Q_\hat{e}^i)^{(t)}$ is less than $1/2^N$, update via $\text{Eliminate}((Q_\hat{e}^i)^{(t)}, F_e^{(t)}, (Y_e^i)^{(t)})$.
- Otherwise,

$$\begin{align*}
(Q_\hat{e}^i)^{(t+1)} &\leftarrow (Q_\hat{e}^i)^{(t)} \setminus (Y_e^i)^{(t+1)} \quad \text{where} \quad (Y_e^i)^{(t+1)} \leftarrow \text{conv}\left((Y_e^i)^{(t)}, \hat{x}^{(t)}\right). \quad (11)
\end{align*}$$

U.4 Else $c \cdot \hat{x}^{(t)} < c \cdot x^{(t)}$, then there exists $e \in E^{(t)}$ such that $x^{(t)} \in e$ and the algorithm updates a questionable interval on the edge-space $e$. Let $(Q_e^i)^{(t)}$ be the questionable interval containing $x^{(t)}$.
- If the length of $(Q_e^i)^{(t)}$ is less than $1/2^N$, update via $\text{Eliminate}((Q_e^i)^{(t)}, F_e^{(t)}, (Y_e^i)^{(t)})$.
- Otherwise,

$$\begin{align*}
(Q_e^i)^{(t+1)} &\leftarrow (Q_e^i)^{(t)} \setminus (F_e^{(t)})^{(t+1)} \quad \text{where} \quad (F_e^{(t)})^{(t+1)} \leftarrow \text{conv}\left(F_e^{(t)}, x^{(t)}\right). \quad (12)
\end{align*}$$

D Polynomial Mistake Bound with Exponential Running Time

In this section we give a simple randomized algorithm for the unknown constraints problem, that in expectation makes a number of mistakes that is only linear in the dimension $d$, the number of rows in the unknown constraint matrix $A$ (denoted by $m$), and the bit precision $N$, but which requires exponential running time. When the number of rows is large, this can represent an exponential improvement over the mistake bound of LearnEdge, which is linear in the number of edges on the polytope $P$ defined by $A$. This algorithm which we describe shortly is a randomized variant of the well known halving algorithm [13]. We leave it as an open problem whether the mistake bound achieved by this algorithm can also be achieved by a computationally efficient algorithm.

Let $K$ be the hypothesis class of all polytopes formed by $m$ constraints in $d$ dimensions, such that each entry of each constraint can be written as a multiple of $1/2^N$ (and without loss of generality, up to scaling, has absolute value at most 1). We then have

$$|K| = 2^{O(dmN)}.$$ 

We write $K^{(t)}$ to denote the polytopes that are consistent with the examples and solutions we have seen up to and including day $t$. Note that $|K^{(t)}| \geq 1$ for every $t$ because there is some polytope (specifically the true unknown polytope $\mathcal{P}$) that is consistent with all the optimal solutions. On each day $t$ we keep track of consistent polytopes and more specifically update the set of consistent polytopes by

$$K^{(t+1)} = \left\{ P \in K^{(t)} : \text{argmax}_{x \in \mathcal{P} \cap N^{(t)}} c \cdot x \right\}, \quad (13)$$

where $N^{(t)}$ is the new constraint on day $t$. The formal description of the algorithm, FCP, is presented in Algorithm 10. To predict at each day, FCP selects a polytope $P^{(t)}$ from $K^{(t)}$ uniformly at random and guesses $\hat{x}^{(t)}$ that solves the following LP: $\max_{x \in P^{(t)} \cap N^{(t)}} c \cdot x$.

We now bound the expected number of mistakes that FCP makes.

Theorem 15. The expected number of mistakes that FCP makes is at most $\log(|K|) = O(dmN)$, where the expectation is over the randomness of FCP and possible randomness of adversary.
Algorithm 10: Find Consistent Polytope FCP

procedure FCP
    \( K^{(1)} = \mathcal{K} \)
    \( \triangleright \) Initialize
    Every day, choose \( \hat{P}^{(t)} \in K^{(t)} \) uniformly at random.
    Guess \( \hat{x}^{(t)} \in \arg\max_{x \in \hat{P}^{(t)} \cap N^{(t)}} c \cdot x \).
    After seeing solution \( x^{(t)} \), set \( K^{(t+1)} \) as in (13).
    Set \( t \leftarrow t + 1 \) and repeat.
end procedure

Proof. First note that the probability that FCP does not make a mistake at day \( t \) can be expressed as \( \frac{|K^{(t+1)}|}{|K^{(t)}|} \). This is because if FCP makes a mistake at day \( t \), it must have selected a polytope that will be eliminated at the next day (also note that FCP selects its polytope from among the consistent set uniformly at random). Now consider the product of these probabilities over all days \( t = 1 \ldots T \).

\[
\prod_{t=1}^{T} (1 - P[\text{Mistake at day } t]) = \prod_{t=1}^{T} \frac{|K^{(t+1)}|}{|K^{(t)}|} = \frac{|K^{(T+1)}|}{|K^{(1)}|}.
\]

Finally, note that the expected number of mistakes is the sum of probabilities of making mistakes over all days. Using the inequality \((1 - x) \leq e^{-x}\) for every \( x \in [0, 1] \) and rearranging terms we get

\[
\sum_{t=1}^{T} P[\text{Mistake at day } t] \leq \log \left( \frac{|K^{(1)}|}{|K^{(T+1)}|} \right) \leq O(dmN),
\]

since \( |K^{(1)}| = 2^{O(dmN)} \) and \( |K^{(T+1)}| \geq 1 \).

E Missing Proofs from Section 4

First we state Theorem 16 from Grotschel et al. [11] about the running time of the Ellipsoid algorithm.

**Theorem 16.** Let \( P \subset \mathbb{R}^D \) be a polytope given as the intersection of linear constraints, each specified with \( N \) bits of precision. Given access to a separation oracle which can return, for each candidate solution \( p \notin P \) a hyperplane with \( N \) bits of precision that separates \( p \) from \( P \), the Ellipsoid algorithm outputs a point \( p' \in P \) or outputs \( P \) is empty after a number of iterations that is at most \( \text{poly}(D,N) \).

We are now ready to bound the number of mistakes that LearnEllipsoid makes.

**Proof of Theorem 13** Whenever LearnEllipsoid makes a mistake, there exists a separating hyperplane (given in (10)) that cause the Ellipsoid algorithm to run for another iteration. When LearnEllipsoid reduces \( F \) to a set that contains a single point up to \( N \) bits of precision then predicting via (9) will ensure we never make a mistake again. Hence, the number of mistakes our learning algorithm can commit against an adversary is bounded by the maximum number of iterations for which the Ellipsoid algorithm can be made to run, in the worst case.

We know that \( x^{(t)} \) each day is on a vertex of the polytope \( P^{(t)} \) which is guaranteed to have coordinates specified with at most \( N \) bits of precision by Assumption 11. Theorem 12 guarantees that the solution \( \hat{x}^{(t)} \) that LearnEllipsoid produces in (9) is a vertex solution of \( P^{(t)} \), so \( \hat{x}^{(t)} \) can also be written with \( N \) bits of precision by Assumption 11. Thus every constraint in \( F \) and hence each separating hyperplane can be written with \( d \cdot N \) bits of precision. By Assumption 10 and Theorem 16, we know that the Ellipsoid algorithm will find a point in the feasible region \( F \) after at most \( \text{poly}(n,d,N) \) many iterations which is the same as the number of mistakes of LearnEllipsoid. ■