Learning First-Order Definite Theories via Object-Based Queries

Joseph Selman and Alan Fern

School of Electrical Engineering and Computer Science
Oregon State University
{selman,afern}@eecs.oregonstate.edu

Abstract. We study the problem of exact learning of first-order definite theories via queries, toward the goal of allowing humans to more efficiently teach first-order concepts to computers. Prior work has shown that first order Horn theories can be learned using a polynomial number of membership and equivalence queries [6]. However, these query types are sometimes unnatural for humans to answer and only capture a small fraction of the information that a human teacher might be able to easily communicate. In this work, we enrich the types of information that can be provided by a human teacher and study the associated learning problem from a theoretical perspective. First, we consider allowing queries that ask the teacher for the relevant objects in a training example. Second, we examine a new query type, called a pairing query, where the teacher provides mappings between objects in two different examples. We present algorithms that leverage these new query types as well as restrictions applied to equivalence queries to significantly reduce or eliminate the required number of membership queries, while preserving polynomial learnability. In addition, we give learnability results for certain cases of imperfect teachers. These results show, in theory, the potential for incorporating object-based queries into first-order learning algorithms in order to reduce human teaching effort.

1 Introduction

This work is motivated by the goal of enabling non-experts in first-order logic to easily and efficiently teach first-order concepts to computers. We are interested in situations where the teacher has a reasonable semantic understanding of the target concept but is unable to provide a definition due to lack of expertise in formal knowledge representation. A key research issue is to understand the types of information that a teacher can provide and how to best use them for learning. Most work on learning first-order concepts limits the teacher to labeling examples as positive or negative, which is a restricted and indirect form of teaching.

In this work, we study algorithms that elicit and utilize information about the relevance and correspondence between objects in first-order training examples. Given that examples are often produced and/or analyzed by teachers, it is reasonable to expect that teachers will be able to at least partially provide this

information. The question we ask is: How can we use such information and what will its impact on learning efficiency be?

We study this question from a theoretical perspective where the problem is to exactly identify a target first-order definite theory using labeled examples and queries to a teacher. Prior theoretical work [10, 6] focused on using equivalence queries to obtain positive and negative examples, and on membership queries to obtain example labels from the teacher. While the number of queries is shown to be polynomial in certain problem parameters, the use of only these queries is quite restricted and membership queries can be problematic in practice. For example, an algorithm may ask membership queries about examples that will appear nonsensical to a typical teacher which is a well-documented problem [4]. In general, past work has not given much consideration to how suitable such queries might be for humans to answer.

We consider the use of two new types of queries that are directly about the objects in examples provided by the teacher. As such, it is plausible that the queries might be more easily answered than queries about synthetic examples constructed by the learner. The first query type is a relevant object query that returns the set of "relevant objects" in a specific example. We show that by using this query type it is possible to significantly reduce the number of required membership queries. Further, we show that membership queries can be reduced or eliminated altogether with simple restriction applied to the equivalence oracle. We also analyze what happens when the teacher's responses are imperfect, showing that there can still be benefits if the imperfection is bounded. The second query type considered is the use of pairing queries, which ask the teacher to match "corresponding" objects between two specific examples. We show that with such queries membership queries can be eliminated.

Our main contribution is a theoretical investigation into the use of object-based queries for learning first-order concepts. To our knowledge this is the first such investigation. As our results are of a worst-case nature, the exact bounds and algorithms are likely not directly applicable to practice. However, similar to past work on identifying Horn theories [6], which led to new practical algorithms [3], we expect that our work will lead to practical algorithms for utilizing object-based feedback from teachers.

2 Previous Work

The problem of exactly learning propositional Horn sentences from membership and equivalence queries was shown to have polynomial runtime with a fixed number of variables and a fixed number of clauses [2]. This algorithm was later generalized to learning first-order Horn theories from equivalence and membership queries for several learning settings, including learning from interpretations and learning from entailment [6]. All of the algorithms in this paper are built upon the algorithms given in [6].

A different algorithm using the same query types was also shown to have polynomial runtime with certain fixed problem parameters for first-order Horn

theories where all the consequents involve the same predicate (called Horn programs) [10]. The generalization process used least-general-generalizations with a much finer minimization step than used here. This algorithm was used to learn control knowledge in a planning setting using goal-decomposition rules [11].

The notion of using relevant objects provided by a teacher to initialize a search-based first-order rule learner was examined [13]. Similarly, a system that used "nudges" and object highlighting to learn various constraints was created for programming games [7]. Both approaches incorporate information about objects from the teacher into experimental systems, but neither provide a theoretical analysis of the learning problem, which is the main emphasis of our work.

