Actively Soliciting Feedback for Query Answers in Keyword Search-Based Data Integration

Zhepeng Yan Nan Zheng Zachary G. Ives
University of Pennsylvania
Philadelphia, PA USA
{zhepeng,nanzheng,zives}@cis.upenn.edu

Partha Pratim Talukdar
Carnegie Mellon University
Pittsburgh, PA USA
ppt@cs.cmu.edu

Cong Yu
Google, Inc.
New York, NY USA
congyu@google.com

ABSTRACT

The problem of scaling up data integration, such that new sources can be quickly utilized as they are discovered, remains elusive: global schemas for integrated data are difficult to develop and expand, and schema and record matching techniques are limited by the fact that data and metadata are often under-specified and must be disambiguated by data experts. One promising approach is to avoid using a global schema, and instead to develop *keyword search*-based data integration — where the system *lazily* discovers *associations* enabling it to join together matches to keywords, and return ranked results. The user is expected to understand the data domain and provide feedback about answers' quality. The system *generalizes* such feedback to learn how to correctly integrate data.

A major open challenge is that under this model, the user only sees and offers feedback on a few "top-k" results: this result set must be carefully selected to include answers of *high relevance* and answers that are *highly informative* when feedback is given on them. Existing systems merely focus on predicting relevance, by composing the scores of various schema and record matching algorithms. In this paper we show how to predict the *uncertainty* associated with a query result's score, as well as how *informative* feedback is on a given result. We build upon these foundations to develop an *active learning* approach to keyword search-based data integration, and we validate the effectiveness of our solution over real data from several very different domains.

1. INTRODUCTION

The vision of rapid information integration remains elusive, despite steady progress in system architectures [13] and in alignment techniques for discovering links among records [11] and schema elements [28]. In general, the approach is to define one or more integrated or *mediated* schemas capturing the data domain, use schema mapping (alignment) and entity resolution (record linking) tools to map data sources into the mediated schema, and finally allow users to pose structured queries against mediated schemas.

A stumbling block is that string, pattern, and structural similarities among data and metadata elements (the core techniques whose outputs are combined by alignment tools) do not suffice to uniquely identify the correspondences between data or metadata items. The resulting ambiguous matches can only be resolved with domain (commonly, human) expertise. As a result, many of today's tools aim for *semi-automated* matching, where the system makes predictions and relies on a human domain expert to correct any mistakes or resolve any uncertainties (e.g., see [28, p. 345]).

There are several shortcomings to having a database administrator inspect the output of a schema matching tool before adding the mapping to an existing system: (1) administrator vetting becomes a bottleneck to the system incorporating sources; (2) the metadata might not clearly describe the data that must be mapped[1]; (3) subtle variations in semantics may only show up in occasional, incorrect query results. Moreover, the mediated schema itself can be a bottleneck to adding new data, as new sources may have concepts that do not yet exist at the global level.

For these reasons, recent work [4, 29, 34, 35] has proposed to complement (or even replace) conventional integration with techniques that do not rely on a mediated schema and schema mappings created by an administrator. Instead the proposal is to adopt a *keyword search over databases* model [5, 19, 20, 25] where matches to individual keywords are assembled into query results by discovering "join trees" that link the matches. This requires discovering paths of *associations* (alignments across records, terms, or schema elements in sources) that can join matching records together. Under this model, the output of alignment algorithms is used directly to answer queries, with no administrator intervention: the system relies on the *end user* to have the domain expertise to vet the results, and to provide *feedback* [34, 35] on the system's ranking of (some) individual query results. Now instead of having a human administrator correct bad associations, the system must *learn the correct score* (possibly zero or infinite) of each individual association, given the user's feedback on query answers that are formed from multiple associations [34].

This model is a form of "pay-as-you-go" integration [13], as it enables the system and its users to focus their attention on those associations that relate to actual information needs. The associations relevant to frequently posed queries should be the ones that receive the most attention and refinement. In fact the pay-as-you-go approach can be used to complement and *inform* more traditional integration techniques: the keyword search log can help a human administrator determine which parts of the data to prioritize integrating, and provide clues for what mappings are most relevant.

However, to successfully learn to integrate data, the system must balance its need to acquire feedback useful for answering future queries, versus the requirement that each user immediately gets the information he or she needs. Today's keyword search systems

Proceedings of the VLDB Endowment, Vol. 6, No. 3
Copyright 2013 VLDB Endowment 2150-8097/13/01... $ 10.00..

[1] Consider, e.g., the situation where users put data into comments fields because there was no appropriate column in the schema.

have approached this problem by simply assuming the query scoring function is accurate: they return the top-k results according to the scoring function, which in turn bases its scores on the predicted (but possibly incorrect) output of matching tools. Under this model the user will attempt to remove false positives but has no way of seeing — and providing feedback on — false negatives.

Such a model works well when the system returns a good mix of correct and invalid results and the user can "separate" them. However, as the number and complexity of sources and their attributes increases, many potential queries are likely to have similar scores, due to inherent uncertainty in combining low-confidence results from various matching algorithms. The number of potential results can grow rapidly as the number of keyword matches increases, whereas the number of results seen by the user remains constrained by the dimensions of the screen and the limits of user attention. Thus, when a keyword-based data integration system selects queries to produce answers, it should not merely choose alignments based on the relative scores of associations — but also the *uncertainty* associated with a given query result, and the *informativeness* of feedback given on that particular result.

In this paper, we use *active learning* to help the system determine which query results to present, given a combination of their predicted score, their inherent uncertainty, and the amount of information gained about other potential queries. Intuitively, the informativeness of feedback on a query result is related to how much uncertainty there is about the result's relevance to the query, and how many other *similar* share features with this result — meaning that feedback on the first result also reduces their uncertainty. We provide a more precise characterization of informativeness later in the paper. Our work goes beyond previous attempts to use uncertainty-directed ranking in the pay-as-you-go-integration space, such as [24] which focused on individual mappings, by looking at the total uncertainty associated with *queries* and their results, and how this uncertainty should be combined with relevance ranking.

The key questions addressed in this paper are how to estimate the utility of a given query to the system and to the user, and how to estimate the uncertainty of a query's score, in applying active learning to the problem of determining the relevance of associations to a query. Specifically, we make the following contributions:

- Techniques for estimating the uncertainty associated with a query, through the notions of entropy and variance, and by combining the probability distributions of the output for individual schema matching or record linking outputs.
- Pruning and active learning techniques that focus the user's attention on the query results most likely to either be relevant, or help the system produce better results.
- A scoring model using *expected model change* to relate the user's model of browsing data to how we should combine and rank both useful and uncertain query answers.
- A method of *clustering* similar join queries, and choosing the most useful representative.
- An experimental evaluation demonstrating the effectiveness of our approach across several real data domains.

Section 2 provides the context of our problem, including the basic workflow of our integration task. Section 3 shows how we assess the *informativeness* of each query. Section 4 then describes how we combine informativeness and predicted score to return ranked query results, and to learn from feedback on them. We experimentally analyze our results in Section 5, describe related work in Section 6, and conclude in Section 7.

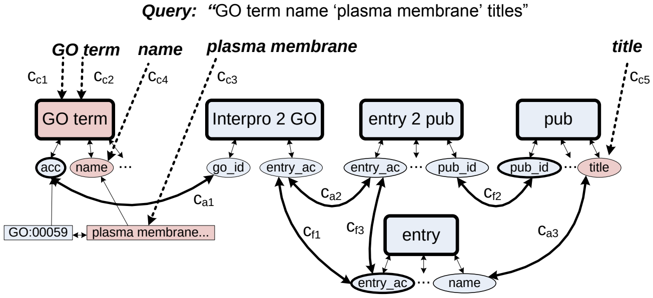

Figure 1: Search graph with keyword search terms. Each keyword may match a node with a *similarity score* and each pair of attributes may also match with a similarity score; this is captured by an edge *cost* c_{ci} that can further be broken into weights plus features.

2. PROBLEM SETTING AND DEFINITION

Our goal in this paper is to effectively obtain and *learn from* user feedback, in order to aid the system in finding the data integration query results of greatest relevance to each user's (ongoing or future) information need. We assume that users express their information need using a *keyword search* query model [5, 18, 19, 20, 25] in which keywords get matched against elements in one or more relations in different data sources. Such elements will be represented as nodes in a graph connected by weighted *association* edges — typically foreign keys, containment relationships between nested objects, or schema alignments and record links that were produced by combining outputs from one or more automated matching algorithms. See Figure 1 for an example, where rounded rectangles represent relations, ellipses represent attributes, and squared rectangles represent fields in tuples; dashed edges represent matches between keywords and schema or data nodes, unlabeled edges represent connections between attributes and relations (has-a) and between fields and attributes (is-a); bidirectional edges with labels represent potential joins with a confidence score computed by one or more schema alignment tools. In answering a query, the system will attempt to find trees within the graph linking the relations or tuples matching the given keywords, as indicated by bold lines in the figure. In general, there may be multiple matches to a search keyword, and there may be multiple edges between nodes, representing different candidates for joining the associated relations. We target users whose goal is to construct a set of correct query answers, and who are thus incentivized to provide feedback indicating which tuples should be removed from or re-ordered within the result set.

