Schema Clustering and Retrieval for Multi-domain Pay-As-You-Go Data Integration Systems

Hatem A. Mahmoud
University of Waterloo
Waterloo, ON, Canada
hamahmoud@cs.uwaterloo.ca

Ashraf Aboulnaga
University of Waterloo
Waterloo, ON, Canada
ashraf@cs.uwaterloo.ca

ABSTRACT

A data integration system offers a single interface to multiple structured data sources. Many application contexts (e.g., searching structured data on the web) involve the integration of large numbers of structured data sources. At web scale, it is impractical to use manual or semi-automatic data integration methods, so a pay-as-you-go approach is more appropriate. A pay-as-you-go approach entails using a fully automatic approximate data integration technique to provide an initial data integration system (i.e., an initial mediated schema, and initial mappings from source schemas to the mediated schema), and then refining the system as it gets used. Previous research has investigated automatic approximate data integration techniques, but all existing techniques require the schemas being integrated to belong to the same conceptual domain. At web scale, it is impractical to classify schemas into domains manually or semi-automatically, which limits the applicability of these techniques. In this paper, we present an approach for clustering schemas into domains without any human intervention and based only on the names of attributes in the schemas. Our clustering approach deals with uncertainty in assigning schemas to domains using a probabilistic model. We also propose a query classifier that determines, for a given a keyword query, the most relevant domains to this query. We experimentally demonstrate the effectiveness of our schema clustering and query classification techniques.

Categories and Subject Descriptors

H.4.0 [**Information Systems Applications**]: General

General Terms

Algorithms, Design, Experimentation

Keywords

data integration, clustering, classification

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

1. INTRODUCTION

As the number of structured data sources on the web continues to increase, so does the difficulty of organizing them and making them accessible. A prominent example of structured data sources on the web is the large number of web sites that provide access to databases through web forms. Such databases hidden behind web forms are usually called the *deep web* or the *hidden web*, and are believed to surpass the *surface web* in quantity and quality [4]. Recent studies by Google estimate an order of 10 million high quality HTML forms [13]. Many other types of structured data sources spanning a wide spectrum of domains are also available on the web, such as HTML tables and downloadable spreadsheets. The need to provide access to a large number of heterogeneous structured data sources also arises on a smaller scale in personal information management and scientific data management applications [9].

One of the approaches used to access such large numbers of heterogeneous structured data sources is to treat their data as mere documents and apply keyword search on them using information retrieval (IR) techniques. For the deep web, various techniques have been proposed to *surface* it, making it searchable via traditional IR techniques [14]. This approach, however, does not take much advantage of the structure of data sources. Another approach that takes advantage of such structure is to use data integration. Data integration systems provide the user with a unified interface to access a set of data sources that provide information about the same real-world domain but have different schemas. Typically, a data integration system is established by first defining a *mediated schema* that represents the domain that is being considered and acts as the user's interface to the system. Mappings are then defined from the schemas of the various data sources to that mediated schema. Creating and maintaining data integration systems has always been an expensive process that consumes much effort. Consequently, much research has been done to facilitate that process by developing techniques that recommend mediated schemas and schema mappings to the user [16].

Different data integration techniques require different levels of user involvement. At web scale, the massive number of data sources makes even semi-automatic data integration techniques impractical. For example, attempts to use semi-automatic integration techniques by Google [14] indicate that a human annotator working on data integration with the help of semi-automatic tools can integrate only 100 schemas on average per day. The other alternative, which is fully automatic data integration, produces imprecise medi-

ated schemas and schema mappings. Therefore, it was suggested [13] that a pay-as-you-go data integration approach is the only way to deal with web-scale data integration. A pay-as-you-go data integration system accepts approximate and incomplete integration as a starting point, and allows further enhancements to be introduced later, whenever deemed necessary. The system starts providing services, e.g. keyword search, without having to wait until full and precise integration takes place. To deal with imprecision in fully automatic integration, prior research [6, 7] proposes using a probabilistic model where several possible mediated schemas are generated, each assigned a probability value. Then, from each data source to each of the generated mediated schemas, several possible mappings are generated, and each mapping is also assigned a probability value. However, all existing fully automatic integration techniques assume that the data sources to be integrated belong to the same domain, so a preprocessing phase is still needed to cluster data sources into domains before data integration takes place [13]. Without such a step, data integration is more likely to produce semantically incoherent mediated schemas and inaccurate mappings to these schemas. Surprisingly, there has been very little work on automatic clustering of data sources into domains.

In this paper, we present an approach for clustering structured data sources into domains based on their schemas. When working on structured web data sources, we are faced with many challenges. First, the only information guaranteed to be available about a data source is attribute names. Even simple information like attribute data types is not always easy to determine. Therefore, our clustering approach relies entirely on attribute names to cluster schemas into domains. Second, we do not know in advance all the domains we should have or how many they are, since the web is essentially about everything. Consequently, we use a clustering algorithm that does not make assumptions about the number or the types of domains in advance. Third, since we are proposing a fully automatic technique, we need to handle uncertainty in deciding which domain a schema should be assigned to. We use a probabilistic model to deal with this uncertainty, where each data source may belong to multiple domains with different probabilities. Typically, after schemas are clustered into domains, existing techniques of schema mediation and mapping will be run on each domain separately. Our work integrates well with previous work on schema mediation and mapping with uncertainty [6, 7].

At query time, we need to give the user the capability to search for relevant domains. For example, a search engine needs to detect when a keyword query contains attribute names that are relevant to one or more of the domains constructed in the clustering phase. More concretely, a keyword query like "departure Toronto destination New York" contains two attribute names that are relevant to the 'travel' domain, namely 'departure' and 'destination'. The search engine can then retrieve the mediated schemas of relevant domains and present them to the user in the form of structured query interfaces as part of the search results page, ranked by their relevance to the query. The user can then pose structured queries over any of those query interfaces and retrieve structured data. We present a technique based on naive Bayes classification to determine the domains relevant to a query and rank these domains according to their relevance to the query. The probabilistic nature of the domains adds more challenges to classification.

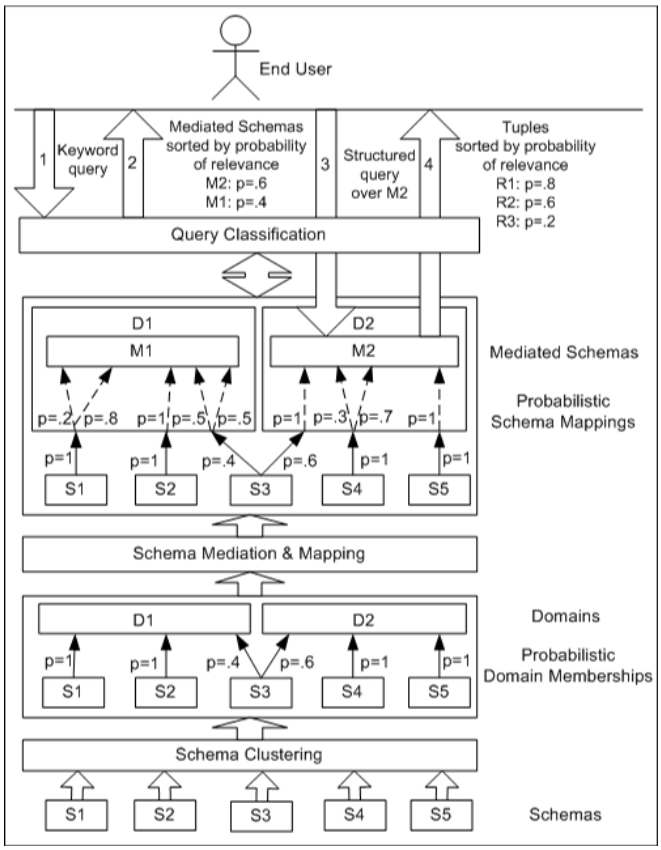

Figure 1: System architecture illustrated via an example of typical use case.

Figure 1 illustrates the architecture of our system with an example of a typical use case. Our contributions in this paper can be summarized as follows:

1. A fully automatic technique for clustering schemas into domains based on attribute names only.
2. A probabilistic approach for handling uncertainty in clustering. Our approach integrates seamlessly with existing approaches for schema mediation and mapping with uncertainty.
3. A technique based on naive Bayes classification to determine the domains relevant to a keyword query and rank those domains according to their relevance to the query. Our classifier takes into account the fact that the domains are probabilistic.
4. An experimental evaluation on schemas from a wide spectrum of domains.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 provides a more formal problem definition, while Section 4 provides an overview of our solution. Section 5 explains our schema clustering approach. Section 6 explains our approach for retrieving domains that are relevant to a keyword query. Our experimental evaluation is presented in Section 7, and we conclude in Section 8.

