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

by

Hatem A. Mahmoud

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2010

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 thesis, I present an approach for clustering schemas into domains without any human intervention and based only on the names of attributes in the schemas. My clustering approach deals with uncertainty in assigning schemas to domains using a probabilistic model. I also propose a query classifier that determines, for a given keyword query, the most relevant domains to this query. I experimentally demonstrate the effectiveness of my schema clustering and query classification techniques.

Acknowledgements

I would like to thank my supervisor Ashraf Aboulnaga for all his support and guidance to me during my research.

Also, I would like to thank Alon Halevy and Anish Das Sarma for sharing the schema set from [8] with me. And thanks to my thesis readers Charles Clarke and Ihab Ilyas for their time and effort.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

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 [5]. Recent studies by Google estimate an order of 10 million high quality HTML forms [20]. Many other types of structured data sources spanning a wide spectrum of domains are also available on the web, such as HTML tables, Fusion Tables [14], 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 [13].

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 [21]. 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 this 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 this process by developing techniques that recommend mediated schemas and schema mappings to the user [25].

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 [21] 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 mediated schemas and schema mappings. Therefore, it was suggested [20] 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 [8, 9] proposes using a probabilistic model where several possible mappings are generated from each data source to the generated mediated schema, and each mapping is assigned a probability value.

All existing fully automatic integration techniques assume that the data sources to be integrated belong to the same domain (e.g., all data sources are about travel, bibliography, people, etc.), so a preprocessing phase is still needed to cluster data sources into domains before data integration takes place [20]. 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.

Once the schemas are clustered into domains, and data integration techniques are applied to each domain, we need to be able to direct the user's queries to their relevant domains at runtime. 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 Cairo" 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 these query interfaces and retrieve structured data.

1.2 Contribution

In this thesis, I present an approach for clustering structured data sources into domains based on their schemas. This problem involves 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, the clustering approach that I propose in this thesis relies entirely on attribute names to cluster schemas into domains. Second, there is no prior knowledge about the types of domains that should be created or how many they are, since the web is essentially about everything. Consequently, I use a clustering algorithm that does not make assumptions about the number or the types of domains in advance. Third, since I am proposing a fully automatic technique, I need to handle uncertainty in deciding which domain a schema should be assigned to. I 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. My work integrates well with previous work on schema mediation and mapping with uncertainty [8, 9].

I also propose a technique based on naive Bayesian classification to determine the domains relevant to a given keyword query and rank these domains according to their relevance to the query. The probabilistic nature of the domains adds more challenges to classification since the query is supposed to be assigned to a domain whose content is uncertain.

The work in this thesis has been published in [22]. The contributions 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. This approach integrates seamlessly with existing approaches for schema mediation and mapping with uncertainty.

3. A technique based on naive Bayesian classification to determine the domains relevant to a keyword query and rank those domains according to their relevance to the query. The classifier takes into account the fact that the domains are probabilistic.

4. An experimental evaluation on schemas from a wide spectrum of domains.

1.3 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 presents background and related work. Chapter 3 provides an overall view of the proposed solution after providing a more formal definition of the problem. Chapter 4 explains the schema clustering approach, then Chapter 5 explains the process of retrieving domains that are relevant to a keyword query. The experimental evaluation is presented in Chapter 6, and finally the conclusion is in Chapter 7.

Chapter 2

Background and Related Work

2.1 Background

2.1.1 Clustering

I use machine learning techniques from the domain of document clustering [2, 27] to deal with schema clustering. Typically, when clustering a set of data objects (e.g., documents, or schemas), the objects are represented as a set of vectors with the same dimensionality. A distance function is defined to measure the distance between each two vectors, and then a clustering algorithm is used to group vectors that are close to each others into clusters.

One of the most widely-used clustering algorithms is k-means [19]. It starts with an initial estimate (possibly a random estimate) of clusters' centroids, then the algorithm works iteratively. At each iteration, each data object is assigned to the cluster of the closest centroid, then the centroids are recomputed based on the data objects in each cluster. The k-means algorithm assumes that the number of clusters is known in advance, and its output depends on the initially estimated centroids. Moreover, it is hard to use k-means in cases where the centroid is hard to define (e.g., for binary feature vectors).

Another possible approach is to use density-based clustering algorithms (e.g., DB-SCAN [11]), which search the data space for regions of high density that are separated from each others by regions of low density. Density-based clustering is more robust than k-means since it is less sensitive to noise, and can handle clusters of arbitrary shapes, and it does not require prior knowledge about the number of clusters. However, it is problematic when the notion of density is hard to define (e.g., in high-dimensional data) or when clusters have varying densities.

A third approach is hierarchical clustering, where the dataset is viewed as a tree of clusters. At the root of the tree is a cluster that contains all the data, while the leaves of the tree are singleton clusters that contain single data objects, and each non-leaf cluster is the union of its children clusters. Hierarchical clustering is either divisive (top-down) or agglomerative (bottom-up).

In divisive clustering, the algorithm starts with the whole data set (i.e., the root of the tree), then breaks it down into two (or more) clusters, then breaks the new clusters down into smaller clusters, and so on. At each iteration, a cluster is chosen to be split based on some criterion. For example, one possible criterion [18] is to pick the cluster with the largest diameter, where the diameter of a cluster is defined as the maximum distance between any two objects in it. A partitioning algorithm (e.g., k-means) is used to split the chosen cluster into smaller ones.

Agglomerative clustering goes in the opposite direction: it starts with singleton clusters (i.e., the leaves of the tree), then it merges them iteratively to produce larger clusters. At each iteration, the closest pair of clusters is chosen to be merged, based on a specified measure of inter-cluster distance. Some common measures of inter-cluster distance include: single-link (i.e., the minimum distance between the objects of the two clusters), complete-link (i.e., the maximum distance between the objects of the two clusters), group average (i.e., the average of the distances between the objects of the two clusters), and centroid-based (i.e., the distance between the centroids of the two clusters).

Hierarchical clustering, whether divisive or agglomerative, typically does not run until the whole tree of clusters is constructed. It is usually terminated once the current level of clusters satisfies a particular condition. For example, we can terminate divisive clustering once the diameters of all clusters are below some threshold, and we can terminate agglomerative clustering once the distance between the closest pair of clusters is above some threshold. Another approach is to terminate hierarchical clustering once a particular number of clusters is reached.

Agglomerative hierarchical clustering avoids many of the limitations of other clustering techniques. For example, it does not require prior knowledge of the number of clusters, its output does not depend on initial estimates, and it can be used with any inter-cluster distance measure, so it is useful when the notions of centroid and density are hard to define. However, it is also expensive. Divisive hierarchical clustering can be more efficient than agglomerative clustering if the tree of clusters is balanced and the number of clusters is very small compared to the number of data objects. Whether agglomerative clustering is more effective than divisive clustering or not is subject to debate (see for example [31]). However, as a bottom line, divisive clustering inherits the limitations of the algorithm that it uses to partition clusters. For example, if k-means is used, then the output of partitioning depends on the initial estimates of the centroids, and we are still limited by our ability to define a meaningful centroid.

Clustering algorithms can be also classified based on whether clusters overlap (i.e., the same data object may belong to multiple clusters) or not. If clusters are allowed to overlap, a data object that is assigned to multiple clusters might have a different degree of membership in each cluster, and in such case it is referred to as fuzzy clustering. Fuzzy clustering is based on fuzzy set theory [30], where elements may belong to multiple sets with membership values that range from 0 to 1. The most common fuzzy clustering algorithm is fuzzy c-means (FCM) [3]. Another approach to handle uncertainty in clustering is through probability theory, which is the model I use in this thesis because it fits well with previous research in data integration with uncertainty.

2.1.2 Classification

Classification is the problem of assigning a data objects to one of several pre-specified classes. When the user poses a keyword query at runtime, the problem of determining the domain(s) relevant to the query can be dealt with as a classification problem. The object to be classified (in our case, the keyword query) needs to be represented as a vector, similar to the vectors of the data objects in the clusters. Then this vector is passed to a classification algorithm that returns the most-relevant class, or a list of classes sorted by relevance.

As with clustering, there are many well-known classification algorithms that can be used, including instance-based classification algorithms (e.g., nearest neighbour [12]), decision tree classification algorithms (e.g., C4.5 [24]), Perceptron-based classification algorithms [26], support vector machines [28], etc. However, the design choice I make in the classification phase is guided by the design choice I made in the clustering phase; that is, to use a probabilistic model to handle uncertainty in clustering. I use naive Bayesian classification which is a machine learning technique that has been used extensively in document classification and other applications [23], and it works effectively on top of the probabilistic model constructed during clustering. Bayesian classification makes use of Bayes' rule to calculate the probability that the query object belongs to one of the classes:

$$Pr(y|x) = \frac{Pr(x|y)Pr(y)}{Pr(x)}$$

where x is the query object and y is a class. The classifier is called "naive" if it assumes that the elements of the vector x are statistically independent given y. This is usually done to make the problem more tractable. The classifier can still perform well under such assumption. Also, although Bayesian classification is expensive, it is possible to move all expensive operations to setup time rather than query time, as demonstrated in Chapter 5.