2.1 Preliminaries

We implement our techniques within the **Q** Query System, which combines keyword search for data integration with feedback-based learning from query answers. Our basic techniques could be adapted to other data integration systems that currently do not perform learning, such as Kite [29] and the system of Bergamaschi et al. [4], as well as keyword search systems for databases, like Discover [20], DBXplorer [1], BANKS [5], and BLINKS [19]. We show the basic architecture of the **Q** System in Figure 2.

Graph and Edge Weights. **Q** initially creates a *search graph*, by cataloging and indexing known sources, and encoding metadata and data items as nodes. Nodes may optionally receive an authority score, e.g., from link analysis [3] or user feedback [35].

Certain *association* edges are then added to capture relationships such as containment, subclassing, membership, and foreign key references. Other edges, describing potential cross-source joins, are *lazily* added by the **Q** system: as a user keyword query is entered, a *query formulator* seeks to find trees within the schema graph that connect nodes matching the search terms; as necessary, the query formulator will call an *association generator* to induce new association edges in the graph, by running schema matching and record linking tools. The **Q** system can directly use the outputs of existing matchers and alignment tools from the literature, such as [9, 10, 28]. To facilitate this, each association edge's *cost* or score is comprised of a set of *feature values*, plus *weights* for each of the features obtained from the schema matching and record linking tools and adjusted from feedback.

Most "modern" schema matching tools comibine the results of different *base matchers*, which focus on different characteristics of the data and/or metadata [28]. In this paper we assume that we have direct access to the base matcher scores from the alignment tools, as is the case with the COMA++ system we used in our experiments [9]. (We can also directly use the final output from any matching tool, but we must have at least partial information about the component scores it uses, so we can estimate its uncertainty.)

2.1.1 Search and Ranking

When the **Q** system is given a keyword query of the form $Q = \{K_1, \ldots, K_m\}$, it first uses a keyword similarity metric[2] to match each keyword $K_i \in Q$ against all search graph nodes (schema and data elements). It "overlays" onto the search graph a node representing each K_i (see keyword nodes, represented as boldfaced italicized words, in Figure 1). It then adds an edge from K_i to each graph node whose label matches the keyword with a similarity score exceeding some threshold. Each such edge is assigned a cost expression (e.g., c_2 in the figure) that represents a dissimilarity or semantic distance, and is lower for closer matches. In turn, an edge weight such as c_2 is actually comprised of two components, a *similarity score* s_2 from the similarity metric, and an adjustable *weight* w_2 that is learned by the system.

With respect to this overlaid search graph, each tree with leaf nodes $K_1 \ldots K_m$ represents a possible join query (each relation node in the tree, or connected to a node in the tree by a zero-cost edge, represents a query atom, and each non-zero-cost edge represents a join or selection condition). Like most keyword search-over-database systems, **Q** generates queries that may produce relevant answers by running an approximate *Steiner tree algorithm* [35] to connect matching nodes in the search graph with the lowest-cost tree, and executes them and unions their results together in ranked order using a *top-k query processing algorithm* [12, 16, 22]. While the **Q** system combines cost components (features) derived from data as well as metadata, in this paper we focus on features that are associated with the metadata and the query — particularly those having to do with predicted schema matches — rather than those derived from specific fields in the data. The tree has an associated *cost function* that weights and sums the costs on the edges (and optionally nodes) in the tree. In turn, the set of underlying costs can be represented as a binary *feature vector* representing the set of participating alignments, and a *weight vector* representing the weight values for those features: the cost represents the dot product of those two vectors.

Probabilistic Interpretation of Cost. Typically we initialize the feature and weight vectors to *negative log likelihoods* of the probabilities that items match — meaning that the overall query cost also represents a negative log likelihood and can be converted back to a probability by computing 2^{-cost}. We refer to this latter term as the *score* of a result.

2.1.2 User Feedback

In general the **Q** system's task is not finished once it has returned a set of query answers. Rather, the user may pose feedback over these results, in the form of constraints on the preferred ordering of the answers, or identification of good or bad results. In general we assume that the user looks over a portion of the k answers returned, and provides (1) a *watermark* indicating the set of results verified and (2) feedback about which results are known to be incorrect. In contrast to information retrieval where the user generally only wants *one* valid answer and does not reuse query results, we expect here that the user wishes to keep the *set* of correct answers to a query, and that he or she may make the results *persistent* in the form of a view. Hence the user is incentivized to provide feedback. We assume in this paper that the task of determining correct vs. incorrect results is context-insensitive, meaning all users' feedback can be combined — as opposed to the more context-specific problem of learning source authoritativeness [35].

2.2 Soliciting Feedback: Active Learning

The **Q** system encourages the user to "curate" the results of the query, distinguishing good answers from bad ones and establishing a preferred ranking order for the results. This leads to a tension between two desiderata: we must provide *some* relevant answers so the user is motivated to look through the results; but we want the system to continuously expand its ability to score new sources and new edges, i.e., increase its recall, meaning that we must also solicit feedback on results that include uncertain edges. These contrasting goals motivate the focus of this paper: an *active learning* [30] approach incorporated into a component called the *suggester* module.

The **Q** system's suggester module ranks queries based on its uncertainty about their score, and how much feedback about their validity aids the system in predicting the score for other queries that have features in common with them. Its top results will typically be merged with the top-scoring query results, giving a mix of items for the user's inspection and feedback. Effective output from the suggester will help the system accelerate learning convergence while reducing the need for user intervention.

To achieve this, we incorporate the idea of active learning from the machine learning literature. Active learning improves the accuracy of learning while reducing the amount of training data: it relies on the ability of the learning algorithm to choose the data from which it learns. This is especially desirable for applications where labeled training data (in our setting, correct scores or costs for association edges, leading to correct query result rankings) is difficult to obtain. Typically, an active learning algorithm has access to an oracle and issues queries on unlabeled data. The oracle answers the learning algorithm by assigning a label associated with the query instance. In our setting, the user serves as the oracle.

The key question in applying active learning is how to select the next unlabeled instance for the oracle's annotation. A common approach is to adopt *uncertainty sampling*, a query strategy based on an uncertainty measure. This measure determines how uncertain the label given to the instance will be, and indicates how much extra information the underlying model may learn. In our setting, a computed query result is a sample, and its label indicates whether it satisfies the user's information need with the correct ranking. We develop a novel means of measuring a query result's uncertainty, given the uncertainty associated with its individual components like join associations.

We also explore another aspect: estimating how much feedback on a single tuple can help label a group of *similar* queries and their results. We develop a clustering strategy where queries sharing common edge and node structure are grouped together into

[2] By default tf-idf over the tuples in the data, although other metrics such as edit distance or n-grams could be used.

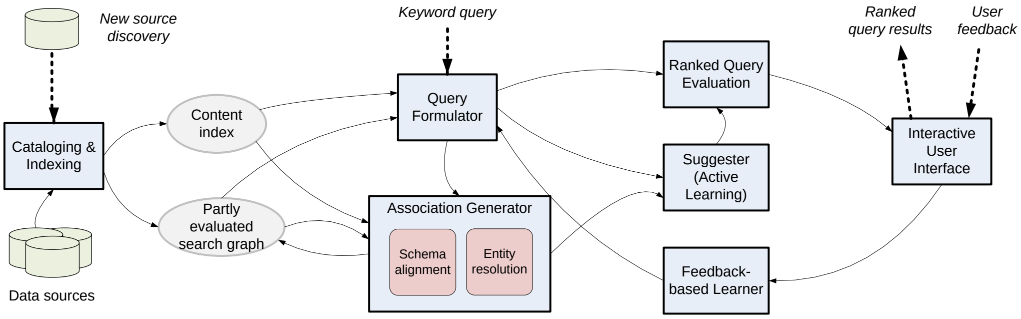

Figure 2: In the Q system, a search graph is generated by indexing known data and metadata as nodes; *association* edges are lazily added. A keyword query is fed to a *query formulator*, which finds matching trees within the search graph. As necessary, it calls an *association generator* to induce new graph edges. This paper focuses on the *suggester* module, which uses active learning to ensure a mix of top-scoring answers and answers for which feedback is most informative. User feedback goes to a learner, which improves the scores of the associations in the graph.

one cluster, and a representative is presented to the user. Such clustering-ranking schemes have been previously used in other active learning applications, such as guided data repair [36] and record linking [2]. A major novelty of our work lies in determining and ranking the uncertainty of clusters of queries. We develop a measure based on the uncertainty score from the alignment(s).

