The Precision of Query Points as a Resource for Learning Convex Polytopes with Membership Queries

Paul Goldberg
University of Warwick
Department of Computer Science
Email: pwg@dcs.warwick.ac.uk

Stephen Kwek
University of Texas at San Antonio
Department of Computer Science
Email: kwek@jazz.cs.utsa.edu

Abstract

We consider the problem of learning convex polytopes from membership queries only, where the learner is initially provided with a single interior point. The class of polytopes learnable in this setting turns out to be those whose vertices can be efficiently enumerated given their bounding hyperplanes. It is an open question whether in general one can enumerate the vertices of a given polytope in time polynomial in the number of vertices. In fact, we show that both problems are equivalent. We also give a query-based algorithm for the related problem of piecewise linear function regression.

The bit complexity of the instances in our queries and the time complexity are polynomial in the bit complexity of the coefficients of the equations defining the bounding hyperplanes. This is consistent with prior established results showing that the weights, not the size, of a neural network determine the complexity of learning. As in previous positive results on learning convex polytopes, the precision of the instances queried can have 'polynomially' high precision. Thus one can view the precision of the input to the membership query oracle as a useful resource. This leads us to investigate learning environments where this precision is limited.

1 Introduction

1.1 Our Main Result

In this paper, we consider learning the concept class of intersections of halfspaces, i.e. convex polytopes, in d-dimensional Euclidean space. The instances in Euclidean space are labeled according to some target convex polytope P. Instances that lie inside P are classified positive while those lying outside P are classified negative. Given an initial positive instance to start with, the learner's task is to determine P exactly by posing membership queries (MQs) to an oracle that returns the classifications of the instances of the learner's choosing.

Clearly, if the equations defining the bounding hyperplanes are allowed to be unrestricted unit-cost real numbers, then learning is impossible even if the instance space is one-dimensional and the target is a half interval. In this case, the learner is simply trying to identify a real number x by asking queries of the form: "$x \leq y$?". Regardless of how many MQs are made, the learner cannot determine x exactly but can only identify arbitrarily small intervals containing x. Thus, we assume that the coefficients of the bounding hyperplanes are rational numbers. Further, we assume that the numerators and denominators have values bounded by m. We call the maximum number of bits, $\lceil \log m \rceil$ needed to encode the numerators and denominators the *bit complexity* of the target concept. The *bit complexity* of an instance is the maximum number of bits needed to encode the numerators and denominators of its (rational) components.

We show that this class of convex polytopes in arbitrary dimension can be identified exactly if we may ask membership queries on instances with bit complexity higher than that of the target concept. Our algorithm runs in time polynomial in the bit complexity β of the target concept, dimension d, and number of vertices and facets (i.e. faces of dimension $d-1$). This is subject to the target polytope belonging to any class for which the vertex enumeration problem is solvable efficiently. The paper [Pro94] gives examples of such classes. An important special case is when the target polytope is a simplex, and we can say that the number of vertices is just $d+1$. In fact, we also show in this paper that both our learning problem and the vertex enumeration problem are essentially equivalent (See Theorem 10). Our algorithm assumes that the bit complexity b of the target is known. In the case where β is not known, our algorithm will produce a polytope that approximates (in terms of its volume) the target by assuming β to be sufficiently small.

1.2 Previous Work on Learning Convex Polytopes

Convex polytopes have been investigated extensively in computational learning theory. They can be viewed as a continuous generalization of CNF Boolean formu-

las which are also very well-studied. By using simple prediction-preserving reductions it can be shown that learnability of the class of convex polytopes without membership queries implies PAC-learnability of CNF formulas [PW90, Lit88]. The latter problem is one of the most challenging open problems in learning theory. In the following, we survey some of the relevant results on the learning of convex polytopes.

1.2.1 Learning Convex Polytopes in Arbitrary Dimension is Hard in General

Mostly negative results have been obtained for learning in the case that the dimension is not held constant. Long and Warmuth [LW93] showed that learning convex polytopes (bounded convex polytopes) in the continuous domain, without membership queries, is as hard as learning polynomial-size circuits, assuming that the chosen representation (whose size dictates a parameter in which the algorithm must be polynomial) is the list of vertices, instead of bounding hyperplanes, of the polytope. It follows that if one-way functions exist, then learning convex polytopes is intractable.

Learning convex polytopes in arbitrary dimension remains difficult even if the number of halfspaces is restricted to some small constant. Blum and Rivest [BR89] showed that finding an intersection of two halfspaces that is *consistent* with a sample of labeled points from the boolean domain, if it exists, is NP-complete. It has been shown that in the boolean domain, exact learning from equivalence and membership queries remains NP-hard if the algorithm is required to find (as ours does) an intersection with the *same* number of halfspaces k, for any fixed $k \geq 3$ [AHHP98, PR94].

1.2.2 Learning Convex Polytopes using Membership Queries

One way of making the learning of convex polytopes tractable using membership queries only is to restrict the number of halfspaces, and sometimes, also the dimension. Bultman and Maass [BM91] presented an algorithm that learns a single halfspace in the discretized domain $\mathbf{Z}_m^2 = (1, \cdots, m)^2$ using $\theta(\log m)$ membership queries in time $O((\log m)^{O(1)})$. Shevchenko [She87] investigated the learning of a single halfspace in $\mathbf{Z}_m^n = (1, ..., m)^n$ in time polynomial in $((\log m)^n)$ by asking $O((\log m)^{(n-1)\lceil \frac{n}{2} \rceil + n})$ membership queries.

Baum [Bau90] presented an algorithm that learns intersections of two halfspaces from examples and membership queries, or from examples alone if the distribution obeys a symmetry condition (homogeneous). This result has been extended by Blum *et al.* [BCGS95] to learn intersections of two (not necessarily homogeneous) halfspaces where the membership queries on points that are distance d from the bounding hyperplanes are unreliable and the distribution has weight 0 within d of the boundary.

1.2.3 Learning Convex Polytopes Using Membership Queries and Labeled Sample

With a membership query oracle, Baum [Bau91] presented an algorithm for solving the consistency problems for, and hence Probably Approximately Correctly (PAC) learning, intersections of s halfspaces in n dimensions in time polynomial in s and n. He assumed that the sample labeled instances are drawn from a distribution that rule out with high confidence pathological configurations with several hyperplanes arbitrarily close to one another or large fractions of the measure right on top of decision boundaries. Subsequently, Kwek and Pitt [KP98] improved on his result by presenting an efficient algorithm that works for arbitrary distributions. Their algorithm makes use of membership queries to resolve the credit assignment problem. That is, it partitions the negative examples into sets where all the examples in each set can be separated by the same bounding hyperplane of the target. This allows that learner to construct a separating halfspace for each of the sets in the partition. Our polytopes learning algorithm presented here can be viewed in some sense as an improvement over these results in that it does not requires an initial sample, but just a single positive instance, to start with. Moreover, we require the learner to exactly identify, instead of PAC learn, the target convex polytope.