2.2 Related Work

Clustering schemas into domains was considered in [17]. However, there are still important differences between my work and that work. The main differences are as follows:

1. The work in [17] considered exactly eight specific domains (flights, cars, movies, etc.) and presented a clustering algorithm that is given the number of clusters in advance. I do not pre-specify the number or the types of domains considered. This is important because, at web-scale data integration, it is hard (even impossible) to determine the number of the available domains. Also the types of domains available on the web are so many that the boundaries between them are usually blurred.

2. The authors of [17] assumed that, for each of the eight domains, there exist *anchor* attributes that do not occur except in the schemas of the domain. I do not assume the existence of such anchor attributes.

3. The clustering algorithm in [17] was based on agglomerative hierarchical clustering as in this thesis, but with a different cluster-to-cluster similarity measure. Basically, the algorithm in [17] assumes that, for each domain, the attribute names follow a distinct multinomial probability distribution, then measured the similarity between every two clusters based on how high the probability that the attributes of both clusters belong to the same multinomial distribution, as determined using chi-square test. My approach uses a simple (yet effective) distance measure based on Jaccard similarity.

4. The work in [17] did not deal with uncertainty in clustering; that is, the assignments of schemas to domains were certain. I use probabilistic schema-to-cluster assignment to handle uncertainty.

Schema clustering is also mentioned in [20] 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, 8, 29].

Chapter 3

System Overview

3.1 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 thesis is to automate the clustering step. Therefore, I consider the two problems of (1) clustering schemas into domains, and (2) retrieving and ranking relevant domains at query time. For the purpose of my research, I define the notion of a domain as follows:

Definition 3.1.1 (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.*

I 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').

My 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). I focus on single-table schemas since most data sources on the web belong to this category. My 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 I 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. I 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. The 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 3.1.1. 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 thesis, but I have to ensure that my solutions integrate well with previous work. At query time, I need to provide the user with the capability to retrieve domains relevant to a keyword query, where a keyword query is a set of terms (i.e., keywords), taking in consideration that the domains are constructed with uncertainty. The query classifier takes as an input a keyword query and a set of domains, and outputs for each domain its degree of relevance to this query. The classifier expects the keyword query to include some keywords that are similar to attribute names from the relevant domain(s) (e.g., "departure Toronto destination Cairo" or "books authored by Stephen King").

3.2 Solution Overview

I 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 [10]. 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, I 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 [16] show 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 my approach on data sources whose data and data types are not plainly exposed (e.g., the deep web). My experiments on the schemas of hundreds of web data sources from diverse domains show that assigning probabilities based on textual similarities works well (Chapter 6).

I 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 [8, 9]. 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 that is constructed 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 this 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 [8]. 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 the tuple is retrieved belongs to that domain (where such probability is computed by my clustering algorithm), 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 (where such probability is computed by the probabilistic schema mapping algorithm, e.g., [8]).

3.3 Architecture

Figure 3.1 illustrates the architecture of the proposed system. It clarifies the two levels of uncertainty in the data integration system: the uncertainty in assigning schemas to domains, and the uncertainty in mapping the schemas in each domain to the domain's mediated schema. Besides, the figure also illustrates a typical use case of the system. In a typical use case, the user poses a keyword query on the system at runtime. The keyword query is forwarded to the query classifier which determines the most relevant domains to this query. The relevant domains are presented to the user in the form of structured query interfaces that correspond to the mediated schemas of the relevant domains, sorted by their relevance to the keyword query. Then the user poses a structured query on one of the mediated schemas to retrieve tuples, and the structured query is forwarded to the data sources in the domain behind this mediated schema. The tuples retrieved from each data source are assigned probability values as explained in the previous section, then all tuples are merged into a single result set to be presented to the user sorted by tuple probability.

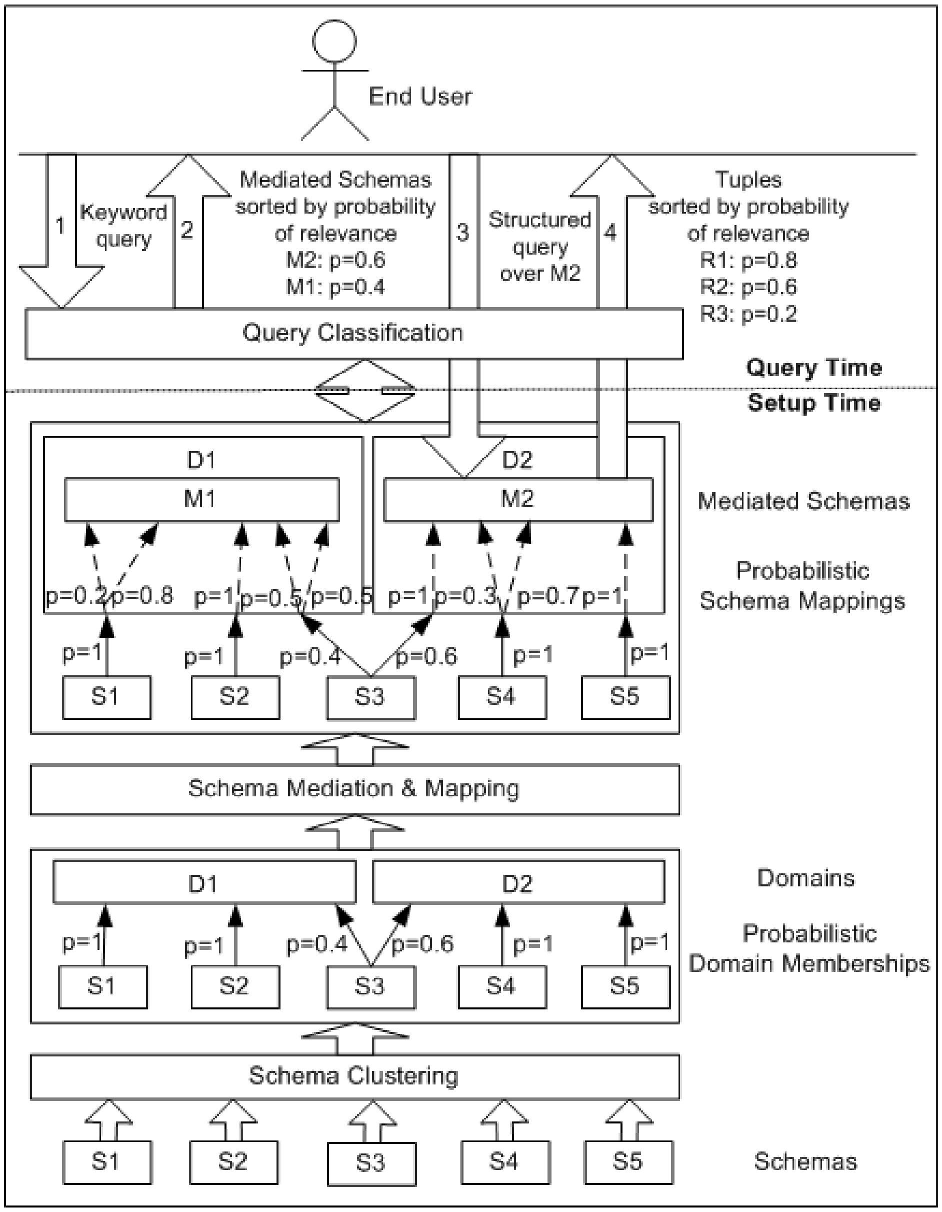


Figure 3.1: System architecture illustrated via an example of a typical use case.

Chapter 4

Schema Clustering

This chapter explains the process of schema clustering, which can be divided into three phases: (1) creating feature vectors, (2) the actual clustering of schemas, and (3)assigning probabilities of schema memberships in domains. Given a set of schemas $S = \{S_1, S_2, \ldots, S_{|S|}\}$ as input, the 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. I 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.

4.1 Creating Feature Vectors

Before proceeding with clustering, I need to characterize each schema with a feature vector. Feature vectors are needed both during the clustering process and during query classification. I 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, I 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. I 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.