Recently various error models were introduced for membership queries [5] and the resulting behavior was analyzed, similar to the (j, f)-based queries discussed here. A different type of algorithm analysis that accounts for noise in learning theory is the PAC-Learning framework introduced by [12]. With query learning, it was shown that any language exactly learnable by equivalence queries is also PAC-Learnable [1]. For examples of first-order logic PAC-Learning algorithms that learn with certain fixed problem parameters, see [9], [11] and [6].

3 Preliminaries

We assume a fixed set of first-order predicates P, each with an associated arity that is upper-bounded by a. An *atom* is a predicate from P applied to the proper number of variables or constants. A *literal* is either an atom (positive literal) or the negation of an atom (negative literal). A *clause* is a disjunction of literals and is a *ground clause* if it contains no variables. The *head* of a clause is its set of positive literals and the *body* is the set of negative literals. A *definite clause* is one that contains exactly one positive, or head, literal. A set of definite clauses is a definite theory. We will be interested in the space of all definite theories over P that do not contain constants, denoted by $\mathcal{H}_D(P)$.

Following prior work [6] and without loss of generality, our algorithms employ the notion of d-subsumption (also known as object-identity subsumption) between clauses. A clause C_1 *d-subsumes* clause C_2 if there exists a substitution θ, where no two variables map to the same object, such that $C_1\theta \subseteq C_2$ (viewing clauses as sets of literals). That is, the substitution must be a 1-1 mapping.

Problem Setup. Our goal is to exactly learn an unknown target hypothesis $T \in \mathcal{H}_D(P)$ by interacting with different types of oracles. These oracle types can be thought of as different ways of interacting with a teacher. In particular we are interested in how many queries must be issued to the oracles in order to learn a hypothesis $H \in \mathcal{H}_D(P)$ that is equivalent to T with respect to coverage of examples as defined below.

In our formulation a *training example* is a definite ground clause. The body of an example can be viewed as a set of facts describing the context of an example and the single head literal can be viewed as a predicate that we would like to predict given the context. Readers familiar with inductive logic programming can view the bodies of our examples as encoding the ground background knowledge

that is relevant to a particular example. For example, if the problem was to learn what configurations of a chessboard represent a state where a king is in check, the body might consist of literals describing the state of the board and the head would be a literal representing the concept of check. Note that the head predicate is not restricted to be the same across all examples.

A theory $T \in \mathcal{H}_D(P)$ *covers* an example E if there is at least one clause in T that d-subsumes E. A *positive example* is one that is covered by the target theory, while all other examples are *negative*. Note that, as detailed in prior work [6], there is no loss in generality in defining coverage in terms of d-subsumption rather than standard subsumption. In particular, for any target theory T defined relative to standard subsumption, there is an equivalent theory (possibly larger) under d-subsumption. Note that under our definition, positive examples are required to be directly covered by at least one clause in the target hypothesis. This is a weaker notion of coverage than entailment, since examples that are entailed may not be considered covered according to subsumption. For example, a theory with the two clauses $P(X) \rightarrow Q(X)$ and $Q(X) \rightarrow R(X)$, entails the example $P(c) \rightarrow R(c)$, but does not cover the example under subsumption. This difference does not limit the learnability of $\mathcal{H}_D(P)$, rather it will influence the behavior of the oracles when generating examples and answering queries.

Standard Query Types. The two most common query types in learning theory are equivalence queries and membership queries. They have been used exclusively in past work on learning first-order definitions. Equivalence queries provide a way for algorithms to obtain new counter-examples for the current hypothesis (examples where the hypothesis and target disagree about coverage). Note that in practice humans are not expected to actually answer equivalence queries, since this would require a full understanding of the current hypothesis and target concept, which is clearly impractical. Rather, equivalence queries are best viewed as a theoretical model for a large set of labeled training examples, which can produce counter examples. In this sense, a practical implementation of any of the algorithms described below would implement an equivalence oracle via such a training set that is automatically checked for consistency with the current hypothesis. This type of equivalence oracle implementation has been used successfully in the past for moving from a theoretical algorithm to practice [3]. Formally, in our setting we have:

Definition 1. *An* equivalence query (EQ) *takes a definite theory H as input and returns "done" if H is equivalent to the target theory T with respect to example coverage. Otherwise, a counter-example is returned.*

Membership queries allow an algorithm to generate an example and ask the oracle to label it. There are no restrictions placed on examples used for MQs which is a known problem [4] with human teachers, as the learner can generate nonsensical examples that confuse the teacher.

Definition 2. *A* membership query (MQ) *takes an example E as input and returns "true" if E is a positive example of the target T. Otherwise "false" is returned.*

Our results will provide worst-case bounds on the number of queries (of various types) needed to exactly learn a target theory. These bounds will depend on certain quantities related to the target theory. Table 1 provides a reference for symbols used to represent these quantities.

