# Visualization of Heterogeneous Data

Mike Cammarano, Xin (Luna) Dong, Bryan Chan, Jeff Klingner, Justin Talbot, Alon Halevy, and Pat Hanrahan

**Abstract**— Both the Resource Description Framework (RDF), used in the semantic web, and Maya Viz u-forms represent data as a graph of objects connected by labeled edges. Existing systems for flexible visualization of this kind of data require manual specification of the possible visualization roles for each data attribute. When the schema is large and unfamiliar, this requirement inhibits exploratory visualization by requiring a costly up-front data integration step. To eliminate this step, we propose an automatic technique for mapping data attributes to visualization attributes. We formulate this as a schema matching problem, finding appropriate paths in the data model for each required visualization attribute in a visualization template.

**Index Terms**—Data integration, RDF, attribute inference.

## 1 Introduction

Recently, there has been tremendous interest in web mashups, which combine data from multiple web services into new visualizations and applications [4, 44]. Mashups require technology both to easily integrate diverse data sources and to easily create visualizations.

A major challenge to creating mashups is the nature of the data on the web. Since the web is not centrally managed, databases do not always conform to agreed upon schemas. Globally, the generation of data can be considered as a loosely coupled bottom-up process [2]. The classic example is Wikipedia which is being created by thousands of people around the world. Another example is GoogleBase [19]; GoogleBase allows anyone to add new records with an arbitrary schema to a shared database. The result is that most data on the web (and also in businesses and governments) is heterogeneous, unstructured, and often incomplete. Researchers in the database community have called such a collection of heterogeneous data a *data space*, and have formulated a long-term research agenda to provide technologies for managing such data spaces [16].

Semantic web technologies like the Resource Description Framework (RDF) [5] and triple-stores attempt to provide a common denominator format within which diverse data sources can be represented. RDF represents data as a graph. Each node in the graph is an object represented by a uniform resource identifier. Edges connect nodes to other nodes or to literals which represent attributes. Although most data on the web is not represented as RDF, the data model is general enough that it provides a convenient unifying abstraction. We will not assume the existence of an ontology – semantics of objects need not be agreed upon and the data may be incomplete.

In this paper, we consider the problem of visualizing heterogeneous collections of data. We describe a system that is able to automatically find the information in the collection of data that is needed to create a visualization. The user starts by creating a query that returns a result set of objects. This query could be from a text-based search engine or from a more structured query browser. The user then selects a type of visualization, for example, a map, time or scatterplot. In order to create the visualization, various attributes of are needed. For example, to place an item on a map, a geolocation needs to be retrieved for each object. In order to find these attributes, the system searches for the attributes it needs by following links between objects in the data space. Once the attributes are found, the visualization is drawn and presented to the user.

Our approach is based on decoupling the schema of the underlying data from the specification of a visualization. By introducing a layer of search to mediate between the user's visualization specification and the actual RDF data, the user can request visualizations without having to know the schema in advance.

The specific contributions of this paper are the following:

- We describe a formalism for specifying visualizations without requiring detailed knowledge of the data sources or their schemas.
- We formulate the problem of matching visualizations to information in the sources as a variant of schema matching. We break the matching problem into two phases: a path indexing phase to enumerate and prioritize which paths to consider, followed by a search for combinations of path instances that attempts to select the best set of paths to use for each object.
- We describe the implementation of our technique and some experiences from fielding it on different scenarios. We evaluate the system by performance as well as accuracy in returning good matches. Examples are given using a variety of visual representations including maps, timelines, scatterplots, and node-link diagrams.

In this paper we will emphasize examples of our technique applied to dbpedia [7], an RDF database automatically extracted from Wikipedia. However, we have also used it for visualizing collections of documents in a digital library, and visualizing personal information like e-mails and address books.

## 2 Overview

Our technique takes a set of object instances and a specification of fields needed for a visualization. For each object instance, it then attempts to choose paths to the attributes that best fit the requirements of the requested fields. This technique is intended to be used as just one component of a larger interactive platform for searching and browsing loosely-coupled heterogeneous data. In particular, this paper will not address the initial query mechanisms for selecting the objects of interest.