3. FINDING INFORMATIVE QUERIES

In this section, we develop mechanisms for measuring the *uncertainty* of a query result, given knowledge of the uncertainty associated with the edge (schema match) and node (relation authoritativeness) components of the search graph. We then take into account the fact that queries may have overlapping edges or relations, meaning that feedback may benefit multiple queries. We seek to focus on returning "more informative" query results.

Our approach is to build over existing schema matchers and their underlying components. However, a challenge is that modern matchers [9, 10, 28] combine the results of many *base matchers*, but return a single similarity score that does not reveal any information about how this was obtained. In order to determine the level of uncertainty associated with each potential alignment, we seek to estimate the *probability distribution* over the range of values, as suggested by [15, 26]. To form this estimate, we compute a predicted distribution over all primitive matchers' scores for each association edge. We learn a weight for each of the base matchers, and use each weighted value as a point within a probability distribution for the possible values for the composite matcher. (The **Q** system learns how to best combine the weights from these base matchers, as described in Section 4.3.2.) Modeling feature weights as random variables enables us to estimate overall relevance of an edge, as well as the amount of uncertainty associated with it. This generalizes our previous model in [35], since the previous feature weight corresponds to a sure event with only one possible value and probability 1.

We consider in Section 3.1 how to compute weight distributions for the search graph using *features* of two types: (1) those suggesting attribute alignments (possible join *edges*) across relations on an attribute; (2) those representing authoritativeness or quality of relations or *nodes*. Section 3.2 then shows how such features' uncertainties can be *combined across multiple relations and joins* to get the uncertainty associated with a query (and its results). Finally, many queries may be similar: Section 3.3 shows how we can *cluster* such queries to maximize our ability to include a diverse set of results within the top-k answers returned to a user.

3.1 Uncertainty in the Search Graph

We first explain how the estimated probability distributions for base matchers are incorporated into our search graph. We divide our discussion into the basic features associated with edges and nodes in the graph, and weights assigned to those features.

3.1.1 *Graph Components and Features*

We formalize the definition of search graph and features in our **Q** system. Assume that we have $\mathbf{R} = \{R_i\}$ as the set of all relations in the schema. Let $G = (V, E)$ be the schema graph, where V is the set of attribute nodes and E is the set of alignment edge. Each alignment edge connects a pair of attributes and represents a possible join on that pair. The global feature set $F = F_a \cup F_r$ contains alignment features $F_a = \{f_{AB} | A, B \in V\}$ and relation features $F_r = \{f_R | R \in \mathbf{R}\}$, where features in F_a describe attribute alignments and features in F_r identify relation qualities. Every edge is mapped to a vector of binary feature values, where each bit specifies if that feature relates to the edge. We denote by f_i^e the value that a feature f_i takes on edge e. In fact, given an edge $(A, B) \in E$, where A is an attribute of relation R_1 and B is an attribute of relation R_2, only the features $f_{R_1}^e$, $f_{R_2}^e$, and f_{AB}^e will have value 1, while all the remaining features take value 0.

Each feature has an associated weight, from which the cost of an edge is derived. The weight of an alignment feature captures the quality of that matching, and the weight of a relation feature describes table authoritativeness. Feature weights are derived from schema matching tools and further adjusted through user feedback. We treat the weights as random variables, to model the uncertainty of predictions of the schema and record alignment tools. For each alignment feature f_{AB}, we denote by $W_{f_{AB}}$ the random variable which maps a possible weight of the alignment feature to a probability value. Hence, $W_{f_{AB}} = w$ is the event that the feature weight takes value w. Similarly, there is a weight random variable $W_{f_{R_i}}$ for each relation feature $f_{R_i} \in F_r$. Thus, the cost of an edge $e = (A, B) \in E$, where A is an attribute of relation R_1 and B is an attribute of relation R_2, is given by the random variable

$$C(e) = \sum_i W(i) f_i^e = W_{f_{R_1}} + W_{f_{R_2}} + W_{f_{AB}}. \qquad (1)$$

We now describe how features are computed for edges and nodes. (Our system also supports features whose values come from the data, e.g., score attributes within tuples [23, 34], and our model generalizes to this. For simplicity we focus on query rather than instance-based attributes.)

3.1.2 *Edge (Schema Alignment) Features*

The **Q** system encodes schema matches (attribute alignments) as *alignment features*, whose weights represent alignment qualities.

For any two attributes A and B, the random variable $W_{f_{AB}}$ represents the score distribution we derive. We first focus on a single pair of attributes, where we treat the set of base matchers' outputs as an *ensemble* of classifiers, and let them *vote* on a prediction.

Schema Matcher Ensemble. The initialization procedure utilizes m matching primitives, i.e., base schema matchers. For any attribute pair (A, B), the i^{th} matching primitive algorithm produces a normalized discrete cost score $0 \leq s_i(A, B) \leq 1$. In our framework, as described before, a low score indicates high similarity.

Voting Heuristic. Since precision varies from different matching algorithms, we can assign each member a normalized preference $p(M_i)$, such that $\sum_i p(M_i) = 1$. This preference can be interpreted as the confidence level of a primitive. It also represents how heavily a primitive contributes to the final aggregated matching score. The values will be trained ahead of time, and will be learned (see Section 4.3.2). For any possible weight value $w \in [0, 1]$, we have the following estimation formula based on voting from the matching ensemble,

$$\Pr(W_{f_{AB}} = w) = \sum_{1 \leq i \leq m} \mathbf{1}(s_i(A, B) = w) p(M_i). \tag{2}$$

The above formula states that the probability of the alignment having a score w is the summation over all weights of matching primitives yielding the same score.

Relevance and Uncertainty. We can reason about both relevance and uncertainty of a particular alignment based on distributions over weight values for alignment features. For example, we can use the expectation $\mathbb{E}(W_{f_{AB}})$ to measure relevance, and then use the entropy of $W_{f_{AB}}$, $\mathbb{H}(W_{f_{AB}})$, or its variance, $\mathbb{V}(W_{f_{AB}})$, to measure uncertainty. We discuss this in more detail in Section 3.2.

3.1.3 Node (Relation Authoritativeness) Features

In some cases the user may have a certain preference for (or bias against) a particular relation R, e.g., due to its authoritativeness. We model this as a feature shared across all edges linking to the node R and its attribute nodes, and we derive an initial weight distribution for this feature.

We exploit the user query log to form this initial weight distribution, and then rely on feedback to adjust the weight. In the **Q** system, users can have each result query saved as a view, and have previous keywords stored as nodes in the schema graph. We assume that these keywords roughly represent the distribution over the user's long-term set of queries. Let $R(A_1, A_2, \ldots A_p)$ be a relation and $V_Q = \{k_1, k_2, \ldots k_q\} \subseteq V$ be the (nonempty) set of keyword nodes from the user's query history. We use matching primitives for each (A_i, k_j) pair to get a score distribution, encoded by the random variable $S(i, j)$.

We can apply Formula 2 to compute $S(i, j)$ as the weight distribution of an alignment feature. Hence, the weight random variable corresponding to the relation feature f_R is given by

$$W_{f_R} = \frac{\sum_{1 \leq i \leq p, 1 \leq j \leq q} S(i, j)}{pq}. \tag{3}$$

Similarly to the case of edges, we can examine both relevance and the amount of uncertainty associated with relation R, through its corresponding feature.

3.2 Composing Uncertainty for Queries

In the **Q** system, keyword queries on the schema graph produce a set of structured queries, each generated from a Steiner tree T which is a subgraph of G. We combine uncertainty from each edge and define the uncertainty of a query by examining its corresponding Steiner tree as follows.

The cost of an edge $e \in E$ with features $f_1, f_2, \ldots$ and associated weights $W_1, W_2, \ldots$, where W_i is a random variable, can be calculated using Formula 1. Abusing the notation a little, we have the cost of a tree T derived from costs of all edges presented in T, as follows

$$C(T) = \sum_{e \in E(T)} C(e) = \sum_i \sum_{e \in E(T)} f_i^e W_{f_i}, \tag{4}$$

where $E(T)$ denotes the set of edges of tree T. We treat each W_i as being independent of the others. While in reality this may not be true, we will show experimentally that this heuristic is effective, and that it simplifies the learning procedure.

Consider a structured query plan modeled by a Steiner Tree T, we can infer relevance and uncertainty of the query from its cost expression. The **Q** system measures query relevance by the expectation $\mathbb{E}(C(T))$, and it captures query uncertainty using either entropy or variance. We describe each next.