Variable	Meaning
T	target definite theory
P	fixed set of predicates
a	largest arity of any predicates in P
k	largest number of distinct variables in a clause of T
m	number of clauses in T
n	largest number of objects in any example

Table 1. Notational reference.

4 Base Algorithm

Our algorithms are based on Khardon's algorithmic framework for learning first-order Horn theories from membership and equivalence queries [6]. As a starting point, we describe a base algorithm for our problem that uses membership and equivalence queries. This algorithm is similar to those in [6] but adapted to the specific learning problem outlined above.[1] Our later algorithms are based on viewing membership queries as the "assembly language of learning queries", which are grouped into functional units and replaced by higher-level object-based queries. As we will see, many of the queries used in the base algorithm can be reformulated as queries about objects.

Overview. Algorithm Learn-MQ gives pseudo-code for our base algorithm. The algorithm maintains a current definite theory hypothesis H and a set of ground clauses S from which H is derived. The algorithm initializes S and H to the empty sets and then enters the main loop which updates S and H on each iteration. Each loop iteration begins by issuing an equivalence query. If the result is "done", then H is correct and is returned as the answer. If a positive counter example is returned (we will see that this is always the case for perfect oracles), then MQs are used to incorporate the positive example into the current hypothesis which results in a current clause being generalized or a new clause being added. Note that there is no if condition for a negative counter example; it will become apparent below that our algorithm will never overgeneralize, making negative counter examples unnecessary.

[1] Khardon's main algorithm was for the setting of learning from interpretations. It was then shown how to place a wrapper around this algorithm for learning from entailment, which most resembles our formulation here. The algorithm described here is defined directly for our learning problem rather than taking a wrapper approach and hence is novel, though the key ideas follow rather directly from Khardon's work.

Positive examples are incorporated into H via two steps. First, the procedure call Min-MQ$(E, \emptyset)$, described below, returns an example $E' \subseteq E$ that is still a positive example of the target theory, but where literals involving "irrelevant objects" have been removed. Here irrelevant objects are those that are not necessary in finding a variable substitution showing a target clause covers E. Given the minimized example, the algorithm then calls Pair-MQ(E', S), described below, in order to merge E' with the set of ground clauses in S. The set S is maintained so that distinct ground clauses correspond to distinct clauses in the target theory. The example E' is combined with an existing clause $s \in S$ if E' and s are covered by a common target clause. Otherwise, if no such s exists then E' is added as a new element to S. Given the updated S, the hypothesis H is set to a variabilized version of S (via the call **variabilize**(s), where distinct constants in s are replaced with distinct variables). Below we review the key functions Min-MQ and Pair-MQ and show the correctness and worst-case number of queries for Learn-MQ.

```
1 S = ∅, H = ∅
2 repeat
3 |   E = EQ(H)
4 |   if E = done then return H
5 |   if E is a positive counter example then
6 |   |   E' = Min-MQ(E, ∅)                    // See text
7 |   |   S = Pair-MQ(E', S)
8 |   H = variabilize(S)
```

Algorithm Learn-MQ: Learns $\mathcal{H}_D(P)$ using EQ and MQ queries.

```
  Data: An example E' and a set of positive examples S
  Result: S is updated by appending E' or by modifying S
1 for each s ∈ S do
2 |   for every pairing J between s and E' do
3 |   |   if MQ(J) then
4 |   |   |   return S = S ∪ {J} − s
5 return S ∪ {E}
```

Algorithm Pair-MQ: Updates the set of examples S with a new example E' using MQs.

Minimizing Examples. Algorithm Min-MQ (omitted due to space constraints) accepts an example and a list of objects already known to be relevant and considers each object o in E (and not in the known relevant list) and removes literals involving objects that are determined to be irrelevant. To test the relevance of object o, all literals in the current E involving o are dropped and

a MQ is issued on the resulting example. If the MQ indicates that the result is still a positive example, then o is considered irrelevant. If the example becomes negative, then we know that o must bind with some variable in a clause of the target theory T so we should not remove it. Importantly, after minimization an example is guaranteed to be positive and will have at most k objects in it, where k is the largest number of variables contained in a single clause of T.

Merging via Pairing Examples. Algorithm Pair-MQ illustrates how a minimized example E' is combined with the current set of ground clauses S. A key operation in the algorithm is *pairing* two examples E_1 and E_2 to produce a new example J as follows: 1) Select a 1-to-1 mapping M between the objects in E_1 and E_2, 2) Let E_1' be a version of E_1 with its objects mapped according to M and return $J = E_1' \cap E_2$.

Example 1. The pairing of

$$E_1 = P(1,2), P(2,3), R(1,1) \rightarrow Q(1,3)$$

and

$$E_2 = P(a,b), P(b,c), R(b,b) \rightarrow Q(a,c)$$

under mapping $\{1/a, 2/b, 3/c\}$ results in:

$$J = P(a,b), P(b,c) \rightarrow Q(a,c).$$

Although the constants from E_2 are used in J, they are used in our algorithms as placeholders for variables.