The limitations of our method presented here are first, that the target polytope must allow its vertices to be efficiently enumerable (a sufficient condition for this is that at most d facets should meet at any vertex). Moreover, the time complexity and number of membership queries are polynomial not just in the number of facets (as in these earlier algorithms) but also the number of vertices. These two constraints are necessary for our purely membership-query based learning setting, since with just membership queries as our resource, every vertex needs to be "inspected" with MQs to verify that it has not been truncated by an additional undiscovered halfspace. Note that for general polytopes, it is very much an open question whether, given linear inequalities representing their bounding hyperplanes, it is possible to list all their vertices in time polynomial in the number of vertices.

Recently, Kwek [Kwe00] also presented an algorithm for our problem of learning convex polytopes using membership queries. However, his result assumes the target is an upper convex polytope (*i.e.*, unbounded from above) and the number of membership queries made is polynomial in the bit complexity and the number of faces (which could be exponential in the number of facets) of the target. He employed duality so that the bounding hyperplanes are points in the dual space and a membership query corresponds to asking whether a hyperplane cuts through the dual of the target (upper convex) polytope. His algorithm is essentially a 'gift-wrapping' algorithm in the dual space that determines the dual of the target. Our algorithm is an improvement over Kwek's result in that the number of membership queries and the time complexity is polynomial in the number of vertices and number of facets. Further, we do not need to assume that the target is unbounded

from above.

1.3 The Precision of the MQ Oracle

At first glance, our result and the positive results in Section 1.2.3 seems to contradict the general consensus that efficient concept learning in arbitrary dimension (even when the number of halfspaces is small) is a difficult task. However, these algorithms are efficient because of the use of a powerful membership query oracle. More specifically, the instances that are input to the oracle can have bit complexity polynomially higher than the bit complexity of the target concept (as in our result) or the bit complexity of the initial labeled sample (as in the PAC learnability results using membership queries noted in Section 1.2.3). We define the *precision* of the membership query oracle as the bit complexity of queried instances.

Our result is consistent with various known results indicating that the size of the coefficients is a more important determinant of the complexity of the target concept than the dimension. Recently, Abbound *et. al.* [AAB+99] showed that the number of membership queries needed to learn a linear threshold function in the Boolean domain with positive integer weights bounded by t requires $O(n^t)$ membership queries. Also, the worst case mistake bounds of Littlestone's Winnow on-line algorithm [Lit88] and its variants [CBLW95, KW94, HKW96] for learning linear threshold functions are linear in the total number of bits needed to encode the weights. Golea *et. al.* [GBLM98] showed that sample complexity of a neural network is determined more by the magnitude of the weights of the network than its size.

This suggests that the precision of the MQ oracle should be viewed as a resource. If this precision is too high, even if it is polynomially higher, the learning algorithm may not be feasible. On the other hand, if it is the same as the bit complexity of the initial labeled sample, then the learning task is intractable. This is because learning intersections of k halfspaces for any $k \geq 3$ using eqivalence and membership queries is NP-hard [AHHP98, PR94]. Thus, one would like to know what can be learned if the precision of the MQ oracle is allowed to be a few bits more than the bit complexity of the target concept or the initial labeled sample so that the results obtained are more feasible, and the MQ oracle is powerful enough to learn a previously intractable task.

To this end, we extend Angluin's exact learning model [Ang88] for Boolean functions to real-valued functions in a natural way. In our extension, an equivalence query oracle returns a point x together with $f(x)$ if the target f differs from the hypothesis h of the learner at point x in the domain. Instead of having a membership query oracle, we have a *valuation query oracle* which returns $f(x)$ on input x. The precision of the valuation query oracle is the bound on the maximum number of bits needed to represent each component of the instances in the valuation queries. The learner's task is to identify the target function in time, and hence with a number of queries, that is polynomial in the representational size of the target function and the number of variables.

In this extended exact learning model, we investigate the learnability of the function class of convex piecewise linear functions $\mathcal{CPLF}$. We can view a convex k-piecewise linear function f with domain $\mathbf{R}^d$ as

$$f(x) = \max\{f_1(x), ..., f_k(x)\}$$

where the f_i's are linear functions. The following observation shows that efficient exact learnability of this function class would imply that DNF is efficiently learnable in the exact model.

Observation 1 *There is a prediction-preserving transformation of DNF Boolean formulas to convex piecewise linear functions.*

Proof: Here, a boolean attribute has value either -1 or +1 (instead of 0 or 1). Each term $T_i = l_1^i ... l_{k_i}^i$ in f can be represented as a function

$$f_i : \frac{1}{k_i} \sum_{j=1}^{k_i} w_j^i l_j^i$$

where $w_j^i = +1$ if l_j^i is a positive literal and -1 otherwise. Let f_0 be the constant function $(n-1)/n$. Consider the convex piecewise linear functions F formed by $f_0, ..., f_t$ where t is the number of terms. It is straightforward to check that a boolean instance x satisfies a term T_i if and only if $f_i(x) = 1$. Conversely, a boolean instance x falsifies a term T_i if and only if $f_i(x) \leq (n-1)/n$. Thus F maps all positive boolean instances to 1 and all negative boolean instances to $(n-1)/n$. □

Note that the above prediction-preserving transformation is a bijection when the domain of the class of convex piecewise linear functions is restricted to $\{-1, +1\}^n$. That is, the instances of our membership queries and valuation queries are in the same domain. Therefore, the existence of an efficient exact learning algorithm for convex piecewise linear functions using equivalence queries and/or valuation queries would imply DNFs are learnable using (improper) equivalence queries and/or membership queries. However, the observation does not apply when the domain of the target convex piecewise linear functions is the real domain and a valuation query oracle is available. In this case, there is no instance in boolean domain of the DNF learning problem that corresponds to an instance in the real domain of the function learning problem.

We show that convex k-piecewise linear functions can be learned with both equivalence queries and valuation queries in time polynomial in the dimension and k. The precision of the valuation query oracle is one bit more than the bit complexity of the counterexamples returned by the equivalence query oracle. Here, we assume that the attribute values of the counterexamples returned by the equivalence query oracle have a decimal representation. If a rational representation is used, then the bit precision may need to be twice the

bit complexity of the counterexamples. When the dimension is fixed, we show that (improper) equivalence queries alone can ensure efficient learning. The last result assumes that the approximating function need not be a convex piecewise linear function.

2 Some Lemmas Needed to Establish Our Main Result

For the sake of simplicity, we assume that the coefficients of the equations defining the target concept are integers[1] with absolute values bounded by β. The following lemma by Maass and Turan states that the vertices are rational points of polynomially bounded bit complexity.

Lemma 2 *[MT94]* [2] *Suppose the coefficients of the equations defining the target concept are integers with absolute values bounded by β, then the vertices are in $\mathcal{Q}^d_{(8d\beta)^{3d}}$ where $\mathcal{Q}_{(8d\beta)^{3d}} = \{\frac{a}{b} : 0 \leq |a|, |b| < (8d\beta)^{3d}\}$.*

In our algorithm, we often need to determine where a line segment with rational endpoints intersects a linear hyperplane with bit complexity β. Note that, the bit complexity of this point of intersection is again a rational point, and the problem is reduced to determining a number (*i.e.*, the intersection point) on the rational line. The next lemma states that this problem can be solved using binary search.

