HAMSTER: Using Search Clicklogs for Schema and Taxonomy Matching

Arnab Nandi[*]
University of Michigan, Ann Arbor
arnab@umich.edu

Philip A. Bernstein
Microsoft Research
phil.bernstein@microsoft.com

ABSTRACT

We address the problem of unsupervised matching of schema information from a large number of data sources into the schema of a data warehouse. The matching process is the first step of a framework to integrate data feeds from third-party data providers into a structured-search engine's data warehouse. Our experiments show that traditional schema-based and instance-based schema matching methods fall short. We propose a new technique based on the search engine's clicklogs. Two schema elements are matched if the distribution of keyword queries that cause click-throughs on their instances are similar. We present experiments on large commercial datasets that show the new technique has much better accuracy than traditional techniques.

1. INTRODUCTION

In this paper, we address the problem of unsupervised matching of schema information from a large number of data sources into the schema of a data warehouse. The application is the use of structured data sources to enhance the results of keyword-based web search. For example, Google, Yahoo and Live Search all provide shopping listings for the query "digital camera" above their traditional web search results, presumably by augmenting their keyword index with structured shopping data. This requires gathering a wide variety of structured data sources into a data warehouse that is indexed by the search engine for keyword queries. These sources are typically provided by third-parties, though they might also be obtained from web sites using information extraction. The sources need to be integrated so that similar data in the warehouse is indexed in the same way by the search engine, thereby improving the relevance of the result of keyword queries. The first step of the integration process is to match incoming data source schemas to the warehouse schema.

For example, suppose we are integrating data sources that describe movies. Our data warehouse of integrated data has a column called Rating, which describes the suitability of the movie for certain audiences (e.g., G, PG-13, R). We need to integrate a new data source which has an XML tag <MPAA> that contains the rating. It is beneficial to map <MPAA> to the column name Rating, so that instances of <MPAA> in the new source are recognized in our index as ratings. This enables the search engine to answer queries about the rating of movies that appear only in this new data source, such as a keyword query "rating Dark Knight".

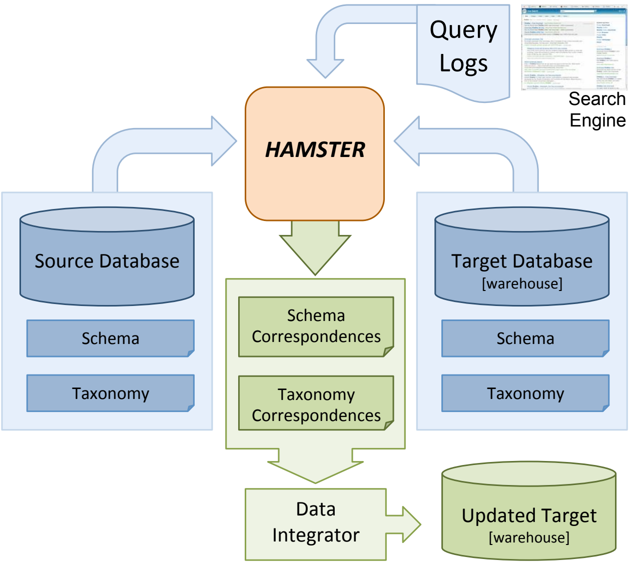

Figure 1: HAMSTER System Architecture

Some values in a data source are categorical. By "categorical," we mean the values come from a controlled vocabulary and are organized into a taxonomy. For better indexing, we need to map the data source's taxonomy to the data warehouse's taxonomy. For example, product catalogs usually categorize each product within a taxonomy. An item "netbook" might have an attribute "class" in a new data source, whose value is the path *computer* ▷ *portable* ▷ *economy* ▷ *small* in the data source's taxonomy. But the data warehouse may classify the item differently, such as *laptop* ▷ *lightweight* ▷ *inexpensive*. To do a good job of answering the query "netbook" over data feeds whose descriptions of netbooks do not contain the word "netbook," we need to map the data feed's class to the data warehouse's class and recognize "netbook" as a term for the latter class.

Our focus is how to generate the required schema and tax-

[*] Work done while at Microsoft Research.

VLDB '09, August 24-28, 2009, Lyon, France
Copyright 2009 VLDB Endowment, ACM 000-0-00000-000-0/00/00.

onomy mappings. In Fig. 1 we describe our system, *HAMSTER* (Human Assisted Mapping of Schema & Taxonomies to Enhance Relevance.) Our matching technique makes use of the search engine's query logs in addition to the usual information about schema structure and instances. To support a structured-search engine, there are many other steps to integrate the data, index it, and answer queries. The details of these steps are beyond the scope of this paper.

There are two aspects of this matching problem that differ from conventional enterprise data integration scenarios. First, it is important that little or no human intervention is required, so we can scale up to integrate a large number of data sources, with new ones arriving all the time. Second, we do not need or expect a perfect match. Although an imperfect match will degrade search quality, the search results may still be acceptable, especially if the imperfection affects only a few search results. By the same token, a schema matching result must be very accurate if it affects the integration of data that shows up in many search results, to avoid degrading search quality.

Despite these two differences, we can still apply conventional schema matching techniques. We need to produce a set of high confidence *correspondences* from the structure of the incoming data to the canonical structure of the warehouse, where each correspondence is a pair of elements, one from each schema. This can be done by extracting "features" from each of the data source's schema elements, and finding the data warehouse schema element whose features are most similar to those of the source element. If the similarity score of the most similar data warehouse element is too low, then the schema matcher does not return any corresponding data warehouse element. Doing this for all elements in the data source results in a set of correspondences, which is called a *mapping.*

There is a large body of work on schema matching that utilizes schema-oriented and instance-oriented information as hints [1, 4, 12, 18, 32]. However, applying these techniques to our scenario is problematic.

Inconsistent labels and structure of the schemas are one source of problems. Schema-oriented matching algorithms work well when naming conventions are standardized and there is a general consensus about how the data should be organized. These properties often hold in enterprise scenarios, but not for heterogeneous data sources gathered from all over the world. The structure can be divergent for many reasons, ranging from different regional and cultural conventions, to different platforms and optimization goals, to different applications for which the data sources were created. For example, we may want to integrate data feeds from a hardware manufacturing company in China with the feeds from an online store front in the U.S.

Instance-oriented matching algorithms require that the instances of the schemas to be matched have common features. Here too, there are many stumbling blocks. The same data can be represented in different units. For example, a laptop's RAM capacity can be represented in megabytes in one data source, and in gigabytes in another. Data conversion is often hard and ambiguous; one gigabyte can mean either 1000 or 1024 megabytes, and there are different policies for rounding off decimal places. Similarly, date and time units have many different formatting conventions. Additionally, data can come from different topic domains. For example the meaning of "rating" in a hardware database has nothing to do with the "rating" field from a review provider. We record some of the real-world challenges we encountered during our mapping task in Section 7.

Given these issues, we were not surprised that our experiments generated unsatisfactory mappings using unsupervised schema- and instance-oriented matching techniques. While some initial success with semi-automated tuning has been reported [34], the tuner still needs to be trained on a manually-produced gold standard. So even if conventional schema matching were satisfactory, a domain expert would need to continually curate the mapping as new data sources arrived.

2. USING CLICKLOGS TO MATCH

Since schema-oriented and instance-oriented techniques did not perform well enough for our purposes, we needed another source of information to drive the matching. We therefore explored the use of query logs extracted from the search engine. These logs contain click-through data that indicates which search results a user clicked on.

A *clicklog* contains $<q, u>$ pairs, each of which indicates that a user clicked on URL u, which was one of the results returned by the search in response to the user's keyword search query q. The intuition that drives our use of clicklogs is this: *if two items in a database are similar, then they should be searched for using similar queries.* To exploit this intuition, for each schema element and taxonomy term, we mine clicklogs to obtain the *query distributions* that led to click-throughs on instances of that element or term. Then for each schema element or taxonomy term S in the source, we identify the schema element or taxonomy term in the target whose query distribution is most similar to that of S.