In general, there will be exponentially many pairings in the number of objects in the examples (which is bounded by k in the algorithm). Note that this operation is similar to the standard least-general generalization (LGG) operator [8] (the LGG can be viewed as the conjunction of all pairings), but unlike the LGG operation, the result is guaranteed to be no larger than the input examples.

The algorithm attempts to find an example $s \in S$ that is covered by a target clause C that also covers E'. It then generalizes s and E' to a new clause J that is a (smaller) positive example that is also covered by C. In this way, examples in S are continually being generalized and the number of examples in S is bounded until they become maximally general. This process is carried out by considering each possible pairing between E' and s and then asking an MQ about the resulting example J. If J is positive, then it replaces s in S with J. This is justified by the following:

Lemma 1. *For any examples E_1 and E_2, we have that E_1 and E_2 are covered by a common target clause C if and only if there exists a pairing J of E_1 and E_2 that is covered by C.*

The proof is omitted due to space constraints, but it is similar to the corresponding result in [6].

Note that, according to this result, the number of examples in S will only grow when a new positive example E is not covered by any of the target clauses

that cover current examples in S. Thus, the size of S will never grow larger than the number of target clauses.

Correctness and Complexity. We now show that Learn-MQ makes progress after each EQ resulting in a bound on the number of queries overall. We first give two simple lemmas.

Lemma 2. *If algorithm Pair-MQ replaces an example $s \in S$ with a new example J, then J contains strictly fewer literals than s.*

Proof. J can never grow larger than s by the definition of pairing. Suppose J and s were the same size. This means that no literals were dropped during the pairing of E and s for some 1-to-1 mapping. This implies that the variabilized version of s in H covers E', which implies that H covers E. However, this is a contradiction, since E' is guaranteed by the main loop of Learn-MQ to be a positive counter example to H. □

Lemma 3. *The size of S in algorithm Learn-MQ will never grow larger than the number of clauses in the target hypothesis.*

Proof. Suppose that $|S| > |T|$. As all examples in S are positive, there must be two examples s_1 and s_2 in S that are covered by the same clause C in T. By Lemma 1, there is a pairing of s_1 and s_2 that is also covered by C. However, we know that one of s_1 or s_2 was added to S in the presence of the other and that would only be done if there was no such pairing, showing a contradiction. □

We can now give the main result, showing that for constant a (max predicate arity) and k (max variables per clause) exact learning of $\mathcal{H}_D(P)$ is possible with a polynomial number of queries.

Theorem 1. *For any $T \in \mathcal{H}_D(P)$, Algorithm Learn-MQ learns an equivalent hypothesis after at most $|P|mk^a$ equivalence queries and $|P|mk^a(n+mk^k)$ membership queries.*

Proof. (sketch) The maximum size of an example (as measured by the number of literals) is $|P|n^a$. For every example, the algorithm removes an object and then asks a membership query. Therefore, the number of membership queries used on minimization for each example is bound by n.

After an example is minimized the maximum size of E' is $|P|k^a$, as there are at most k objects in a binding to a clause C in T. The algorithm now searches all mappings of objects to find a correct pairing. For two examples, both with k objects, the number of mappings is k^k. In the worst case, we will search all mappings for all elements in S (limited by m as shown by Lemma 3). Therefore the number of membership queries used for pairing an example is bound by mk^k.

Because every pairing removes at least one literal from an element in S, by Lemma 2 the number of examples returned by EQs is bound by $|P|mk^a$, as it is easily verified that EQs will only return positive counter-examples

Finally, the number of membership queries is bound by $|P|mk^a(n+mk^k)$. □

5 Object-based Queries

While EQs and MQs have been shown to be sufficient for efficient learning of $\mathcal{H}_D(P)$ with constant k and a, there is no reason to believe that such query types are ideal for use with human teachers. In particular, MQs are arguably the most primitive form of query and provide a relatively small amount of information compared to the teacher effort required to answer them (this involves analyzing a novel example which may or may not make sense from a semantic perspective).

Here we focus on reducing the number of membership queries required for learning by allowing for queries about the relevance of objects in an example. It is reasonable to expect that natural interfaces for such queries can be constructed in many teaching domains, where the teacher is able to mark the relevant objects in examples that they have already created or analyzed. For example, if the goal was to teach the legal moves of chess, the interface would allow for the selection of pieces and squares on a chess board.

5.1 Relevant object queries

In Min-MQ, membership queries are used to discover the objects necessary for a positive example to remain covered by the target. Since this information is likely obvious to a human teacher in many situations, it makes sense to consider a new type of query that directly asks for the same information.