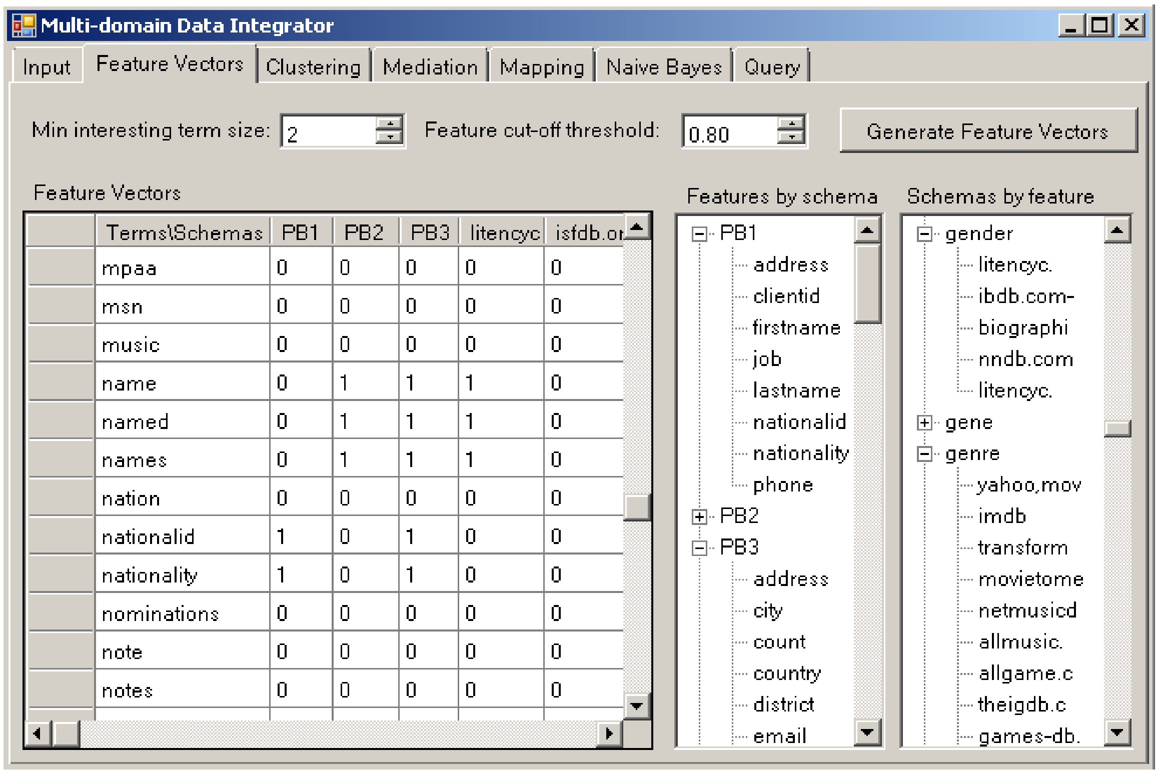
Algorithm 1 describes how feature vectors are created. First, for each schema $S_i \in S$, I extract all the terms from S_i by splitting its attribute names over a set of pre-defined delimiters, like white spaces, slashes, and underscores. For example, given the following schema {Class_ID,Day/Time, Professor Name, Subject}, the set of extracted terms will be {Class, ID, Day, Time, Professor, Name, Subject}. I 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'). I convert all terms to a canonical form for better comparisons (e.g., all characters to lower case), then I 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 $S = \{S_1, S_2, \ldots, S_{|S|}\}$

 3:     for all $S_i \in S$ 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 $\cup_{i=1}^{|S|} T_i$ into a vector $L$
10:     for all $S_i \in S$ 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 $\max_{t \in T_i} t\_sim(L_j, t) \geq \tau_{t\_sim}$ then
14:                 $F^i_j \leftarrow 1$
15:             else
16:                 $F^i_j \leftarrow 0$
17:             end if
18:         end for
19:     end for
20:     return $F = \{F^1, F^2, \ldots, F^{|S|}\}$
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 $dim\ L = | \cup_{i=1}^{|T|} T_i|$. I 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



Min interesting term size: 2
Feature cut-off threshold: 0.80

Feature Vectors

	Terms\Schemas	PB1	PB2	PB3	litencyc	isfdb.or
	mpaa	0	0	0	0	0
	msn	0	0	0	0	0
	music	0	0	0	0	0
	name	0	1	1	1	0
	named	0	1	1	1	0
	names	0	1	1	1	0
	nation	0	0	0	0	0
	nationalid	1	0	1	0	0
	nationality	1	0	1	0	0
	nominations	0	0	0	0	0
	note	0	0	0	0	0
	notes	0	0	0	0	0

Features by schema

```
⊟ PB1
    address
    clientid
    firstname
    job
    lastname
    nationalid
    nationality
    phone
⊞ PB2
⊟ PB3
    address
    city
    count
    country
    district
    email
```

Schemas by feature

```
⊟ gender
    litencyc.
    ibdb.com-
    biographi
    nndb.com
    litencyc.
⊞ gene
⊟ genre
    yahoo,mov
    imdb
    transform
    movietome
    netmusicd
    allmusic.
    allgame.c
    theigdb.c
    games-db.
```

Figure 4.1: The system interface while creating feature vectors.

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, I 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. I then compare this maximum to a threshold τ_{t_sim} that I 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 [7]. In this thesis, I 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. I 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 [15]. 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. Figure 4.1 shows the system interface at the phase of generating feature vectors.

4.2 Clustering Algorithm

I use hierarchical agglomerative clustering as described in Algorithm 2.

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:
        $\underset{(U_i^{(k)}, U_j^{(k)}) \in U^{(k)} \times U^{(k)};\ i \neq j}{\arg\max} \ c\_sim(U_i^{(k)}, U_j^{(k)})$
    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:
            $\underset{(U_i^{(k)}, U_j^{(k)}) \in U^{(k)} \times U^{(k)};\ i \neq j}{\arg\max} \ c\_sim(U_i^{(k)}, U_j^{(k)})$
    end while
    return $C = U^{(k)}$
end procedure
```

First, I 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 function between the two schemas S_i and S_j, where $1 \leq i, j \leq |S|$. I use the Jaccard coefficient as a similarity measure since it is known to be suitable for high dimensional binary feature vectors [27]. Thus,

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

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, I 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, I denote the set of clusters that I have as $U^{(k)}$. Since I 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 I merge the two closest (most similar) clusters into one new cluster, i.e., $|U^{(k+1)}| = |U^{(k)}| - 1$.

I 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 $U_j^{(k)}$. Experiments in Section 6.2 show that other cluster similarity measures can also be used to 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, $U_c^{(k+1)} = U_c^{(k)}$. Consequently,

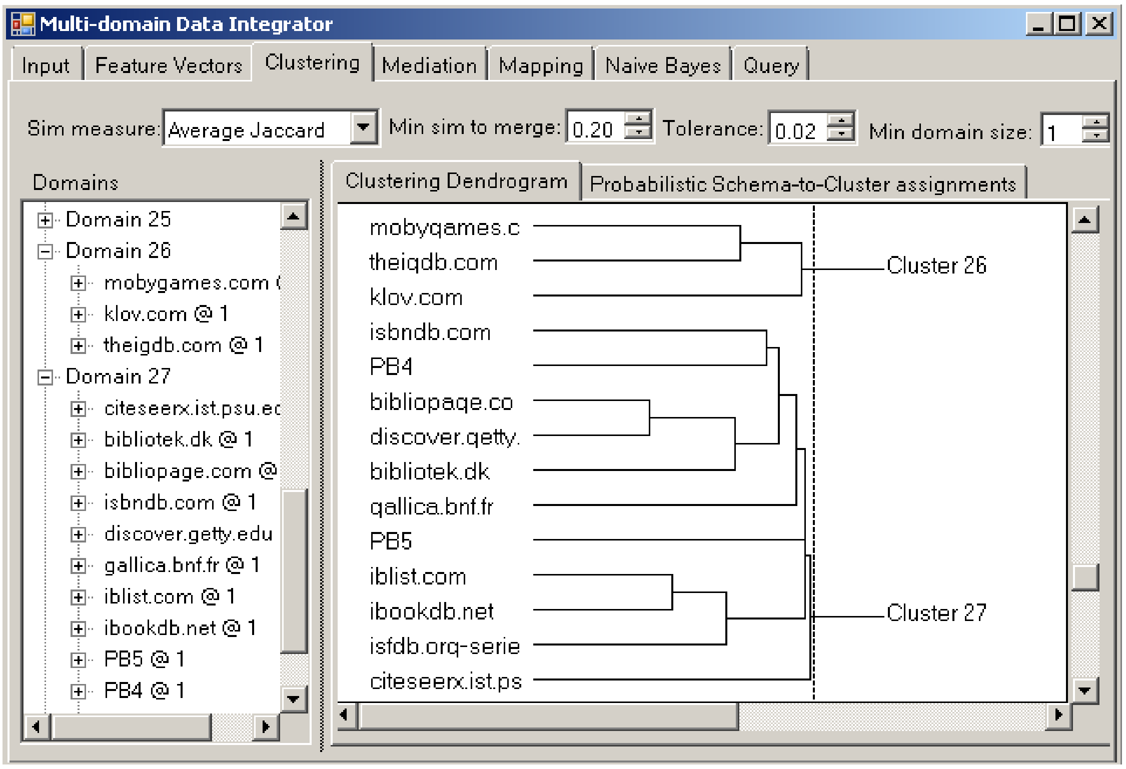
$$U^{(k+1)} = (U^{(k)} \setminus \{U_a^{(k)}, U_b^{(k)}\}) \;\cup\; \{U_{ab}^{(k+1)}\}$$

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 $U_{ab}^{(k+1)}$, I 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, the 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. The experiments



Sim measure: Average Jaccard Min sim to merge: 0.20 Tolerance: 0.02 Min domain size: 1

Figure 4.2: The system interface during schema clustering.

in Section 6.2 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 the clustering algorithm. Figure 4.2 shows the system interface during the schema clustering phase.

4.3 Assigning Probabilities

The main source of uncertainty in schema clustering are 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, I 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.

Algorithm 3 Assign Probabilities