Lemma 3 *Let x be an arbitrary number in $\mathcal{Q}_m = \{\frac{a}{b} : a, b \in \{0, \cdots, m\}\}$. Suppose we are given an oracle that takes a rational input $y \in \mathcal{Q}$ and answers whether "$x \leq y$". Then we can identify the number x in $O(\log(m))$ time by making $3\lceil\log(m)\rceil$ queries to the oracle.*

Proof: See Appendix A. □

Let H denote the set of bounding hyperplanes forming our target polytope. Without loss of generality, let us assume that the initial positive instance is the origin o. The next two lemmas allow us to find a point lying on an unidentified hyperplane. With this point, Lemma 7 states that the unidentified hyperplane can be constructed.

Lemma 4 *Let H denote the set of bounding hyperplanes forming our target polytope. Suppose that the boundary of our current hypothesis is a proper subset H' of H such that the polytope P' formed is not bounded. Then we can determine an instance p lying on some bounding hyperplane in $H \setminus H'$.*

Proof: We begin by finding a semi-infinite ray with one end at the origin which is contained in the (unbounded) polytope. Such a ray can be expressed as the set of points of the form $\alpha(\lambda_1, \lambda_2, \ldots, \lambda_d)$ where α ranges over the non-negative real numbers. Suppose a linear threshold function f that defines a facet of P' is given by $\gamma_1 x_1 + \gamma_2 x_2 + \ldots + \gamma_d x_d \leq \tau$ (where the x_i's are coordinates and the γ_i's are the parameters of f). Then f imposes the following linear constraint on the parameters of the ray: $\lambda_1\gamma_1 + \lambda_2\gamma_2 + \ldots + \lambda_d\gamma_d \leq \tau$. Hence finding an unbounded ray contained in P' can be solved by linear programming.

Given a suitable ray R, there should exist a point p on R which is on the border of the target polytope P (since P is assumed to be bounded) and necessarily lying on a facet (*i.e.* a $(d-1)$-face) of P which is given by a linear threshold function other than the ones that define the facets of P'.

We can find p by binary search using membership queries, and p may be found to whatever precision we are allowed with the membership queries. The origin is already known to lie in the interior of P, and we may identify a point on R that is exterior to P – starting from an initial guess of $\alpha = 1$, we keep doubling our guess until the point $\alpha\langle\lambda_1, \cdots, \lambda_d\rangle$ is outside the polytope. By Lemma 2, the number of guesses is polynomially bounded. □

Lemma 5 *Suppose our current hypothesis is an intersection of a proper subset H' of H such that the polytope formed is bounded (convex polytope) P'. Suppose that each vertex of P is in the intersection of exactly d facets of P. Then we can determine an instance p lying on some bounding hyperplane in $H \setminus H'$.*

Proof: Let P be the target polytope, P' the current hypothesis (formed from a subset of P's bounding hyperplanes), so that $P \subseteq P'$. The general idea is to try to enumerate the vertices of P', checking that each one is a vertex of P. Starting from a vertex v of P', where v is the intersection of d of the halfspaces defining P'; call this set $S_v = \{H_1, \ldots, H_d\}$. A vertex v' adjacent to v will be defined by halfspaces $S_{v'} = S_v \cup \{H'\} \setminus \{H_i\}$ for some $H_i \in S_v$, $H' \notin S_v$. For each appropriate H_i, H' we can

1. see if v' is a new vertex
2. see if v' is a vertex of P.

If v' is not a vertex of P, we will find a new hyperplane of P cutting the line segment between the origin and v'. (We then use Lemma 7 to identify the new hyperplane and find a vertex that it contains.) If v' is a vertex of P, add it to our collection and subsequently test neighbours of v'. □

The above lemma makes the assumption that each vertex is the intersection of exactly d of the faces (and no more). This holds for simplices and hypercubes for example (although in the latter case there are exponentially many vertices). The assumption is needed

[1] We can do so since any linear equation with rational coefficients can be expressed using integer coefficients by multiplying all the coefficients by the least common multiple of the denominators of the coefficients.

[2] The original statement of this lemma is different from our version. The original lemma states that if the instances space is $[0, \cdots, m]^d$ then any linear halfspace is equivalent to one with coefficients in $[0, \cdots, (8dm)^{3d}]$. Our version is a dual of the original lemma.

since the algorithm necessarily performs vertex enumeration, which as noted previously is an open problem in the general case. A solution would allow the algorithm presented here to apply in general (in time polynomial in the number of vertices together with the number of facets).

Lemma 6 *Suppose p is the point found in Lemma 4 or Lemma 5, and has bit complexity b. That is, the denominator of each component is at most 2^b. (1) If p does not lie on a bounding hyperplane h then the distance from p to h is at least $r = 1/2^b d\beta^2$. (2) Further the ball $B(p,r)$ with center p and radius r lies entirely inside the current hypothesis.*

Proof: Suppose $h : \sum_{i=1}^{d} c_i x_i = c_{d+1}$ is a bounding hyperplane that does not contain p. Then the shortest distance between p and h is

$$\frac{c \cdot (p - \frac{c_{d+1}}{c_i} u_i)}{||c||}$$

where c_i is a coefficient that is non-zero and u_i is the ith unit vector. This distance is at least $1/2^b \beta \sqrt{d\beta^2}$ which is larger than $1/2^b d\beta^2$.

To prove the second statement, it suffices to show that the point p does not lie on the boundary of the current hypothesis. This is clearly true for the case of Lemma 4. The same can be shown to hold for the point p in Lemma 5 as follows. Suppose on the contrary that p lies on the boundary of our hypothesis. Consider an arbitrary two-dimensional space S containing the line ov where v is the vertex of P' that contains p (see the proof of Lemma 5). The intersection of the target and S is a convex polygon with p and v lying on the boundary, and o in the interior. This is impossible since p lies between o and v! □

Lemma 7 *Given the point p constructed from Lemma 4 or 5, the learner can construct a bounding hyperplane defining h that is not in our hypothesis.*

Proof: For simplicity, let us assume the following.

Assumption 1: p does not lie on a facet of dimension smaller than $d-2$. That is, p lies in the interior of a face f of the target that has not been determined.

Assumption 2: p does not lie on an axis.

We will remove these two assumptions later in our proof. With these two assumptions, we can find d mutually independent points $q_1, ..., q_d$ that lie on some face f as follows.

Let S_i be the two-dimensional subspace that contains o and the line $l_i = p + tu_i$ where $t \in R$ and u_i is the ith unit vector(see Figure 1). Note that S_i would have been a one-dimensional if p lies on the ith-axis (*i.e.*, Assumption 2 is not true). The intersection of the target with S_i is a convex polygon P_i since S_i contains an interior point o and a boundary point p. Further there is a face f such that $f \cap S_i$ is an edge which p lies on. Denote this edge by e_i. Consider the two points $p_i^- = p - ru_i$ and $p_i^+ = p + ru_i$, which are on the sphere $B(p,r)$. If both points are in the target concept, then by convexity and the fact that p lies on the edge e_i, both points must lie on e_i and hence f. Thus, we can arbitrarily set q_i to either of these points.