Definition 3. *A* relevant object query (ROQ) *takes a positive example E as input and returns a minimal set of objects Q bound in a substitution θ such that for some clause $C \in T$, $C\theta \subseteq E$.*

Note that when an example is covered by multiple target clauses, there is not a unique answer to an ROQ. Given a set of objects Q and an example E, we let $E[Q]$ denote a new example that discards any literal in E that involves an object outside of Q. It is easy to verify that if a set of objects Q' is a strict subset of $\text{ROQ}(E)$, then $E[Q']$ will not be covered by the target theory. Thus, a single ROQ can play the same role as Min-MQ in the base algorithm. This yields the most basic modification to Learn-MQ called *Learn-MQ-ROQ* that uses ROQs for example minimization instead of Min-MQ. In particular, Learn-MQ-ROQ is identical to Learn-MQ except that a call to Min-MQ for example E is replaced with $E[\text{ROQ}(E)]$. This results in a decrease in the number of MQs with the addition of ROQs.

Proposition 1. *For any $T \in \mathcal{H}_D(P)$, Learn-MQ-ROQ learns an equivalent hypothesis after at most $|P|mk^a$ equivalence queries and ROQ queries, and reduces the number of membership queries over Learn-MQ by $|P|mk^a n$. Learn-MQ-ROQ does not depend on the maximum number of objects per example n.*

If, on average, the teacher cost of answering ROQs is less than a factor of n more than answering MQs, then there is an overall reduction in the teaching cost. This is a plausible situation, particularly in domains with many objects where a human teacher can often easily ignore the potentially large number of irrelevant objects.

5.2 Eliminating membership queries

With the aid of relevant object queries, one might wonder if it is possible to eliminate membership queries completely. The only other use of MQs in algorithm Learn-MQ is in Pair-MQ, where membership queries are used to discover if the pairing generated is still positive.

Instead of using a new query type to discover such pairings (such an approach is used in algorithm Learn-PQ), we instead apply a simple restriction on the equivalence oracle. Specifically, we require that the EQ always return a negative counter-example if one exists. Such an oracle is called *negatively-biased*.

Definition 4. *An oracle that answers equivalence queries is called* negatively-biased *if it always returns a negative counter-example for H when one exists.*

With this restriction we can guarantee that, upon a positive counter-example, our hypothesis H is guaranteed to be correct (albeit under-specified). Instead of searching for pairings using membership queries, we will search for the correct pairing by testing each one out in the hypothesis. If the pairing is incorrect, we will get a negative counter example that the new clause incorrectly covers. If we get a positive example, then the pairing must be correct.

Algorithm Learn-ROQ demonstrates how such an approach would work. A positive counter-example is minimized using Min-ROQ as we saw before. Then, the algorithm steps through every element in S, testing all possible pairings. A pairing is tested by directly adding it to the hypothesis and asking an equivalence query. If a negative counter-example is returned, the search continues. Otherwise, the algorithm replaces S with S' (the candidate for S) and starts over with the most recent positive example. If all pairings are eliminated via negative counter-examples then the example is appended to S as a new clause. A new positive example is received and the algorithm continues as before.

```
1  S = ∅, H = ∅, E = EQ(H)
2  repeat
3  |  if E = done then return H
4  |  E' = Min-ROQ(E, ∅)
5  |  for each s ∈ S do
6  |  |  for every pairing J between s and E' do
7  |  |  |  S' = S ∪ {J}
8  |  |  |  H = variabilize(S')
9  |  |  |  E = EQ(H)
10 |  |  |  if E is a positive counter example then
11 |  |  |  |  S = S'
12 |  |  |  |  Goto line 3
13 |  S = S ∪ {E'}
14 |  H = variabilize(S)
15 |  E = EQ(H)
```

Algorithm Learn-ROQ: Learns $\mathcal{H}_D(P)$ using EQ and ROQ queries.

Theorem 2. *The number of equivalence queries in algorithm Learn-ROQ is bound by* $|P|mk^a(1 + mk^k)$*. No membership queries are used. The number of relevant object queries is bound by* $|P|mk^a$*.*

Proof. (sketch) As before, the algorithm removes at most one literal or adds a new clause on every positive example. The number of positive examples is bound by $|P|mk^a$. Each example requires one ROQ, yielding the same bounds.

If a negative example is returned from the EQ oracle, then a pairing was incorrect and the algorithm tries to find a new one. There are mk^k total pairings for a single example, so each positive example requires at most mk^k negative examples to test pairings. The total number of examples is $|P|mk^a(1 + mk^k)$, which is equal to the number of EQs. □

While Learn-ROQ no longer uses membership queries, the number of equivalence queries has greatly increased. This is beneficial in situations where examples are plentiful but asking for new labelings is costly. A great deal of the examples ($|P|m^2k^{(a+k)}$ at most) provided by the equivalence query are negative examples, implying that this algorithm might be a good match for domains where negative examples are easy to provide.