```
1:  procedure AssignProbabilities
2:      input: Set of schemas $S = \{S_1, S_2, \ldots, S_{|S|}\}$
3:      input: Set of clusters $C = \{C_1, C_2, \ldots, C_{|C|}\}$

4:      for all $C_r \in C$ do
5:          Define a domain $D_r$
6:      end for
7:      for all $S_i \in S$ do
8:          for all $C_r \in C$ do
9:              if $s\_c\_sim(S_i, C_r) \geq \tau_{c\_sim}$ and
10:             $\frac{s\_c\_sim(S_i, C_r)}{\max_{C_j \in C} s\_c\_sim(S_i, C_j)} \geq 1 - \theta$ then
11:                 $Pr(S_i \in D_r) \leftarrow \frac{s\_c\_sim(S_i, C_r)}{\sum_{D_j \in D(S_i)} s\_c\_sim(S_i, C_j)}$
12:             else
13:                 $Pr(S_i \in D_r) \leftarrow 0$
14:             end if
15:         end for
16:     end for
17:     return $\{(S_i, D_r, Pr(S_i \in D_r)) : \textit{for all } S_i \textit{ and } D_r\}$
18: end procedure
```

Since I am going to assign some schemas to multiple domains, while each schema in S belongs to one and only one cluster in C, I need to separate the concept of clusters from the concept of domains. I use the term *clusters* to refer to sets of schemas that partition S, like those returned by Algorithm 2. I use the term *domains* to refer to sets of schemas too; however, every schema in S may belong to multiple domains with different probabilities.

I construct domains from the clusters returned by Algorithm 2 as follows. First, I 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. I then examine every schema $S_i \in S$; if S_i is sufficiently similar to multiple clusters then I 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, I 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, $\frac{s\_c\_sim(S_i, C_r)}{\max\limits_{C_j \in C} s\_c\_sim(S_i, C_j)} \geq 1 - \theta$, 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. The value used in the experiments of this thesis is $\theta = 0.02$.

For each schema $S_i \in S$, let $D(S_i)$ be defined as follows:

$$D(S_i) = \{D_r : s\_c\_sim(S_i, C_r) \geq \tau_{c\_sim} \; and \; \frac{s\_c\_sim(S_i, C_r)}{\max\limits_{C_j \in C} s\_c\_sim(S_i, C_j)} \geq 1 - \theta\}$$

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,

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

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. In practice, most schemas will belong to one domain with probability 1, while only a few schemas in each domain will belong to the domain with probabilities between 0 and 1 exclusive.

After schema clustering, a fully automatic schema mediation and mapping technique can be run to generate a 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. The following section explains how this is typically done.

4.4 Probabilistic Mediation and Mapping

Between schema clustering and query classification, we need to create a mediated schema for each domain and map all the domain's schemas to that mediated schema. There is already previous research that deals with that problem [8], and it is not among the goals of this thesis to add to that research. However, I need to show how my work integrates with previous work. Therefore, this section serves to complete the picture rather than to develop new schema mediation and mapping techniques.

Any existing method of schema mediation and mapping can be used as long as it provides the following for each domain $D_r \in D$, given $S(D_r)$ as an input:

- A mediated schema $M_r = \{A_1^{M_r}, A_2^{M_r}, \ldots, A_{|M_r|}^{M_r}\}$, where $A_j^{M_r}$ is a *mediated attribute* for $j = 1, 2, \ldots, |M_r|$. A mediated attribute is not necessarily a single attribute. Depending on the algorithm used for schema mediation, it may be a set of *similar* attributes extracted from $S(D_r)$ and clustered together. Typically, attribute similarity should be based on the same similarity function *t_sim* that is used in creating feature vectors.

- For each schema $S_i \in S(D_r)$, a probabilistic schema mapping $\Phi^{S_i,M_r} = \{(\phi_j^{S_i,M_r}, Pr(\phi_j^{S_i,M_r}));\ where\ j = 1, 2, ..., |\Phi^{S_i,M_r}|\}$. $\phi_j^{S_i,M_r}$ is a schema mapping from S_i to M_r; that is, a function that maps tuples from S_i to M_r. $Pr(\phi_j^{S_i,M_r})$ is the probability that such mapping is correct. I make the simplifying assumption that $Pr(\phi_j^{S_i,M_r})$ is independent of $Pr(S_k \in D_r)$, for all $S_k \in S(D_r)$. So the probabilities assigned to schema mappings by the schema mapping algorithm are only based on the attribute names of the schemas in $S(D_r)$.

I use the same symbol S_i when referring to the data source whose schema is S_i so as to simplify the notation. I also use the phrase *raw tuples* to refer to tuples retrieved from S_i but not mapped yet to M_r, while those already mapped to M_r are referred to as *mapped tuples*.

At query-time, whenever a query is posed over M_r, the query is dispatched to every schema S_i in $S(D_r)$. Let the set of raw tuples retrieved from S_i be $R^i = \{R_j^i : j = 1, 2, \ldots, |R^i|\}$. The probabilistic schema mapping Φ^{S_i,M_r} is then applied to every raw tuple $R_j^i \in R^i$ to map it into a tuple compatible with the schema M_r. Since Φ^{S_i,M_r} actually comprises several mappings, each having its own probability, we have for each raw tuple $R_j^i \in R^i$ a set of mapped tuples, each with its own probability. If, for the same raw tuple R_j^i, two (or more) mapped tuples are exactly the same, e.g. $\phi_l^{S_i,M_r}(R_j^i) = \phi_{l'}^{S_i,M_r}(R_j^i)$ where $l \neq l'$, then they will be consolidated into one mapped tuple whose probability is the sum of the probabilities of the original tuples; that is, $Pr(\phi_l^{S_i,M_r}) + Pr(\phi_{l'}^{S_i,M_r})$.

The mapped tuples retrieved from different data sources, all of them already mapped to M_r, will be gathered into a single result set R^{all} to be returned to the user. In our case, different data sources belong to D_r with different probabilities. Therefore, the overall probability assigned to every mapped tuple $\phi_l^{S_i,M_r}(R_j^i)$ in the result set R^{all} will be the product of two probabilities: (1) the probability that the schema S_i, from which the tuple R_j^i was retrieved, belongs to the domain D_r, and (2) the probability that the mapping $\phi_l^{S_i,M_r}$, that was used to map the tuple R_j^i into M_r, is correct. That is,

$$Pr(\phi_l^{S_i,M_r}(R_j^i) \in R^{all}) = Pr(\phi_l^{S_i,M_r}) \ . \ Pr(S_i \in D_r)$$

Finally, if two (or more) tuples in the final result set R^{all}, retrieved from different data sources, are exactly the same, e.g. $\phi_l^{S_i,M_r}(R_j^i) = \phi_{l'}^{S_{i'},M_r}(R_{j'}^{i'})$, where $i \neq i'$, then they will be consolidated into one tuple whose probability is

$$1 - \left[\left(1 - Pr(\phi_l^{S_i,M_r}(R_j^i) \in R^{all})\right)\left(1 - Pr(\phi_{l'}^{S_{i'},M_r}(R_{j'}^{i'}) \in R^{all})\right)\right]$$

Chapter 5

Query Classification