For example, a user looking for small laptops may issue the query "netbooks," and then click on two of the URLs that were returned, one for "eee pc" and one for "hp mininote." This establishes that "eee pc" and "hp mininote" are related. Hence, even though the "eee pc" is considered its own product taxonomy term ("eee") by Asus, it can be matched with the "mininote" taxonomy term from HP, because the respective items from both companies were clicked on when searching for "netbooks," "under 10-inch laptops" and "sub notebooks". If one were to consider all the queries that led to categories from each source, we expect to see a high overlap between the queries of similar categories.

Clicklogs present unique advantages as a similarity metric. First, they are generated by users, and are hence independent of the data provider's naming conventions with respect to schema and taxonomy. Moreover, query information is self-updating over time. Users automatically enrich the data with new and diverse lexicons, capturing various colloquialisms that come into use. For example, using conventional approaches, to handle the recent popularity of new small form factor laptops, a human would have to manually update the search engine's thesaurus to reflect that "netbooks" and "sub notebooks" are synonyms. This would have to be done for each language that the search engine supports—a big expense that does not scale well. Additionally, clicklogs provide a wealth of information for a wide variety of topics. If there is user interest in a domain, then there will be clicklog data that can be mined. Clicklogs are also more resilient to spamming attacks. Current systems can be tricked by mislabeling an incoming feed causing data to be wrongly integrated. By contrast, to influence a clicklog, a byzantine data provider would have to issue a large number of search queries and click on URLs, posing as different users or IPs to achieve similar effect.

We outline the contributions made by this paper.

1. We introduce the problem of data integration for structured-search engines and show how it differs from traditional enterprise data integration scenarios (Section 1).
2. We introduce "query distributions" derived from search engine clicklogs as a feature that can be used to determine the similarity of schema elements (Section 2).
3. We show how to reduce the problem of matching taxonomy paths that appear as data values into the problem of schema matching (Section 3).
4. We introduce new techniques for deriving query distributions that are associated with schema elements and taxonomy terms (Section 4).
5. We introduce the use of *surrogate* items to leverage query distributions for items that do not appear in clicklogs (Section 5).
6. We report on experiments that show the use of query distributions improves schema matching results compared to conventional schema-based and instance-based techniques (Section 6).
7. We show how to integrate the query distribution technique into an existing schema matching framework (Sections 6.2 and 8).

3. COMBINING SCHEMA & TAXONOMIES

In this section, we describe how to reduce the taxonomy mapping problem into the schema matching problem, and hence re-use our schema-matching framework and concepts to perform both operations.

First, we require our incoming data feed to be in XML format, and the warehouse to be a collection of XML entities. Since XML data can be represented as a tree, we can consider schema mapping to be a problem of mapping between nodes in two trees, one representing the data feed's XML structure and one representing the warehouse schema.

The mapping process first extracts a set of features from the structure and content of the XML data. It then uses a set of similarity metrics to select correspondences from the tree representing the incoming data schema to the tree representing the data warehouse schema. For example, consider the following XML feed:

```
<feed>
  <laptop>
    <name>ASUS eeePC</name>
    <class>Portables ▷ Economy ▷ Smallsize</class>
    <market>Americas ▷ USA</market>
  </laptop>
</feed>
```

For the schema mapping task, the words "ASUS" and "eeePC" are considered as features for the schema element "name". With instance-oriented matching, the warehouse schema element whose instances contained many mentions of "ASUS" would be selected to match "name".

Taxonomy terms usually appear in data sources as data values within entities, not as schema elements. We need to map each of these data source values into the corresponding warehouse taxonomy path. To do this, we use a "pivot" operation that converts the categorical part of the XML element into a virtual XML schema, including other fields as needed. In the example, we can perform the pivot operation first on the categorical field "class," keeping "name" as a feature. This converts the feed into:

```
<feed>
  <laptop>
    <Portables>
      <Economy>
        <Smallsize>ASUS eeePC</Smallsize>
      </Economy>
    </Portables>
  </laptop>
</feed>
```

This virtual XML allows us to overload the schema mapping operations to also perform taxonomy matching. This is done for every categorical field we need to map (e.g. "market"), resulting in a taxonomy tree with many leaf nodes. We focus on mapping only the leaf nodes of each tree.

4. USING CLICKLOGS AS FEATURES

To drive the schema matching task, we aggregate the clicklog into a *clicklog summary*. Each entry is of the form <*q, n, u*>, where *n* is the number of times that a user clicked on URL *u* when it was presented as a result of search query *q*. For example, an entry <*laptop, 5,* `http://asus.com/eeepc`> means that for the query "laptop," the search result with URL `http://asus.com/eeepc` was clicked 5 times. All other information, such as unique identifiers for the user and the search session, is discarded for privacy reasons.

4.1 Generating Query Distributions

To match two schema elements or taxonomy terms, we extract the query distribution across all clicklog entries that correspond to each element. We then generate a correspondence between the two elements if their query distributions are sufficiently similar.

To do this we need to associate each schema or taxonomy element with the set of clicklog entries whose URLs refer to that element. We do this in two steps. In the first step, we associate each element to be matched with the set of entities that are instances of the element. These entities are the values of the element that are found in the data feed. For example, if an entry in a data feed associates the taxonomy term "Computers ▷ Laptop ▷ Small Laptops" with the value "HP MiniNote," then we regard "HP MiniNote" as an entity that is an instance of that taxonomy term.

In the second step, we associate each entity with the URL in the clicklog summary that describes that entity. This turns out to be an easy task, because most modern websites are database driven. Since each website has to query the database to generate a webpage for the requested URL, websites typically embed a unique key value of the entity in the URL itself and have a catalog that publishes the definition of keys. For example, Amazon.com uses a unique "ASIN number" to identify each product in their inventory. This ASIN number also appears as a part of each Amazon.com product page URL. Thus, the URL "`http://amazon.com/dp/B0006HU400`" is about the product with ASIN number B0006HU4OO, or "Apple Macbook Pro" (similarly, in our clicklog example below, "macbookpro" and "mininote" could be primary keys to identify the corresponding items in the database). Hence, to find the URL for an Amazon.com product item, we can simply look up the product item's ASIN number, and then select all clicklogs entries whose URL contains this ASIN number. This convention was found to be true for all major database-driven website frameworks and almost all the top shopping websites we inspected[1].

The ***query distribution of an entity*** is the normalized frequency distribution of keyword queries where the query returned that entity and the user selected it (i.e., clicked on it). For example, according to the clicklog in Table 1a, of the 25 queries that led to the click of the entity "eeePC" (denoted by `http://asus.com/eeepc`), 5 were for "laptop," 15 for "netbook" and the remaining 5 for "cheap notebook." Hence, after normalization, the query distribution is *{"laptop":0.2, "netbook":0.6, "cheap notebook":0.2}*.