Entropy. In information theory, *entropy* roughly represents the expected number of questions to be asked to decode a distribution. The entropy for a given random variable X is defined as:

$$\mathbb{H}(X) = -\sum_{x \in X} \Pr(X = x) \log \Pr(X = x).$$

However, since the distribution of $C(T)$ can be a set of possible values each with a uniform probability, we need more contextual information to derive more meaningful entropy values. Let $D = [s_{\min}, s_{\max}]$ be the domain of all scores, and $\{B_1, B_2, \cdots, B_b\}$ be the scoring "bins" which uniformly partition D into several ranges. Each B_i represents the range $[s_{\min} + (i-1)\frac{s_{\max} - s_{\min}}{b}, s_{\min} + i\frac{s_{\max} - s_{\min}}{b})$. Let G_j denote the event $C(T) \in B_j$. We have $\Pr(G_j) = \sum_c \mathbf{1}(c \in B_j) \Pr(C(T) = c)$. In this case, we consider the sample space to be all possible B_i. Hence, we can define the entropy value as follows

$$\mathbb{H}(C(T)) = -\sum_{1 \leq j \leq b} \Pr(G_j) \log \Pr(G_j). \tag{5}$$

In the **Q** system, given a tree T, we can compute its entropy by maintaining the distribution over total cost when traversing T. As Formula 4 suggests, when edge e is visited, we maintain the sum $SF(i) = \sum_{e \in E(T)} f_i^e$ for feature f_i. Finally, we can compute the cost distribution by independence assumptions on W_i, and therefore derive the entropy. This can be done by dynamic programming as shown in Algorithm 1

Variance. Much like entropy, the variance value of a probabilistic distribution describes how diverse is the range of its possible outcomes. Using Formula 4 and independence assumption on different weights, we can compute the variance value as follows

$$\mathbb{V}(C(T)) = \sum_i SF^2(i) \mathbb{V}(W_i). \tag{6}$$

3.3 Clustering Queries

In active learning, the goal is to select query results such that the system maximizes its ability to learn (in this case, from user feedback). The uncertainty measure developed previously identifies the single sample (query result) with lowest confidence, in isolation. However, in the **Q** system, the user labels derived query results as positive or negative, but the feedback is converted into a modification of weights on individual features on edges or nodes. Such features may be **shared with other queries and their results** —

Algorithm 1 Computing entropy/variance for a query, or a tree

Input: A Steiner tree T
Output: Entropy and variance values for the total cost $C(T)$

```
for all feature $f_i$ do
    $SF(i) \leftarrow 0$
end for
for all edge $e \in E(T)$ do
    for all feature $f_i$ appears on $e$ do
        $SF(i) \leftarrow SF(i) + f_i^e$
    end for
end for
for all $d$ in the value domain $D$ do
    $P^0(d) \leftarrow 0$
end for
$P^0(0) \leftarrow 1$.
for $i \leftarrow 1$ to $|F|$ do
    for all $d \in D$ do
        $P^i(d) \leftarrow 0$.
    end for
    for all $d$ in domain $D$ s.t. $P^{i-1}(d) > 0$ do
        for all possible value $w$ which $W_i$ can take do
            $P^i(d+w) \leftarrow P^{i-1}(d)\Pr(W_i = w)$
        end for
    end for
end for
for all bin $B_i$ of total cost do
    $\Pr(B_i) = \sum_{d \in B_i} P^{|F|}(d)$
end for
return $H = -\sum_{B_i} \Pr(B_i) \log \Pr(B_i)$ and
    $V = \sum_i SF^2(i) V(W_i)$.
```

meaning that the system may improve its uncertainty on *multiple* queries from feedback on a single result. Ideally, we can find a few "representative" query results to return in the top-k results, and learn about many other results' scores from these representatives.

Our clustering strategy targets this problem. It presents to the user the results of a query (Steiner tree) that shares some highly uncertain edges with other, also-highly-uncertain, queries — such that feedback given on results from the first query (tree) can also reduce the uncertainty of the other queries. To achieve this, we must estimate common uncertain information between two Steiner trees and how informative a given Steiner tree is with respect to a keyword query. We use these to cluster overlapping queries and choose a representative query per cluster.

Clustering Algorithm. Our clustering algorithm takes a set of query trees $\{T_1, T_2, \cdots\}$ as input and clusters them into k groups, one associated with each top-k answer, similar to the k-means algorithm. (Note that in our domain it is intractable to compute the entire set of query results and then perform hierarchical agglomerative clustering; instead we can only produce some partial set of results and return k of them. Hence k-clustering makes sense.)

We define the *center* tree T of a set of trees to be a tree where each edge e has an associated appearance frequency $r_T(e)$, which is the ratio of number of trees having e to the size of the set. For any general Steiner tree T, $r_T(e)$ is defined as $\mathbf{1}(e \in E(T))$. Now, we define the similarity between two trees as follows.

$$\hat{S}(T_1, T_2) = \sum_{e \in E(T_1) \cap E(T_2)} \mathbb{U}(e) \min(r_{T_1}(e), r_{T_2}(e)).$$

This similarity roughly estimates the amount of common uncertainty of the two trees by summing up uncertainty values on their common edges. It can estimate the similarity between two Steiner trees as well as similarity between a Steiner tree and a "center" tree of a set. We use standard k-means clustering over this similarity function to build clusters.

Choosing Cluster Representatives. Once we have a cluster of similar queries, the next key question is how to choose one from them for feedback, such that the **Q** system can learn as much information as possible. We determine such a representative based on a notion of *informativeness*. Intuitively, the informativeness of a query with respect to a cluster of queries measures how much uncertainty this particular query shares with other trees in the cluster. Formally, given a cluster of trees C and a tree $T \in C$, we define the informativeness as follows:

$$I_C(T) = \sum_{T' \in C, T' \neq T} \sum_{e \in E(T) \cap E(T')} \mathbb{U}(e) \qquad (7)$$

where $\mathbb{U}(X)$ is any uncertainty measurement over a random variable X. Thus, we vote on a representative according to the above informativeness formalism, and choose the most informative tree to represent the cluster.

4. RANKING AND LEARNING

The previous section showed how to estimate the relevance of a query, how to cluster queries with shared edges, and how to identify representative queries based on informativeness. Once the **Q** system identifies the set of Steiner trees and clusters, it must rank the clusters such that they answer user's initial query and maximize the utility of potential user's feedback.

In this section, we consider two closely related issues. First, we need to *rank* the answers to a keyword query $Q = \{K_1, K_2, \ldots, K_m\}$, taking both relevance and uncertainty into account. We then consider how the system can learn from the user's feedback and update the scores and probability distributions associated with individual features.

4.1 Basic Ranking of Query Results

The cost of a query, i.e., a Steiner tree, derived in Formula 4, is a random variable from which our **Q** system determines its rank. A query's rank should depend on *relevance*, i.e., how likely is a query result to satisfy the user's information need. However, the query also has a certain amount of *uncertainty* in its score, which indicates how much extra information the **Q** system can learn from possible user feedback given on results derived from the query. A query's amount of uncertainty should also determine its rank since the user only sees a few top results and the system needs to maximize its learning gain. We have shown in Section 3 how to compute these two measurements for a given query. We now consider how to rank queries based on these values.

There is a tension between these two goals, rendering it semantically difficult to aggregate them for ranking and learning. We cannot directly use the decision-theoretical notion of stochastic dominance, nor skyline-based ranking, as these only produce *partial* orderings of results. Moreover, they fail to consider how the ordering affects the way a user provides feedback. Our trade-off between relevance and uncertainty in the long run is very similar to the problem of "exploration versus exploitation" in machine learning [30]. However, we must adapt existing techniques, because some of the key metrics are intractable to obtain in our setting.

We first consider two orthogonal notions of ranking, one based on the predicted relevance of results, and the other based purely

Algorithm 2 Computing top ranked queries

```
Input: Schema graph $G$
Output: A ranked list of trees

1: Compute all top-$k'$ Steiner trees $\{T_i\}$
   w.r.t. minimum expected cost
2: Cluster these trees into $k$ clusters,
   each with representative $T_{r(c_i)}$
3: Rank all clusters using one of the ranking methods
```

on uncertainty, followed by a weighted combination of the two. In Section 4.2 we will consider a more sophisticated means of combining the different facets.

Predicted-Relevance Ranking. A natural method of ranking is based on *predicted relevance*, i.e., the score obtained by combining the *expected values* of the features in the Steiner tree (query). Most keyword search-based systems, including prior versions of the **Q** system, adopt this ranking semantics. The final answer set in this model is a list of the k lowest-cost trees, in increasing order of expected cost. The top-k queries in this model can be computed by taking the graph, computing the expected cost for each edge and assigning it as the edge weight, then running a k-best Steiner tree approximation algorithm [35], which is tractable in practice.

Uncertainty Ranking. Conceivably, one could instead rank queries (and answers) according to the level of associated uncertainty. This is in some sense what systems supporting active learning typically do: focus the user's attention on the results that have the highest uncertainty, and thus the highest utility in learning how to rank.