Next, suppose one of these two points, call it p_i, is outside the target. Let s be the point $p - r' \cdot \vec{po}$ where $r' = \max\left(r, \left(\frac{r}{||po||}\right)^2\right)$. Clearly, s is in the ball $B(p,r)$ centered at p with radius r. Since s lies between the two positive points o and p, it is also in the target polytope. Further by statement 1 of Lemma 6, no vertex of P_i is inside $B(p,r)$. Therefore the point of intersection q_i, of the line $\overline{p_i s_i}$ and P_i is on the edge e_i and hence on f. This point can be determined by the binary search of Lemma 3.

Now, if Assumption 1 is not true, the points $q_1, \cdots, q_d$ produced by the above procedure may not lie on the same face. To circumvent this difficulty, instead of picking p deterministically, we randomly pick a point p' to replace p in a similar fashion as selecting q_i as follows. We randomly pick a vector u that has polynomial bit complexity and have Euclidean norm smaller than r. We let $p^+ = p + u$ and $p^- = p - u$. If both points are inside the target concept, then we select another u until we have one of p^+ and p^- outside the target. Note that since the measure of the boundary of a face is zero (relative to the face), we are almost certain that the desired p^+ and p^- can be determined in the first trial. In the worst case, we can try another u with almost certainty of succeeding. Say p^+ is outside the target. Replacing p_i by p^+ in the procedure for determining q_i, we can determine another point p' that is on e and hence also on the boundary. Again, since the measure of the boundary of a face is zero, p' is very likely to fall on the interior of an undermined face. Using Lemma 8, we can verify that the points $q_1, \cdots, q_d$ obtained is in the interior of a face with almost certainty. In the worst case, we can repeat the process until we get the desired p'. Replacing p by p', we can determine the desired points $q_1, ..., q_k$. Note that with the new p, we need to recalculate r base on the bit complexity of p' (which is still polynomial).

Finally, suppose Assumption 2 is not true and p is on the ith-axis. In this case, the above algorithm can still determine the $d-1$ points $q_1, ..., q_{i-1}, q_{i+1}, ..., q_d$ which together with p gives us a collection of d mutually independent points that lie on the same face. □

Lemma 8 *Suppose we are given the set of points $Q = \{q_1, \cdots, q_d\}$ constructed in Lemma 7. By posing at most d membership queries, we can determine if all the points in Q lie on the same face.*

Proof: Let q_i and q_j be two arbitrary points in Q. Consider the two dimensional subspace S containing p, q_i and q_j (see Figure 2). If q_i does not lie on a face f_j that q_j (and p) lie, then $q_i' = p + q_i\vec{p}$ lies on the opposite side of pq_j and hence h_j. Therefore q_i' is outside

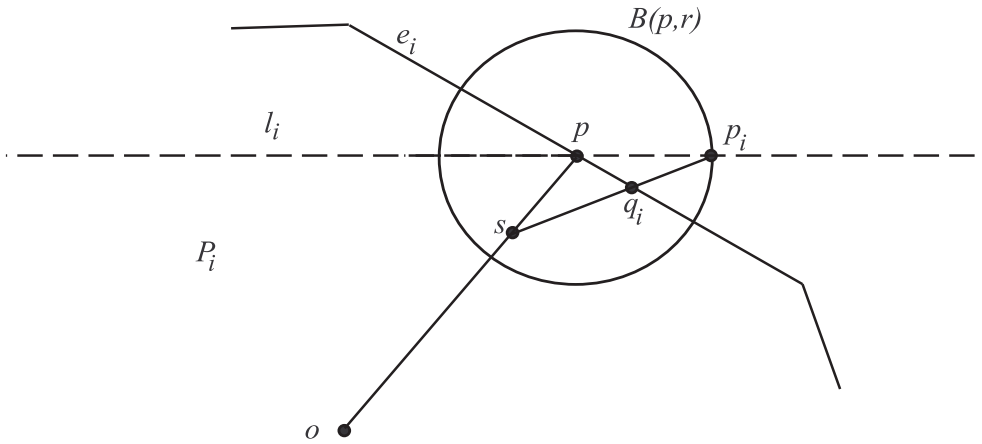

Figure 1: The point q_i is obtained by performing a binary search to find a boundary point between $p_i = p + ru_i$ and $s = p - r'\vec{op}$.

the target concept. On the other hand, if the faces which q_i and q_j lie are the same, then by Lemma 6, the ball $B(p, r)$ intersects S in a circle with center at p and radius r. Hence q_i', which is inside this circle, is inside the target. Therefore, all the points in Q lie on the same face if and only if all the q_i''s are also inside the target. □

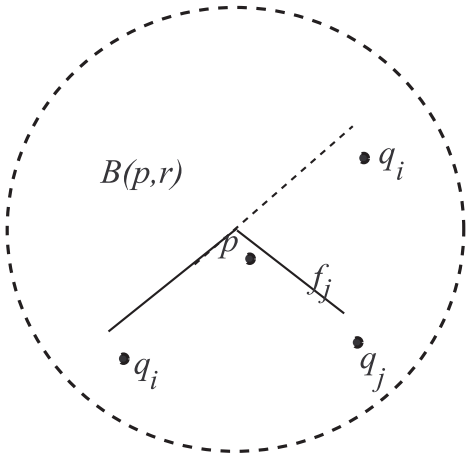

Figure 2: All the q_i's are lie on the same face if and only if all the q_i''s are inside the target polytope.

3 Learning Convex Polytopes with Membership Queries

With the lemmas obtained in the previous section, it is straightforward to design an algorithm that learns convex polytopes using membership queries (see Figure 3. Briefly, with the lemmas in Section 2, we can simply use Lemma 5 to find a suitable point lying on the interior of an unknown face. Using Lemma 7, we can construct the bounding hyperplane that contains this face. We keep doing this until the hypothesis becomes bounded. We then repeat the same process with Lemma 5 until all the vertices in the hypothesis are also vertices in the target, and output the hypothesis.

```
LearnPolytope
1.  V = ∅ {V will be vertex set of target polytope}
2.  F = ∅ {F will be set of bounding hyperplanes}
3.  while R = findray(F) succeeds (i.e. finds a ray)
4.      p = threshold(R, 2b)
5.      f = facet(p)
6.      F = F ∪ {f}
7.  end while { by now, F defines a bounded polytope}
8.  repeat
9.      let v be a vertex of polytope(F) not in V
10.     if v ∈ P (satisfies MQ) then V = V ∪ {v}
11.     else p = threshold(origin, v)
12.     f = facet(p)
13.     F = F ∪ {f}
14. until no new vertices v are found.
```

Figure 3: An algorithm for learning convex polytopes containing the origin with MQs

The algorithm uses procedures $threshold(R, a)$ which finds (to a given precision a) a threshold between positive and negative classification for points along a given ray R (using lemma 5). Procedure $findray(F)$ finds a semi-infinite ray from the origin that is contained in a convex polytope (using lemma 4). It succeeds provided that the polytope bounded by set F of hyperplanes is

unbounded. $facet(p)$ finds a bounding halfspace containing threshold point p (using lemma 7.)