We contrast our technique against two existing mechanisms for synthesizing visualizations from databases of objects and their attributes. One common approach to visualizing objects of many different types is to have a `display` method for each class. For example, most object-oriented programming languages include a method to convert an object to a string for display purposes. The `display` method can take as input any of the properties of the instance to compute the visualization. The obvious disadvantage of this method is that the visualization is solely determined by the class of an object, and cannot be tailored to a particular context or task.

- *Mike Cammarano, Bryan Chan, Jeff Klingner, Justin Talbot, and Pat Hanrahan are with Stanford University. E-mail: {mcammara, bryanc, klingner, jtalbot}@stanford.edu, hanrahan@cs.stanford.edu.*
- *Xin (Luna) Dong is with University of Washington, E-mail: lunadong@cs.washington.edu.*
- *Alon Halevy is with Google, E-mail: halevy@google.com.*

*Manuscript received 31 March 2007; accepted 1 August 2007; posted online 27 October 2007.* Published 14 September 2007.
*For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.*

1077-2626/07/$25.00 © 2007 IEEE Published by the IEEE Computer Society

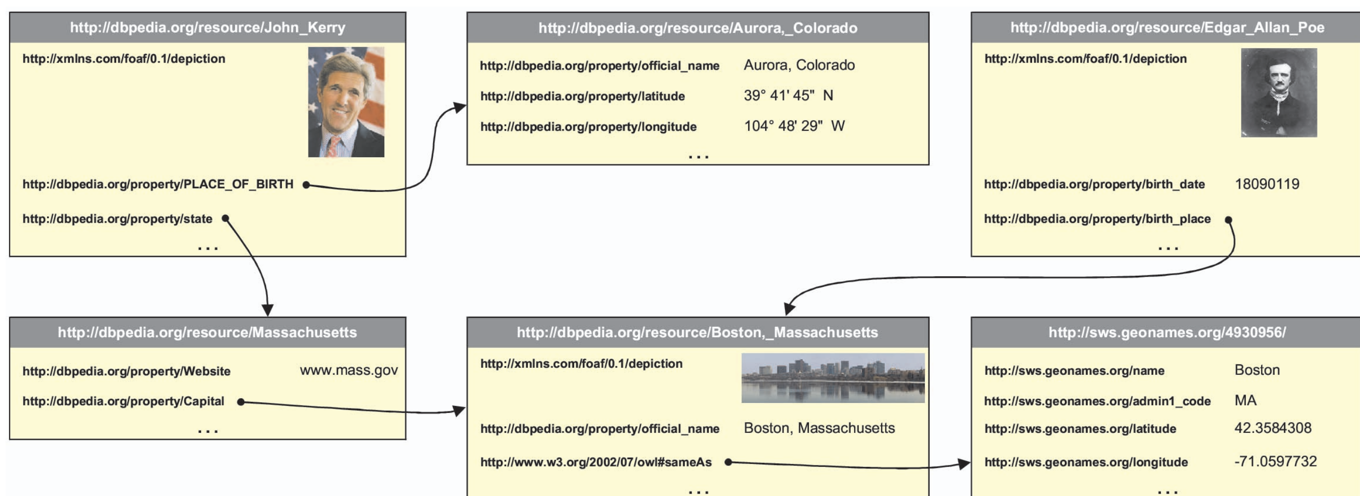


Fig. 1. A small portion of the dbpedia RDF graph illustrating the heterogeneity of representations for people and places. Each box in the diagram depicts an object and several of its literal valued attributes. Associations between objects are shown by arrows.

A more flexible method for specifying visualizations can be found in the *Maya Viz* system [3], which separates the visualization method from the class definition. In this approach, the visualization only requres that data objects have specific attributes conforming to known semantic roles. This allows two advantages. First, any set of objects can be displayed as long as each object has the required properties, even if they differ on other attributes. For example, if an object has a timestamp field, then it can be displayed on a timeline. Second, by separating the visualization method from the object definition, multiple visualizations can be created for each object. However, this method requires up-front data integration, since objects must be annotated with roles clarifying how their attributes should be interpreted.