In this chapter I investigate the issue of answering keyword queries posed over a multi-domain data integration system by retrieving and ranking relevant domains. I use a naive Bayesian 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 this, some of the keywords in the query need to be similar to some attribute names in the relevant domains. The design of the classifier ensures that expensive operations are performed at system setup time rather than query time.

5.1 The Naive Bayesian Classifier

At query time, the user poses keyword queries on the system. The system proposed in this thesis deals with keyword queries that include keywords that are similar to some of the attribute names in the schema domains (e.g., "departure Toronto destination Cairo"). Whenever a keyword query is posed over the system, the system generates a feature vector to characterize the keyword query in the same manner as it does for every schema in S (Section 4.1). Let Q denote the set of keywords in the keyword query entered by the user. I convert all keywords into the canonical form, and remove stop words and extremely small keywords. The result is the set of terms T_Q. I then construct the binary feature vector F^Q. Let F_j^Q 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. I set the value of the feature F_j^Q to 1 if there exists a term in T_Q that is sufficiently similar to L_j; that is, if $\max\limits_{t \in T_Q} t\_sim(L_j, t) \geq \tau_{t\_sim}$. Otherwise, $F_j^Q = 0$.

The 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 this probability as $Pr(D_r|F^Q)$.

According to Bayes' rule:

$$Pr(D_r|F^Q) = \frac{Pr(F^Q|D_r)Pr(D_r)}{Pr(F^Q)} \tag{5.1}$$

$Pr(F^Q|D_r)$ 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, I may assign Q to the domain that has the maximum posterior probability (i.e., $\arg\max\limits_{D_r}\ Pr(D_r|F^Q)$), or I may return a list of relevant domains ranked by posterior probabilities. Note that, in Equation 5.1, I 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, I only need to compute $Pr(F^Q|D_r)Pr(D_r)$.

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

$$Pr(F^Q|D_r)Pr(D_r) \ = \ Pr(D_r) \prod_{j=1}^{dim\ L} Pr(F_j^Q|D_r) \tag{5.2}$$

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 4.3. Therefore, $Pr(D_r)$ can be expressed in terms of the following summation over all subsets of $S(D_r)$:

$$Pr(D_r) \ = \sum_{S' \subseteq S(D_r)} Pr(D_r|D_r = S')Pr(D_r = S') \tag{5.3}$$

Similarly, for all j,

$$Pr(F_j^Q|D_r) = \sum_{S' \subseteq S(D_r)} Pr(F_j^Q|D_r = S', D_r)\ Pr(D_r = S'|D_r) \tag{5.4}$$

I analyze the probabilities on the right-hand sides of Equations 5.3 and 5.4 as follows:

1. The probability $Pr(D_r|D_r = S')$ is estimated as:

$$Pr(D_r|D_r = S') = \frac{|S'|}{|S|} \tag{5.5}$$

2. The probability $Pr(D_r = S')$ can be expressed as the probability of the following conjunction:

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

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

$$Pr(D_r = S') = \prod_{S_i \in S'} Pr(S_i \in D_r) \prod_{S_i \notin S'} Pr(S_i \notin D_r) \tag{5.6}$$

3. $Pr(F_j^Q | D_r = S', D_r)$ 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:

$$Pr(F_j^Q | D_r = S', D_r) = \begin{cases} \dfrac{\sum_{S_i \in S'} F_j^i}{|S'|} & ; \ if \ F_j^Q = 1 \\ 1 - \dfrac{\sum_{S_i \in S'} F_j^i}{|S'|} & ; \ if \ F_j^Q = 0 \end{cases} \tag{5.7}$$

4. The probability $Pr(D_r = S' | D_r)$ can be computed using Bayes' rule as follows:

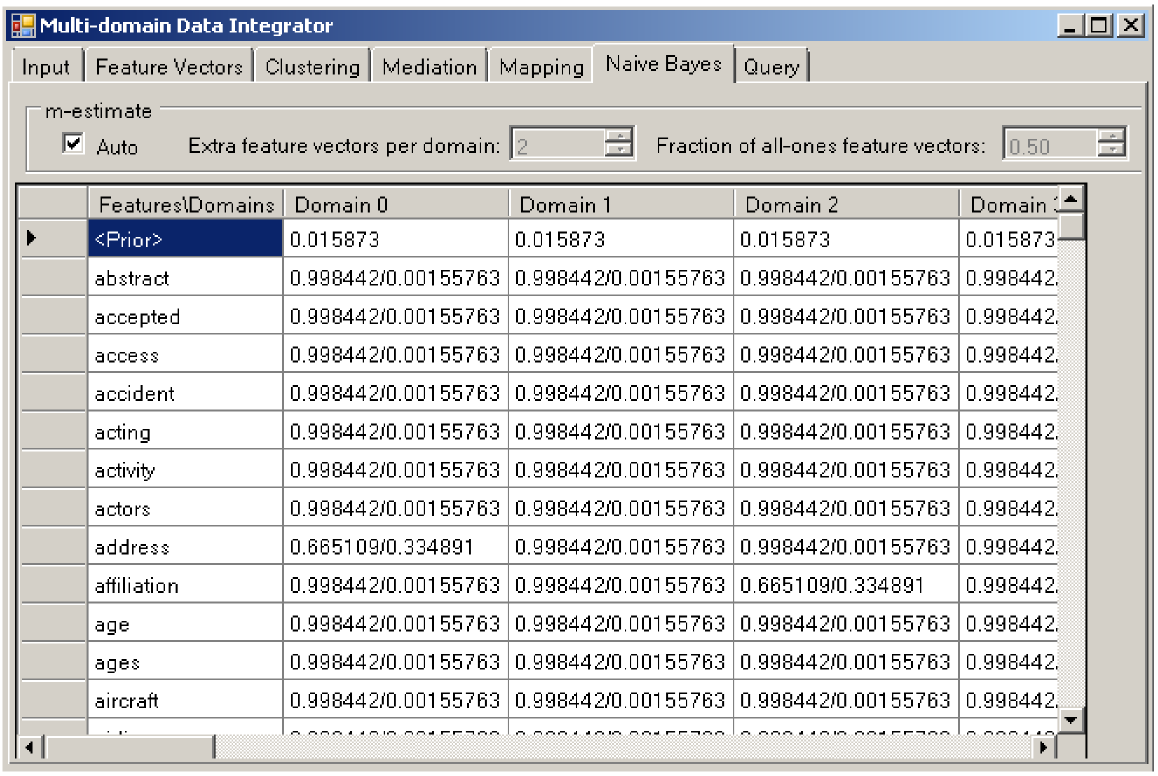
$$Pr(D_r = S' | D_r) = \frac{Pr(D_r | D_r = S') Pr(D_r = S')}{Pr(D_r)} \tag{5.8}$$

   Note that the probabilities on the right-hand side of Equation 5.8 are already computed as part of Equation 5.3.

By substituting Equations 5.5, 5.6, 5.7, and 5.8 into Equations 5.3 and 5.4, and then substituting Equations 5.3 and 5.4 into Equation 5.2, I obtain the posterior probabilities required to rank domains. Figure 5.1 shows the system interface during the construction of the query classifier.

5.2 Robustness Issues

There are two issues with Equation 5.7. The first issue 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,


Multi-domain Data Integrator

m-estimate

☑ Auto Extra feature vectors per domain: 2 Fraction of all-ones feature vectors: 0.50

Features\Domains	Domain 0	Domain 1	Domain 2	Domain
<Prior>	0.015873	0.015873	0.015873	0.015873
abstract	0.998442/0.00155763	0.998442/0.00155763	0.998442/0.00155763	0.998442
accepted	0.998442/0.00155763	0.998442/0.00155763	0.998442/0.00155763	0.998442
access	0.998442/0.00155763	0.998442/0.00155763	0.998442/0.00155763	0.998442
accident	0.998442/0.00155763	0.998442/0.00155763	0.998442/0.00155763	0.998442
acting	0.998442/0.00155763	0.998442/0.00155763	0.998442/0.00155763	0.998442
activity	0.998442/0.00155763	0.998442/0.00155763	0.998442/0.00155763	0.998442
actors	0.998442/0.00155763	0.998442/0.00155763	0.998442/0.00155763	0.998442
address	0.665109/0.334891	0.998442/0.00155763	0.998442/0.00155763	0.998442
affiliation	0.998442/0.00155763	0.998442/0.00155763	0.665109/0.334891	0.998442
age	0.998442/0.00155763	0.998442/0.00155763	0.998442/0.00155763	0.998442
ages	0.998442/0.00155763	0.998442/0.00155763	0.998442/0.00155763	0.998442
aircraft	0.998442/0.00155763	0.998442/0.00155763	0.998442/0.00155763	0.998442

Figure 5.1: The system interface during the construction of the query classifier.

so $|S'|$ may be zero. The second issue is that, 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 this, assume that $F_j^i = 0$ for all $S_i \in S(D_r)$. Then, according to Equation 5.7 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 5.4, the probability $Pr(F_j^Q|D_r)$ will also be zero. Eventually, by substituting $Pr(F_j^Q|D_r)$ in Equation 5.2, 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 I use the *m-estimate* of probabilities [4]. Basically, for each domain D_r, and for each subset $S' \subseteq S(D_r)$, I 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 5.7 can be rewritten as follows:

$$Pr(F_j^Q | D_r = S', D_r) = \begin{cases} \dfrac{\sum_{S_i \in S'} F_j^i + p.m}{|S'| + m} & ;\ if\ F_j^Q = 1 \\ 1 - \dfrac{\sum_{S_i \in S'} F_j^i + p.m}{|S'| + m} & ;\ if\ F_j^Q = 0 \end{cases} \tag{5.9}$$

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, I 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. I set $p = 1/dim\ L$ and $m = 1 + |S'|$, which gives much more bias towards missing terms, and keeps the number of added schemas reasonable compared to the size of the domain.

5.3 Time Complexity

Bayesian classification is expensive, but all of the expensive operations in my case can be done at setup time rather than query time. Equations 5.5, 5.6, and 5.8 are all independent of the user's query. Equation 5.9 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) I can still compute Equation 5.9 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 5.2 can be pre-computed and stored at setup time. At query time, calculating Equation 5.2 for all domains takes $O(|D|\ dim\ L)$ running time per query.