The bound on the bit complexity of the instances used in our algorithm is polynomially bounded. However, the expression for a reasonable bound based on the basic parameters - dimension, β and the number of faces in our target concept, is quite complex. Further, the main intention of this paper is simply to illustrate that convex polytopes can be learned in polynomial time which was previously not known. Giving a detailed analysis of the algorithm will obscure the main idea behind our work. Hence, instead of deriving an expression for the bound, we derive a bound that is polynomial in terms of other polynomial functions and simply give its closed from without giving the details of the derivation.

The bit complexity of the point p in Lemma 4 is some polynomial $poly_{LP}(k, d, \beta)$ where $poly_{LP}(k, d, \beta)$ is the bit complexity of the output produced by solving a linear equation with k constraints, d unknowns and the coefficients in the linear program have bit complexity β. By Lemma 2, the bit complexity of a vertex in P is $3d\log(8md)$. The bit complexity of the point p in Lemma 5 is

$$poly_{intersect}(d-1, \beta, \max(poly_{LP}(k, d, \beta), 3d\log(8md)), 0)$$

where $poly_{intersect}(d-1, \beta, a, b)$ is the bit complexity of the point of intersection of a $d-1$-dimension linear hyperplane with a line segment with one endpoint having bit complexity a and the other having bit complexity b. It is easy to show that $poly_{intersect}(d, \beta, a, b) \leq 2d\beta(a + b)$. Let $bit(a)$ denote the bit complexity of a point a. In the proof of Lemma 7, the bit complexity of the point p^+ is $bit(p)+bit(u)$ where u is the offset of p^+ from p and it has polynomial bit complexity. Since p' is the intersection of a bounding hyperplane with the line segment $\overline{op^+}$, $bit(p') = poly_{intersect}(d, \beta, bit(p^+), 0)$. Now, $s = (1-r)\vec{op'}$ and $p_i = p+ru_i$, and hence both have bit complexity $-\log r+bit(p')$. Finally, q_i is the intersection of a bounding hyperplane with the line segment $\overline{sp_i}$. Therefore $bit(q_i) = poly_{intersect}(d, \beta, bit(s), bit(p_i))$. Thus, all the points considered here have polynomial bit complexity. Using the fact that $poly_{intersect}(d, \beta, a, b) \leq 2d\beta(a + b)$, one can bound their bit complexity by

$$\mathcal{B} = O\left(d^3\beta^3 \max(poly_{LP}(k, d, \beta), 3d\log(8\beta d))\right).$$

The binary search is always performed on a line segment where the endpoints and the desired intersection points have bit complexity bounded by $\mathcal{B}$. Therefore, by Lemma 3, the number of membership queries in our binary search is bounded by $3\mathcal{B}$. Further, the number of binary searches performed is bounded by $O(kd + v)$ where k is the number of faces and v is the number of vertices in out target polytope. Therefore, the total number of membership queries is $O(\mathcal{B}(kd + v))$.

The time complexity of our algorithm is dominated by the time to perform the binary search and solving at most k linear programming problems with d variables and d constraints. Thus, the time complexity is $O\left((\mathcal{B}(kd + v) + k) + k\mathcal{LP}(k, d)\right)$, where $\mathcal{LP}(k, d)$ is the time complexity for solving a linear programming problems with k constraints and d variables.

Theorem 9 *Suppose the target concept comes from a class of convex k-polytopes in $\mathcal{R}^d$ where the vertex enumeration problem can be solved efficiently. Further, suppose it has bit complexity β and v vertices. Then the target can be determined by asking*

$$O(\mathcal{B}(kd + v))$$

membership queries on instances that have bit complexity bounded by

$$\mathcal{B} = O\left(d^3\beta^3 \max(poly_{LP}(k, d, \beta), 3d\log(8\beta d))\right)$$

in time

$$O\left((\mathcal{B}(kd + v) + k) + k\mathcal{LP}(k, d)\right) .$$

Here, $poly_{LP}(k, d, \beta)$ is the bit complexity of the output produced by solving a linear equation with k constraints, d unknowns and the coefficients in the linear program have bit complexity β. $\mathcal{LP}(k, d)$ is the time complexity for solving a linear programming problems with k constraints and d variables. □

In fact, we show in the next theorem that the converse of Theorem 9 is true.

Theorem 10 *The problem of learning convex polytopes using membership queries and the vertex enumeration problem are equivalent.*

Proof: In the dual space, the vertices and bounding hyperplanes of a polytope P correspond to the bounding hyperplanes and vertices in the dual $D(P)$ of P. Thus, if we can determine all the bounding hyperplanes in $D(P)$ then we have solved the vertex enumeration problem. Further, a point p is inside $D(P)$ if and only if p is a convex combination of the vertices in $D(P)$. Since these vertices corresponds to the hyperplanes of P which are known, we can determine if p is in $D(P)$. □

4 Learning Convex Piecewise Linear Functions

Theorem 11 *In domain $\mathbf{R}^d$, $\mathcal{CPLF}$ can be learned using at most $k(d+1)$ equivalence queries and $k^2(d+1)$ valuation queries. Here, k is the number of linear functions in the target. The time complexity of the algorithm is*

$$O\left(k(d+1)LP(k(d+1), d+1)\right)$$

where $LP(m, n)$ is the time complexity for solving a linear program with m constraints and n variables. If the counterexamples returned have decimal representation, then the precision of the valuation query oracle is one bit more than the bit complexity of the instances returned by the equivalence query oracle. If the counterexamples returned are in $\mathcal{Q}_m = \{\frac{p}{q} : 0 \leq p, q \leq m\}$, then the precision of the valuation query oracle needed is at most $4\lceil\log m\rceil + 2$.

Proof: First, suppose we have an oracle $\mathcal{O}$ that on input x_1 and x_2, answers whether there exits an i such that $f(x_1) = f_i(x_1)$ and $f_i(x_2) = f(x_2)$. That is, the oracle tells the learner whether the points $\langle x_1, f(x_1)\rangle$ and $\langle x_2, f(x_2)\rangle$ fall on the same linear surface of the target function. Using $\mathcal{O}$, the learner can separate the labeled counterexamples returned by the equivalence query oracle into bags $P_1, ..., P_j, j \leq k$ such that all the points in the same bag lie on the same linear surface of f. Note that a point may appear in more than one bag if it lies on a facet. The hypothesis produced by the learner is a convex j-piecewise linear function

$$h(x) = \max(h_1(x), ..., h_j(x))$$

such that if $x \in P_i$ then $h(x) = h_i(x)$. The individual h_i can be constructed by using linear programming to find a linear function that fits the points in P_i but lies below all the points in $\cup_{j \neq i} P_j$. The resulting algorithm is shown in Figure 4.