An *aggregate class* is the set of entities that are instances of a given schema element or taxonomy term. The ***query distribution for an aggregate class*** is the normalized frequency distribution of keyword queries that resulted in the selection of any of the member instances. Table 1b presents query distributions for 3 aggregate classes.

query	freq	url
laptop	70	http://searchengine.com/product/macbookpro
laptop	25	http://searchengine.com/product/mininote
laptop	5	http://asus.com/eeepc
netbook	5	http://searchengine.com/product/macbookpro
netbook	20	http://searchengine.com/product/mininote
netbook	15	http://asus.com/eeepc
cheap netbook	5	http://asus.com/eeepc

Database: Aggregate class	query distribution
Warehouse: "...▷ Small Laptops"	{"laptop": 25/45, "netbook":20/45}
Warehouse: "...▷ Professional Use"	{"laptop":70/75, "netbook":5/75}
Asus.com:"eee"	{"laptop":5/25, "netbook":15/25, "cheap netbook":5/25}

Table 1: (a) A sample clicklog. (b) Inferred query distributions for 3 aggregate classes

4.2 Using Query Distributions

Given an incoming data source, we need to generate a mapping between the aggregate classes (taxonomy and schema elements) of the incoming data and those of the data warehouse. To do this, we do a pairwise comparison between the query distributions of each aggregate class of the incoming and warehouse data sources. Similarity scores above a certain tunable threshold are considered to be valid candidate correspondences.

To compare query distributions, we need a comparison metric. For now, we use Jaccard similarity. We consider other alternatives at the end of this section.

To revisit our previous example, suppose the warehouse contains only one *HP Mininote* small laptop product item, with the taxonomy term "Computers ▷ Laptop ▷ Small Laptops". Suppose the warehouse also contains the *Apple Macbook Pro* item, the only laptop that falls under "Computers ▷ Laptop ▷ Professional Use". Now, suppose Asus, another laptop manufacturer, wishes to include its data in our index. It uploads an XML feed to our system, as described previously. The "eee PC" item is assigned the taxonomy term "eee" in the Asus feed, and our task is to map "eee" to the appropriate warehouse taxonomy term.

To do this, we generate query distributions for the aggregate classes representing each of the two warehouse categories, and then compare them with the query distribution for the aggregate class representing the ASUS feed taxonomy term "eee". We analyze our clicklogs in Table 1a and observe that 100 people have searched (and clicked a result) for the word "laptop". Seventy of them clicked on the Apple Macbook Pro item, 25 on the HP MiniNote item, and 5 on the link for the Asus eee PC item in the incoming feed. For the query "netbook," we observe 40 queries, 5 of which clicked on Macbook, 20 on the MiniNote product, and 15 on the eee PC. For the query "cheap netbook," 5 out of 5 queries resulted in clicks to eeePC.

We count not only the number of clicks on the items in the warehouse such as the Apple Macbook Pro, but also clicks on the Asus items, since the search engine indexes the Asus web site. In addition, we have a mapping of product pages on asus.com to entities of the incoming feed, since each page's URL is constructed from the primary key of the entity. So when someone clicks on an asus.com result, we can translate it to a click to an Asus item.

Hence, the query distribution for the aggregate class representing the data provider's "eee" taxonomy term is {"laptop":5, "netbook":15, "cheap netbook":5}. For the aggregate class representing the data warehouse's "Computers ▷ Laptop ▷ Small Laptops" taxonomy term, the distribution is {"laptop": 25, "netbook":20}, and for "Computers ▷ Laptop ▷ Professional Use," the query distribution is {"laptop":70, "netbook":5}. (For clarity, we have not normalized the query distributions in this paragraph.)

After processing the clicklogs to generate query distributions of the aggregate classes, we can now compare each pair (i.e., Warehouse class "Computers ▷ Laptop ▷ Small Laptops" vs Asus.com class "eee" and Warehouse class "Computers ▷ Laptop ▷ Professional Use" vs Asus.com class "eee") and generate correspondences as follows:

```
GENERATE-CORRESPONDENCES(Classes $WC$, Classes $IC$)
for each class $wc$ in $WC$
  do for each query $ic$ in $IC$
       do EMIT($wc, ic$, COMPARE-DISTRIBUTIONS($wc, ic$))
```

```
COMPARE-DISTRIBUTIONS(Distribution $D_H$, Distribution $D_F$)
$score \leftarrow 0$
for each query $q_h$ in $D_H$
  do for each query $q_f$ in $D_F$
       do $minFreq = \text{LESSER}(D_H[q_h], D_F[q_f])$
          $score \leftarrow score + \text{JACCARD}(q_h, q_f) * minFreq$
return $score$
```

Jaccard similarity JACCARD($q1, q2$) of two queries $q1$, $q2$ is defined as:

[1] Alexa.com Top Shopping Sites

$$Jaccard = \frac{|Words(q1) \cap Words(q2)|}{|Words(q1) \cup Words(q2)|}$$

For example, consider comparing the query distributions for the aggregate classes of ASUS taxonomy term "eee" {"laptop":0.2, "netbook":0.6, "cheap netbook":0.2} and the warehouse taxonomy term "Small Laptops" {"laptop":0.56, "netbook":0.44}. Comparing each combination of queries, the score is $(1 \times 0.2 + 1 \times 0.44 + 0.5 \times 0.2) = 0.74$. On the other hand, the score for comparing "eee" with the warehouse taxonomy term "Professional Use" would be $(1 \times 0.2 + 1 \times 0.07 + 0.5 \times 0.07) = 0.31$, which is much less than the previous similarity score. Clearly, "Computers ▷ Laptop ▷ Small Laptops" is the recommended correspondence for incoming taxonomy term "eee".

Jaccard similarity can be replaced by other functions, such as WordDistance:

$$WordDistance(n) = Len(Words(q1) \cap Words(q2))^n$$

Jaccard compensates for large phrases that are common, looking at only the ratio of common vs. uncommon words. WordDistance allows for exponential biasing of overlaps, also considering the length of the common words. Another alternative is simply to use the exact string comparison function, hence only counting the queries that are identical in both distributions. We call this alternative the Min variant. It is significantly faster because it does not have to do word-level text analysis. We compare and contrast these functions in Section 6.4.

4.3 Remarks

In this section we offer three remarks about using clicklogs and query distributions for schema matching. First, we offer some intuition by comparing them to schema-based and instance-based matching using join operations as an analogy. In schema-based matching, each warehouse class in the set of classes WC is compared with each incoming provider class in the set of classes IC, possibly generating a correspondence. So in a sense, we are performing the join

$$WC \bowtie_{SB} IC$$

where the schema-based join operator $\bowtie_{SB}$ generates correspondences based on the text similarity of the left and right sides. In instance-based matching, we add another join on each side to map classes to their underlying entities, $Items_W$ and $Items_I$, and then do an instance-based join $\bowtie_{IB}$:

$$(WC \bowtie Items_W) \bowtie_{IB} (Items_I \bowtie IC)$$

In clicklog-based matching, we introduce a further query-distribution-based join $\bowtie_{QD}$ with the clicklog CL:

$$(WC \bowtie (Items_W \bowtie CL)) \bowtie_{QD} ((CL \bowtie Items_I) \bowtie IC)$$

This intuitive view highlights the fact that clicklog-based schema matching is not competitive with schema-based or instance-based methods. It simply adds another feature that can be used to improve the result of other schema-matching methods.

Second, we observe that query distributions are a general feature of schemas that can be understood independent of their use in schema matching. They have the following properties, which might be useful in contexts beyond schema matching, such as classification applications.

- The query distributions of similar entities are similar. (E.g., if the Toshiba m500 and Toshiba x60 are similar items, then the query distributions for the Toshiba m500 and Toshiba x60 are similar.)
- Query distributions of similar aggregate classes are similar.
- The query distribution of a database item is closest to the aggregate class it belongs to. (This allows us to use query distributions for classification purposes.)

We verified these properties in experiments that are described in Section 6.5.

Third, we observe that the fact that two entities have similar query distributions depends on their ranking in the results of search queries. Thus, our approach leverages the search engine's ability to identify related entities (i.e., URLs) in response to a query. Indeed, one could argue that our query-distribution method does only that. However, this discounts the benefit provided by users, since they generate search keywords and click on relevant items more than irrelevant ones. We are leveraging their vocabulary and judgment, not just the search engine's ranking.