To analyze the setup time needed for Equation 5.2 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. I 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 5.6, and consequently S' will not contribute to the summation in Equation 5.3. Additionally, by substituting $Pr(D_r = S')$ into Equation 5.8, $Pr(D_r = S' | D_r)$ will also be zero, and again S' will not contribute to the summation in Equation 5.4. Therefore, when computing the summations in Equations 5.8 and 5.4, I 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, the probabilities in Equation 5.2 can be calculated 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 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 4.3).

Chapter 6

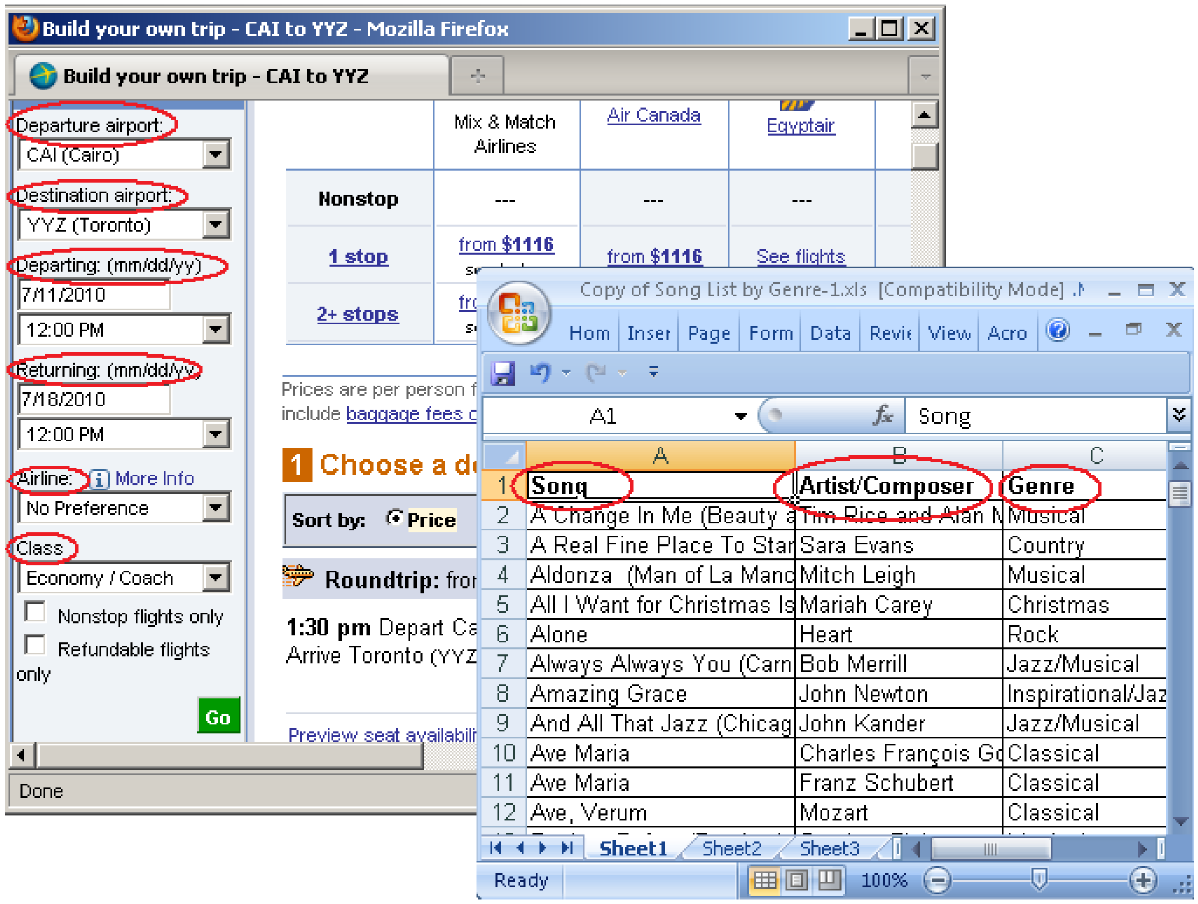
Experimental Evaluation

6.1 Experimental Setup

I implement the back end schema clustering and classification algorithms in C++, while the front end GUI is implemented in C#, and I use this prototype to evaluate the effectiveness of my schema clustering and query classification techniques. I run my experiments on a Windows Vista machine, with Intel Centrino Duo 2GHz processor and 3GB RAM. The goal of my experiments is to demonstrate that my 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 I need schemas of web data sources labeled with their correct domains, and I 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 I describe next.

6.1.1 Schemas Used

I use the schema set used in [8], which I obtained from the authors of that paper. This schema set consists of 2323 schemas from 5 different domains (bibliography, cars, courses, movies, and people) that were extracted from Google's web index, and I refer to it as the DDH set after the initials of the authors of [8]. Examples of schemas from the DDH data set include {title, authors, year_of_publish, conference_name} from the 'bibliography' domain and {year, type, make, model} from the 'cars' domain. 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 my schema clustering and query classification methods on such diverse and overlapping domains, I collect my own


Build your own trip - CAI to YYZ - Mozilla Firefox

Build your own trip - CAI to YYZ

Departure airport:
CAI (Cairo)
Destination airport:
YYZ (Toronto)
Departing: (mm/dd/yy)
7/11/2010
12:00 PM
Returning: (mm/dd/yy)
7/18/2010
12:00 PM
Airline: More Info
No Preference
Class
Economy / Coach
Nonstop flights only
Refundable flights only
Go

	Mix & Match Airlines	Air Canada	Egyptair
Nonstop	---	---	---
1 stop	from $1116	from $1116	See flights
2+ stops			

Prices are per person f
include baggage fees c

1 Choose a d

Sort by: Price

Roundtrip: fror

1:30 pm Depart Ca
Arrive Toronto (YYZ

Preview seat availabili

Done

Copy of Song List by Genre-1.xls [Compatibility Mode]

Hom Inser Page Form Data Revie View Acro

A1 Song

	A	B	C
1	Song	Artist/Composer	Genre
2	A Change In Me (Beauty a	Tim Rice and Alan M	Musical
3	A Real Fine Place To Star	Sara Evans	Country
4	Aldonza (Man of La Manc	Mitch Leigh	Musical
5	All I Want for Christmas Is	Mariah Carey	Christmas
6	Alone	Heart	Rock
7	Always Always You (Carn	Bob Merrill	Jazz/Musical
8	Amazing Grace	John Newton	Inspirational/Jaz
9	And All That Jazz (Chicag	John Kander	Jazz/Musical
10	Ave Maria	Charles François Go	Classical
11	Ave Maria	Franz Schubert	Classical
12	Ave, Verum	Mozart	Classical

Sheet1 Sheet2 Sheet3

Ready 100%

Figure 6.1: Extracting schemas from deep web data sources and downloadable spreadsheets. Attribute names are circled in red.

schema sets manually through web search and through lists of hidden web data sources that are available on-line (e.g., in Wikipedia).

I extract two sets of schemas from two types of data sources as illustrated in Figure 6.1. The first schema set, which I refer to as DW, is extracted from deep web data sources. For this schema set, I 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. For example, the schema extracted from the web page of expedia.com that is shown in Figure 6.1 is {departure airport, destination airport, departing (mm/dd/yy), returning (mm/dd/yy), airline, class}. The second schema set, which I refer to as SS, is extracted from downloadable spreadsheets that I 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. For example, the schema extracted from the spreadsheet that is shown in Figure 6.1 is {song, artist/composer, genre}. 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.

6.1.2 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 6.1.1 *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 my 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*

	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	97
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 6.1: Statistics about schema sets.

algorithms. The question is: If the clustering algorithm clusters S_1 and S_2 together, is it a false positive? If it does not cluster them together, is it a false negative?

In order to deal with this problem, I perform my experiments as follows. For the two schema sets DW and SS, I manually associate each schema S_i with a set of labels $B(S_i)$ according to what I perceive as potential domains for S_i. Example domain labels that I use include 'movies', 'bibliography', and 'people' (see Appendix A for a complete list). As a concrete example, the schema {first name, last name, function, gender, description, date of birth, place of birth, date of death, place of death} that is extracted from a deep web data source is associated with the label 'people', while the schema {Name, Grade, School, District, Project} that is extracted from a downloadable spreadsheet is associated with the four labels 'schools', 'people' 'awards', and 'projects'. Each schema in the data set is labeled with at least one label. Table 6.1 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 associated 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. I run the clustering algorithm on the schema sets and examine the set of domains D that is produced by the clustering algorithm, and I label each domain $D_r \in D$ with the set of *dominant labels* within this domain. Usually there is only one dominant label, but sometimes there are several labels that equally dominate the domain (i.e., there exist two or more labels, each of them is associated with n schemas in the domain, where n is the largest number of schemas associated with any label in 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. I 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. I also sum probabilities as a weighted counting of schemas when estimating precision and recall. If more than one label equally dominate the domain, I 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,

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

I 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. I also compute the fraction of schemas assigned to non-homogeneous domains as one of my performance measures. Another special case is a domain with only one schema. This happens when a schema is not sufficiently similar to any other schemas in S, given the value used for the threshold τ_{c_sim}. I 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 $a \neq b$. I use the term *fragmentation* to refer to this case and I measure the degree of fragmentation in my 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, I 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