In contrast, we want to support casual exploration of unfamiliar data sets without requiring data integration in advance. These data sets may suffer from incomplete data and inconsistent naming conventions. Our approach is to tightly integrate search with visualization. Each visualization will be able to invoke graph searches that aim to provide "best-effort" retrieval of the requested properties. We propose multiple heuristics to guide these searches. Nevertheless, they cannot be expected to have perfect precision or recall, so missing or incorrect values must be handled. Our visualizations can make potential errors easy to discover by showing a confidence score associated with each item, as well as the lineage (the paths followed in the RDF graph to obtain it).

### 2.1 Heterogeneous Data

We are interested in working with data sets with missing or inconsistently named data. We will base most of our examples on the dbpedia corpus, a large and rich data source that exhibits these challenges.

Figure 1 shows a small portion of the dbpedia RDF graph. It consists of 6 objects: 2 representing public figures, and 4 representing geographic locations. This example helps to illustrate the heterogeneity present in dbpedia data describing people and places. In particular, note that:

- There are multiple distinct attributes with similar semantics.
- Each object has an incomplete subset of the possible attributes.
- The above effects can be compounded when following sequences of multiple associations.
- Consequently, retrieving semantically equivalent information pertaining to two different objects may require traversing completely different paths in the graph.

For example, while the attribute describing senator John Kerry's place of birth is named PLACE_OF_BIRTH, the analogous attribute for Edgar Allan Poe is named birth_place. Similarly, note the different representations for latitude and longitude. The object describing the city of Aurora, Colorado has attributes for these values in the dbpedia namespace, and the values are encoded as strings in degrees, minutes, seconds format. The entry for Boston, Massachusetts does not have attributes directly describing geolocation. However, Boston is associated with an object from the auxiliary geonames database (also available from dbpedia.org) via the sameAs relation. The geonames object describing Boston does have latitude and longitude attributes. They are formatted as signed decimal values, as opposed to the string format used for Aurora, Colorado.

Next, consider following a sequence of several associations to retrieve a distant attribute. For example, the geographic coordinates of John Kerry's birthplace can be found by first following the PLACE_OF_BIRTH association, and then the latitude and longitude attributes, respectively. We can write these paths as:

dbp:PLACE_OF_BIRTH.dbp:latitude
dbp:PLACE_OF_BIRTH.dbp:longitude

Note that we abbreviate (or omit) the namespaces of predicates within the paper text for ease of reading. In order to obtain the analogous geocoordinates for Edgar Allan Poe's birthplace, completely different paths must be followed. In this case:

dbp:birth_place.owl:sameAs.geo:latitude
dbp:birth_place.owl:sameAs.geo:longitude

In addition to needing to traverse different association paths to obtain analogous fields for different source objects, this case would also require converting the results into a common format. Recall that one pair of coordinates are expressed as decimals and the other as strings encoding the sexagesimal degrees-minutes-seconds format.

Several other observations may help to convey the heterogeneity of dbpedia. Searching for attribute names containing the string "birth" reveals at least 12 different attribute names that all appear to describe dates of birth. There is also great variability in which attributes are available. Consider the relations that originate with one of the 100 current U.S. senators. There are 70 different kinds of relations observed, but each senator uses, on average, only 27 of them with some having as few as 15 and some as many as 41.

Figure 2 is a visualization of dbpedia data that attempts to place U.S. senators on a map according to their state. In this case, the object describing a senator does not have an attribute for geolocation. However, as we have previously noted, it can be reasonable to infer geolocation attributes from associated objects like their home state. Our technique allows needed attributes to be automatically inherited from associated objects when the visualization requires it.