The problem with this approach is that the entropy of a tree does not follow the principle of optimality with respect to the entropy of its subtrees. There is no obvious way to determine the top-k trees with respect to entropy, without enumerating all trees. Moreover, since a tree's entropy is not directly related to its predicted relevance, the highest-entropy query answers may not be useful to the user. Similarly, query answers with high variance values may not help answer the user's information need. For these reasons we next consider a more feasible hybrid strategy we term *mixed ranking*.

Mixed Ranking. The notion behind mixed ranking is that top answers should include a predicted relevance component to try to satisfy the user's information need, while still including some uncertain answers that are useful from an active learning perspective. This can be achieved as follows. We first compute a large subset of the queries predicted to be relevant, e.g., $2k$ most relevant query trees, and *then* choose from among these (possibly clustering them if they are highly overlapping) according to their uncertainty scores. Since we can tractably obtain approximate Steiner trees and compute entropy or variance for an individual tree, the overall mixed ranking is tractable in practice.

The scheme incentivizes the user to provide feedback: some of the query answers are likely to be of good quality, but they be mixed with bad answers. The user will be able to see that a small amount of feedback may result in an even more complete answer set.

4.2 Ranking by Expected Model Change

The previous ranking semantics fail to consider *position bias*: the user typically examines results from top to bottom and stops at some point. This behavior suggests that top items are more likely to receive feedback. Even for the same set of results, different orderings may yield different amounts of feedback to the system. To address this problem, we present a novel ranking semantics using the notion of *expected model change* [32] and taking the user's browsing behavior into account. Briefly, the expected model change resulted from an unlabeled sample quantifies the estimated change to the current model *if we knew its label*. Typically, if a sample has a higher value of expected model change, the system is likely to learn more information if its label is revealed. Furthermore, our ranking by expected model change algorithm yields *a provable relevance guarantee*: we show a cost lower bound for the list of top-k results obtained from this ranking method. This is desirable because query answers need to satisfy the user's information need.

4.2.1 Browsing and Feedback Model

We adopt a user browsing model very similar to [7, 17, 33]. The user starts by looking at the first result in the list and gives feedback on it. Then he or she continues to the second result with some probability p, or terminates his or browsing with probability $(1-p)$. If he does examine and give feedback on the second result, he will repeat the above behavior for the third result and so on. The user's browsing procedure terminates if he reaches the end of the list, or when he or she stops at a certain position. This stop position is what we refer to as the *watermark*, and we assume the user has vetted each result up to the watermark, and given negative feedback on any answers known to be incorrect.

Formally, if we let $r_1, r_2, \cdots, r_k$ be the top-k results where r_i is displayed at position i and let F_i denote the event that r_i is examined and feedback is given on it, we can model the user's browsing behavior as follows.

$$\begin{aligned}\Pr(F_1 = 1) &= \beta_1,\\ \Pr(F_i = 1|F_{i-1} = 0) &= 0,\\ \Pr(F_i = 1|F_{i-1} = 1) &= \beta_i,\end{aligned}$$

where $$0 \leq \beta_j \leq 1, \forall j.$$

The second formula assumes that the user stops browsing if he does not examine the previous result in the list. The third formula quantifies the probability that the user continues to the result at position i if he has examined the result at position $i-1$. We generally assume that probabilities for continuing may vary for different positions. We also do not assume these conditional probabilities diminish as position moves from top to bottom.

4.2.2 Expected Model Change

Given the browsing model, we now formalize the definition for expected model change. This will allow us to quantify the aggregated amount of relevance and that amount of uncertainty associated with the top-k results. Consider the top results $\{r_1, \cdots, r_k\}$, where each r_i has a corresponding Steiner tree T_i, whose cost is a random variable. Informally, the expected model change for r_i is the expected amount of uncertainty reduction if r_i is given feedback to and no other trees in the top list receive feedback. The expected amount of uncertainty reduction can be computed by simulating our learning module, described in Section 4.3. The learning algorithm takes the list of top-k trees and feedbacks given to each tree (positive, negative or no feedback) as input, and changes probability distributions over feature weights in the schema graph accordingly. This results in change of cost variables for graph edges, and therefore the uncertainty measure of cost functions associated with the schema graph G. We denote by G^+ the new graph if r_i is given a positive feedback, and by G^- the new graph if r_i is given a negative feedback. We also denote by $\mathbb{U}(G)$ the total amount of uncertainty associated with schema graph G, obtained by $\mathbb{U}(G) = \sum_{e \in E(G)} \mathbb{U}(e)$, where $\mathbb{U}$ is a given uncertainty measure, for instance, variance or entropy. Similarly, $\mathbb{U}(G^+)$ and $\mathbb{U}(G^-)$ describe amount of uncertainty associated with G^+ and

G^-, respectively. If we let ∇L_i be the amount of uncertainty reduction if r_i is given a positive feedback and $\nabla \overline{L_i}$ be the amount of uncertainty reduction if r_i is given a negative feedback, we can compute these values as follows.

$$\nabla L_i = \mathbb{U}(G) - \mathbb{U}(G^+), \qquad \nabla \overline{L_i} = \mathbb{U}(G) - \mathbb{U}(G^-).$$

According to the probabilistic interpretation of edge cost described in Section 2, we assume that $\alpha_i = 2^{-E[C(T_i)]}$ estimates the probability that r_i meets the user's information need, which in turn estimates the probability that r_i receives a positive feedback. Hence, the expected model change if r_i receives a feedback is given by

$$J_i = \alpha_i \nabla L_i + (1 - \alpha_i) \nabla \overline{L_i}.$$

Now that we have defined the expected model change for a given unlabeled query, we consider how to maximize the expected total amount of uncertainty reduction by incorporating our user browsing model. For a given query r_i, the user must first examine it in order to provide feedback. Hence, we can compute the **Q** system's expected utility from learning feedback given to r_i, denoted by $Y_s(i)$, as follows

$$Y_s(i) = \prod_{j=1}^{i} \beta_j J_i = \prod_{j=1}^{i} \beta_j (\alpha_i \nabla L_i + (1 - \alpha_i) \nabla \overline{L_i}).$$

Denote $B_i = \prod_{j=1}^{i} \beta_j$, which is the probability that the user has examined r_i. Similarly, we can obtain the relevance estimation, or the user's utility, for r_i, denoted by $Y_u(i)$ as

$$Y_u(i) = \prod_{j=1}^{i} \beta_j \alpha_i = B_i \alpha_i.$$

Combining utility scores for each individual result, we obtain the following two objective functions, one utility function for **Q** system's learning, and the other for the overall relevance.

$$Y_s = \sum_{i=1}^{k} Y_s(i), \qquad Y_u = \sum_{i=1}^{k} Y_u(i).$$

We apply an active learning strategy to maximize Y_s with respect to an ordering of a candidate result set, so that the **Q** system can learn as much information as possible. We will show that greedily ranking by J_i maximizes this objective function. Furthermore, we will show that Y_u has a lower bound if we rank results by J_i. This indicates that the total amount of relevance can be guaranteed.

THEOREM 1. *Y_s is maximized if $\{r_i\}$ is ranked by descending J_i. Moreover, $Y_u \geq \frac{Y_s^*}{\sqrt{\sum_{i=1}^{k} J_i^2}}$, where Y_s^* is the optimal maximized value.*

Proof Sketch. We omit the detailed proof here due to space constraints. The first part of the theorem can be proved based on an *exchange argument*. The key step for proving the second part is to apply the Cauchy-Schwarz inequality.

The above theorem establishes a nice connection between exploration and exploitation: actively learning by the **Q** system still provides relevance lower bound. In practice, since computing the full ranking requires enumerating all Steiner trees, which takes exponential time, we instead compute a group of most relevant Steiner trees and rank them by their values of expected model change.

4.3 Learning from User Feedback

Recall that the keyword search-based data integration model identifies correct attribute alignments by learning from user feedback over query answers. In our **Q** system implementation, two modules — the interactive user interface and the feedback-based learner — enable this capability. Once a keyword query is issued, the **Q** system returns the results computed by the top-k Steiner trees[3], using one of the ranking algorithms. The system then converts these trees to conjunctive queries, executes these queries, and returns tuple answers [35]. The **Q** system displays resulting tuples to the user annotated with *provenance*, in the form of a tree describing the query or queries that produced the answer.

The user examines a portion of the result set of tuples and gives feedback, either positive (via explicit positive marking or by specifying the "watermark") or negative (via explicit negative marking). From the cumulative feedback, the **Q** system learns which features in the schema graph, i.e., attribute alignments and data source qualities, are most relevant to the user.

In reality, there are two sets of weight parameters that must be learned: those for correcting edge alignments, which take uncertainty and the weighted scores of base schema matchers into account (Section 4.3.1), and the weights that should be given by default to each of those individual base schema matchers, before further feedback is given (Section 4.3.2).