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

I 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

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

I 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 my clustering approach, I measure the effect of changing τ_{c_sim} on the performance measures like precision, recall, fragmentation, unclustered schemas, and non-homogeneous domains. I also compare the performance of my clustering algorithm when other cluster-to-cluster similarity measures are used instead of the average Jaccard similarity that is described in Section 4.2. The other alternatives I 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 $U_j^{(k)}$.

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

6.1.3 Generating Queries

To evaluate my query classification algorithm I 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 I simulate as follows. I let the number of keywords in each query range from 1 to 10, with 100 queries generated for each number in this range. I use the same domain labeling terminology as in Section 6.1.2. For each randomly generated query Q_{rand}, I 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:

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

That is, 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}, I start generating the keywords of the query. For simplicity, I 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 4.1; that is, $T_{all} = \cup_{S_i \in S} T_i$. I 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, $Freq(t_l, B_j) = |\{S_i : S_i \in S(B_j)\ and\ t_l \in T_i\}|$. When picking terms for B_{rand}, I filter out the terms that do not exist in a sufficiently large fraction of schemas in $S(B_{rand})$. The fraction that I use for DW and SS is 0.25, while in the case DDH I 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, I need to estimate for each of the remaining terms the probability that the term will be used in a query that targets B_{rand}. I use the following formula to estimate the degree by which a term t_l distinguishes a label B_j from other labels:

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

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. I 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, I 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.

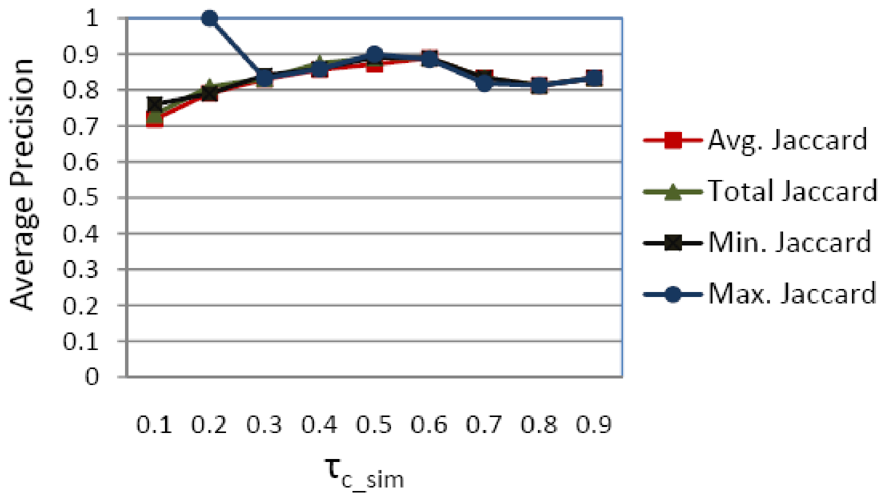


Figure 6.2: Average precision.

Examples of randomly generated queries include: the single keyword query "make" targeted at the domain 'cars', the 3-keywords query "pages author words" targeted at the domain 'bibliography', and the 4-keywords query "class hours bldg location" targeted at the domain 'courses'.

6.2 Schema Clustering Quality

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

First I run my 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, I run the clustering algorithm on the union of the two schema sets DW and SS. Figures 6.2 to 6.6 show 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 figures show that all the similarity measures perform almost the same, except for Max. Jaccard which performs 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

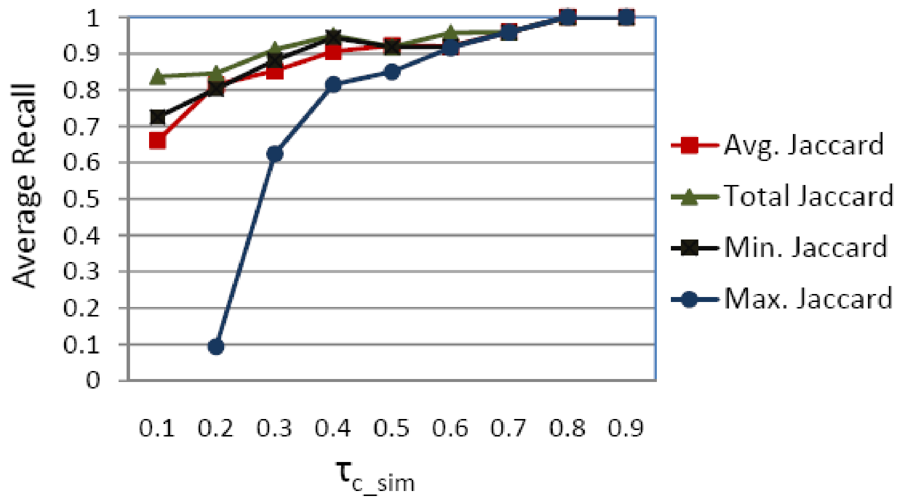


Figure 6.3: Average recall.

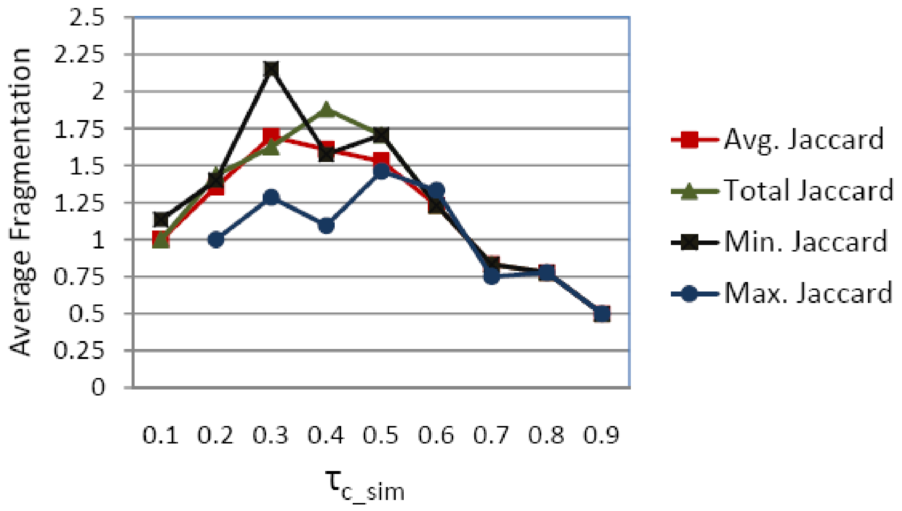


Figure 6.4: Average fragmentation.

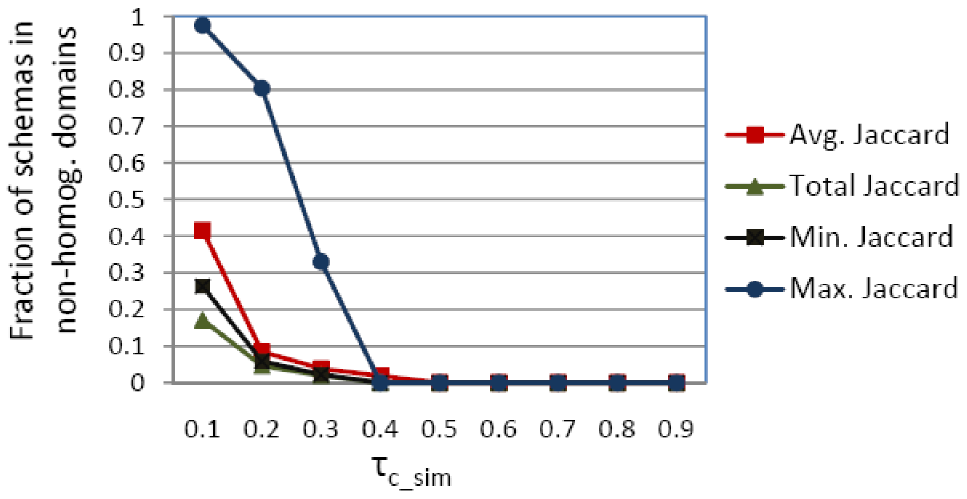


Figure 6.5: Fraction of schemas in non-homogeneous domains.

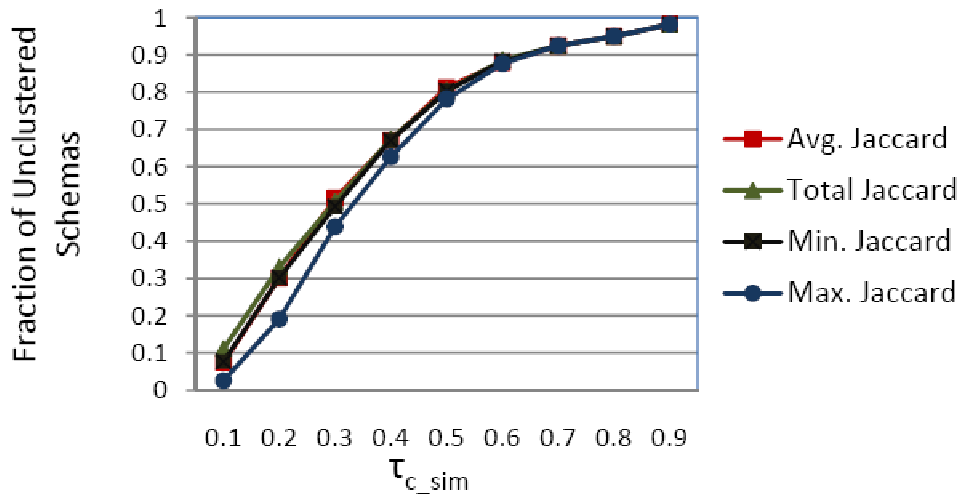


Figure 6.6: Fraction of unclustered schemas.

	$\tau_{c\_sim} = 0.2$			$\tau_{c\_sim} = 0.3$		
	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 6.2: Evaluation of schema clustering.

Avg. Jaccard or Min. Jaccard, so it too is not recommended. I recommend either Avg. Jaccard or Min. Jaccard. I use Avg. Jaccard as my default similarity measure as described in Section 4.2.

The figures also illustrate 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 6.1.1) 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 6.2 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 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}.

6.3 Effect On Mediation And Mapping

Although it is possible in principle to perform schema mediation and mapping without prior clustering, my experiments show that there are serious problems that arise when doing that. In this section, I describe two problems that I observed when doing schema mediation and mapping on my schema sets without prior clustering. For the purpose of my experiments, I use the probabilistic schema mediation and mapping algorithms described in [8].

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 [8] the threshold is set to 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

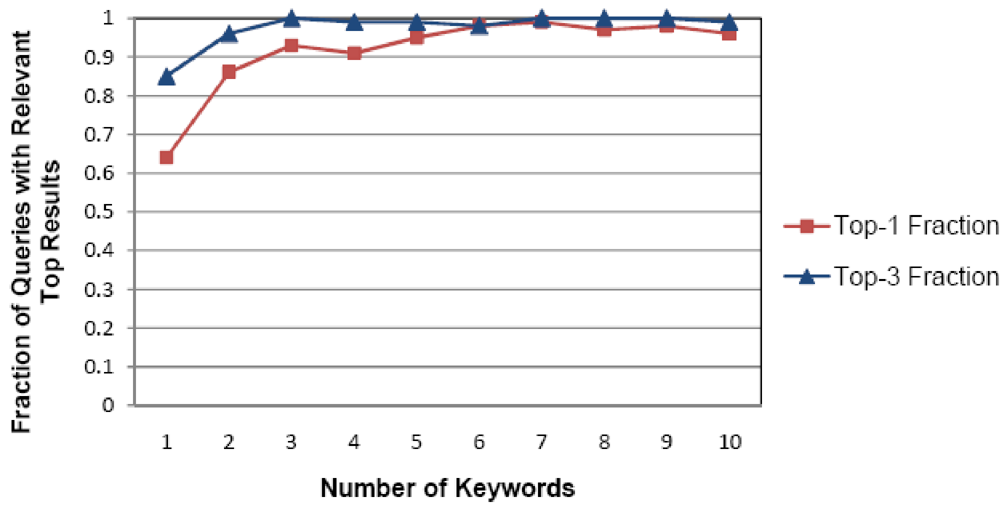


Figure 6.7: Query classification quality.

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 my experiments, when doing schema clustering, mediation, and mapping, using typical parameters, the total end-to-end running time is always less than 25 minutes. I conclude that schema clustering before mediation and mapping improves quality and scalability.

6.4 Query Classification Quality

In this section, I present experiments to evaluate the accuracy of my query classifier. First, I run my clustering algorithm to cluster schemas into domains. Next, I construct a naive Bayesian classifier as described in Chapter 5 based on the domains that are generated from clustering. I then use this classifier to classify queries that I generate randomly as described in Section 6.1.3. 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, I 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. I 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, I only compute the top-1 fraction.

My 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 6.7. As the number of keywords per query increases, classification accuracy increases until the top-1 fraction becomes almost 1. My results show that the classifier works well, even though the keyword queries generated by my 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, I also measure the running time needed to construct the naive Bayesian 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.

Chapter 7

Conclusion and Future Work

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. I deal with this problem and propose a schema clustering approach that utilizes techniques from document clustering. My solution depends on attribute names only, and therefore is general enough to handle the different types of data source on the web. I use a probabilistic model to handle the uncertainty in assigning schemas to domains, which fits perfectly with previous work on data integration with uncertainty. I also propose a technique based on naive Bayesian classification that reasons on top of my probabilistic model in order to assign keyword queries posed by the user at runtime to the most relevant domains.