Finally, Learn-ROQ provides motivation for an algorithm that successfully learns with mislabeled negatives. Often many real-world learning problems have a strong world model (i.e. creating an example by randomly inserting literals into a clause is very unlikely to produce a positive example). Unfortunately, the problem of a single mislabeled example is difficult to recover from with the framework in this paper, as there is no mechanism for noise. We believe a solution would be very useful in practice, and hope future work builds on this approach.

5.3 Imperfect relevance oracles

In all likelihood, a human answering ROQs would be unlikely to behave perfectly. This could be due to human error or in some cases due to interface issues. For example, in some domains there may be classes of objects for which the human is able to interact with as well as other objects that are internal to the domain encoding and hence are inaccessible. In such cases, imperfect heuristics might be used to determine relevance for the second class of objects.

Here we consider learning in the presence of specific types of imperfect ROQ oracle that are either verbose (includes objects that are not relevant) or conservative (does not include all the relevant objects). We will require the MQ and EQ oracles to operate perfectly. This restriction is not unreasonable as different query types may have different costs or underlying mechanisms associated with them.

Verbose Oracles. Recall that for an example E covered by multiple target clauses there are multiple possible ROQ answers, depending which covering clause C the oracle considers. When an oracle answers based on a clause C, we say that its answer is relative to C.

Definition 5. *An ROQ oracle is* (j, f)-verbose *if for at most j clauses in the target theory T, the oracle's answer Q' relative to any of those j clauses is a superset of the true set of relevant objects with at most f additional objects.*

Note that we can directly use a (j, f)-verbose oracle to answer ROQs in Learn-MQ-ROQ and still guarantee correct learning. However, the queries required increases with the amount of verbosity as quantified by j and f.

Theorem 3. *For any $T \in \mathcal{H}_D(P)$, Learn-MQ-ROQ, using a (j, f)-verbose ROQ oracle, learns an equivalent hypothesis. Compared to a perfect ROQ oracle, the number of EQs and ROQs increases by at most $|P|j(k+f)^a$ and the total number of MQs is now bound by $(|P|j(k+f)^a + |P|(m-j)k^a)(j(k+f)^{(k+f)} + (m-j)k^k)$.*

Proof. (sketch) If we have an extra f objects returned from a ROQ for j clauses in T, then the size of an example after minimization is bounded by $|P|(k+f)^a$. On the other $m - j$ clauses, the size remains the same as before ($|P|k^a$). Therefore, if each example removes one literal, the total number of examples (and hence EQs) is bound by $|P|j(k + f)^a + |P|(m - j)k^a$. This is the same bounds for ROQs.

As before, the worst case for membership queries is when all possible pairings must be searched. The total number of pairings for an example is bound by $(m - j)k^k$ for clauses in S that the oracle does not make a mistake on, and $j(k + f)^{(k+f)}$ for the clauses that it does. Adding this up and multiplying by the number of examples gives the bounds $(|P|j(k + f)^a + |P|(m - j)k^a)(j(k + f)^{(k+f)} + (m - j)k^k)$. □

Unfortunately, it appears that for domains where n (max objects) is not exceedingly large, directly using Learn-MQ-ROQ with a (j, f)-verbose oracle can result in many more membership queries compared to the base learning algorithm Learn-MQ. It is an open question about whether it is possible to improve the dependence on j and f.

Conservative Oracles. Next, we restrict the oracle in the opposite direction where relevant objects can be missing.

Definition 6. *An ROQ oracle is* (j, f)-conservative *if, for at most j clauses in the target theory T, the oracle's answer Q' relative to any of those j clauses is a subset of the true set of relevant objects with no more than f missing relevant objects.*

Because relevant objects are missing, we cannot directly use Learn-MQ-ROQ or Learn-ROQ as our examples may be overly general. This is because conservative answers lead to overly general minimized examples, which break the assumptions of the merging processes. Instead we give a new algorithm Learn-MQ-ROQ-Conservative that is similar to the base algorithm. An important change is that the pairing algorithm Pair-MQ is slightly modified so that it only considers pairing together examples that contain the same number of objects. This new modified algorithm is referred to as *Pair-MQ-Con*.