2. RELATED WORK

We use machine learning techniques from the domain of document clustering [2] to deal with schema clustering. When determining which domains are relevant to a query, we use naive Bayes classification, which is a machine learning technique that has been used extensively in document classification and other applications [15]. The idea of using probabilistic models to deal with uncertainty in data integration was considered in [6] and [7]. However, it was used in the context of probabilistic schema mediation and mapping, while the problem of clustering schemas into domains was left open. Our using of probabilistic models in clustering integrates seamlessly with [6] and [7]. Clustering schemas into domains has been considered in [12]. However, while [12] presents a customized algorithm that is designed to fit eight predefined domains (like cars and movies), our clustering approach is designed to handle any number of arbitrary and overlapping domains. Furthermore, the approach in [12] assumes that for each domain there are *anchor* attributes that do not occur except in that domain, while we do not rely on the existence of anchor attributes. Additionally, our approach deals with uncertainty in clustering via probabilistic models, and uses simple (yet effective) distance measures in clustering. Schema clustering is also mentioned in [13] as part of the proposed pay-as-you-go architecture, but without details on how to deal with the numerous challenges that arise when clustering web data sources. Finally, clustering is also used as a tool in another phase of data integration, namely schema mediation, where it is the attributes that are clustered not the schemas [1, 6, 18].

3. PROBLEM DEFINITION

Existing techniques of automatic data integration assume that the data sources to be integrated belong to the same domain. For these techniques to work on a large number of data sources from multiple domains, there has to be an initial step in which the data sources are clustered into domains. The objective of this paper is to automate the clustering step. Therefore, we consider the two problems of (1) clustering schemas into domains, and (2) retrieving and ranking relevant domains at query time. For the purpose of our research, we define the notion of a domain as follows:

DEFINITION 3.1 (DOMAIN). *A domain is a set of single-table schemas with sufficiently large intra-domain similarity and sufficiently large inter-domain dissimilarity, according to some measure of similarity.*

We also define a schema as a set of attribute names, and an attribute name as a set of terms (e.g., the attribute name 'First Name' consists of the terms 'First' and 'Name').

Our system takes as an input a set of single-table schemas, where each schema is extracted from a structured data source (e.g., a web form, an HTML table, a downloadable spreadsheet). We focus on single-table schemas since most data sources on the web belong to that category. Our approach works on schemas without assuming any access to the actual databases behind these schemas, so as to be general enough to handle deep web data sources without the need to surface them. Moreover, the only information we need to know about a schema is the attribute names, which is often the only information that is available. So, for example, attribute data types are not required. We also do not assume any prior information about the exact number or nature of potential domains. Consequently, domains need to be *discovered* from the available schemas. Our problem generally involves uncertainty in determining whether two schemas belong to the same domain or not. The output of the clustering phase is a set of domains, where each domain is a set of schemas as in Definition 3.1. In a typical pay-as-you-go system, each output domain will be fed as an input to a schema mediation and mapping algorithm. Schema mediation and mapping is already a well-studied problem and is not a focus of this paper, but we have to ensure that our solutions integrate well with previous work. At query time, we need to provide the user with the capability to retrieve domains relevant to a keyword query, taking in consideration that our domains are constructed with uncertainty. We define a keyword query as a set of terms. The query processing phase takes as an input a keyword query and a set of domains, and outputs for each domain its degree of relevance to that query.

4. SOLUTION OVERVIEW

We use *hierarchical agglomerative clustering* to group schemas into domains. This algorithm operates by iteratively merging similar schemas together into clusters and merging similar clusters together into larger clusters, until a maximum level of inter-cluster dissimilarity is reached [8]. Since textual similarity among attribute names is the basis upon which mediated schemas and schema mappings are usually generated, it is reasonable to rely on the same basis when measuring schema-to-schema similarity during schema clustering. Therefore, we assume that the probability that two schemas belong to the same domain can be determined based on the textual similarity between the attribute names of the two schemas. Previous empirical studies [11] have shown that attribute names within the same domain tend to be similar across different schemas. Moreover, relying only on attribute names makes it possible to apply our approach on data sources whose data and data types are not plainly exposed (e.g. the deep web). Our experiments on the schemas of hundreds of web data sources from diverse domains show that assigning probabilities based on textual similarities works well.

We handle uncertainty in schema clustering based on a probabilistic model. Besides being mathematically appropriate, using a probabilistic model is consistent with previous research that deals with uncertainty in pay-as-you-go data integration systems [6, 7]. The steps of constructing the probabilistic model and drawing inferences from it can be summarized as follows:

1. Each schema is represented by a feature vector, which we construct based on the terms extracted from the attribute names of the schema.
2. Hierarchical agglomerative clustering is applied to the feature vectors of the schemas to group them into domains.
3. Schemas that have equal or close similarities to multiple domains are assigned to each of these domains with different probabilities. The probabilities are based on the similarities between schemas and domains.
4. When a user poses a keyword query over the system, naive Bayes classification is used to determine, for each

domain, the probability that the query belongs to that domain. Relevant domains are then ranked based on probability values.

Between Steps 3 and 4, existing techniques from previous research can be used to generate a mediated schema for each domain and then generate probabilistic mappings from the schemas in the domain to the domain's mediated schema. The generated schema mappings are also probabilistic so as to handle the uncertainty in determining which attributes in a source schema correspond to which attributes in the mediated schema [6]. A probabilistic mapping from a source schema to a mediated schema is basically a set of possible mappings, each assigned a probability.

The probability assigned to any individual tuple retrieved from a domain at query time is the product of two probabilities: (1) the probability that the schema from which that tuple is retrieved belongs to that domain (obtained from our work), and (2) the probability that the schema mapping based on which the tuple was mapped to the mediated schema of the domain is the correct mapping (obtained from the probabilistic schema mapping technique, e.g., [6]).

5. SCHEMA CLUSTERING

Given a set of schemas $S = \{S_1, S_2, \ldots, S_{|S|}\}$ as input, our target is to output a set of clusters $C = \{C_1, C_2, \ldots, C_{|C|}\}$, where $C_r \subseteq S$, for all $C_r \in C$. Optimally, for all $i, j = 1, 2, \ldots, |S|$, and for all $r = 1, 2, \ldots, |C|$, the two schemas S_i and S_j should belong to C_r if and only if S_i and S_j represent the same real-world domain. However, we have no means by which we can determine automatically and with absolute certainty whether any two given schemas represent the same domain or not. We have to rely on approximate methods and accept best-effort results, which is an essential aspect of the pay-as-you-go approach. We assume that the probability that two schemas belong to the same domain can be determined based on the textual similarity between the attribute names of the two schemas.

5.1 Creating Feature Vectors

Before proceeding with clustering, we need to characterize each schema with a feature vector. Feature vectors are needed both during the clustering process and during query classification. We use a vector space model similar to that used in document clustering [2]; that is, if there are d distinct terms in all given schemas, we characterize each schema with a vector comprised of d binary features, one feature for each distinct term to indicate whether this term exists in the schema or not. We use binary features instead of, for example, counting the frequency of terms in schemas, because schema attributes usually contain a few terms, so binary features are sufficient.

Algorithm 1 describes how feature vectors are created. First, for each schema $S_i \in S$, we extract all the terms from S_i by splitting its attribute names over a set of pre-defined delimiters, like white spaces, slashes and underscores. For example, given the following schema {Class_ID,Day/Time, Professor Name, Subject}, the set of extracted terms will be {Class, ID, Day, Time, Professor, Name, Subject}. We also split attribute names that consist of several capital-started terms concatenated to each others (e.g. 'MaxNumberOfStudents' is split into 'Max', 'Number', 'Of' and 'Students'). Splitting attribute names is motivated by the observation that individual terms within the attributes names of schemas in a single domain can cluster together better than the whole attribute names; since they tend to be less sensitive to rephrasing (e.g., 'Professor Name' versus 'Name of the Professor'). We convert all terms to a canonical form for better comparisons (e.g. all characters to lower case), then we remove stop words and extremely short terms (e.g. terms with less than three letters). The result is the set $T = \{T_1, T_2, \ldots, T_{|T|}\}$, where T_i is the set of terms extracted from the schema S_i.