Experiments show that, despite the limited amount of information available about the data sources at clustering time, and despite the great amount of overlapping and irregularity in the schemas of the data sources, the proposed clustering techniques work well, and are not very sensitive to minor variations in parameters. The query classifier also demonstrates good performance even though the experiments are run using automatically generated queries that involve some randomness.

A possible direction of future work is to refine the solution to remove the exponential factor in the running time needed to construct the query classifier (see Section 5.3). Even if the running time can be effectively controlled through the parameter θ, the existence of an exponential factor in the running time is an issue that needs to be resolved. One approach to handle this issue is by approximating the probability distributions that require such exponential time.

Another direction of future work is to develop techniques that incorporate user feedback to update the system. User feedback can be either explicit, where the user directly assesses the correctness of clustering (e.g., by informing the system that a schema should be assigned to another cluster rather than the one determined by the system), or implicit, where the system automatically infers the correctness of clustering by monitoring user interaction (e.g., clicking on search results). A different approach is to solicit automatic feedback from the data retrieved from each data source at query time. That is, to determine whether the tuples retrieved from the data sources in a given cluster are consistent with each others, according to some measure of consistency, and use this to assess the correctness of clustering.

It is also possible to extend the system by supporting other query models beyond the current notion of keyword queries, considering data sources more general than single-table sources, or working with other types of data sources such as RDF data.

APPENDICES

Appendix A

Domain Labels Used in Experiments

TOC, access, accounts, activities, airdisasters, alcohol, animals, applications, architecture, art, articles, attributes, awards, banks, bibliography, blogs, boardgames, buildings, business, cartoons, categories, channels, chemistry, chess, codeofconduct, comics, competitions, concerts, contacts, courses, databases, degrees, departments, drink, environment, events, exams, exposures, factories, fellowships, festivals, food, games, gender, genes, grants, healthplans, hotels, housing, industry, inflation, insurance, interments, internships, invoices, items, jobs, librarians, licenses, licensing, locations, math, media, medications, money, movies, music, names, nurseries, organizations, people, plans, plants, producers, projects, race, religious, research, roads, robots, schedule, schemas, schools, series, sessions, shows, sports, subjects, suppliers, taxes, teachers, theatres, tourism, tracking, videos, vulnerabilities, windows

Bibliography

[1] Ashraf Aboulnaga and Kareem El Gebaly. μbe: User guided source selection and schema mediation for internet scale data integration. In *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2007.

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

[3] James C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Kluwer Academic Publishers, 1981.

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

[5] Kevin C.-C. Chang, Bin He, Chengkai Li, Mitesh Patel, and Zhen Zhang. Structured databases on the web: observations and implications. *SIGMOD Record*, 33(3), 2004.

[6] Kevin C.-C. Chang, Bin He, and Zhen Zhang. Toward large scale integration: Building a metaquerier over databases on the web. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2005.

[7] William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. A comparison of string distance metrics for name-matching tasks. In *Proceedings of the Workshop on Data Cleaning and Object Consolidation at the International Conference on Knowledge Discovery and Data Mining (KDD)*, 2003.

[8] Anish Das Sarma, Xin Dong, and Alon Halevy. Bootstrapping pay-as-you-go data integration systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2008.

[9] Xin Dong, Alon Halevy, and Cong Yu. Data integration with uncertainty. *VLDB Journal*, 18(2), 2009.

[10] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* Wiley-Interscience, second edition, 2000.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, 1996.

[12] Evelyn Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination. Technical report, USAF School of Aviation Medicine, Randolph Field, Texas, 1951.

[13] Michael Franklin, Alon Halevy, and David Maier. From databases to dataspaces: a new abstraction for information management. *SIGMOD Record*, 34(4), 2005.

[14] Hector Gonzalez, Alon Halevy, Christian S. Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, and Warren Shen. Google fusion tables: data management, integration and collaboration in the cloud. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2010.

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

[16] Bin He and Kevin C.-C. Chang. Statistical schema matching across web query interfaces. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2003.

[17] Bin He, Tao Tao, and Kevin C.-C. Chang. Organizing structured web sources by query schemas: a clustering approach. In *Proceedings of the Conference on Information and Knowledge Management (CIKM)*, 2004.

[18] Leonard Kaufman and Peter Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* Wiley-Interscience, 1990.

[19] J. B. Macqueen. Some methods of classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967.

[20] Jayant Madhavan, Shirley Cohen, Xin Dong, Alon Halevy, Shawn Jeffery, David Ko, and Cong Yu. Web-scale data integration: You can afford to pay as you go. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2007.

[21] Jayant Madhavan, David Ko, Lucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's deep web crawl. *Proceedings of the VLDB Endowment*, 1(2), 2008.

[22] Hatem A. Mahmoud and Ashraf Aboulnaga. Schema clustering and retrieval for multi-domain pay-as-you-go data integration systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 2010.

[23] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[24] John R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[25] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4), 2001.

[26] Frank Rosenblatt. *Principles of Neurodynamics*. Spartan, New York, 1962.

[27] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley Longman, 2005.

[28] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, 1995.

[29] Wensheng Wu, AnHai Doan, and Clement T. Yu. Merging interface schemas on the deep web via clustering aggregation. In *Proceedings of the International Conference on Data Mining (ICDM)*, 2005.

[30] Lotfy A. Zadeh. Fuzzy sets. *Information and Control*, 8, 1965.

[31] Ying Zhao and George Karypis. Hierarchical clustering algorithms for document datasets. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery at the ACM SIGMOD International Conference on Management of Data*, 2002.