We can simulate the oracle $\mathcal{O}$ by using valuation oracle as follows. Given two labeled points $\langle x_1, f(x_1)\rangle$ and $\langle x_2, f(x_2)\rangle$ as input to $\mathcal{O}$, we use the valuation oracle to determine $f(x')$ where x' is the average of x_1 and x_2. If $\langle x_1, f(x_1)\rangle$, $\langle x_2, f(x_2)\rangle$ and $\langle x', f(x')\rangle$ are collinear then we return YES, otherwise we return NO. The bit complexity of x' is at most one bit more than that of x_1 and x_2 when decimal representation is adopted. If the representation is $\mathcal{Q}_m$ then it is at most $4\lceil \log m \rceil + 2$. To see the latter, consider the (worst) case where $x_1 = \frac{a}{b}$ and $x_2 = \frac{c}{d}$ such that are a, b, c and d are mutually relatively prime. Say $a = m-1$, $b = m-2$, $c = m-3$ and $d = m-4$. Then $x' = (ad + bc)/2bd$ which has bit complexity at most $4\lceil \log m \rceil + 2$.

Clearly, once P_i contains $d+1$ (mutually independent) points, the learner can exactly determine the linear function that defines the values of the points in P_i. Thus, the number of counterexamples seen by the learner is bounded by $k(d+1)$. The number of valuation queries needed is at most k for each counterexample. Thus, the total number of valuation queries needed is at most $k^2(d+1)$. The time complexity in each while loop is dominated by the complexity for solving a linear program in Line 9 and 13. Thus, the time complexity of the entire algorithm is $O\left(k(d+1)LP(k(d+1), d+1)\right)$. □

Learn$\mathcal{CPLF}$-1 in Theorem 11 uses valuation query oracle to separate a set of counterexamples C according to which hyperplanes they lie on. However, if the dimension is constant then we do not need a valuation query oracle to learn $\mathcal{CPLF}$. Figure 5 describe an efficient algorithm that learns $\mathcal{CPLF}$ using only equivalence queries. The learner maintains a collection of sets of labeled instances $A = \{P_1, ..., P_j\}$. Each of these sets has size at most $d+1$. Initially A is empty. Let P_i' denote the set of instances obtained by ignoring the labels of the instances in P_i. To make a prediction on x, the learner first determines those sets P_i's which x is linearly dependent on. We call these sets *relevant* and the others *irrelevant*. If there is no relevant set then the learner simply makes a random guess. Otherwise, for each relevant set $P_i' = \{x_1^i, ..., x_{k_i}^i\}$, say $x = \sum_{j=1}^{k_i} a_j^k x_j^k$, the learner let $P_i(x) = \sum_{j=1}^{k_j} a_j^k f(x_j^k)$. (Note that the values $f(x_j^k)$'s can be obtained from P_i.) The learner then guesses $f(x)$ to be $P_{i^*}(x)$ where $i^* = \arg\max_{i:P_i' \text{ is relevant}} P_i(x)$. When the prediction is a mistake, the learner receives a counterexample $\langle x, f(x)\rangle$. The learner then updates A and the hypothesis h according to the following types of error.

Type 1: The relevant set is not empty and our prediction $P_{i^*}(x)$ is greater than $f(x)$. Here, the learner eliminates from A all the relevant sets P_i such that $P_i(x) > f(x)$.

Type 2: The relevant set is empty or $P_{i^*}(x) < f(x)$. In this case, for each irrelevant set P_i in A, we introduce a new set $P_i \cup \{\langle x, f(x)\rangle\}$ into A.

```
Learn𝒞𝒫ℒℱ-2
1.  A ← ∅
2.  while our hypothesis is not the same as the target
3.      get the counterexample ⟨x, f(x)⟩
4.      if ∃ some relevant sets P_i such that P_i(x) > f(x)
5.          remove from A all such sets
6.      else
7.          for each irrelevant set P_i in A
8.              A ← A ∪ {P_i ∪ {⟨x, f(x)⟩}}
```

Figure 5: An algorithm for learning convex piecewise linear functions in fixed dimension that uses only equivalence queries.

Claim 12 *Suppose C^* is the set of counterexamples seen during Type 1 mistakes. Let $B(C^*)$ denotes the collection of bags obtained in Learn$\mathcal{CPLF}$-1 using $\mathcal{O}$. After each update of A, $B(C^*) \subseteq A$.*

Proof: Initially both A and C^* are both empty and therefore the claim is true to begin with. First, consider the updates due to type 1 mistakes. A set P_i that is being removed from A must be relevant and hence x is a linear combination of P_i. Say $x = \sum_{j=1}^{k_i} a_j^k x_j^k$. By definition of $B(C^*)$, the values of the points in P_i are defined by the same linear function f_l and hence $f_l(x) = \sum_{j=1}^{k_i} a_j^k f_l(x_j^k)$. Therefore $P_i(x) = f_l(x) \leq f(x)$ and P_i is not removed. Next, consider type 2 updates. Before the update, the bag P_i in $B(C^*)$ that contains x must be irrelevant. For otherwise, x is a linear combination of P_i and the above analysis implies that $P_i(x) \leq f(x)$. However $f(x) \neq P_i(x)$ since our prediction is less than $f(x)$. In other words, the points in P_i lie on a different linear surface than x. Thus, $P_i \cup \{x\}$ is not in $B(C^*)$. Hence, all the bags in $B(C^*)$ that contain x must be in the new sets that are added to A. □

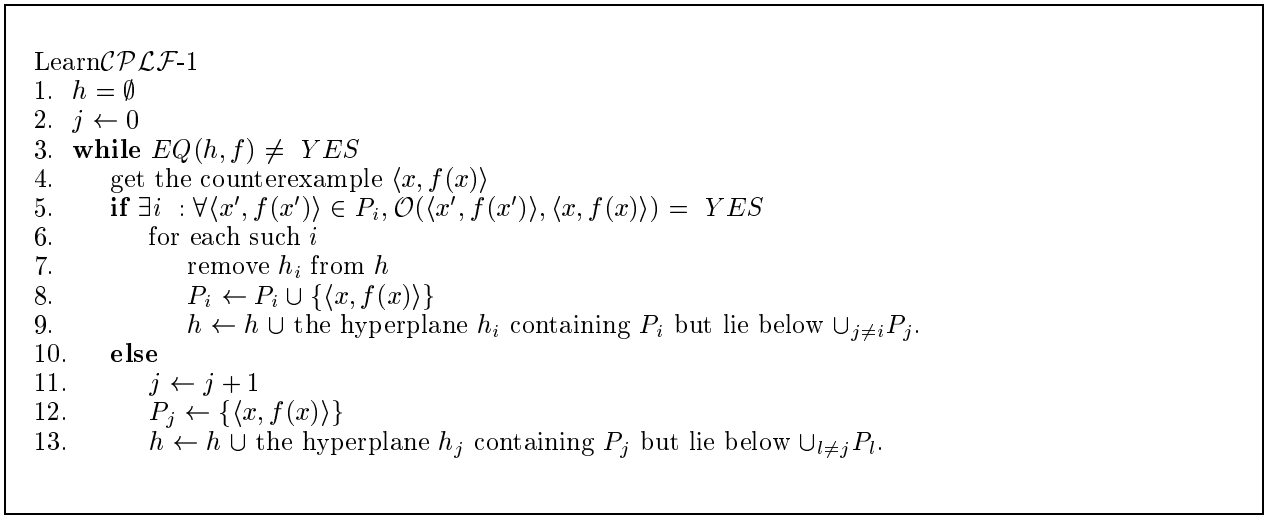

Figure 4: An algorithm for learning convex piecewise linear functions in arbitrary dimension.