4.3.1 Learning Edge Costs

Our prior implementation for the **Q** system [35] incorporates the MIRA [6] online learning algorithm, an online approximation to support vector machines, to receive feedback from the user in a streaming fashion and to update weight values. In a nutshell, MIRA attempts to find a new weight vector which is closest to the previous one and which satisfies constraints formalized from the feedback.

The implementation in this paper requires that feature weights be random variables instead of scalar values. The challenge, then, is how to adapt the online learning algorithm so that it can deal with probability distributions over feature weights. We adapt MIRA to our new setting as follows. The user indicates that a particular tree T^* should be the top ranked tree among the set of all top-k trees B. Let $\mathbf{w}$ be the vector where $\mathbf{w}_i = \mathbb{E}(W_i)$. We directly apply the MIRA algorithm, which takes the weight vector $\mathbf{w}$ as input and returns a new weight vector $\mathbf{w}'$ as output.

Finally, we compare $\mathbf{w}_i$ with $\mathbf{w'}_i$ for each i. If $w_i = w'_i$, then we keep the existing probability distribution for the weight on feature f_i, because f_i does not separate correct alignments and incorrect ones in the top-k trees and its weight does not need adjustment. On the other hand, if $w_i \neq w'_i$ for some i, the **Q** system will update this feature weight to a sure event: the weight random variable W_i will have value $\mathbf{w'}_i$ with probability 1. We show pseudocode for our learning algorithm in Algorithm 3, and the loop in Line 12 computes the new weight update. We use symmetric difference between two trees as the loss function, as in [35]:

$$L(T, T') = |E(T) \backslash E(T')| + |E(T') \backslash E(T)| \tag{8}$$

The online learning algorithm takes an *initial* score for an edge, based on a weighted combination of schema matcher outputs, and adjusts it. We next discuss how we set the initial score weights.

4.3.2 Learning Weights for Schema Matchers

[3] For simplicity we describe the outcome as if each query produces one result, although the system actually iteratively enumerates top-scoring queries, even beyond k such queries, until it gets k answers.

Algorithm 3 Online learner

Input: Search graph G, user feedback stream U, required number of query trees k
Output: Updated weights W

```
Initialize $W$
$r = 0$
**while** $U$ is not exhausted **do**
    $r = r + 1$
    $(S_r, T_r) = U.Next()$
    $w_i^{(r)} = \mathbb{E}(W_i)$
    $C_{r-1}(i,j) = \mathbf{w}^{(r-1)} \cdot \mathbf{f}_{ij} \;\; \forall (i,j) \in E(G)$
    $B = KBestSteiner(G, S_r, C_{r-1}, K)$
    $\mathbf{w}^{(r)} = \arg\min_{\mathbf{w}} ||\mathbf{w} - \mathbf{w}^{(r-1)}||$
    s.t. $C(T, \mathbf{w}) - C(T_r, \mathbf{w}) \geq L(T_r, T), \;\; \forall T \in B$
        $\mathbf{w} \cdot \mathbf{f}_{ij} > 0 \; \forall (i,j) \in E(G)$
    **for all** i **do**
        **if** $w_i^{(r-1)} \neq w_i^{(r)}$ **then**
            Update $W_i$ s.t. $\Pr(W_i = w_i^{(r)}) = 1$
        **end if**
    **end for**
**end while**
**return** $W$
```

The schema matching literature suggests that different matchers should have different weights in order to achieve a matching prediction with good precision and recall [28]. Such weight distributions may be different for different databases.

In the **Q** system, matcher weights are first applied to form probability distributions over alignment scores which compute query relevance and uncertainty. As the user poses more queries, the **Q** system learns the cost of individual attribute alignment. Under this model, it may seem that matcher weights are no longer needed after initialization. However, consider the case where new data sources are added into the current system. In order to apply the same active learning module, we will still have to compute probability distributions over alignment features for new relations. The results of such estimations directly depend on matcher weights. Hence, we need to learn, and periodically re-learn, such weights in order to perform more accurate predictions for new data sources. Note that, however, the newly learned matcher weights should not be propagated to edges whose costs are already updated from user's feedback.

We periodically use a linear regression model to relearn matcher weights: we choose this learning method because the goal of learning the best parameter settings fits naturally into the regression setting. Alternatives like Naive Bayesian or SVM learning are more appropriate for classification tasks.

The aggregate matching score for an attribute alignment is a weighted sum of individual matching scores obtained from base matchers. Each alignment edge in the schema graph (after several rounds of learning edge costs) is a labeled sample, where the expected value of the alignment feature weight serves as the aggregated score. Formally, let $\{M_1, M_2, \cdots, M_m\}$ be the set of matches and $p(M_i)$ be the normalized matcher weight of M_i. For each attribute alignment edge $e = (A, B)$, the estimated aggregated matching, computed by $\mathbb{E}(W_{f_{AB}})$, is its label and we assume that the score correctly reflects alignment quality. Each matcher M_i produces score $s_i(A, B)$ on edge (A, B). The learning procedure aims to find $\{p(M_i)\}$ and to minimize the loss objective function

$$\min \sum_{e=(A,B)} ((\sum_i p(M_i) s_i(A,B)) - \mathbb{E}(W_{f_{AB}}))^2.$$

This is a classic linear regression problem, and we can invoke the learning procedure periodically.

5. EXPERIMENTAL ANALYSIS

We now experimentally evaluate the different options for the suggester module, and their impact on learning, in the **Q** system. We seek to validate that active learning improves our ability to distinguish correct from incorrect schema mapping edges and return better query results, and to understand the differences among the ranking models and uncertainty metrics.

Datasets. A major challenge in conducting keyword search experiments across integrated data sources is that it is very difficult to identify the complete set of correct ("gold") alignments, and even more difficult to identify the set of correct answers. To simulate this in a controlled way that matches real data, we focus on real data where the possible joins are known. We chose three well-curated datasets and removed information about foreign keys, meaning the **Q** system must use schema alignment tools to discover potential edges, and learning to improve its knowledge of the correct scores. The datasets were chosen to represent very different domains, and include the bioinformatics testbed used in [34], which combines the widely referenced Interpro and GeneOntology (GO) datasets; the popular Internet Movie Database; and the Mondial geographic encyclopedia. For each dataset, we choose a subset of the tables. Details about the datasets are shown below.

Name	Bioinformatics	IMDB	Mondial
No. Tables	9	7	9
No. Attributes	48	21	36
No. True alignments	9	6	9
Size (in MB)	172	1082	7

Query Workload. For each dataset, we generated a workload comprising 7 (for Bioinformatics) or 10 (for IMDB and Mondial) keyword queries, whose results are revisited (and new feedback is given) three times. The queries were based on common-knowledge searches, and keywords with low selectivity were emphasized. Each of the visits (phases) is done in a randomly permuted order. Sample queries include "isomerase protein" for Bioinformatics and "Greece 'health organization'" for Mondial. Queries cover most possible join paths, including some with high and some with low uncertainty. For example, the path (roles.movie_id, movies.id) in IMDB has low uncertainty, while the path (interpro_interpro2go.go_id, GO_term.id) in Bioinformatics has high uncertainty.

Methodology. Experiments were conducted using our implementation of the **Q** system, which comprises approximately 55,000 lines of Java code. Evaluation was done using on an Intel Xeon CPU (2.83GHz, 2 processors) Windows Server 2008 (64-bit) machine with 8GB RAM, using JDK 1.60_11 (64-bit). For each dataset, the **Q** system first loads its schema (without knowing foreign keys) and constructs the schema graph. We run in parallel a set of schema matching primitives from the COMA++ schema matcher to compute a weight distribution between $[0, 1]$ for every alignment feature. We prune potential edges for which all matching primitives give low similarity scores. We also assign one of 10 possible uniform weight distributions over $[0, 1]$ to each node (relation) feature.

Once the schema graph is constructed, we iteratively pose keyword queries. In each iteration, the **Q** system returns the top-k ranked Steiner trees for the keyword query, according to one of our ranking algorithms to be evaluated. Then we simulate the user's feedback: a Steiner tree receives positive feedback if all of its edges are correct alignments (according to the actual schema information not provided to the system), and negative feedback otherwise. Note that our definition of correct Steiner tree is very strict, as there might be additional alignments that are not specified in the schema. The **Q** system then uses the feedback to learn adjustments to the feature weights, and updates the costs of edges in the schema graph.

We consider two dimensions: the uncertainty metric, namely entropy and variance; and the means of scoring results from relevance and uncertainty, including using relevance only, mixed ranking, and expected model change. Combining these options, we look at predicted relevance (**Relevance**), mixed ranking using entropy (**Mixed Ent**), mixed ranking using variance (**Mixed Var**), expected model change using entropy (**EMC Ent**), and expected model change using variance (**EMC Var**). We also present results for our query clustering scheme: here we only present the **EMC Var** ranking scheme, which proved to be the most effective.