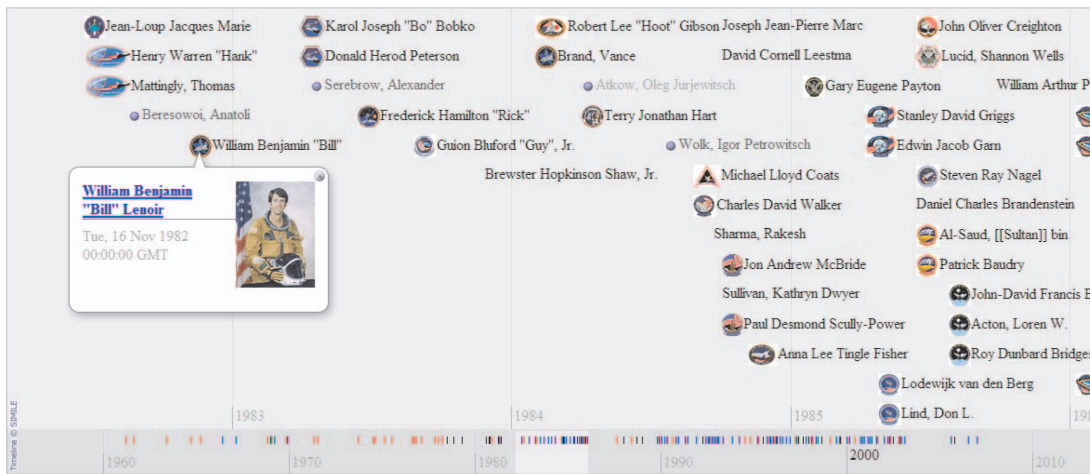


Fig. 3. A timeline of manned spaceflight. The data is extracted automatically from dbpedia and then displayed using the SIMILE timeline widget [6].

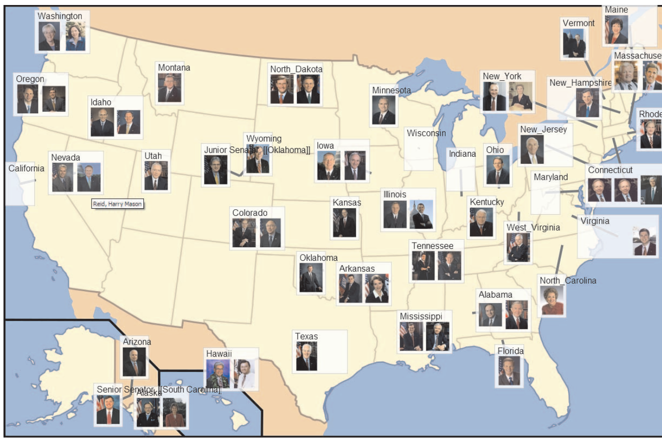


Fig. 2. A map marking states with their senators, based on data automatically extracted from dbpedia.

## 3 Formalism

This paper considers the following problem: given a set of objects and a visualization, find for each object the attributes required by the visualization. To define our problem formally, we first define the data model and the specification of a visualization.

### 3.1 Data Model

We model our data as a set of *object* instances. Objects have a set of *attributes*, each of which can take one or several values. Objects can also be linked with other objects by *associations*. A *class* represents a set of similar objects and summarizes the related attributes and associations.

With this model, we can view our data as a labeled directed graph. Specifically, each node in our graph corresponds to either an object or a literal. Edges from an object to a literal are attributes, and edges from one object to another are associations.

Note that this abstract data model is equivalent to that of RDF. Each subject–predicate–object triple in an RDF model corresponds to a directed edge from the subject to the object, labeled with the predicate.

### 3.2 Visualization Specification

We specify a visualization using a *schema* and an *encoding*. This approach for formalizing a visualization is based on the work of Bertin [9] and others [28, 34, 38]. Formally, a visualization is specified by a set of triples $\{(T_1,N_1,E_1),\ldots,(T_k,N_k,E_k)\}$, where for each $i \in [1,k]$, $(T_i,N_i,E_i)$ represents a *visualization attribute*: $T_i$ is the type of the attribute, $N_i$ is the name of the attribute, and $E_i$ is the visual encoding for the attribute. The encodings represent mappings to visual variables. Typical encodings include spatial position ($x$ and $y$), *color*, *size*, *text*, etc. In some cases encodings simply map fields onto parameterized templates; we express these encodings as *template.parameter*.