We can test this claim by viewing simple search engine-based clustering as a special case of instance-based matching, with the content of the entities as features, and the search engine's ranking as a similarity function. As we will see in Sec. 6.2, search-based clustering does not perform as well as query distributions. One reason is the benefit of user clicks; search queries provide high-quality human-generated features for each URL (i.e., entity). Another is that search engine results are not based solely on content similarity. They are biased by popularity metrics such as PageRank. Hence, the most similar item may rank below a less similar but more popular item. Since query distributions span multiple queries, the association of two items due to their overlapping query distributions is much stronger.

5. FINDING SURROGATES

The use of query distributions requires that data sources have a web presence, so that their results show up in web search results and are clicked on. Some data providers do not satisfy this requirement. However, they are likely to have established competitors with similar data, which do have significant presence in the clicklogs. Thus, we propose a method by which we identify *surrogate clicklogs* for any data source without significant web presence. For each candidate entity in the feed that does not have a significant presence in the clicklogs (i.e. clicklog volume is less than a threshold), we look for an entity in our collection of feeds that is most similar to the candidate, and use its clicklog data to generate a query distribution for the candidate object.

```
Get-Surrogate-ClickLog(Entity $e$)
  $query \leftarrow$ DB-String($e$)
  $similarItems \leftarrow$ Similar-Search($targetDB, query$)
  $surrogateUrl \leftarrow similarItems[0].url$
  return Get-ClickLog($surrogateUrl$)
```

The functions DB-String and Similar-Search are user implemented. For example, we use the surrogate method for our third task in the following section. Since one of our datasets, called the *Whale* dataset, does not have a web presence, we use clicklog information from Amazon.com as a substitute. For our purpose, given an instance in Whale, the DB-String function returns the concatenation of the entity's "name" and "brandname" attributes as the query string, i.e., *return item.name + " " + item.brandname.* For the "Similar-Search" function, we used the Yahoo! Web Search API with the "site:amazon.com inurl:/dp/" filter to find the appropriate Amazon.com product item and url (every Amazon.com item has a unique product page). This lets Yahoo! search for pages only within the *amazon.com* domain, which contain "/dp/" in their URL. In line 4, we simply pick the top result among those returned, and use its URL as our surrogate URL. The search engine's clicklog is then searched for this URL to generate a surrogate clicklog. For this taxonomy mapping task, we first sample 100 items per taxonomy term for the incoming Whale data.

For each of the 1500 categories in Whale, we sample 100 items from its inventory. Then, we search for them on Amazon, and end up finding 3651 items representing 853 Whale categories.

6. EXPERIMENTS AND EVALUATION

In this section, we compare the efficacy of our mapping algorithm against other methods, and explore various other facets of the process. We begin in Section 6.1.1 by describing our metrics and algorithms. We then describe the data sets and mapping tasks in Section 6.1.2, and describe the algorithms compared in Section 6.1.3. We also study the effect of clicklog size on mapping quality in Section 6.3, and evaluate the use of *surrogate* clicklogs. Then, in Section 6.4 we consider different functions for comparing query distributions to pick the ideal one. We conclude by presenting evidence in Section 6.5 that verifies our claims that query distributions are a valid similarity metric.

6.1 Experimental Setup

6.1.1 Evaluation Metrics

In order to evaluate our mapping algorithm, we first need to define the methods and metrics for evaluation. For each of the integration tasks described in Section 6.1.2, we derive or construct gold standards listing the correct mappings. A mapping is a set of *correspondences.* Each correspondence is a pair of elements, one from each schema (or taxonomy).

Each algorithm is then run, producing a set of correspondences. For each set of correspondences, we compute recall (i.e., the fraction of correct correspondences that we produced) and precision (i.e., the fraction of correspondences in the produced set that are correct). Ideally, mapping algorithms should have both high recall and high precision. When this is not possible, we prefer to have high precision at the cost of reduced recall. Since mappings produced are used down the pipeline in automated and semi-automated processes for the search engine, an erroneous mapping could have an amplified impact in overall result quality. Thus it is important to maintain high precision and minimize any errors that we introduce into the system, even if it is at the cost of reduced recall.

Recall and precision are defined using the notions of true(T) and false(F) positives(P) and negatives(N), where $Recall = TP/(TP + FN)$ and $Precision = TP/(TP + FP)$.

Generated correspondences that were wrong (whose left side existed in the gold standard, but not the right side) or didn't exist (even the left side was not present) in the gold standard are considered as false positives. Correspondences whose left side existed in the gold standard, but matching correspondences that did not exist in the test mapping were counted as false negatives. Correspondences produced that exist in the gold standard were marked true positives, while true negatives are not considered.

Since a practical framework may contain more than one algorithm to perform data integration, we are interested in comparing the output of our *Query-Distribution* method against other methods, namely *Schema-based* and *Instance-based* methods, which will be described in the following paragraphs. Differences and similarities of outputs give us a deeper understanding of the strengths and weaknesses of each method, which can then be used to come up with a combined meta-mapping algorithm. We construct a *Consensus* based algorithm in this light, and study its performance.

6.1.2 Datasets

To derive query distributions, we used a sample of query log data from the Live.com web search engine, restricted to only English U.S. queries. Blank queries, pornographic or offensive keywords and navigational queries were removed using blacklists. As expected, our query log displays properties similar to published studies on search query logs, and follows Zipf's law: many of the queries have a low frequency of occurrence and a few of them are highly frequent. Clicklogs are attained by filtering those entries that lead to a valid URL. Query distributions were then generated by mapping URLs to entities. For example in Task B below, 1.8 million search queries in our query log sample resulted in clicks to pages with URLs starting with http://amazon.com and contained a valid ASIN number. The URLs were then mapped to an Amazon product using the ASIN id. Each product had a query distribution averaging 13 unique (i.e., different) search queries (min 1, max 181, stdev 22).

To study many different aspects of query-distribution based mappings, we need to pose a variety of questions. However, we were unable to find a single real dataset that captures this variety. Instead, our experiments evaluate three data integration tasks. Each task is an actual data integration challenge we encountered during the development of our search engine, and each poses unique challenges typical of real-world scenarios. Our first task is to integrate the schema of an incoming data feed. The second task involves the mapping of taxonomies. The third task evaluates the use of surrogate clicklogs.

Task A. Schema Integration: Our data warehouse's *Movie* database contains movie and showtime information gathered from a number of data sources. The warehouse Movie schema contains 5 domains for *movies*, *people*, *showtimes*, *series* and *companies*, each with 4-5 schema elements. For example, each item of the movies domain has the elements "Name," "Runtime," "Rating," among others. We consider the inclusion of an additional data provider (let us call it *XYZ Movies*) which provides feeds called *Movie* and *Theatre.* We focus on the mapping of schema elements within the feed to schema elements in the warehouse. In our case, we consider the mapping of the data warehouse's *movies* domain with the *Movie* from our data provider. The warehouse *movies* domain contains 30 different XML elements and 16 attributes. The incoming *Movie* was originally in CSV format with 24 fields, converted to a flat XML stream. A gold standard was hand-created for this. We use a sample of 10,000 entities from the data warehouse, and a sample of around 1400 entities for the incoming data.

Task B. Taxonomy Integration: The data warehouse's *Shopping* domain contains millions of computers and electronics items, each of which is assigned exactly one taxonomy term from about 6000 leaf categories. The incoming Amazon.com dataset is a collection of 70,000 computers and electronics items, each having multiple categories. We study a sample of 258 products that existed in both catalogs. We manually derive a gold standard by identifying the instances in both catalogs, and then use the taxonomy term information on each side (613 in Amazon, 43 in our warehouse) to produce 901 correspondences we consider to be correct. The taxonomy correspondences were observed to be consistent in the following sense: If an item was assigned the "laptops" taxonomy term in Amazon, and the same item was assigned the taxonomy term "portables" in the warehouse, then all other items under "laptops" were also assigned "portables" in the warehouse. 93% of the Amazon categories consistently mapped to the same warehouse taxonomy term.