Upon a positive example, Learn-MQ-ROQ-Conservative minimizes it using the ROQ oracle and then searches for pairings. If no pairing was found between

E' (the minimized example) and the set of positive examples S, the algorithm checks to see if E' is truly a positive example using an MQ. If it is not positive then a relevant object was missed (the oracle made a mistake). The mistake is resolved by re-minimizing the example using Min-MQ. Since all objects given by the conservative ROQ are relevant, they are passed to Min-MQ so the routine does not spend time testing them. After minimization the correct example is now passed to Pair-MQ-Con for pairing. Using Pair-MQ-Con in place of Pair-MQ implies that the algorithm will never try to pair E' to the same example s more than once. If a relevant object is missing and E' does not pair with anything in S, then the number of objects in E' will necessarily increase (as the membership query will detect it and employ Min-MQ. Because the size has increased, E' will not be checked against previously checked elements in S and avoid a significant amount of redundant work.

```
1  S = ∅, H = ∅
2  repeat
3  |  E = EQ(H)
4  |  if E = done then return H
5  |  if E is a positive counter example then
6  |  |  E' = Min-ROQ(E)
7  |  |  S = Pair-MQ-Con(E', S)
8  |  |  if E' ∈ S (E' did not pair with any elements) and MQ(E') is false then
9  |  |  |  S = S − {E'}
10 |  |  |  E' = Min-MQ(E, objects(E'))
11 |  |  |  S = Pair-MQ-Con(E', S)
12 |  H = variabilize(S).
```

Algorithm Learn-MQ-ROQ-Conservative: Learns $\mathcal{H}_D(P)$ using EQ, MQ, and conservative ROQ queries.

Theorem 4. *For any $T \in \mathcal{H}_D(P)$, Learn-MQ-ROQ-Conservative, using a (j, f)-conservative ROQ oracle, learns an equivalent hypothesis. Compared to Learn-MQ-ROQ with a perfect oracle the maximum number of EQs and ROQs are unchanged and the total number of MQs is bound by $|P|jk^a(mk^k + 1 + n - (k - f)) + |P|m(m - j)k^{(a+k)}$.*

Proof. (sketch) The number of examples remains the same as Learn-MQ-ROQ ($|P|mk^a$), giving the bounds on equivalence queries and relevant object queries. If the relevant object oracle makes a mistake ($j > 0$), then the algorithm searches for pairings. The worst case occurs when all clauses in the target hypothesis have the same number of variables, so this search for pairings adds no membership queries as E' has $k - f$ variables.

Before generating H, the algorithm asks a membership query to verify the example is actually positive (which it is not). The search for the true number of relevant objects in an example takes $n - (k - f)$ membership queries, as that is

how many objects there are that might possibly be relevant. Finally, the search for pairings is restarted, giving mk^k total pairings that can be found.

When an example does not have a mistake, the number of membership queries remains the same as the original algorithm (mk^k for finding the correct pairing). Multiplying the number of MQs for each example by the size of each type of example gives $|P|jk^a(mk^k+1+n-(k-f))+|P|m(m-j)k^{(a+k)}$, which is what is provided above. □

Algorithm Learn-MQ-ROQ-Conservative has a much more reasonable increase of membership queries with respect to j and f compared to the verbose setting. Instead of an exponential increase of MQs with f, the change is now linear. It remains an open problem as to whether it is possible to remove membership queries altogether when using a conservative ROQ oracle.

Verbose vs. Conservative. The above results show that clearly, when MQs are available, a conservative oracle is superior to a verbose oracle with respect to the number of MQs. This is because it is much easier to identify an example missing relevant objects than an example with extra objects. This has implications for domains where heuristics are used to judge relevance of objects that are not available via an interface for a human teacher to judge relevance. In particular, it suggests that when MQs are available using a conservative heuristic is desirable.

5.4 Pairing queries

The majority of the MQs in Learn-MQ are used in the search over pairings. However, it is plausible that often a teacher will be able to directly indicate which objects between two existing examples "correspond" with respect to the target concept. For this purpose we introduce pairing queries.

Definition 7. *A* pairing query (PQ) *is a query that, given two positive examples E_1 and E_2, returns* **false** *if there is no clause $C \in T$ that covers both E_1 and E_2. Otherwise it picks a target clause C that covers both examples via substitutions θ_1 and θ_2 and returns a 1-to-1 mapping between objects in θ_1 and objects in θ_2 where objects are mapped together if they correspond to the same variable in θ_1 and θ_2.*

Note that by definition the mapping returned by a PQ will only involve relevant objects. In this sense, PQs are strictly more powerful than ROQs. Pairing queries can be directly placed in Learn-MQ-ROQ by replacing the loop that searches over pairings. The algorithm directly asks if two examples pair together and asks for a mapping P. If the query is false, the algorithm proceeds through the set S, appending the example if no pairing is found. Otherwise, a pairing is calculated using the mapping P and the algorithm proceeds normally. This is demonstrated in algorithm Learn-PQ. From previous results we can easily bound the required number of EQs and PQs.