Parameter Settings. We used $k = 5$ as the number of top queries to compute answers for each keyword search. To consider both relevance and uncertainty, we actually have the **Q** system fetch the top-$2k$ most relevant Steiner trees, and to choose from among these the top-k trees according to the combined ranking metric of study. When clustering, we use the top-$4k$ trees, which we combine into $2k$ clusters and then choose the top-scoring k results. We use the following COMA++ [9] schema matching primitives: data type similarity, string edit-distance, string q-gram distance, semantic similarity, and instance matchers. Initial matcher weights are trained offline before the experiment using a very small set of example attribute pairs.

We consider the following questions:

- Which active learning schemes most reduce the amount of training required to distinguish between gold and invalid schema alignments? Does clustering help? (Section 5.1)
- Which schemes require the least feedback to help the system discover correct alignments? (Section 5.2)
- Do the improvements in learning correct edges translate into an improvement in initial (before feedback) answer quality for keyword searches? (Section 5.3)

5.1 Speed of Learning

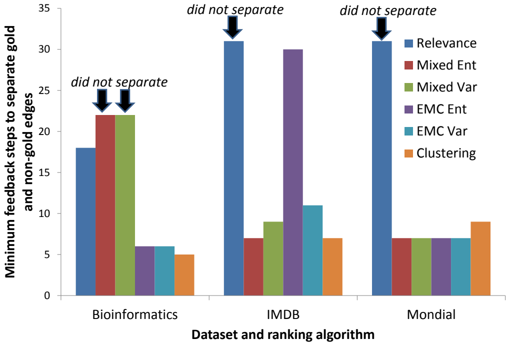

Figure 3: Speed of learning to distinguish gold vs. invalid edges

The focus of our work on active learning is speeding up (in terms of feedback steps required) the **Q** system's ability to discriminate between valid (gold) and invalid edges. Hence in this first experiment we measure how many feedback steps are required to *separate* gold-standard and erroneous edges, where our test for separation is whether the intervals corresponding to the mean plus or minus one standard deviation, for gold-standard and erroneous edges, are non-overlapping and remain non-overlapping.

We performed a comprehensive set of experiments comparing the various algorithms across our three datasets. Space constraints preclude presentation of all results, so we summarize in Figure 3 how many feedback steps (rounds) were required to separate the intervals. Note that there are a maximum of 21 rounds for the Bioinformatics dataset and 30 for the other datasets. If separation was not achieved within this limit, we mark this on the figure.

For both IMDB and Mondial, all of the active learning algorithms perform better than the standard (**Relevance**) method, which does not manage to achieve separation. For Mondial, which has a relatively small set of edges, most methods have comparable performance (with **Clustering** requiring slightly more work). For IMDB, the mixed ranking method **Mixed Ent** and the **Clustering** method (which uses expected model change plus variance) are most effective. The mixed ranking scheme **Mixed Var** and its expected model change variation **EMC Var** also perform well. Surprisingly, expected model change plus entropy is ineffective here, given that entropy itself is useful. For Bioinformatics, both of the expected model change-based schemes only require 6 steps to achieve separation, and the clustering algorithm does even better, requiring 5 steps. These outperform **Relevance**, which uses 18 steps, and the mixed ranking methods do not achieve separation.

To give a greater sense of the differences in costs, we provide more detail for the Bioinformatics dataset. This is representative of the other results, and focused on the target domain of the **Q** system. We see in Figures 4–6 the one-standard-deviation intervals around the mean score for gold-standard and invalid edges in the bioinformatics dataset, as each round of feedback is given to the system. (Recall that high scores mean high costs or dissimilarity.) In this dataset it takes 18 rounds to achieve separation of intervals for the **Relevance** method (Figure 4). The **Mixed Ent** method (Figure 5) never achieves separation within the 21 rounds of feedback (as its top-k answer set does not include results with several of the error-producing features). The **Clustering** method (Figure 6) performs best, achieving separation after only 5 feedback steps.

Although learning produces a large gap between the average costs of gold and non-gold edges, in a few cases, full separation is not achieved. Here the base schema matchers' scores for the incorrectly categorized edges have low variance and entropy — the matchers are "certain" about wrong alignments — so such edges do not show up in the top-k results and receive feedback.

Overall, expected model change with variance, with or without clustering, does best in learning to separate gold and invalid edges.

5.2 Recovering Gold Edges

We next study when active learning helps the **Q** system to find a more complete set of gold edges. The overall *edge recall* (ratio of gold edges to edges the system predicts) is determined by a combination of query load (which determines the set of nodes and to some extent trees) and ranking scheme (which determines feedback), and it may not always hit 1.0. We show in Figure 7 how rapidly the system reaches its maximum recall value given our limited number of queries and feedback steps (21 for Bioinformatics and 30 for the others). We include in the captions, in parentheses, the maximum recall value achieved for each dataset.

On average, active learning (particularly the expected model change-based methods including **Clustering**) significantly speeds up the rate at which the system achieves maximum recall. For Bioinformatics all of our new methods converge more quickly than **Relevance**, but the mixed ranking methods do not achieve separation and thus include a significant number of invalid edges. For IMDB and Mondial, the **Relevance** method is the one that does not achieve separation. We see that here the various active learning methods have comparable performance.

For more detail on the methods' behavior, we show in Figures 8 and 9 a visualization of a subset of the **Q** search graph, showing

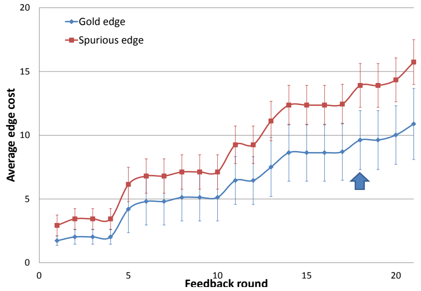

Figure 4: Separation for Relevance in the Bioinformatics dataset

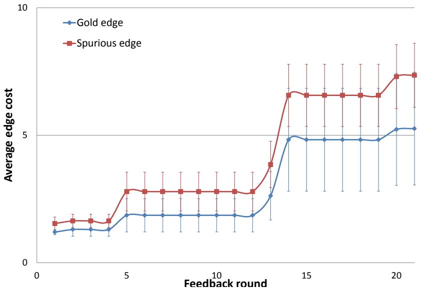

Figure 5: Separation for Mixed Ent in the Bioinformatics dataset

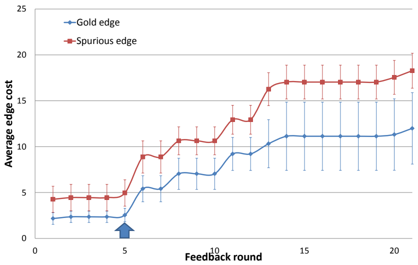

Figure 6: Separation for Clustering in the Bioinformatics dataset

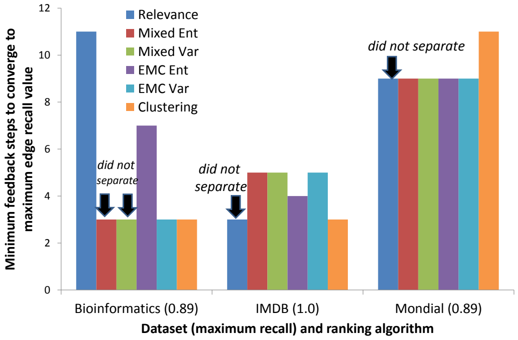

Figure 7: Feedback rounds required to reach maximum recall

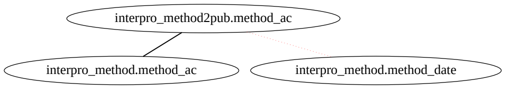
interpro_method2pub.method_ac
interpro_method.method_ac
interpro_method.method_date

Figure 8: Feedback on edges using Relevance: width is proportional to feedback steps; dotted lines indicate an invalid edge.

various relations and the edges among them. Dotted edges indicate invalid edges predicted by the schema matchers, and the width of the line indicates the amount of feedback given. We see that the **Relevance** method (Figure 8) provides less overall feedback on the edges, and explores fewer edges, than the **EMC Ent** method of Figure 9. We conclude that expected model-change-based methods achieve the best combination of separation plus edge recall. To give an idea of how edge recall changes over time, we plot this for all three of our datasets versus feedback rounds in Figure 10. Observe that maximum recall is achieved within 5-9 rounds of feedback.

5.3 Initial Query Answer Quality

While the system's goal is to learn correct rankings for the edges, the user's goal is to retrieve good query answers. Our final experiment measures how quickly the system — given feedback on the results from a set of *training* queries — can achieve 100% precision in the set of answers returned for a different but related *test* set of 5 queries. Figure 11 shows the number of feedback rounds required over the training data; arrows over the bars indicate that full precision was not achieved even after the maximum number of steps. The figure shows that for the first two datasets, active learning makes a significant difference, and that the **Clustering** algorithm shows measurable benefits over the expected model change methods. For Mondial the number of edges used in queries is small, so all methods receive adequate feedback to achieve perfect answers.