Task C. Schema & Taxonomy Integration, Use of surrogates: The *Whale* (name anonymized) database contained a list of 2.6 million computers and electronics product offers and discount information. Each offer was assigned exactly one taxonomy term out of a taxonomy of around 1500 leaf categories. Other fields included product name, brand name and various pricing options, of which we only used the brand name and product name fields. We attempt to integrate this with the warehouse Shopping database from the previous task B. Unlike the previous task, we do not have any instance information from the warehouse side. This is because the Shopping database is split in an ad hoc manner across multiple partitions, and sampling entities for each of the 6000 categories is extremely expensive to compute. A correspondence of each of the 1500 Whale leaf categories to exactly one of the 6000 warehouse categories was done manually to create the gold standard.

6.1.3 Algorithms

For each task, we compare our query distribution based methods against two other methods. The first method utilizes the "Schema-based" techniques of the matching algorithm described in [32]. This method performs matching based on lexical properties of labels, and the structure of the schema tree (and in our case, also the taxonomy tree). We call this method *Schema-based* mapping.

The second method, which we call the *Instance-based* method, uses the content of the warehouse's entities as features, and schema elements and taxonomy terms as labels to be input to a Naive Bayes classifier. A sample of the incoming data is then classified using this model. The schema element or taxonomy term of the incoming data sample and the output label are considered as a correspondence, and all correspondences above a certain score threshold are considered.

We also compare a *consensus* algorithm that combines the correspondences generated from the Schema-based, Instance-based, and query-distribution methods by summing the normalized scores for each candidate correspondence. The consensus based method is meant *solely* to understand the interactions between the various algorithms. Our focus is the impact of various features on mapping quality. More advanced "meta-mappers" listed in the related work section can always be run using the mentioned mapping algorithms as input, generating better results.

6.2 Evaluating Integration Quality

We now present recall and precision numbers for each task and method. Each algorithm returned a set of correspondences with a similarity "score," signifying the algorithm's confidence that the correspondence was correct. The score was thresholded at increasing levels to calculate the various combinations of recall and precision.

Schema Integration: For Task A, the Schema-based method was able to achieve 88% precision and 47% recall, thanks to the use of standard names such as "Director" / "director" and "Title" / "movie_title" in the schema. The Schema-based method was not able to map elements such as "Genre" / "Category" or "MPAA" / "Rating". The Instance-based method was much better at this, since the datasets contained identical or similar values for these fields. Overall, the Instance-based method was able to get 71% precision with 73% recall. The combination of the two in the consensus method, was able to correctly map all but five elements. For each of these elements, the correct correspondence was among the top-3 choices suggested.

While such high recall and precision is sometimes feasible with Schema-based and Instance-based methods, this may not be achievable for incoming feeds whose data does not overlap, or are in languages or formats different from the warehouse. In this case, a non-instance based method such as query distributions is needed. We thus attempt to map the incoming *XYZ Movies* data stream, using surrogate clicklogs of `imdb.com`. Various parts of the data were mapped to appropriate URLs. For example, the "awards" field for the movie *Titanic* was linked to `http://www.imdb.com/title/tt0120338/awards`. This method produced 11 correspondences, 5 of which were correct. We observed that query distribution worked well for the frequently queried parts of the database. Schema elements that are often part of user queries, such as "awards," "category," "person," and "movie_title" were correctly mapped. Elements whose instances are not usually part of user queries, such as "country code," "MPAA," and "runtime," were mapped incorrectly.

Taxonomy Integration: For Task B, we can see in Fig. 2 that the query distribution method clearly outperforms the others, with 92% precision for 100% recall. Correspondences are also of high confidence; for each taxonomy term, the score for the top correspondence suggestion was over five times the score of other suggestions on average. Instance-based matching achieved a low 11% precision and 95% recall, because the classifier was using the name and brand name as its input features, but products were named differently in the warehouse and at Amazon. While the names were different in the two sources, users still searched for them using similar keywords, resulting in a successful mapping by query distribution.

We now evaluate the use of the search engine for instance-

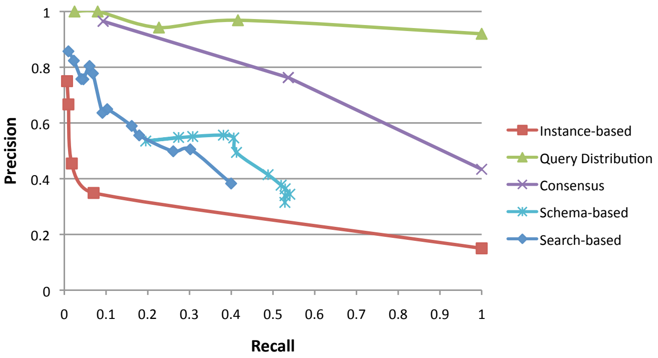

Figure 2: Taxonomy Integration in Task B

based mapping, which was suggested in Section 4.3.

To match the Amazon.com and warehouse taxonomies, we built a similarity matrix M whose rows are Amazon terms and whose columns are warehouse terms, initializing all entries to zero. We queried the search engine for each Amazon.com item's product name, restricting the result to pages (and hence entity instances) from our warehouse's website. Let A be an Amazon item and W be a search result for A from our warehouse. For each taxonomy term t_A for A and t_W for W, we add one to $M[t_A, t_W]$. We match t_A and t_W if $M[t_A, t_W]$ exceeds a threshold, which was varied to get the range of precision-recall points shown in Fig. 2 .

As we can see in the figure, recall for this method is bounded to 40%, while precision is between that of the Schema-based and Instance-based methods. When we performed the experiment in reverse, searching for warehouse objects restricting the result to Amazon.com pages, we found that the precision again bordered around 50%, while the recall was even worse, tapering off at 33%.

We believe these numbers are a strong indicator of our previous assertions. Query distribution is an important feature for schema matching. Queries and clicks from users performed better than schema-based similarity of the aggregate class, or instance-based similarity on the data. Not only does it lend to increased precision, but it also increases recall by creating a multitude of mappings, based on the search queries by the user.

The high quality of query-distribution based matching is primarily because we are looking at products with a lot of clicklog data. We do not expect all of our data to have such high-traffic clicklog information, and thus detailed query distributions. A study of the effect of clicklog size on quality is presented in the next section. While all three methods concurred on 43 correct correspondences, query distributions also correctly found 27 correspondences not discovered by the Schema-based method. This shows that query distributions don't just compete but complement the other methods, motivating the use of the Consensus algorithm, which performs much better than the Schema-based method.

Although the consensus algorithm was not as good as the query-distribution method, we believe this is due to our naive meta-mapper that simply added the similarity scores. A more sophisticated and well-tuned consensus algorithm should perform no worse than the component matchers and in most cases better than any of them, including the query-distribution method.

Schema & Taxonomy Integration, Use of surrogates: With Task C, we inspect a slightly larger dataset, where we achieve 61% precision and 53% recall with the Schema-based method, and 40% precision and 10% recall using a surrogate query distribution, using 100 items per Whale taxonomy term. On a sample a tenth the size (10 items per taxonomy term), we got half the recall for the same precision.

Note that the precision of the query-distribution method will suffer while using surrogates, since there is an obvious source of error introduced by looking for appropriate similar items on the web. Twenty of the correct correspondences were common with the Schema-based method, while 58 (3% of total correct correspondences) were those that Schema-based got wrong or did not suggest. Thus, the query distribution method lent an extra 3% of recall that can be leveraged. Upon combining the two methods using consensus, we are able to achieve 56% precision and 60% recall, demonstrating that we are able to extend the recall of the correspondences using query information.