Algorithm 1 Create Feature Vectors

```
1: procedure CreateFeatureVectors
2:     input: Set of schemas S = {S_1, S_2, ..., S_|S|}
3:     for all S_i ∈ S do
4:         Define the set of terms T_i
5:         Extract all terms from S_i's attribute names to T_i
6:         Convert all terms in T_i into a canonical form
7:         Remove very small terms and stop words from T_i
8:     end for
9:     Sort all terms in ∪_{i=1}^{|S|} T_i into a vector L
10:    for all S_i ∈ S do
11:        Define a binary vector F^i, where dim F^i = dim L
12:        for all terms L_j in L do
13:            if max_{t∈T_i} t_sim(L_j, t) ≥ τ_{t_sim} then
14:                F^i_j ← 1
15:            else
16:                F^i_j ← 0
17:            end if
18:        end for
19:    end for
20:    return F = {F^1, F^2, ..., F^|S|}
21: end procedure
```

Next, all terms in $\cup_{i=1}^{|T|} T_i$ are sorted into a vector of terms $L =< L_1, L_2, \ldots, L_{dim\ L} >$, where $dim\ L = |\cup_{i=1}^{|T|} T_i|$. We then create, for each $S_i \in S$, a binary feature vector F^i, such that $dim\ F^i = dim\ L$. Let F^i_j denote the j^{th} feature in F^i. The vector F^i characterizes S_i by indicating, for each term L_j in L whether S_i contains a term that is *sufficiently similar* to L_j or not; if yes then $F^i_j = 1$, otherwise $F^i_j = 0$.

For each $S_i \in S$, F^i is computed as follows. Let t_sim be a function that takes two terms t and t' as input and returns a real value in the range $[0, 1]$ that indicates how similar the two terms are. For each term L_j in L, we compute $\max_{t \in T_i} t\_sim(L_j, t)$; that is, the maximum among all the similarities between L_j and each of the terms in S_i. We then compare that maximum to a threshold τ_{t_sim} that we set based on our knowledge of the similarity function t_sim. If $\max_{t \in T_i} t\_sim(L_j, t) \geq \tau_{t\_sim}$ then $F^i_j = 1$, otherwise $F^i_j = 0$.

There are already several well-studied functions for measuring term similarity [5]. In our work, we use a function that is based on the longest common substring. Let the function $LCS(t_i, t_j)$ denote the longest common substring between the two terms t_i and t_j, and the function $len(t)$ denote the number of characters in the term t; then

$$t\_sim(t_i, t_j) = \frac{2.len(LCS(t_i, t_j))}{len(t_i) + len(t_j)}$$

That is, the length of the longest common substring divided

by the average of the lengths of the two terms. We pick a high value for τ_{t_sim}, for example 0.8, to ensure sufficient similarity. The longest common substring can be computed efficiently in linear time using suffix trees [10]. Another possible alternative for the term similarity function t_sim is to use a function that recognizes two terms to be similar if and only if they have the same stem.

5.2 Clustering Algorithm

We use hierarchical agglomerative clustering as described in Algorithm 2. We choose hierarchical clustering because we do not know the appropriate number of clusters in advance, and hierarchical clustering does not require prior knowledge of this number.

Algorithm 2 Cluster Schema

procedure ClusterSchema
  input: Set of schemas $S = \{S_1, S_2, \ldots, S_{|S|}\}$
  $k \leftarrow 1$
  $U^{(k)} \leftarrow \{\{S_1\}, \{S_2\}, \ldots, \{S_{|S|}\}\}$
  Let $(U_a^{(k)}, U_b^{(k)})$ be:
    $\arg\max_{(U_i^{(k)}, U_j^{(k)}) \in U^{(k)} \times U^{(k)};\ i \neq j} c\_sim(U_i^{(k)}, U_j^{(k)})$
  while $c\_sim(U_a^{(k)}, U_b^{(k)}) \geq \tau_{c\_sim}$ **do**
    $U_{ab}^{(k+1)} \leftarrow U_a^{(k)} \cup U_b^{(k)}$
    $U^{(k+1)} \leftarrow (U^{(k)} \setminus \{U_a^{(k)}, U_b^{(k)}\}) \cup \{U_{ab}^{(k+1)}\}$
    $k \leftarrow k + 1$
    $U^{(k)} \leftarrow \{\{S_1\}, \{S_2\}, \ldots, \{S_{|S|}\}\}$
    Let $(U_a^{(k)}, U_b^{(k)})$ be:
      $\arg\max_{(U_i^{(k)}, U_j^{(k)}) \in U^{(k)} \times U^{(k)};\ i \neq j} c\_sim(U_i^{(k)}, U_j^{(k)})$
  end while
  return $C = U^{(k)}$
end procedure

First, we measure the similarity between every two schemas by measuring the similarity between their feature vectors. Let the function $s\_sim(S_i, S_j)$ be the similarity measure between the two schemas S_i and S_j, where $1 \leq i, j \leq |S|$. We use the Jaccard coefficient as a similarity measure since it is known to be suitable for high dimensional binary feature vectors [17]. Thus,

$$s\_sim(S_i, S_j) = Jaccard(F^i, F^j) = \frac{|\{r : F_r^i = 1 \text{ and } F_r^j = 1\}|}{|\{r : F_r^i = 1 \text{ or } F_r^j = 1\}|}$$

All schema-to-schema similarities should be computed and memoized (i.e., cached) in advance so as to avoid recomputing them multiple times during clustering.

Next, we proceed to clustering. Initially, every schema is considered a singleton cluster in its own right. Then agglomerative hierarchical clustering operates iteratively by merging the *most similar* pair of clusters among the set of available clusters into one new cluster, based on some measure of cluster similarity. At the beginning of each iteration k, we denote the set of clusters that we have as $U^{(k)}$. Since we start by placing every schema in a singleton cluster, $U^{(1)} = \{\{S_1\}, \{S_2\}, \ldots, \{S_{|S|}\}\}$. After each iteration k, the number of clusters shrinks by one as we merge the two closest (most similar) clusters into one new cluster, i.e., $|U^{(k+1)}| = |U^{(k)}| - 1$. We define the similarity between any two clusters $U_i^{(k)}$ and $U_j^{(k)}$ as follows:

$$c\_sim(U_i^{(k)}, U_j^{(k)}) = \frac{1}{|U_i^{(k)}||U_j^{(k)}|} \sum_{S_a \in U_i^{(k)}} \sum_{S_b \in U_j^{(k)}} s\_sim(S_a, S_b)$$

That is, the average of the similarities between every schema in $U_i^{(k)}$ and every schema in $U_j^{(k)}$. We show in all our experiments (Section 7.2) that other cluster similarity measures can be also used and give similar results.

For each iteration k, let the closest pair of clusters be $U_a^{(k)}$ and $U_b^{(k)}$, then the new (merged) cluster will be the union of $U_a^{(k)}$ and $U_b^{(k)}$; that is, $U_{ab}^{(k+1)} = U_a^{(k)} \cup U_b^{(k)}$. For every other cluster $U_c^{(k)} \in U^{(k)} \setminus \{U_a^{(k)}, U_b^{(k)}\}$, $U_c^{(k)}$ remains the same in $U^{(k+1)}$; that is, $U_c^{(k+1)} = U_c^{(k)}$. Thus we have

$$U^{(k+1)} = (U^{(k)} \setminus \{U_a^{(k)}, U_b^{(k)}\}) \cup \{U_{ab}^{(k+1)}\}$$

For every pair of clusters in $U^{(k+1)}$ not including $U_{ab}^{(k+1)}$, inter-cluster similarities remain the same as they were in the previous iteration, so there is no need to recompute them. For $U_{ab}^{(k+1)}$, we can compute its similarity to every other cluster $U_c^{(k+1)} \in U^{(k+1)} \setminus \{U_{ab}^{(k+1)}\}$ in a constant amount of time by utilizing the memoized values from the previous iteration as follows:

$$c\_sim(U_c^{(k+1)}, U_{ab}^{(k+1)}) = \frac{|U_a^{(k)}|.c\_sim(U_c^{(k)}, U_a^{(k)}) + |U_b^{(k)}|.c\_sim(U_c^{(k)}, U_b^{(k)})}{|U_a^{(k)}| + |U_b^{(k)}|}$$

Thus, our memoization can be updated in $O(|U^{(k+1)}|)$ running time.

Clustering stops when the most similar pair of clusters $(U_a^{(k)}, U_b^{(k)})$ is not similar enough; that is, $c\_sim(U_a^{(k)}, U_b^{(k)}) < \tau_{c\_sim}$, where τ_{c_sim} is a pre-defined threshold. Our experiments in Section 7.2 elaborate on the choice of τ_{c_sim}. Let the last set of clusters produced before the algorithm stops be $C = \{C_1, C_2, \ldots, C_{|C|}\}$. The set C is the output of our clustering algorithm.