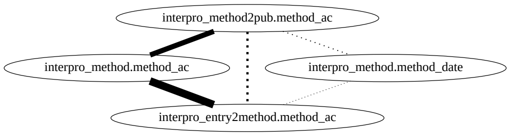
interpro_method2pub.method_ac
interpro_method.method_ac
interpro_method.method_date
interpro_entry2method.method_ac

Figure 9: Feedback on edges using EMC Ent: width is proportional to feedback steps; dotted lines indicate an invalid edge.

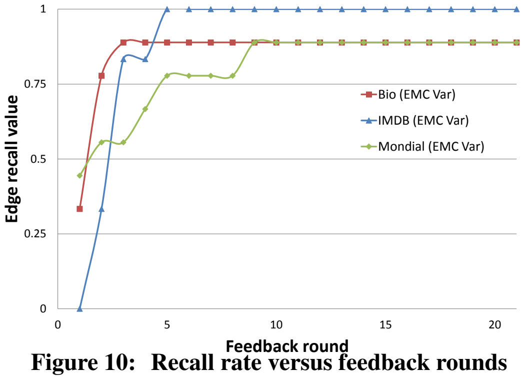

Figure 10: Recall rate versus feedback rounds

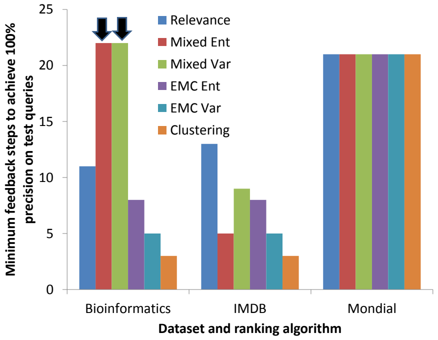

Figure 11: Quality of initial query answers

5.4 Discussion

Overall, we conclude that **EMC Var** is effective in virtually all scenarios, generally beating **EMC Ent**. The **Clustering** method produces the best initial answer quality, though its speed of learning and edge recall is sometimes slightly slower than the **EMC** methods. While our active learning techniques show significant benefits, there remains room for improvement. We have assumed that the different base matchers' scores will be relatively uncorrelated, allowing us to detect uncertainty. Section 5.1 showed that sometimes, though, the base matchers do not provide enough score diversity to indicate potentially incorrect alignments. In the future we hope to study whether a greater diversity of base matchers would help.

6. RELATED WORK

We previously discussed related work on keyword search in Section 2, including our prior work. The **Q** system incorporates off-the-shelf schema matchers. Like most modern matchers [28], our system combines output from multiple sub-matchers [9, 10, 27] (in particular we use their base matchers and learn to compose them). Our focus is on a general *architecture* for incorporating the output of matchers while obtaining entropy information. The problem of modeling uncertainty in schema matching is discussed in [14].

The problem of selecting the *most informative* feedback has been studied in data cleaning [36] and data integration. For integration, approaches include focusing on highest-value candidate schema matches for dataspaces [24] and on active learning for refining record linking in [2]. These are related in spirit to our approach, but they get feedback over *individual alignments* whereas we seek to understand the uncertainty associated with an entire query, and combine high-scoring and uncertain queries' results.

While active learning is a popular area of machine learning [30], standard techniques cannot be directly used on tree-structured queries in which individual edges have uncertainty. Three strategies have been primarily used in prior research: (1) the least confident strategy considers only the most likely prediction; (2) the maximum margin strategy considers the top two predictions; (3) the entropy maximization strategy considers all predictions, which can be exponential in the size of the structured object predicted. In contrast, we explore an intermediate setting by clustering k-best predictions, and considering a representative tree (query) from each cluster.

Prior work on active learning over structured output has sought to select the next *instance* upon which to receive feedback, with feedback directly over the predicted objects [30, Sec. 2.4]. Our work differs in keeping the instance (keyword query) the same, and soliciting feedback over different trees (interpretations) of the given query. With the notable exception of [21], most previous work on active learning over structured output involved sequences [8, 31], whereas in the **Q** system we infer trees.Although cluster-based active learning has been found to be useful in previous research [30, Sec. 5.2], such work has focused on classification and not structured prediction. Moreover, clustering in those cases is performed over the input instances, rather than the output Steiner trees corresponding to the given keyword query.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we studied several techniques for incorporating active learning into keyword search-based data integration. The primary challenges were how to estimate the amount of uncertainty associated with a query built from edges induced during schema matching, and how to rank results in a way that maximizes utility to the user and the system, simultaneously. We showed experimentally that the notion of expected model change, particularly using clustering, was highly effective. In future work we plan to deploy the system in the `www.ieeg.org` neuroscience portal and conduct user studies to see how it performs in practice.

Acknowledgments

We thank Burr Settles for his advice on active learning, and the anonymous reviewers for their feedback. This work was funded in part by NSF IIS-1050448, CNS-0721541, and a grant from Google.

8. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE*, 2002.

[2] A. Arasu, M. Götz, and R. Kaushik. On active learning of record matching packages. In *SIGMOD Conference*, pages 783–794, 2010.

[3] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *VLDB*, 2004.

[4] S. Bergamaschi, E. Domnori, F. Guerra, R. Trillo Lado, and Y. Velegrakis. Keyword search over relational databases: a metadata approach. In *SIGMOD*, 2011.

[5] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *ICDE*, pages 431–440, 2002.

[6] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7:551–585, 2006.

[7] N. Craswell, O. Zoeter, M. J. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *WSDM*, pages 87–94, 2008.

[8] A. Culotta and A. McCallum. Reducing labeling effort for structured prediction tasks. In *AAAI*, pages 746–751, 2005.

[9] H. H. Do and E. Rahm. Matching large schemas: Approaches and evaluation. *Inf. Syst.*, 32(6), 2007.

[10] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD*, 2001.

[11] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 19(1), 2007.

[12] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4), June 2003.

[13] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Rec.*, 34(4), 2005.

[14] A. Gal. *Uncertain Schema Matching*. Synthesis Lectures on Data Management. Morgan and Claypool, 2011.

[15] A. Gal and T. Sagi. Tuning the ensemble selection process of schema matchers. *Inf. Syst.*, 35(8):845–859, 2010.

[16] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an RDBMS for web data integration. In *WWW*, 2003.

[17] F. Guo, C. Liu, A. Kannan, T. Minka, M. J. Taylor, Y. M. Wang, and C. Faloutsos. Click chain model in web search. In *WWW*, pages 11–20, 2009.

[18] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD*, 2003.

[19] H. He, H. Wang, J. Yang, and P. S. Yu. Blinks: ranked keyword searches on graphs. In *SIGMOD*, 2007.

[20] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681, 2002.

[21] R. Hwa. Sample selection for statistical parsing. *Computational Linguistics*, 30(3):253–276, 2004.

[22] I. F. Ilyas, W. G. Aref, and A. K. Elmagarmid. Supporting top-k join queries in relational databases. In *VLDB*, 2003.

[23] M. Jacob and Z. G. Ives. Sharing work in keyword search over databases. In *SIGMOD Conference*, 2011.

[24] S. R. Jeffery, M. J. Franklin, and A. Y. Halevy. Pay-as-you-go user feedback for dataspace systems. In *SIGMOD*, 2008.

[25] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.

[26] A. Marie and A. Gal. Managing uncertainty in schema matcher ensembles. In *SUM*, pages 60–73, 2007.

[27] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *ICDE*, 2002.

[28] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4), 2001.

[29] M. Sayyadian, H. LeKhac, A. Doan, and L. Gravano. Efficient keyword search across heterogeneous relational databases. In *ICDE*, 2007.

[30] B. Settles. *Active Learning*. Morgan & Claypool, 2012.

[31] B. Settles and M. Craven. An analysis of active learning strategies for sequence labeling tasks. In *EMNLP*, 2008.

[32] B. Settles, M. Craven, and S. Ray. Multiple-instance active learning. In *NIPS*, 2007.

[33] S. Shen, B. Hu, W. Chen, and Q. Yang. Personalized click model through collaborative filtering. In *WSDM*, pages 323–332, 2012.

[34] P. P. Talukdar, Z. G. Ives, and F. Pereira. Automatically incorporating new sources in keyword search-based data integration. In *SIGMOD Conference*, 2010.

[35] P. P. Talukdar, M. Jacob, M. S. Mehmood, K. Crammer, Z. G. Ives, F. Pereira, and S. Guha. Learning to create data-integrating queries. In *VLDB*, 2008.

[36] M. Yakout, A. K. Elmagarmid, J. Neville, M. Ouzzani, and I. F. Ilyas. Guided data repair. *PVLDB*, 4(5), 2011.