In mapping the schema for Task C, 26 elements in the Whale schema were matched against 25 elements in the warehouse schema. Fifteen were correctly mapped when using the Schema-based method, while all of them were mapped when using the Instance-based method, by using 2200 warehouse items and 150 Whale items.

6.3 Effect of Clicklog size

The efficacy of the query distribution methods is largely impacted by the volume of the clicklogs. A large clicklog sample covers a larger diversity of possibly common words. We revisit Task B, varying the size of the clicklog by progressively halving it. In Fig. 3, we present the performance of matching items and categories. The leftmost point represents the case of using 1/32 of the clicklog. The rightmost point uses the full clicklog. From left to right, the points in between represent 1/16, 1/8, 1/4, and 1/2. As we can see, there is a sharp drop in recall as the clicklog shrinks. Precision also drops, but much less dramatically than recall.

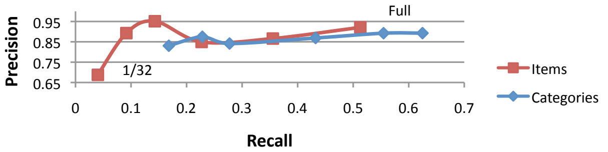

Figure 3: Varying clicklog size : Recall decreases as the size of the clicklog is decreased

6.4 Understanding Query Distributions

To use query distributions, we need a comparison function that compares two distributions. We propose a variety of functions, and study their effect on recall and precision by varying the threshold score above which a correspondence is considered a candidate.

Figures 4 and 5 show the behavior of these metrics for integrating the Task B dataset both at the document matching level and at the taxonomy term level. We noticed that there is not much difference between the different metrics. We choose the Jaccard metric (a query similarity based metric defined in Sec. 4.2) for most purposes due to its flexibility

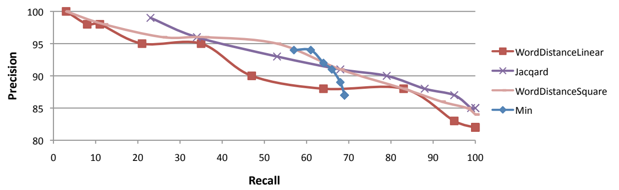

Figure 4: Query Distribution Metrics : Task B categories

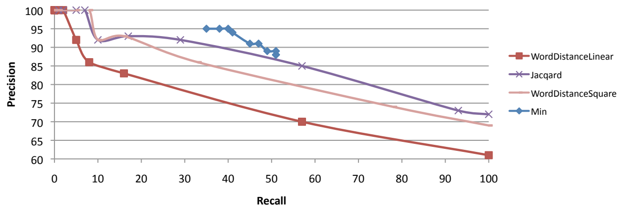

Figure 5: Query Distribution Metrics : Task B items

across varying precision and recall levels, but it should be noted that Min (metric that considers only identical queries, also defined in Sec. 4.2) is much faster since it does not have to do any string processing.

6.5 Claim Experiments

As described in Section 4.3, we first need to verify that query distributions are good measures of similarity. To do this, we pick the 40 categories from the data warehouse taxonomy whose products have the most clicks in our clicklog. We pick 10 items for each taxonomy term, resulting in 400 products. We found 258 of these items also listed at Amazon.com, representing a total of 613 categories (i.e., many items had more than one taxonomy term).

Table 2 presents the comparison of "query distributions" using the WordDistance metric. For each cell, query distributions of the corresponding row and column headers were compared, and the best match was picked. For example, the cell for *warehouse products* vs. *warehouse categories* denotes the ability to classify each product to a taxonomy term, by comparing the query distributions of the product and the taxonomy term. Here, 383 of 400 (or 95.75%) items were categorized correctly. Additionally, we calculated the "confidence" of our choice; this is the ratio of the scores of the top ranking item to the second ranking item, on average. The confidence numbers were observed to be high in all cases, ranging from 2.9x when matching warehouse products against themselves, to 5.6x when mapping Amazon categories to the warehouse categories, to 21x when matching Amazon products to themselves.

	Amazon products	Amazon categories	Warehouse Products	Warehouse categories
Amazon products	257/258 (99.6%)	241/258 (93.4%)	189/258 (73.3%)	226/258 (87.6%)
Amazon categories		373/613 (60.8%)	204/400 (51%)	525/613 (85.6%)
Warehouse Products			392/400 (98%)	383/400 (95.75%)
Warehouse categories				40/40 (100%)

Table 2: Comparing Query Distributions

As we can see in Table 2, almost 100% of query distributions are unique to both products and categories with very high confidence. An exception is the matching of Amazon categories, since a product can have more than one taxonomy term. Hence two categories end up having identical query distributions, and hence scores, in which case we arbitrarily choose one of the categories that have the identical top score. Also, the query distributions of products are closest to that of their categories. Additionally, 85% of Amazon categories map correctly to the right warehouse taxonomy term, again with high confidence. Based on our observations we consider our claims to be verified.

7. DATA IN THE REAL WORLD

While performing the tasks in the previous section, we learned a fair bit about the challenges encountered when dealing with real world data.

One of the primary challenges encountered was the heterogeneity of data models and conventions used. The data sources were in a wide variety of data models, from XML streams, to tab separated values, to SQL data dumps. Even within each data model, there are numerous conventions with regards to schema and data formats, such as:

Levels of normalization: Some data providers perform heavy normalization resulting in a large number of relations / XML entity types. Others encapsulate all their data into a single table / entity type with many optional fields.

In-band signaling: Many data values contain encodings and special characters that are references and lookups to various parts of the database. An example of this is to have a "description" field from our running example for laptops, where entity names are encoded into the text, such as "The laptop is a charm to use, and is a clear winner when compared to the $laptopid:1345$." The field $laptopid:1345$ is then replaced with a linked reference to another laptop by the application layer.

Attributes vs. Elements: There is great variation in XML data regarding the use of attribute values. Many datasets did not contain any attribute values, while one dataset was essentially a feed of a single entity type, which contained a large number of attributes in it. Our approach to this was to treat all attributes as subelements.

Partial Data: The data provided is often a "cutaway" of the original source, where certain parts of the database are missing for practical or privacy purposes. There are often many dangling references and unusable columns.

Multiple levels of detail: Providers have varying levels of granularity in categorical data. While one provider may classify a laptop item as "computer," another may file the same laptop under "laptops ▷ ultraportables ▷ luxury".

Provenance information: A large portion of the data is unusable for search access. For example, some data encountered were provenance and bookkeeping, such as the cardinality of other tables in the database and the time and date of last updates.

Domain specific attributes: Often the data provider uses a proprietary contraction whose translation is available only in the application logic, for example "en-us" to signify a US English keyboard.

Formatting choices: There is considerable variation in format. This is not restricted to just date and time formats. Providers invented their own formats, such as "56789:" in the "decades active" field for a person's biography, denoting that the person was alive from the 1950s to present.

Unit conversion: Quantitative data is often expressed in different interchangeable units, such as Fahrenheit vs. Celsius, hours vs. minutes. Also, the number of significant digits is variable. While one source can say 1.4GHz, the other may mention 1.38Ghz. Approximation is extremely sensitive to semantics. For example it cannot be applied to the terms 802.11, 802.2 and 802.3, since they most probably refer to networking protocols in the hardware domain.

Many of these are instance-level problems that can be solved at the entity extraction and entity reconciliation stages of the integration. However, before moving to the reconciliation stage, we still need a general schema and taxonomy alignment that can guide the following integration steps.