Only the $T_i$ and $N_i$ terms need to be considered by the search algorithm. The $E_i$ visual encodings are then applied to the results returned by the search. Each of our examples will be accompanied by a brief description of how the the visual encodings are implemented.

Note that the name used to specify a visualization attribute $N_i$ need not be a predicate appearing on the graph. This is a strength of our search-centric approach.

**Example 3.1.** First, consider the visualization used in Figure 2, where U.S. senators are shown on a map according to their state. The desired fields are the senators' pictures, and the names and geographical coordinates of their associated states. Accordingly, the visualization specification given to the search algorithm is:

$\{(decimal,$ state latitude, $y),$
$(decimal,$ state longitude, $x),$
$(string,$ name, $tooltip),$
$(Img,$ image, $icon),$
$(string,$ state, $text)\}$

The visual encodings are implemented using a javascript map component that positions each label and icon over a background map image at the specified geospatial coordinates. □

**Example 3.2.** Next, suppose we want to visualize the space race by plotting astronauts and cosmonauts on a timeline of their respective missions. We will use the SIMILE timeline component [6] to perform the visual encoding for this example. The following visualization specification describes the fields to search for and how to map them to the input parameters supported by the timeline component:

$\{(date,$ mission date, $timeline.start),$
$(Img,$ missioninsignia, $timeline.icon),$
$(string,$ name, $timeline.title),$
$(Img,$ image, $timeline.image),$
$(string,$ nationality, $timeline.color)\}$

Figure 3 shows a screenshot of the results obtained when we apply this visualization specification to the list of all astronauts. □

### 3.3 Satisfying Visualization Requirements

Given a set of object instances $\mathcal{N} = \{n_1,\ldots,n_l\}$ and a selected visualization $\{(T_1,N_1,E_1),\ldots,(T_k,N_k,E_k)\}$, our goal is to find for each object $n_i, i \in [1,l]$, the set of required inputs. Hence, to apply the visualization, we need to find $k$ paths for each of the objects in $\mathcal{N}$. Every such path translates into the node sequence followed from the object.

The path must terminate with an attribute leading to a value of the required type. Although edges in the graph are directed, we allow them to be traversed in either direction. Edges followed "backwards" will be prefixed by a caret in our path notation.

We need a mechanism to evaluate whether a candidate set of paths to attributes corresponds well to the requested set of visualization attributes. For example, suppose we have a visualization attribute ($string$, birthplacename, $E_i$), and we're visualizing a set of object instances for people including John Kerry and Edgar Allan Poe. For some object instances the path may be

dbp:PLACE_OF_BIRTH.dbp:official_name

while for others the path may be

dbp:birth_place.dbp:official_name.

Both of these paths should be ranked highly. On the other hand, a candidate visualization in which the path was:

foaf:spouse.dbp:birth_place.dbp:official_name

would be much less suitable.

We formally define the *visualization matching* problem as follows.

**Definition 3.3.** Let $o$ be an object instance and $\{(T_1, N_1, E_1), \ldots, (T_k, N_k, E_k)\}$ be a visualization. Visualization matching finds a tuple of paths $(p_1, \ldots, p_k)$, where for each $i \in [1, k]$,

- the path $p_i$ begins at the node that represents object $o$,
- the end of $p_i$ is a node whose type matches $T_i$,
- the path $p_i$ semantically matches $N_i$. □

The next section will describe the search and ranking algorithms. We will introduce several heuristics for assessing the quality of candidate paths by considering properties like branching factor and the discriminability of the literals.

## 4 Technique

A visualization specification defines a set of required parameters. For each object $o$ that we want to display, we must find paths that fit the requirements of the current visualization. Since many paths may match a visualization attribute, the main objective is to rank and filter out irrelevant paths and then return a result that best represents the remaining path choices for object $o$.

We propose a two-phase visualization matching algorithm designed to solve this problem.