5.3 Assigning Probabilities

The main source of uncertainty in schema clustering is the schemas that lie on the boundaries between clusters. Actually, in some cases, assigning these boundary schemas to clusters is arbitrary. For example, consider the case when agglomerative clustering is running and there exist three clusters $U_1^{(k)}$, $U_2^{(k)}$ and $U_3^{(k)}$. It is possible to have $c\_sim(U_1^{(k)}, U_2^{(k)}) = c\_sim(U_1^{(k)}, U_3^{(k)}) \geq \tau_{c\_sim}$. If no other pair of clusters is as similar as $(U_1^{(k)}, U_2^{(k)})$ and $(U_1^{(k)}, U_3^{(k)})$, then either $U_2^{(k)}$ or $U_3^{(k)}$ will be merged with $U_1^{(k)}$. The choice will typically be arbitrary. Other possible sources of uncertainty include cases of very small differences between $c\_sim(U_1^{(k)}, U_2^{(k)})$ and $c\_sim(U_1^{(k)}, U_3^{(k)})$. Thus, we consider assigning a single schema to multiple domains with different probabilities if it has sufficient similarity to all of them. Algorithm 3 explains how these probabilities are assigned.

Since we are going to assign some schemas to multiple domains, while each schema in S belongs to one and only one cluster in C, we need to separate the concept of *domains* from the concept of *clusters*. We use the notion of clusters to

Algorithm 3 Assign Probabilities

procedure AssignProbabilities
  input: Set of schemas $S = \{S_1, S_2, \ldots, S_{|S|}\}$
  input: Set of clusters $C = \{C_1, C_2, \ldots, C_{|C|}\}$
  for all $C_r \in C$ **do**
    Define a domain D_r
  end for
  for all $S_i \in S$ **do**
    for all $C_r \in C$ **do**
      if $s\_c\_sim(S_i, C_r) \geq \tau_{c\_sim}$ **and** $\frac{s\_c\_sim(S_i, C_r)}{\max_{C_j \in C} s\_c\_sim(S_i, C_j)} \geq 1 - \theta$ **then**
        $Pr(S_i \in D_r) \leftarrow \frac{s\_c\_sim(S_i, C_r)}{\sum_{D_j \in D(S_i)} s\_c\_sim(S_i, C_j)}$
      else
        $Pr(S_i \in D_r) \leftarrow 0$
      end if
    end for
  end for
  return $\{(S_i, D_r, Pr(S_i \in D_r)) : for\ all\ S_i\ and\ D_r\}$
end procedure

refer to sets of schemas that partition S, like those returned by Algorithm 2. We use the notion of domains to refer to sets of schemas too; however, every schema in S may belong to multiple domains with different probabilities.

We construct domains from the clusters returned by Algorithm 2 as follows. First, we consider the existence of a cluster as an indicator of the existence of a domain, so the number of domains equals the number of clusters. Let the set of domains be $D = \{D_1, D_2, \ldots, D_{|D|}\}$, where $|D| = |C|$, and each domain $D_r \in D$ corresponds to a cluster $C_r \in C$, for all r. We then examine every schema $S_i \in S$; if S_i is sufficiently similar to multiple clusters then we assign S_i to the domains corresponding to these clusters with different probabilities based on the similarities between S_i and each of these clusters.

The similarity between a schema $S_i \in S$ and a cluster $C_r \in C$ is measured as follows:

$$s\_c\_sim(S_i, C_r) = \frac{1}{|C_r|} \sum_{S_j \in C_r} s\_sim(S_i, S_j)$$

That is, the average of the schema similarities between S_i and all the schemas in C_r. For any schema S_i to be assigned to any domain D_r, two conditions must be satisfied. First, the value of $s\_c\_sim(S_i, C_r)$ must be at least τ_{c_sim}. Second, we require that the ratio between $s\_c\_sim(S_i, C_r)$ and the maximum similarity between S_i and any other cluster be sufficiently large; that is, $\frac{s\_c\_sim(S_i, C_r)}{\max_{C_j \in C} s\_c\_sim(S_i, C_j)} \geq 1 - \theta$, for some $\theta \in [0, 1]$. The threshold θ quantifies the degree of uncertainty allowed when assigning schemas to domains; a higher θ means higher uncertainty. In our experiments, we set $\theta = 0.02$.

For each schema $S_i \in S$, let $D(S_i) = \{D_r : s\_c\_sim(S_i, C_r) \geq \tau_{c\_sim}\ and\ \frac{s\_c\_sim(S_i, C_r)}{\max_{C_j \in C} s\_c\_sim(S_i, C_j)} \geq 1 - \theta\}$. Also, for each domain $D_r \in D$, let $S(D_r) = \{S_i : D_r \in D(S_i)\}$. For all $S_i \notin S(D_r)$, $Pr(S_i \in D_r) = 0$. Otherwise, for all $S_i \in S(D_r)$, the probability that S_i belongs to D_r is estimated as the schema-to-cluster similarity between S_i and the C_r, normalized so that all the probabilities assigned to S_i sum up to 1. That is,

$$Pr(S_i \in D_r) = \begin{cases} \frac{s\_c\_sim(S_i, C_r)}{\sum_{D_j \in D(S_i)} s\_c\_sim(S_i, C_j)} & ;\ if\ S_i \in S(D_r) \\ 0 & ;\ otherwise \end{cases}$$

The output of this phase is the set of triples $\{(S_i, D_r, Pr(S_i \in D_r)) : for\ all\ S_i \in S\ and\ D_r \in D\}$. Triples with $Pr(S_i \in D_r) = 0$ do not need to be represented explicitly.

After schema clustering, a fully automatic schema mediation and mapping technique (e.g., [6]) can be run to generate a probabilistic mediated schema for each domain D_r, and generate probabilistic mappings from each schema $S_i \in S(D_r)$ to the mediated schema of D_r.

6. QUERY CLASSIFICATION

In this section we investigate the issue of answering keyword queries posed over our multi-domain data integration system by retrieving and ranking relevant domains. We use a naive Bayes classifier to determine the probability that a keyword query belongs to any of the domains that are constructed during the clustering phase. For the classifier to do that, some of the keywords in the query need to be similar to some attribute names in the relevant domains. The design of our classifier ensures that expensive operations are performed at system setup time rather than query time.

At query time, we create a feature vector to characterize the keyword query in the same manner as we did for every schema in S in Section 5.1. Let Q denote the set of keywords in the keyword query entered by the user. We convert all keywords into our canonical form, and remove stop words and extremely small keywords. The result is the set of terms T_Q. We then construct the binary feature vector F^Q. Let F^Q_j be the j^{th} feature in F^Q, and L_j be the j^{th} term in the vector L that contains all terms in all schemas. We set the value of the feature F^Q_j to 1 if there exists a term in T_Q that is sufficiently similar to L_j; that is, if $\max_{t \in T_Q} t\_sim(L_j, t) \geq \tau_{t\_sim}$. Otherwise, $F^Q_j = 0$.

Our target is to determine the posterior probability for each domain D_r; that is, given F^Q, the probability that Q belongs to D_r. Let us denote that probability as $Pr(D_r|F^Q)$. According to Bayes' rule:

$$Pr(D_r|F^Q) = \frac{Pr(F^Q|D_r)Pr(D_r)}{Pr(F^Q)} \quad (1)$$

$Pr(F^Q|D_r)$ is the probability that an arbitrary schema S_{rand}, randomly chosen from the domain D_r, has a feature vector equal to F^Q. $Pr(D_r)$ is the probability that an arbitrary schema S_{rand}, randomly chosen from S, belongs to D_r. $Pr(F^Q)$ is the probability that an arbitrary schema S_{rand}, randomly chosen from S, has a feature vector equal to F^Q.

Based on application context, we may assign Q to the domain that has the maximum posterior probability (i.e., $\arg\max_{D_r} Pr(D_r|F^Q)$), or we may return a list of relevant

domains ranked by posterior probabilities. Note that, in Equation 1, we do not need to compute $Pr(F^Q)$ since it is constant for all $D_r \in D$ and thus it does not affect the relative order of posterior probabilities. Consequently, for each domain D_r, we only need to compute $Pr(F^Q|D_r)Pr(D_r)$.

We make the fundamental assumption of the naive Bayes classifier, which is the assumption that all features are conditionally independent given the domain. Consequently,

$$Pr(F^Q|D_r)Pr(D_r) \;=\; Pr(D_r) \prod_{j=1}^{dim\ L} Pr(F_j^Q|D_r) \qquad (2)$$

where $Pr(F_j^Q|D_r)$ is the probability that an arbitrary schema S_{rand}, randomly chosen from the domain D_r, has its j^{th} feature F_j^{rand} equal to F_j^Q.