A possible solution to this is to have programmers on both sides, the search engine (the warehouse) and the third party data provider, to create standardized streams of data that conform to a canonical schema. While there has been a lot of work in developing universal semantic standards for data feeds [21, 22, 29, 40], it is easy to see why such an approach does not scale. Using a standards-based schema means that we can consider only the data types and domains where standards have been agreed upon. This restricts growth of the schema or taxonomy; any addition or deletion to the schema or taxonomy needs to be reflected in a new version of the standard. If the universal standard is extensible by independent data providers, then the extensions still have to be integrated across providers. Schema and taxonomy evolution are hard to cope with in a standardized environment, and the burden shifts to the third party data provider to do most of the heavy lifting required to conform to a standard. Since a search engine provider would like to make the path of adoption as easy as possible, it is critical to reduce the time investment involved in submitting new data by using automated methods such as ours.

8. RELATED AND FUTURE WORK

The problems of schema and taxonomy mapping are considered critical steps in the process of data integration [24] and have been widely studied. [32, 36] document the large body of work done towards mapping schemas, while [18, 31] discuss ontology mapping. In this paper, we inherit approaches from both sides, and propose a technique to solve both problems with the same framework.

Existing approaches for schema mapping use structural and linguistic similarities of the schema elements. Schema-based matching has been performed using string matching techniques [9], and has been combined with other approaches such as using synonym tables [8]. Learning approaches have been applied as well [12], using a semi-automatic approach with classifiers, integrity constraints and user feedback. The GLUE system [14] employs relaxation labeling to match ontologies. He and Chang [19] use a hidden generative model to interpret schemas in different websites. Similarity Flooding [28] calculates similarity across nodes of a graph (such as a schema graph) using a method inspired by network packet flooding.

The heterogeneity of application scenarios and methods makes it hard for schema mapping techniques to be applied and compared consistently. In our case, we leverage aspects of the data integration pipeline for a search engine. Unlike typical enterprise scenarios [30, 37], we cannot always expect to have well-documented schemas. Moreover, our schema is often inferred from instance data in the incoming XML feeds. Care must be taken to minimize human effort, keeping the data ingestion pipeline as unsupervised as possible. Our scenario is perhaps most similar to catalog integration [1] and e-business domain [38], but is purely a one-way task, integrating incoming data into the warehouse.

The heart of our schema matching technique is the use of clicklogs extracted from search query logs. The idea of using structured query logs to aid in schema matching was introduced by Elmeleegy et al. in [16]. They used SQL query logs as hints to indicate mapping of columns and a genetic algorithm to find the best set of correspondences across multiple matchers and features. They provide an excellent analysis of how to use the *structure* of the SQL queries (such as joins, group-bys, and aggregate functions) on two databases as clues to infer correspondences. By contrast, unlike traditional SQL databases, our search engine is purely keyword based; we cannot exploit any richness of query structure here. Hence, we look at the *content* of the query itself, using the similarity of keyword query phrases as clues for mapping. Since SQL queries perform exact or partial string matching, looking at the content of the queries would be identical to performing conventional instance-based matching, without the log. In contrast, web keyword queries are not always partial string matches to the content of the database item. For example, the word "netbook" may never be mentioned on the ASUS eeePC page (either due to branding, or because the page is in Spanish and uses the word *subportátil* instead), but a search for "netbook" can still lead to the eeePC page as a search result since other pages on the Web linked to it using the anchor text "netbook". Conversely, the eeePC page contained many mentions of the word "laptop". But since it is not a popular full size laptop, it will not have considerable presence in the clicklogs, despite showing up as a top-10 result. These nuances are unique to web search clicklogs, and our query distribution method correctly captures them to infer correspondences.

McCann et al. propose an intriguing technique of "crowd-sourcing" the schema mapping task [27] to volunteer integrators. We exploit user input too, but in a *passive* manner, implicitly using the consumers of our integration for the actual integration task. Unlike their method, we do not require explicit interaction with the user, hence avoiding such problems as adversarial behavior (i.e., spamming) and proper incentivisation of user feedback. Both methods share the burden of identifying users who can be treated as expert integrators, which can be done by clustering query and click behavior of all users[5].

In [4] and [25], the concept of "reuse-oriented" matching is presented. The current match operation is augmented with information gleaned from established corpora and previous successful mappings. Such concepts can be extended to query distributions as well; for example one can boost query co-occurrences by computing statistics over query logs for successful mappings or the entire search engine.

"Meta-mapping," which combines the outputs of multiple matchers, has been investigated in various scenarios.

Both LSD [12] and GLUE [14] use a multi-strategy approach by combining the mappings from a set of learners using a meta-learner. Unsupervised rank aggregation by maximizing ranker agreement [23] and using techniques from evidential reasoning [20] have been discussed. COMA++[4, 17] proposes a variety of match strategies and provides exhaustive infrastructure to evaluate, compose and combine matchers. Such methods can be used to combine Query-Distribution mappings with other sources. In [10] the continual post-integration improvement of "mediated schemas" is discussed to make them more amenable for future integration. We believe these techniques make for excellent inputs towards building our overall data integration framework, and would work very well in conjunction with the ideas proposed in this paper.

As a solution to structured search, the construction of an integrated warehouse is not novel. Google Base [21] already indexes a large amount of data, integrated from a variety of data providers. Data providers are encouraged to use Google's attributes. While universal data standards exist [40, 22], we still expect a large number of data providers to use proprietary schemas and taxonomies. This forces data providers to manually write code to translate from the proprietary formats to the standardized format for the warehouse, an undesirable task. Alternative paradigms include federated search over heterogeneous databases by automatically discovering relationships across the multiple databases [35] or using an A*-based search to execute queries over a merged schema [11]. Both approaches, while different from ours, are perfectly acceptable, and lead to mapping problems similar to ours. Adapting our techniques for these approaches as a suitable topic for future work.

While query log information has been exploited for relevance ranking in search, HAMSTER is the first system to our knowledge that uses keyword query distributions to map schema and taxonomies. Literature on query distribution from the image processing community concentrate on histograms with numerical bins, while we are concerned with query strings as bin labels. [33] proposes *Earth Mover's Distance* as a similarity metric, which is the amount of work required to transform one distribution to another. [7] compares different methods to compare query log information for web search by analyzing the overlap in query terms, result ranks and result contents. We leverage these ideas and propose our own similarity metrics in Section 4.

9. CONCLUSION

With the continual growth of structured data available on the web, it is increasingly important to be able to sift and search through these mountains of structured data. Data integration thus becomes a necessary step to provide a unified search interface. Mapping schema and taxonomy are critical parts of the data integration pipeline, allowing the search engine to effectively ingest large numbers of incoming feeds in an unsupervised or semi-supervised manner.

In this paper, we explored the scenario of schema and taxonomy integration in the context of a keyword search engine. We proposed a single framework to tackle both problems, and presented the issues involved in building the framework. We discovered that semi-supervised mapping is an achievable goal for our tasks, automatically generating 40-60% of the correspondences with a precision of above 70%. Keeping our overall framework in mind, we looked into the use of information previously unavailable in traditional mapping scenarios. We proposed the use of click information from the search engine's query log as a feature in our mapping framework. We discovered that clicklogs are a promising source of features, generating correspondences that many current methods cannot find. This is because current methods analyze only the content of the data, while we use information *outside* of the actual data, i.e., users' queries, to propose mappings. The clicklog method is ideal when the data comes from disparate sources and has little overlap with the contents of the warehouse, a common situation in our experience. We observed that query distributions can produce very high (nearly perfect) precision mappings, and that the recall is proportional to the amount of the clicklog information available.