The same argument in the above proof together with the fact that the points in each bag in A are mutually independent suggests that once all the sets in $B(C^*)$ have $d+1$ instances and $|B(C^*)| = k$ then Type 2 mistake ceases to occur. Therefore, the number of Type 2 mistakes is at most $k(d+1)$. The number of sets introduced by Type 2 mistakes is at most the number of possible subsets of $k(d+1)$ points that have size at most $(d+1)$. Further, each Type 1 mistakes eliminates at least one set from A and hence there are at most $O\left((k(d+1))^{d+1}\right)$ type 1 mistakes. Therefore, the total number of equivalence queries is at most $O\left((k(d+1))^{d+1}\right)$. The time complexity for checking whether there is a relevant set in each iteration is $O\left((k(d+1))^{d+1}d^2\right)$ which gives rise to the total time complexity.

Theorem 13 *The class of convex k-piecewise linear functions in domain $\mathbf{R}^d$, d is constant, can be learned using at most $O\left((k(d+1))^{d+1}\right)$ (improper) equivalence queries in time $O\left((k(d+1))^{2(d+1)}d^2\right)$.* □

Note that the Learn$\mathcal{CPLF}$-1 uses an improper equivalence query oracle where the hypothesis is not in $\mathcal{CPLF}$. A straightforward attempt to convert it to an algorithm that uses proper equivalence query oracle would be to maintain a hypothesis $h = \max(h_1, ..., h_j)$ where h_i is the linear function that fits P_i but lies below $\cup_{j\neq i} P_j$. However, it is not clear how the learner should update A and the hypothesis when the prediction $h(x)$ is higher than $f(x)$ (*i.e.*, type 1 error). We can no longer simply discard P_i when $h_i(x) > f(x)$. We might have to modify h_i to lie below x instead and only discard P_i when we cannot construct h_i that fits P_i but lies below all the other points. However, the number of counterexamples seen before we can discard P_i may be very large.

References

[AAB+99] Elias Abboud, Nader Agha, Nader H. Bshouty, Nizar Radwan, and Fathi Saleh. Learning threshold functions with small weights using membership queries. In *Proc. 12th Annu. Conf. on Comput. Learning Theory*, pages 318–322. ACM Press, New York, NY, 1999.

[AHHP98] H. Aizenstein, T. Hegedus, L. Hellerstein, and L. Pitt. Complexity theoretic hardness results for query learning. *Computational Complexity*, 7:19–53, 1998.

[Ang88] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, April 1988.

[Bau90] E. Baum. A polynomial time algorithm that learns two hidden net units. *Neural Computation*, 2:510–522, 1990.

[Bau91] E. Baum. Neural net algorithms that learn in polynomial time from examples and queries. *IEEE Transactions on Neural Networks*, 2:5–19, 1991.

[BCGS95] Avrim Blum, Prasad Chalasani, Sally A. Goldman, and Donna K. Slonim. Learning with unreliable boundary queries. In *Proc. 8th Annu. Conf. on Comput. Learning Theory*, pages 98–107. ACM Press, New York, NY, 1995.

[BM91] W. J. Bultman and W. Maass. Fast identification of geometric objects with membership queries. In *Proc. 4th Annu. Workshop on Comput. Learning Theory*, pages 337–353, San Mateo, CA, 1991. Morgan Kaufmann.

[BR89] A. Blum and R. L. Rivest. Training a 3-node neural net is NP-Complete. In *Advances in Neural Information Processing Systems I*, pages 494–501. Morgan Kaufmann, 1989.

[CBLW95] N. Cesa-Bianchi, P. Long, and M. K. Warmuth. Worst-case quadratic loss bounds for on-line prediction of linear functions by gradient descent. *IEEE Transactions on Neural Networks*, 1995. To appear. An extended abstract appeared in COLT '93.

[GBLM98] Mostefa Golea, Peter Bartlett, Wee Sun Lee, and Llew Mason. Generalization in decision trees and DNF: Does size matter? In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Sys-*

tems, volume 10. The MIT Press, 1998.

[HKW96] D. P. Helmbold, J. Kivinen, and M. K. Warmuth. Worst-case loss bounds for sigmoided linear neurons. In *Proc. 1996 Neural Information Processing Conference*, 1996. To appear.

[KP98] Stephen Kwek and Leonard Pitt. PAC learning intersections of halfspaces with membership queries. *Algorithmica: Special Issue on Computational Learning Theory*, pages 53–75, 1998.

[KW94] J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. Technical Report UCSC-CRL-94-16, University of California, Santa Cruz, Computer Research Laboratory, June 1994. Revised December 7, 1995. An extended abstract to appeared in the STOC 95, pp. 209-218.

[Kwe00] Stephen Kwek. An efficient algorithm for learning upper convex polyhedra using membership queries. In *Proc. International Symposium of Artificial Intelligence nd Mathematics*, 2000.

[Lit88] N. Littlestone. Learning when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988.

[LW93] P. M. Long and M. K. Warmuth. Composite geometric concepts and polynomial predictability. *Inform. Comput.*, 1993. To appear.

[MT94] Wolgang Maass and György Turán. *How fast can a threshold gate learn?*, chapter 13, pages 381–414. MIT Press, 1994. Earlier versions appeared in FOCS89 and FOCS90.

[PR94] K. Pillaipakkamnatt and V. Raghavan. On the limits of proper learnability of subclasses of DNF formulas. In *Proc. 7th Annu. ACM Workshop on Comput. Learning Theory*, pages 118–129. ACM Press, New York, NY, 1994.

[Pro94] J.S. Provan. Efficient enumeration of the vertices of polyhedra associated with network lp's. *Mathematical Programming*, 63:47–64, 1994.

[PW90] L. Pitt and M. K. Warmuth. Prediction preserving reducibility. *J. of Comput. Syst. Sci.*, 41(3):430–467, December 1990. Special issue of the for the *Third Annual Conference of Structure in Complexity Theory* (Washington, DC., June 88).

[She87] V. Shevchenko. On deciphering a threshold function of many-values logic. *Grokii State University*, pages 155–163, 1987.

Appendix A: Binary Search on a Rational Line

Let x be an arbitrary number in $\mathcal{Q}_m = \left\{\frac{a}{b} : a, b \in \{0, \cdots, m\}\right\}$. By querying whether $x \leq m$, we can determine if the denominator of x is 0. Thus, without loss of generality, we assume that this is not the case.

Note that x can be expressed as $\lfloor x \rfloor + \frac{a}{b}$ where a and b are relatively prime and $a < b$. Using binary search, we can determine $\lfloor x \rfloor$ by making $\lceil \log(m) \rceil + 1$ queries. To exactly determine the fractional part, we perform a binary search on the unit interval $[\lfloor x \rfloor, \lfloor x \rfloor + 1]$ so that we know $\frac{\alpha}{m^2} \leq \frac{a}{b} \leq \frac{\alpha+1}{m^2}$. This can be done by asking $2\lceil \log(m) \rceil$ - 1 queries.