The values of $Pr(D_r)$ and $Pr(F_j^Q|D_r)$, for all j, depend on which schemas are assigned to which domains. This assignment is determined based on another probability distribution as described in Section 5.3. Therefore, $Pr(D_r)$ can be expressed in terms of the following summation over all subsets of $S(D_r)$:

$$Pr(D_r) \;=\; \sum_{S' \subseteq S(D_r)} Pr(D_r|D_r = S')Pr(D_r = S') \qquad (3)$$

Similarly, for all j,

$$Pr(F_j^Q|D_r) = \sum_{S' \subseteq S(D_r)} Pr(F_j^Q|D_r = S', D_r)\ Pr(D_r = S'|D_r) \qquad (4)$$

We analyze the probabilities on the right-hand sides of Equations 3 and 4 as follows. First, the probability $Pr(D_r|D_r = S')$ is estimated as:

$$Pr(D_r|D_r = S') = \frac{|S'|}{|S|} \qquad (5)$$

Second, $Pr(D_r = S')$ can be expressed as the probability of the following conjunction:

$$Pr(D_r = S') = Pr\left(\bigwedge_{S_i \in S'} S_i \in D_r \bigwedge_{S_i \notin S'} S_i \notin D_r\right)$$

We make a second simplifying assumption by assuming that the assignments of schemas to domains are statistically independent. Consequently,

$$Pr(D_r = S') = \prod_{S_i \in S'} Pr(S_i \in D_r) \prod_{S_i \notin S'} Pr(S_i \notin D_r) \qquad (6)$$

Third, $Pr(F_j^Q|D_r = S', D_r)$ is the probability that the j^{th} feature of an arbitrary schema S_{rand}, randomly chosen from S', equals F_j^Q. This probability is estimated as follows:

$$Pr(F_j^Q|D_r = S', D_r) = \frac{|\{S_i : Si \in S'\ and\ F_j^i = F_j^Q\}|}{|S'|}$$

Since all features are binary, the last equation can be rewritten as follows:

$$Pr(F_j^Q|D_r = S', D_r) = \begin{cases} \dfrac{\sum_{S_i \in S'} F_j^i}{|S'|} & if\ F_j^Q = 1 \\ 1 - \dfrac{\sum_{S_i \in S'} F_j^i}{|S'|} & if\ F_j^Q = 0 \end{cases} \qquad (7)$$

However, there are two problems with Equation 7. The first problem is that $|S'|$ can be zero; if, for all $S_i \in S(D_r)$, $Pr(S_i \in D_r) \neq 1$, then there is a non-zero probability that D_r is empty, so $|S'|$ may be zero. The second problem is related to robustness. If a query has an *extra term* (i.e., a term that does not exist in any of the schemas in $S(D_r)$), then no matter how many other terms are common between the query and the schemas in $S(D_r)$, the probability that the query belongs to D_r will be zero. To see that, assume that $F_j^i = 0$ for all $S_i \in S(D_r)$. Then, according to Equation 7 the probability $Pr(F_j^Q|D_r = S', D_r)$ will be zero, for all $S' \subseteq S(D_r)$. Substituting $Pr(F_j^Q|D_r = S', D_r)$ in Equation 4, the probability $Pr(F_j^Q|D_r)$ will also be zero. Eventually, by substituting $Pr(F_j^Q|D_r)$ in Equation 2, the posterior probability will be zero. Similarly, it is easy to see that, if a query has a *missing term* (i.e., a term that exists in all the schemas in $S(D_r)$ but not in the query), then no matter how many other terms are common between the query and the schemas in $S(D_r)$, the probability that the query belongs to D_r will be zero. To solve these two problems we use the *m-estimate* of probabilities [3]. Basically, for each domain D_r, and for each subset $S' \subseteq S(D_r)$, we act as if S' has m additional schemas, some of them have all their features set to 1, while the others have all their features set to 0. Consequently, Equation 7 can be rewritten as follows:

$$Pr(F_j^Q|D_r = S', D_r) = \begin{cases} \dfrac{\sum_{S_i \in S'} F_j^i + p.m}{|S'| + m} & if\ F_j^Q = 1 \\ 1 - \dfrac{\sum_{S_i \in S'} F_j^i + p.m}{|S'| + m} & if\ F_j^Q = 0 \end{cases} \qquad (8)$$

where $p \in (0, 1)$ is the fraction of additional schemas that have all their features set to 1. A typical choice would be to set $p = 0.5$ so as to give the classifier no bias towards either extra terms or missing terms. However, we need to consider the fact that keyword queries are usually short. A typical keyword query will contain a small subset of the terms in the schemas of $S(D_r)$, plus a small number of extra terms, so it is much more likely to have missing terms than extra terms. We set $m = 1 + |S'|$ and $p = 1/dim\ L$, which gives stronger bias towards missing terms.

Finally, the probability $Pr(D_r = S'|D_r)$ can be computed using Bayes' rule as follows:

$$Pr(D_r = S'|D_r) = \frac{Pr(D_r|D_r = S')Pr(D_r = S')}{Pr(D_r)} \qquad (9)$$

Note that the probabilities on the right-hand side of Equation 9 are already computed as part of Equation 3. By substituting Equations 5, 6, 8 and 9 into Equations 3 and 4, and then substituting Equations 3 and 4 into Equation 2, we obtain the posterior probabilities required to rank domains.

Bayesian classification is expensive, but all of the expensive operations in our case can be done at setup time rather than query time. Equations 5, 6 and 9 are all independent of the user's query. Equation 8 depends on the value of the query feature F_j^Q, but since F_j^Q may assume one of only two values (either 0 or 1) we can still compute Equation 8 at setup time for both $F_j^Q = 0$ and $F_j^Q = 1$. Therefore, all the probabilities used on the right-hand side of Equation 2 can be pre-computed and stored at setup time. At query time, calculating Equation 2 for all domains takes $O(|D|\ dim\ L)$ running time per query.

To analyze the setup time needed for Equation 2 let us first define the set of *uncertain schemas* for each domain D_r as $\hat{S}(D_r) = \{S_i : S_i \in S(D_r)$ *and* $Pr(S_i \in D_r) \neq 1\}$. $\hat{S}(D_r)$ is the set of all schemas that belong to D_r with probabilities strictly smaller than 1 and strictly greater than 0. We will also use the term *certain schemas* to refer to schemas that belong to D_r with probability 1; that is, $S(D_r) \backslash \hat{S}(D_r)$. Any subset $S' \subseteq S(D_r)$ may include or exclude any uncertain schemas and still maintain a non-zero probability; that is, $Pr(D_r = S') \neq 0$. However, if any certain schema is excluded from S', then $Pr(D_r = S')$ will be zero according to Equation 6, and consequently S' will not contribute to the summation in Equation 3. Additionally, by substituting $Pr(D_r = S')$ into Equation 9, $Pr(D_r = S'|D_r)$ will also be zero, and again S' will not contribute to the summation in Equation 4. Therefore, when computing the summations in Equations 9 and 4, we only need to consider the subsets that contain all the certain schemas in $S(D_r)$. This prunes the number of subsets to be considered from $2^{|S(D_r)|}$ to $2^{|\hat{S}(D_r)|}$. By memoizing intermediate values, we can calculate the probabilities in Equation 2 for all domains at setup time in $O(\max_{D_r \in D}\{|\hat{S}(D_r)|\ 2^{|\hat{S}(D_r)|}\}\ |D|\ dim\ L\ +\ |S|\ dim\ L)$ running time. The growth rate of setup time is dominated by the need to enumerate all possible combinations of uncertain schemas for each domain. Thus, the time to construct the classifier depends on the number of uncertain schemas much more strongly than the total number of schemas. Uncertain schemas are schemas that lie on the boundaries between domains; that is, schemas with close similarities to multiple domains. Typically, they should not constitute significant portions of their domains. Whenever necessary, the number of uncertain schemas can be decreased by decreasing the parameter θ (Section 5.3).

7. EXPERIMENTS

7.1 Experimental Setup

We implement our algorithms in C++ and use this prototype to evaluate the effectiveness of our schema clustering and query classification techniques. We run our experiments on a Windows Vista machine, with Intel Centrino Duo 2GHz processor and 3GB RAM. The goal of our experiments is to demonstrate that our techniques can correctly cluster schemas of data sources available on the web into domains, and can classify keyword queries into appropriate domains. For these experiments we need schemas of web data sources labeled with their correct domains, and we need queries that are also labeled with domains. Generating and labeling schemas and queries in a meaningful way in itself poses some interesting challenges, which we describe next.