Going forward, we foresee a multitude of directions where advancements can be made. We currently require a few hours to extract relevant clicklogs, and less than an hour to produce the actual mappings. The former time could be reduced drastically by properly indexing clicklog data for our needs, while the latter can be reduced to near instant time by proper optimization and parallelization (the nature of our algorithm makes it especially amenable to a Map-Reduce style adaptation). By reducing the mapping generation time, we can provide the data providers with an interactive experience, showing them how their data will be integrated right when they upload their data. Additionally, clicklogs are noisy, in that users often click through to URLs that turn out to be irrelevant. Removing noise should improve our results and would be worthwhile future work.

While we consider the keyword queries as structure-less phrases, it may be possible to discover an ad hoc structure in them. For example, the query "macbook pro prices" can be translated to "[name] prices". The unsupervised templatization of keyword queries has been shown to be possible at web scale [39]. We plan to incorporate this technique into our framework, allowing us to analyze the (fairly limited) structure of the queries as an additional input feature for the mappers. Furthermore, the clustering of related queries [41, 6] can discover connections between query terms, allowing us to generate even more input features. Another direction to explore is mapper confidence. In the presence of multiple mappers, how do we determine that the clicklog information is the best source of mappings for a particular schema or taxonomy element? A simple option is to threshold the absolute score values. Another option is to look at the amount and quality of the clicklog used to make the mapping proposals, since we expect sections with a large number of users whose clicks agree with each other to produce better mappings than sections with a few disagreeing users.

As documented by similar web-scale integration projects [26], the integration pipeline is required to provide the best-effort answers at all times, while continually improving itself by incorporating feedback. An interesting source of feedback information we possess is query log information, which can be exploited to measure the search satisfaction of the user. One possible idea is to use *search satisfaction* as an objective function for mapping quality. To do this, one could leverage the method of sample testing, where a small fraction of the search engine users are presented with a modified search mechanism. Various aspects of their behavior, such as order of clicks, session time, answers to polls or surveys, etc. are used to measure the effectiveness of the modification.

While each mapping usually consists of the top correspondence match for each entity, one could instead consider the top-k correspondences for each item, resulting in multiple possible mapping configurations. Each mapping configuration is run as a sample test, and the mapping that results in the most satisfactory user experience is then picked as the final mapping answer. A method like this would be an excellent complement to the work in this paper.

10. ACKNOWLEDGEMENTS

We thank Thomas Bergstraesser and Bert Casper for suggesting the problem, Smitha Kalappurakkal for help with the data and Sergey Melnik for many helpful discussions.

11. REFERENCES

[1] R. Agrawal and R. Srikant. On Integrating Catalogs. *WWW*, 2001.

[2] S. Amer-Yahia. A Database Solution to Search 2.0 (keynote talk). *WebDB*, 2007.

[3] E. Amitay and A. Broder. Introduction to Special issue on Query Log Analysis: Technology and Ethics. *TWEB*, 2008.

[4] D. Aumueller, H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. *SIGMOD*, 2005.

[5] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query Recommendation using Query Logs in Search Engines. *International Workshop on Clustering Information over the Web*, 2004.

[6] R. Baeza-Yates, C. Hurtado, and M. Mendoza. Query recommendation using query logs in search engines. *EDBT Workshops*, 2004.

[7] E. Balfe and B. Smyth. A Comparative Analysis of Query Similarity Metrics for Community-Based Web Search. *LNCS*, 2005.

[8] P. Bernstein, J. Madhavan, and E. Rahm. Generic Schema Matching with Cupid. *VLDB Journal*, 2001.

[9] P. Bernstein, S. Melnik, and J. Churchill. Incremental Schema Matching. *VLDB*, 2006.

[10] X. Chai, M. Sayyadian, A. Doan, A. Rosenthal, and L. Seligman. Analyzing and Revising Data Integration Schemas to improve their Matchability. *VLDB*, 2008.

[11] W. Cohen. Integration of Heterogeneous Databases without Common Domains using Queries based on Textual Similarity. *SIGMOD Record*, 1998.

[12] A. Doan, P. Domingos, and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach. *SIGMOD Record*, 2001.

[13] A. Doan and A. Halevy. Semantic Integration Research in the Database Community: A Brief Survey. *AI Magazine*, 2005.

[14] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Ontology Matching: A Machine Learning Approach. *Handbook on Ontologies in Information Systems*, 2004.

[15] X. Dong, A. Halevy, and J. Madhavan. Reference Reconciliation in Complex Information Spaces. *SIGMOD*, 2005.

[16] H. Elmeleegy, M. Ouzzani, and A. Elmagarmid. Usage-Based Schema Matching. *ICDE*, 2008.

[17] D. Engmann and S. Massmann. Instance matching with COMA++. In *BTW Workshop*, 2007.

[18] J. Euzenat and P. Shvaiko. *Ontology Matching.* Springer-Verlag, 2007.

[19] B. He and K. Chang. Statistical Schema Integration across the Deep Web. *SIGMOD*, 2003.

[20] J. Hong, H. Zhongtian, and D. Bell. An Evidential Approach to Query Interface Matching on the Deep Web. *VLDB*, 2008.

[21] W. Hsieh, J. Madhavan, and R. Pike. Data management projects at Google. *SIGMOD*, 2006.

[22] R. Khare and T. Çelik. Microformats: a Pragmatic Path to the Semantic Web. *WWW*, 2006.

[23] A. Klementiev, D. Roth, and K. Small. An Unsupervised Learning Algorithm for Rank Aggregation. *ECML*, 2007.

[24] M. Lenzerini. Data Integration: a Theoretical Perspective. *PODS*, 2002.

[25] J. Madhavan, P. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *ICDE*, 2005.

[26] J. Madhavan, S. Jeffery, S. Cohen, X. Dong, D. Ko, C. Yu, and A. Halevy. Web-scale Data Integration: You can only afford to Pay As You Go. *CIDR*, 2007.

[27] R. McCann, W. Shen, and A. Doan. Matching Schemas in Online Communities: A Web 2.0 Approach. *Proceedings of ICDE*, 2008.

[28] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. *ICDE*, 2002.

[29] P. Mika. Microsearch: An Interface for Semantic Search. *Semantic Search*, 2008.

[30] P. Mork, A. Rosenthal, L. Seligman, J. Korb, and K. Samuel. Integration Workbench: Integrating Schema Integration Tools. *Workshop on Database Interoperability at ICDE*, 2006.

[31] N. Noy. Semantic Integration: A Survey of Ontology-based Approaches. *SIGMOD Record*, 2004.

[32] E. Rahm and P. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB Journal*, 2001.

[33] Y. Rubner, C. Tomasi, and L. Guibas. The Earth Mover's Distance as a Metric for Image Retrieval. *International Journal of Computer Vision*, 2000.

[34] M. Sayyadian, Y. Lee, A. Doan, and A. Rosenthal. Tuning Schema matching Software using Synthetic Scenarios. *VLDB*, 2005.

[35] M. Sayyadian, H. LeKhac, A. Doan, and L. Gravano. Efficient Keyword Search across Heterogeneous Relational Databases. *ICDE*, 2007.

[36] P. Shvaiko and J. Euzenat. A Survey of Schema-Based Matching Approaches. *LNCS*, 2005.

[37] K. Smith, P. Mork, L. Seligman, A. Rosenthal, M. Morse, D. Allen, and M. Li. The Role of Schema Metching in Large Enterprises. *CIDR*, 2009.

[38] M. Stonebraker and J. Hellerstein. Content Integration for e-business. *SIGMOD*, 2001.

[39] B. Tan and F. Peng. Unsupervised query segmentation using generative language models and wikipedia. *WWW*, 2008.

[40] S. Weibel, J. Kunze, C. Lagoze, and M. Wolf. Dublin Core Metadata for Resource Discovery. *Internet Engineering Task Force RFC*, 1998.

[41] J. Wen, J. Nie, and H. Zhang. Query clustering using user logs. *ACM TOIS*, 2002.