Proposition 2. *For any $T \in \mathcal{H}_D(P)$, Learn-PQ, learns an equivalent hypothesis using no more than $|P|mk^a$ EQs and $|P|m^2k^a$ PQs. No MQs or ROQs are required.*

It is unclear how much more difficult it would be for PQs to be answered compared to ROQs; ultimately experience with real users is needed. It is not difficult to imagine a natural interface for PQs in many domains. A system could be implemented that shows the two examples to a user (e.g. two chess boards) and asks for a matching between objects that have the same role in each example. This is similar to asking "What do these two examples have in common?" which is a question that seems natural in a learning environment. The problem of learning with noisy pairing queries is currently open and an important future direction.

```
1  S = ∅, H = ∅
2  repeat
3  |  E = EQ(H)
4  |  if E = done then return H
5  |  if E is a positive counter example then
6  |  |  for each s ∈ S do
7  |  |  |  P = PQ(E, S_i)
8  |  |  |  if P is not false then
9  |  |  |  |  Let J be the pairing generated using mapping P with E and S_i,
   |  |  |  |  where objects not found in P are removed by dropping literals
   |  |  |  |  that reference them.
10 |  |  |  |  S = S ∪ {J} − s
11 |  |  if no pairings were found then
12 |  |  |  S = S ∪ {E}
13 |  H = variabilize(S)
```

Algorithm Learn-PQ: Learns $\mathcal{H}_D(P)$ using EQ and PQ queries.

6 Summary and Future Work

We have shown algorithms that allow for object-based queries to learn in a polynomially-bounded number of queries. We specifically focused on relevant object queries and pairing queries, but these are by no means the only object-based queries that may be useful. It is our hope that the results shown above motivate a further examination into new query types.

While it was claimed that object-based queries might be easier for a human to use than traditional membership queries, no user study has been performed to evaluate this claim. This motivates a user study where users act as various oracle types to try to teach a concept. Users could also be asked what sort of information they would like to be able to express, possibly giving motivation for a new query type or annotation.

Finally, we examined two types of error-bound object-based queries. Both types were deterministic, but such a restriction may not be practical. A probabilistic model of oracle responses may be more likely in practice. Other error

models can also be considered, such as a mixing of the two error types, or providing a large set of negative examples that are correct with some probability.

Acknowledgements The authors acknowledge support of the Defense Advanced Research Projects Agency (DARPA) under grant HR0011-07-C-0060. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DARPA.

References

1. Angluin, D.: Queries and concept learning. Machine Learning 2, 319–342 (1988), http://dx.doi.org/10.1023/A:1022821128753, 10.1023/A:1022821128753
2. Angluin, D., Frazier, M., Pitt, L.: Learning conjunctions of horn clauses. Machine Learning 9, 147–164 (1992), http://dx.doi.org/10.1007/BF00992675
3. Arias, M., Khardon, R., Maloberti, J.: Learning horn expressions with LOGAN-H. The Journal of Machine Learning Research 8, 549–587 (2007)
4. Baum, E.B., Lang, K.: Query learning can work poorly when a human oracle is used. In: IJCNN (1992)
5. Feldman, V., Shah, S.: Separating models of learning with faulty teachers. Theor. Comput. Sci. 410, 1903–1912 (April 2009), http://portal.acm.org/citation.cfm?id=1519541.1519713
6. Khardon, R.: Learning function-free horn expressions. Machine Learning 37, 241–275 (1999), http://dx.doi.org/10.1023/A:1007610422992
7. McDaniel, R.G., Myers, B.A.: Getting more out of programming-by-demonstration. In: Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit. pp. 442–449. CHI '99, ACM, New York, NY, USA (1999), http://doi.acm.org/10.1145/302979.303127
8. Plotkin, G.D.: A Note on Inductive Generalization. Machine Intelligence 5, 153–163 (1970)
9. Raedt, L.D., Dzeroski, S.: First-order jk-clausal theories are pac-learnable. Artificial Intelligence 70(1-2), 375 – 392 (1994), http://www.sciencedirect.com/science/article/B6TYF-47WT97K-F/2/55a8bf6608387d13b5bfa7a55e4b50d2
10. Reddy, C., Tadepalli, P.: Learning first-order acyclic horn programs from entailment. In: Page, D. (ed.) Inductive Logic Programming, Lecture Notes in Computer Science, vol. 1446, pp. 23–37. Springer Berlin / Heidelberg (1998), http://dx.doi.org/10.1007/BFb0027308
11. Reddy, C., Tadepalli, P.: Learning horn definitions: Theory and an application to planning. New Generation Computing 17, 77–98 (1999), http://dx.doi.org/10.1007/BF03037583, 10.1007/BF03037583
12. Valiant, L.G.: A theory of the learnable. Commun. ACM 27, 1134–1142 (November 1984), http://doi.acm.org/10.1145/1968.1972
13. Walker, T., Natarajan, S., Kunapuli, G., Shavlik, J., Page, D.: Automation of ilp setup and search via user provided relevance and type information. In: Inductive Logic Programming (2010)