7.1.1 Schemas Used

We use the schema set used in [6], which we obtained from the authors of that paper. That schema set consists of 2323 schemas from 5 different domains that were extracted from Google's web index, and we refer to it as the DDH set after the initials of the authors of [6]. The domains in DDH are few and sharply separated, and thus are expected to lend themselves perfectly to clustering. Data sources on the web are not restricted to a small number of well-defined domains, but rather come from extremely diverse and overlapping domains. To test our schema clustering and query classification methods on such diverse and overlapping domains, we collect our own schema sets manually through web search and through lists of hidden web data sources that are available on-line (e.g., in Wikipedia). We extracted two sets of schemas from two types of data sources. The first schema set, which we refer to as DW, is extracted from deep web data sources. For that schema set, we find the web form interfaces to deep web data sources, and manually extract the attribute names in each form. These attribute names form the schema of the data source. The second schema set, which we refer to as SS, is extracted from downloadable spreadsheets that we obtained using Google's "search by file type" option. The schema of a spreadsheet consists of the manually extracted headers of the columns in the spreadsheet. The attribute names in DW schemas tend to be phrased in a better way and are more accurately indicative of the domain than the ones in SS schemas. In both schema sets, around 25% of the schemas are unique in the sense that a human would not cluster any of them with any other schemas in their sets. These unique schemas are expected to remain unclustered after the clustering algorithm terminates.

7.1.2 Evaluating Schema Clustering

To evaluate the effectiveness of schema clustering, a typical approach would be to manually assign a label to each schema indicating its domain, and then to measure the effectiveness of the clustering algorithm at grouping schemas with the same domain label. This approach works well for the DDH schema set since the domains are sharply separated, but it does not work well for DW and SS. For DW and SS, the boundaries between different domains are not always obvious, and a single schema may be correctly classified into several domains. The following example illustrates this problem.

EXAMPLE 7.1. *Consider the following two schemas, extracted from two different data sources, both providing information about faculty members:*
S_1:(faculty member, office phone, email, fax)
S_2:(name, position, affiliation, research interests)
Although both schemas are concerned with faculty members, they provide different types of information. In principle, S_1 and S_2 should be clustered together since a user looking for information about faculty members may find both of them useful. However, considering the fact that the objective of clustering in our case is to serve as a preliminary step before schema mediation and mapping, clustering S_1 and S_2 together may not be so useful since the two schemas together do not provide a good input for schema mediation and mapping algorithms. The question is: If the clustering algorithm clusters S_1 and S_2 together, is that a false positive? If it does not cluster them together, is that a false negative?

In order to deal with this problem, we perform our experiments as follows. For the two schema sets DW and SS, we manually associate each schema S_i with a set of labels $B(S_i)$ according to what we perceive as potential domains for S_i. Example domain labels that we use include 'movies', 'bibliography', and 'people'. Each schema is labeled with at least one label. Table 1 provides detailed statistics about the labels used for DW, SS, and their union. These numbers indicate that a few labels have the majority of schemas as-

	DW	SS	Both
Number of Schemas	63	252	315
Max. terms per schema	72	119	119
Avg. terms per schema	14	12.4	12.8
Number of labels used	24	85	98
Max. labels per schema	2	4	4
Avg. labels per schema	1	1.5	1.4
Max. schemas per label	13	67	67
Avg. schemas per label	2.8	4.4	4.5

Table 1: Statistics about schema sets.

sociated with them, while the majority of labels have a few schemas.

Let the set of all labels used be $B = \cup_{i=1}^{|S|} B(S_i) = \{B_1, B_2, \ldots, B_{|B|}\}$. Also, let $S(B_j)$ denote the set of all schemas labeled with B_j. We run the clustering algorithm on our schema sets and examine the set of domains D that is produced by the clustering algorithm, and we label each domain $D_r \in D$ with the set of *dominant labels* within that domain. Usually there is only one dominant label but sometimes there are several labels that equally dominate the domain. Let $B(D_r)$ denote the set of dominant labels in the domain D_r. Also, let $D(B_j)$ denote the set of domains dominated by B_j. We determine dominant labels as follows:

$$B(D_r) = \arg\max_{B_j \in B} \sum_{S_i \in S(B_j)} Pr(S_i \in D_r)$$

Summing the probabilities should be interpreted as a *weighted counting* of the schemas in D_r and is not intended to have a probabilistic meaning. We also sum probabilities as a weighted counting of schemas when estimating precision and recall. If more than one label equally dominate the domain, we include them all in $B(D_r)$.

A special case is when the dominant label of D_r does not have an absolute majority; that is,

$$\max_{B_j \in B} \sum_{S_i \in S(B_j)} Pr(S_i \in D_r) < \frac{1}{2} \sum_{S_i \in S} Pr(S_i \in D_r)$$

We then call D_r a *non-homogeneous domain*. A non-homogeneous domain is treated as if it has no dominant label; i.e. $B(D_r) = \phi$. When computing precision and recall, schemas assigned to non-homogeneous domains are all counted as false negatives. We also compute the fraction of schemas assigned to non-homogeneous domains as one of our performance measures. Another special case is a domain with only one schema. That happens when a schema is not sufficiently similar to any other schemas in S, given the value used for the threshold τ_{c_sim}. We measure the fraction of *unclustered schemas*, and exclude them from other performance measurements like precision and recall. One last case to be considered is when two different domains are dominated by the same label; i.e. $B(D_a) \cap B(D_b) \neq \phi$ where $a \neq b$. We use the term *fragmentation* to refer to that case and we measure the degree of fragmentation in our experiments by computing the average number of domains dominated by each label; that is, $\frac{1}{|B|} \sum_{B_j \in B} |D(B_j)|$. Finally, we measure *precision* and *recall* as follows.

Precision: For each schema $S_i \in S(D_r)$, if $B(S_i) \cap B(D_r) \neq \phi$ then S_i contributes to the true positives of D_r, denoted as TP_{D_r}, by the probability of membership of S_i in D_r.

$$TP_{D_r} = \sum_{S_i : B(S_i) \cap B(D_r) \neq \phi} Pr(S_i \in D_r)$$

Similarly, the false positives of D_r, denoted as FP_{D_r}, are estimated as

$$FP_{D_r} = \sum_{S_i : B(S_i) \cap B(D_r) = \phi} Pr(S_i \in D_r)$$

We therefore estimate the average precision as

$$\frac{1}{|D|} \sum_{D_r \in D} \frac{TP_{D_r}}{TP_{D_r} + FP_{D_r}}$$

Recall: For each domain D_r, if $B_j \in B(D_r)$ and there exists $S_i \in S(D_r)$ such that $B_j \in B(S_i)$, then S_i contributes to the true positives of B_j, denoted as TP_{B_j}, by the probability of membership of S_i in D_r; that is,

$$TP_{B_j} = \sum_{D_r \in D(B_j)} \sum_{S_i \in S(B_j)} Pr(S_i \in D_r)$$

Similarly, the false negatives of B_j, denoted as FN_{B_j}, are estimated as

$$FN_{B_j} = \sum_{D_r \notin D(B_j)} \sum_{S_i \in S(B_j)} Pr(S_i \in D_r)$$

We therefore estimate the average recall as

$$\frac{1}{|B|} \sum_{B_j \in B} \frac{TP_{B_j}}{TP_{B_j} + FN_{B_j}}$$

To evaluate our clustering approach, we measure the effect of changing t_{c_sim} on the performance measures like precision, recall, fragmentation, unclustered schemas, and non-homogeneous domains. We also compare the performance of our clustering algorithm when other cluster-to-cluster similarity measures are used instead of the average Jaccard similarity that is described in Section 5.2. The other alternatives we consider for cluster-to-cluster similarity are Min. Jaccard, Max. Jaccard, and Total Jaccard. These three similarity measures can be defined as follows. Let $U_i^{(k)}$ and $U_j^{(k)}$ be two clusters at a given iteration k.