Lemma 14 *Suppose $\frac{a}{b}, \frac{c}{d} \in \mathcal{Q}_m$ and $\frac{a}{b}, \frac{c}{d} \in [\frac{\alpha}{m^2}, \frac{\alpha+1}{m^2}]$. Then $\frac{a}{b} = \frac{c}{d}$.*

Proof: By way of contradiction, suppose $\frac{a}{b} \neq \frac{c}{d}$, and say $\frac{a}{b} > \frac{c}{d}$. Then

$$0 \leq \frac{a}{b} - \frac{c}{d} \leq \frac{1}{m^2} \Rightarrow 0 < \frac{ad - bc}{bd} < \frac{1}{m^2} \Rightarrow 0 < ad - bc < 1.$$

The last inequality is true since $bd \leq m^2$ and if $bd = m^2$ then $ad - bc \geq m$. This inequality is impossible since a, b, c, d are integers and $\frac{a}{b} \neq \frac{c}{d}$. □

Suppose we know the desired $\frac{a}{b}$ is in $I = [\frac{\alpha}{m^2}, \frac{\alpha+1}{m^2}]$ as in Lemma 14. Clearly, since $\frac{a}{b}$ is the only fraction in $\mathcal{Q}_m$ that is also in I, all fractions in I that is not equal to $\frac{a}{b}$ must have denominator greater than b. Thus, it suffices to find the fraction that has the smallest denominator in I. The next lemma states that such a fraction can be determined in time $O(\log(m))$ without asking any further queries. Thus Lemma 15 completes the proof of Lemma 3.

Lemma 15 *Given an interval $I = [\frac{\alpha}{\beta}, \frac{\gamma}{\delta}]$, there exists a fraction $\frac{a_{min}(I)}{b_{min}(I)}$ in $I = \left[\frac{\alpha}{\beta}, \frac{\gamma}{\delta}\right]$ such that for all $\frac{a}{b} \in I$, $a_{min}(I) \leq a, b_{min}(I) \leq b$. Further, we can determine this fraction in time $O(\log(\max(\alpha, \beta, \gamma, \delta)))$.*

Proof: We prove the existence of $a_{min}(I)$ and $b_{min}(I)$ by constructing it using a recursive algorithm. Our algorithm has the same flavor as Euclid's algorithm for finding the greatest common divisor of two integers. Let $\frac{a}{b}$ be an arbitrary fraction in I. We consider the following two cases.

Case 1: the interval I does not contain any integer.
In this case, we have

$$\frac{\alpha}{\beta} \leq \frac{a}{b} \leq \frac{\gamma}{\delta}$$

and

$$\left\lfloor \frac{\alpha}{\beta} \right\rfloor = \left\lfloor \frac{a}{b} \right\rfloor = \left\lfloor \frac{\gamma}{\delta} \right\rfloor.$$

We can express a as

$$a = \left\lfloor \frac{\alpha}{\beta} \right\rfloor b + (a \bmod b) \qquad (1)$$

Let $a' = a \bmod b$, $\alpha' = \alpha \bmod \beta$ and $\gamma' = \gamma \bmod \delta$. Then, we also have the following inequality.

$$\frac{\delta}{\gamma'} \leq \frac{b}{a'} \leq \frac{\beta}{\alpha'}.$$

That is, $\frac{b}{a'} \in I'$ where $I' = \left[\frac{\delta}{\gamma'}, \frac{\beta}{\alpha'}\right]$. Notice that if there exists $\hat{b}, \hat{a'} \in I'$ such that for all $\frac{b}{a'} \in I', b \geq \hat{b}$ and $a' \geq \hat{a'}$, then substituting $\hat{b}$ for b and $\hat{a'}$ for a' in Equation (1) gives us the smallest a among all feasible b and a' such that $\frac{b}{a'} \in I$. That is, to prove the existence of $a_{min}(I)$ and $b_{min}(I)$, it suffices to prove the existence of $a_{min}(I')$ and $b_{min}(I')$. Similarly, to determine $a_{min}(I)/b_{min}(I)$, it is sufficient to solve the problem with the interval I'.

If I' contains an integer, then the problem instance is reduced to Case 2. Thus, suppose I' does not contain an integer. Notice that $\gamma' \leq \gamma$ and $\alpha' \leq \alpha$.

Suppose $\gamma = \gamma'$ and $\alpha' = \alpha$, then by repeating the above argument, we have

$$\frac{\alpha'}{\beta'} \leq \frac{a'}{b'} \leq \frac{\gamma'}{\delta'}$$

where $b' = b \bmod a'$, $\beta' = \beta \bmod \alpha'$ and $\delta' = \delta \bmod \gamma'$. That is, the problem is reduced to finding $a_{min}(I'')$ and $b_{min}(I'')$ where $I'' = [\frac{\alpha'}{\beta'}, \frac{\gamma'}{\delta'}]$. Further, since $\gamma' < \delta$ and $\alpha' < \beta$, we have $\delta' < \delta$ and $\gamma' < \gamma$.

In other words, by reducing the problem instance (*i.e.* an interval) in this manner, we are sure that $\alpha' + \beta' + \gamma' + \delta' < \alpha + \beta + \gamma + \delta$. Eventually the problem instance must contain an integer and the algorithm terminates (see Case 2). In the worst case, we stop when the interval being considered is $[\frac{1}{1}, \frac{1}{1}]$.

Case 2: the interval I contains an integer. Suppose I contains the integers $z_1 < \cdots < z_k$. We claim that $\forall \frac{a}{b} \in I, a \geq z_1$. This is clearly true if $z_1 = 1$ or $b = 1$ or $\frac{a}{b} \geq z_1$. Thus, suppose $z_1 - 1 < \frac{a}{b} < z_1$ and $b \neq 1$ and $z_1 \neq 1$. Then $a > y(z_1 - 1)$ which implies $a \geq z_1$. Hence we have $a_{min}(I) = z_1$ and $b_{min}(I) = 1$. □

```
findFraction(α, β, γ, δ) : a, b
    if ⌊α/β⌋ = ⌊γ/δ⌋ and α/β ∉ Z                    (Case 1)
        b, a' ← findFraction(δ, γ mod δ, β, α mod β)
        a = ⌊α/β⌋ b + a'                            (Equation 1)
        return a, b
    else        (Case 2)
        return a = ⌈α/β⌉, b = 1
```

Figure 6: An algorithm for finding a fraction $\frac{a_{min}(I)}{b_{min}(I)} \in I = \left[\frac{\alpha}{\beta}, \frac{\gamma}{\delta}\right]$ such that for all $\frac{x}{y} \in I$, $x \geq a_{min}(I), y \geq b_{min}(I)$.

Figure 1 shows an algorithm for determining $a_{min}(I)$ and $b_{min}(I)$. The method we used to reduce the denominator and numerator of the endpoints of I is the same as Euclid's algorithm for finding the greatest common divisor of two numbers $x < y$. The time complexity for Euclid's algorithm is $O(\mathcal{F}^{-1}(\max(x, y))) = O(\log(\max(x, y)))$ where $\mathcal{F}^{-1}(\alpha)$ to be the largest k such that α is less than the kth Fibonacci number. Thus, we have the desired time complexity. □