The *path indexing* phase matches each visualization attribute to a set of path templates called *s*chema paths. These are sequences of predicates, which the next phase uses as a roadmap to find the attributes for each object. Working at this summary level increases efficiency on data sets with a provided schema and ensures a globally consistent ranking on paths. This consistency is important since using the same schema path for as many objects as possible produces a less confusing visualization than independently choosing different paths for each object.

The second phase, *instance matching*, takes an object and finds the best assignment of attributes from the database for the visualization attributes in the visualization specification. This search in the database is guided by the schema paths from the first phase. We describe several heuristics to handle multiple possible matches.

### 4.1 Path Indexing

Each attribute required for the visualization is specified by a name and a type. For each class of objects in the data sources and each visualization attribute, the path indexing process finds paths through the schema that end with the specified type and that semantically match the specified name. For data sets like dbpedia where building a concise schema is difficult we consider all the objects to be in the same class and use nearby paths up to a maximum distance for schema paths. In a way, this represents the schema induced by the subgraph around the objects.

Note that the paths generated at this stage do not have attribute values from individual object instances. They are only sequences of predicates that act as "path templates" for finding real attribute values later on. For this reason, we refer to these as *schema paths*.

Path indexing is split into two stages. First we find available schema paths, and then rank these by how well they fit the visualization specification.

#### 4.1.1 Path Enumeration

We generate schema paths for each attribute in the visualization specification and use a number of heuristics to prune away bad paths as early as possible.

When there is a schema available, it provides a summary of associations between classes. This permits an efficient enumeration of paths by following these associations. For dbpedia, the absence of a known schema and size of the database make the cost of generating all paths prohibitive. Instead, we enumerate schema paths based only on those instances involved in a visualization.

We will use the notation $C_0.A_1.C_1.\ldots.A_n.C_n.a$ for a path, where $C_i$ are classes, $A_i$ are associations, and $a$ is an attribute. For example, one possible path to a latitude for a Person is Person.birth_place.City.sameAs.Geoname.latitude. Note that for any given ($a$,$C$) pair, there may be zero, one, or many schema paths.

A number of heuristics help to filter away irrelevant paths. Given our intuitive preference for short, direct paths, the algorithm will consider only those paths that do not contain loops and are shorter than a specified length bound. We allowed a maximum path length of 4 for the U.S. senators visualization in example 3.1, and paths up to length 3 for all other examples. The longer the paths, the more time-consuming the search, so it is advantageous to avoid wasting effort serching overly-long paths.

In addition to the loop and path length constraints, we also limited the branching factor for each association in a chain. Associations that have a high branching factor describe one-to-many relationships, whereas we typically want to retrieve attributes that are functionally dependent on the initial objects. Consequently, only associations with low branching factor are followed. Specifically, only associations with a branching factor less than 4 were used. The choice of 4 as the maximum branching factor was ad hoc. Different tasks and data sets might benefit from a different value for this parameter.

#### 4.1.2 Path Ranking

We now rank the available schema paths against the name for each visualization attribute. This is handled as a word-matching problem between the name and the schema path.

Words are extracted from a path $C_0.A_1.C_1.\ldots.A_n.C_n.a$ by considering word delimeters and camel-casing. For example Person.placeOfBirth.latitude turns into the bag of words {place, of, birth, latitude}. Similarly a visualization attribute birth_latitude would translate into {birth, latitude}.

To determine how well a schema path matches a visualization attribute, we use two measures: The term frequency-inverse document frequency (TF/IDF) score [36] between the two word sets and the length of the schema path.

TF/IDF takes a set of search words (visualization attribute), and finds documents (schema paths) with many matching words. Frequent words like "of" and "the" have less influence on the score.

This score alone is insufficient, since longer schema paths repeating a word in the visualization attribute will tend to score higher, yet a shorter match is intuitively the more direct and likely correct answer. Thus we give shorter paths priority by scaling the TF/IDF score by the factor $\frac{l \cdot \alpha + 1}{l \cdot \alpha}$ where $l$ is the length of the schema path and $\alpha$ is a constant that decides much to increase the weight of shorter paths. For our examples, we found that longer paths rarely made better matches, so we used a strong bias of $\alpha = \frac{1}{4}$.