Min. Jaccard: The minimum of the Jaccard similarities between every schema in $U_i^{(k)}$ and every schema in $U_j^{(k)}$.

$$c\_sim(U_i^{(k)}, U_j^{(k)}) = \min_{S_a \in U_i^{(k)}, S_b \in U_j^{(k)}} s\_sim(S_a, S_b)$$

Max. Jaccard: The maximum of the Jaccard similarities between every schema in $U_i^{(k)}$ and every schema in $U_j^{(k)}$.

$$c\_sim(U_i^{(k)}, U_j^{(k)}) = \max_{S_a \in U_i^{(k)}, S_b \in U_j^{(k)}} s\_sim(S_a, S_b)$$

Total Jaccard: The number of terms common between all the schemas in $U_i^{(k)}$ and $U_j^{(k)}$ divided by the number of all terms that exist in any of the schemas in $U_i^{(k)}$ or $U_j^{(k)}$.

$$c\_sim(U_i^{(k)}, U_j^{(k)}) = \frac{|\{l : \bigwedge_{S_a \in U_i^{(k)}} F_l^a = 1 \bigwedge_{S_b \in U_j^{(k)}} F_l^b = 1\}|}{|\{l : \bigvee_{S_a \in U_i^{(k)}} F_l^a = 1 \bigvee_{S_b \in U_j^{(k)}} F_l^b = 1\}|}$$

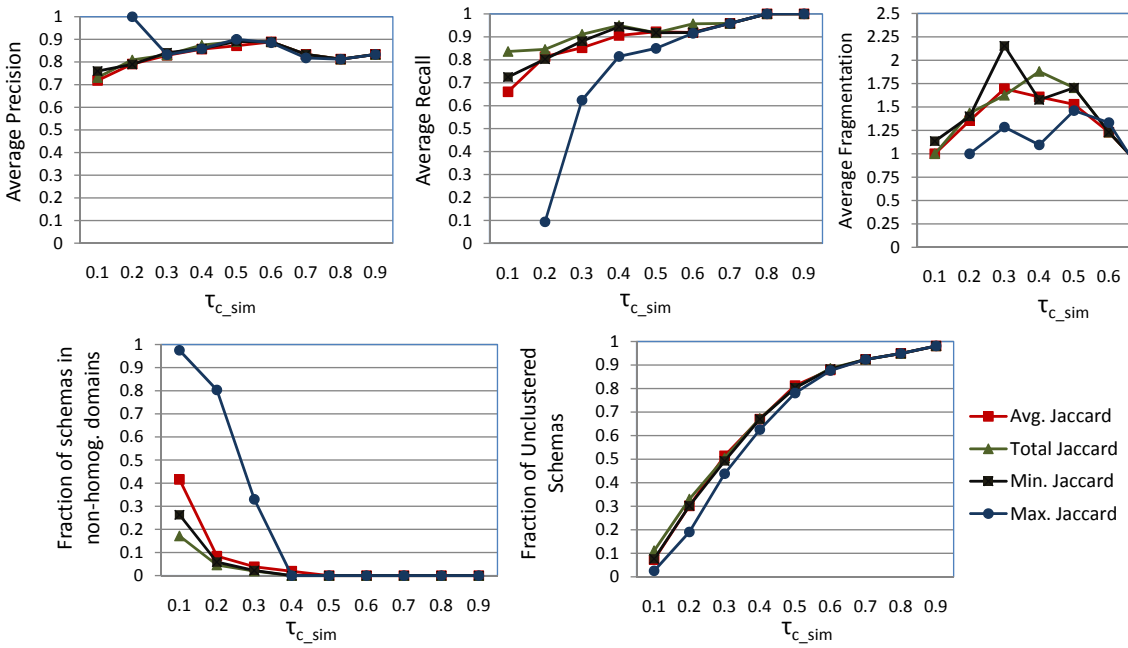

Figure 2: Schema clustering quality.

7.1.3 Generating Queries

To evaluate our query classification algorithm we need to simulate a typical query formulation process in which the user enters a query that includes some attribute names with a particular domain in mind. This query formulation process is a random process that we simulate as follows. We let the number of keywords in each query range from 1 to 10, with 100 queries generated for each number in this range. We use the same domain labeling terminology as in Section 7.1.2. For each randomly generated query Q_{rand}, we pick from B a random label B_{rand} for Q_{rand} to target. The label B_{rand} is selected based on the following probability distribution:

$$Pr(B_{rand}) = \frac{|S(B_{rand})|}{\sum_{j=1}^{|B|} |S(B_j)|}$$

Therefore, a label associated with a larger number of schemas will receive a larger number of queries, ensuring a balanced distribution of queries. Having selected a label B_{rand}, we start generating the keywords of the query. For simplicity, we treat the multiple keywords in the same query as a set of conditionally-independent and identically-distributed random variables given B_{rand}. Let T_{all} be the set of all terms extracted from all schemas as explained in Section 5.1; that is, $T_{all} = \cup_{S_i \in S} T_i$. We need to pick from T_{all} some keywords that a user will typically associate with B_{rand} as characteristic keywords that distinguish it from other labels. For each term $t_l \in T_{all}$, let $Freq(t_l, B_j)$ indicate the number of schemas in $S(B_j)$ that contain the term t_l; that is, $Freq(t_l, B_j) = |\{S_i : S_i \in S(B_j) \text{ and } t_l \in T_i\}|$. When picking terms for B_{rand}, we filter out the terms that do not exist in a sufficiently large fraction of schemas in $S(B_{rand})$. The fraction that we use for DW and SS is 0.25, while in the case DDH we use only 0.1 since the size of $S(B_{rand})$ in the case of DDH is counted in hundreds. After filtering out infrequent terms, we need to estimate for each of the remaining terms the probability that the term will be used in a query that targets B_{rand}. We use the following formula to estimate the degree by which a term t_l distinguishes a label B_j from other labels:

$$\lambda(t_l, B_j) = \frac{Freq(t_l, B_j)}{\sum_{t_a \in T_{all}} Freq(t_a, B_j)} \Big/ \frac{1}{|B|} \sum_{B_b \in B} \frac{Freq(t_l, B_b)}{\sum_{t_a \in T_{all}} Freq(t_a, B_b)}$$

That is, the ratio between the relative frequency of t_l in B_j, and the average relative frequency of t_l in all domain labels. We normalize $\lambda(t_l, B_j)$ such that, given a label B_j, the summation of the normalized $\lambda(t_l, B_j)$, for all t_l, equals 1. The normalized value of $\lambda(t_l, B_j)$ is used as the probability of picking the term t_l given that the label B_j has already been picked. Therefore,

$$Pr(t_{rand}|B_{rand}) = \frac{\lambda(t_{rand}, B_{rand})}{\sum_{t_a \in T_{all}} \lambda(t_a, B_{rand})}$$

This way, we assign higher probabilities to the terms that exist in $S(B_{rand})$ with higher ratios relative to their existence in the schemas of other labels.

7.2 Schema Clustering Quality

We compare the effectiveness of our clustering algorithm when using the four similarity measures: Min. Jaccard, Max. Jaccard, Avg. Jaccard and Total Jaccard. We also measure the effect of changing the value of τ_{c_sim} on the quality of clustering.

First we run our clustering algorithm on the DDH schema set. The clustering algorithm works perfectly on DDH, giving precision and recall values above 0.99 for all $\tau_{c\_sim} \geq 0.2$ and for all similarity measures, except Max. Jaccard which gives low recall for $\tau_{c\_sim} < 0.5$. The perfect performance of the clustering algorithm on DDH is expected since the schemas in DDH belong to a few well-separated domains.

Next we run the clustering algorithm on the union of the two schema sets DW and SS. Figure 2 shows the performance of the clustering algorithm on the union of DW and SS, using the four similarity measures, as τ_{c_sim} varies from 0.1 to 0.9. The figure shows that all the similarity measures perform almost the same, except for Max. Jaccard which performs

	$\tau_{c\_sim} = 0.2$			$\tau_{c\_sim} = 0.3$		
	DW	SS	Both	DW	SS	Both
Precision	0.75	0.84	0.81	0.85	0.87	0.82
Recall	0.93	0.77	0.78	0.98	0.86	0.86
Unclustered	0.25	0.37	0.29	0.48	0.56	0.50
Non-homog.	0	0.11	0.13	0	0.03	0.04
Fragmentation	1	1.77	1.29	1.38	1.67	1.58

Table 2: Evaluation of schema clustering.

worse than the rest under some settings, and is therefore not recommended. Total Jaccard, which is more expensive than the rest, provides no substantial gains over Avg. Jaccard or Min. Jaccard, so it too is not recommended. We recommend either Avg. Jaccard or Min. Jaccard. We use Avg. Jaccard as our default similarity measure as described in Section 5.2. The figure also illustrates how τ_{c_sim} affects the effectiveness of clustering. As τ_{c_sim} increases, precision and recall increase, and the fraction of schemas in non-homogeneous domains decreases, which are all desirable effects. However, as τ_{c_sim} increases, the number of unclustered schemas also increases, which is undesirable. However, we should take into account that 25% of the schemas are already unique (as mentioned in Section 7.1.1) and should therefore remain unclustered. At the extreme value of $\tau_{c\_sim} = 0.9$, all schemas are unclustered. Therefore, we have a trade-off between the number of unclustered schemas and the quality of clustering as measured through precision, recall and the fraction of schemas in non-homogeneous domains. Fragmentation, which does not include unclustered schemas or non-homogeneous domains, generally increases as the value of τ_{c_sim} increases from 0.1 to 0.5, since higher values of τ_{c_sim} prohibit similar clusters from getting merged before the clustering algorithm terminates, and therefore they get fragmented. Starting from around 0.5, as the value of τ_{c_sim} increases fragmentation decreases because τ_{c_sim} is becoming so high that it breaks many domains down into unclustered schemas. As more domains get broken down into unclustered schemas (which are not counted as domains), the number of domains significantly decreases. Therefore, there is much less potential to have a label associated with multiple domains. This set of experiments suggests setting τ_{c_sim} between 0.2 and 0.3. It also shows that clustering is robust since it is not very sensitive to minor changes in τ_{c_sim}.

Table 2 presents results from a set of experiments that focuses on the performance of the clustering algorithm for $\tau_{c\_sim} = 0.2$ and 0.3. This set of experiments is performed on each of the two sets of schemas DW and SS separately, and on the union of DW and SS. As we saw previously, increasing the value of τ_{c_sim} from 0.2 to 0.3 increases precision and recall, and decreases the fraction of schemas in non-homogeneous domains, but it also increases the fraction of unclustered schemas. The performance measures are generally better for DW than SS because SS is more noisy and less rigidly structured than DW. The performance on the union of DW and SS is between the performance on the individual sets, which is expected. The important observations are that clustering quality is high, and varying τ_{c_sim} does not cause major variations in any of the clustering performance measures. From these experiments, we see that the clustering algorithm produces high quality results for different data sets, and can be effectively and robustly controlled using the parameter τ_{c_sim}.

7.3 Effect On Mediation And Mapping

Although it is possible in principle to perform schema mediation and mapping without prior clustering, our experiments show that there are serious problems that arise when doing that. In this section, we describe two problems that we observed when doing schema mediation and mapping on our schema sets without prior clustering. For the purpose of our experiments, we use the probabilistic schema mediation and mapping algorithms described in [6].

The first problem is related to the semantic coherence of mediated attributes. It is common to encounter two attributes from two different domains having exactly the same name but with different meanings depending on the domain. For example, in the DW schema set, the attribute 'family name' is used in a schema from the 'people' domain to refer to the last name of a person, and in a schema from the 'biology' domain to refer to the family of a living organism (i.e., a taxonomic rank). When performing mediation and mapping on DW without clustering the schemas first, these two attributes are associated with each others in a single mediated attribute. At runtime, when posing a query on the mediated schema to retrieve values from the attribute 'family name', the result is an incoherent set of values obtained from both data sources. This problem does not arise when schemas are clustered before mediation.

The second problem is related to the size of the mediated schema. One of the techniques used in schema mediation to make it tractable is to use an attribute frequency threshold to filter out attributes that appear in only a small fraction of schemas (e.g., in [6] the threshold is 0.1). However, this threshold is problematic if no clustering is done before mediation. In that case, the threshold will eliminate most or all the attributes from the domains that have fewer schemas than other domains, causing these small domains to be under-represented or completely absent in the mediated schema. For example, when performing schema mediation on the DDH schema set with a threshold of 0.1 and without clustering, the result is a mediated schema in which 2 of the 5 domains of DDH are absent. Even after reducing the threshold to 0.01, the smallest domain, namely 'people', is still under-represented with only 4 attributes in the mediated schema, not including the most relevant attributes like 'phone', 'address', and 'email'. Picking a very small threshold value will cause larger domains to be over-represented by including a large number of infrequent and uninteresting attributes in the mediated schema. Going to the extreme of completely eliminating the threshold (i.e., using a threshold of 0) results in a meaningless mediated schema that is merely a union of all attributes from all schemas (12060 mediated attributes in the case of DDH). Besides being meaningless, this huge number of mediated attributes significantly increases the running time of schema mediation and mapping. The total running time for mediation and mapping in this case is 5 hours, while in all our experiments, when doing schema clustering, mediation, and mapping, using typical parameters, the total end-to-end running time is always less than 25 minutes. We conclude that schema clustering before mediation and mapping improves quality and scalability.

7.4 Query Classification Quality

In this section, we present experiments to evaluate the accuracy of our naive Bayes classifier. First, we run our clustering algorithm to cluster schemas into domains. Next, we

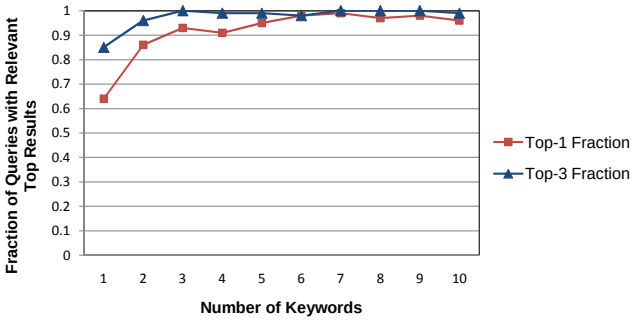

Figure 3: Query classification quality.

construct a naive Bayes classifier as described in Section 6 based on the domains that are generated from clustering. We then use that classifier to classify queries that we generate randomly as described in Section 7.1.3. The classifier returns a ranked list of domains, sorted descendingly according to their relevance to the query. For each query size from 1 to 10 we compute the top-1 fraction, which is the fraction of queries for which the top-ranked domain identified by the classifier is labeled with the same label B_{rand} as the query. We also compute the top-3 fraction, which the fraction of queries for which at least one of the top three domains is labeled with the same label as the query. The top-3 fraction is meaningful only for DW and SS since the number labels is relatively large. For DDH, where the number of labels is only 5, we only compute the top-1 fraction.

Our experiments on DDH give almost perfect results, with the top-1 fraction being 1 for all query sizes, except for single-keyword queries where the top-1 fraction drops slightly to about 0.95. The classification results on the union of DW and SS are shown in Figure 3. As the number of keywords per query increases, classification accuracy increases until the top-1 fraction becomes almost 1. Our results show that the classifier works well, even though the keyword queries generated by our random query generator sometimes include very non-indicative keywords due to the random nature of the query generator. For small query sizes, it is quite common to generate a query that is dominated by non-indicative keywords. In addition to quality, we also measure the running time needed to construct the naive Bayes classifier. For the large schema set DDH, it takes only about 5 minutes to construct the classifier, while for the union of DW and SS it takes less than a minute.

8. CONCLUSION

The growing number of structured data sources on the web has entailed growing interest in data integration for these sources. Existing data integration techniques operate on data sources that belong to a single domain. At web scale, it is infeasible to cluster data sources into domains manually. We deal with this problem and propose a schema clustering approach that leverages techniques from document clustering. We use a probabilistic model to handle the uncertainty in assigning schemas to domains, which fits with previous work on data integration with uncertainty. We also propose a technique based on naive Bayes classification that reasons on top of our probabilistic model in order to assign keyword queries posed by users to the most relevant domains.

Acknowledgements

We thank Alon Halevy and Anish Das Sarma for sharing the schema set from [6] with us. This work was supported by a Google Research Award and by the Natural Sciences and Engineering Research Council of Canada (NSERC) through the Business Intelligence Network strategic networks grant.

9. REFERENCES

[1] A. Aboulnaga and K. El Gebaly. μbe: User guided source selection and schema mediation for internet scale data integration. In *ICDE*, 2007.

[2] N. O. Andrews and E. A. Fox. Recent developments in document clustering. Technical report, Computer Science, Virginia Tech, 2007.

[3] B. Cestnik. Estimating probabilities: A crucial task in machine learning. In *Proc. of the 9th European Conference on Artificial Intelligence*, 1990.

[4] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: observations and implications. *SIGMOD Rec.*, 33(3), 2004.

[5] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proc. of the workshop on Data Cleaning and Object Consolidation at KDD*, 2003.

[6] A. Das Sarma, X. Dong, and A. Halevy. Bootstrapping pay-as-you-go data integration systems. In *SIGMOD*, 2008.

[7] X. L. Dong, A. Halevy, and C. Yu. Data integration with uncertainty. *VLDBJ*, 18(2), 2009.

[8] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, second edition, 2000.

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

[10] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[11] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *SIGMOD*, 2003.

[12] B. He, T. Tao, and K. C.-C. Chang. Organizing structured web sources by query schemas: a clustering approach. In *CIKM*, 2004.

[13] J. Madhavan, S. Cohen, X. Dong, A. Halevy, S. Jeffery, D. Ko, and C. Yu. Web-scale data integration: You can afford to pay as you go. In *CIDR*, 2007.

[14] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's deep web crawl. *Proc. VLDB Endow.*, 1(2), 2008.

[15] T. M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

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

[17] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley Longman, 2005.

[18] W. Wu, A. Doan, and C. T. Yu. Merging interface schemas on the deep web via clustering aggregation. In *ICDM*, 2005.