These two elements are combined to produce an overall score:

$$Score = S \cdot \frac{l \cdot \alpha + 1}{l \cdot \alpha}$$

□

### 4.2 Instance Matching

At query time, we have a set of object instances that we want to display using a particular visualization, and we need to find for each instance particular attribute values that satisfy the visualization specification.

Consider an instance $o$ and a visualization specification:

$$\{(T_1, N_1, E_1), \ldots, (T_k, N_k, E_k)\}.$$

Instance matching returns a tuple of paths $(p_1, \ldots, p_k)$, where for each $i \in [1,k]$, $p_i$ is a path starting from $o$ and ending with a value node of type $T_i$. The path $p_i$ should be an instance of the schema path proposed as a match to $(T_i, N_i)$ during path indexing. Note that the path indexing algorithm generally proposes multiple candidate schema paths for each attribute, and each path can yield multiple path instances for object $o$. Since we typically use only one value in the visualization, we need to choose the best value among the alternative combinations of instance paths. The score computed for each schema path by the path indexing algorithm is the main ranking criterion, but we now describe two additional heuristics.

#### 4.2.1 Ranking Function

**Majority-Rule Heuristic:** When multiple attribute values are reached along paths with equal scores, we can apply the *majority-rule* voting heuristic. For example, suppose multiple equally ranked paths from a Person instance to a latitude attribute yield the results {"49° 15' N", "37° 55' N", "49° 15' N"}. Given these alternatives, the majority rule heuristic suggests we should return "49° 15' N", the most frequently occurring value.

Specifically, consider a ranked list $L$ returned by path indexing. Let $P = \{p_1 \ldots p_m\}$ be the set of path instances whose corresponding schema paths have the same matching score in $L$. We denote by $v(p_i)$ the attribute value at the end of the path $p_i$, and by $|v(p_i)|$ the number of times value $v(p_i)$ appears in paths in $P$. We assign a *majority-rule score* $m$ to each path $p_i$:

$$m(p_i) = \frac{|v(p_i)|}{\max_{p \in P}(|v(p)|)}$$

**Common-Path Heuristic:** The second heuristic we have is the *common-path* heuristic. The path scores from the path indexing and the majority-rule heuristic apply to each path independently. However, there may be implicit dependencies between required attributes. In the following example we illustrate this sort of dependency.

**Example 4.1.** Consider the astronaut timeline example where an astronaut may have flown on multiple missions, each with a mission date and a mission insignia. Suppose we query this data source with the visualization specification:

$$\{(date, \mathsf{missiondate}, E_1), (Img, \mathsf{missioninsignia}, E_2)\}$$

The schema paths that might be used for these fields are:

dbp:mission.dbp:launch
dbp:mission.dbp:insignia

In the dbpedia dataset, good matches for both of these paths pass through a node representing the mission. Intuitively, for each object instance, we should prefer attribute values found at the end of paths that pass through the same intermediate node. Suppose the following four paths are present:

dbr:Buzz_Aldrin.mission.dbr:Gemini_12.launch="1966-11-11"
dbr:Buzz_Aldrin.mission.dbr:Gemini_12.insignia="Gemini_12_insignia.png"
dbr:Buzz_Aldrin.mission.dbr:Apollo_11.launch="1969-07-16"
dbr:Buzz_Aldrin.mission.dbr:Apollo_11.insignia="Apollo_11_insignia.png"

Then, the tuples ("1966-11-11", "Gemini_12_insignia.png") and ("1969-07-16", "Apollo_11_insignia.png") would be preferred to those which spuriously pair one mission's launch date with the other mission's insignia. This dependency between the attributes could not be made explicit in the visualization specification since we assume no knowledge of the schema. However, we apply a common-path heuristic to automatically discover this type of implicit dependency between attributes. We give a higher score to a tuple of attributes which were reached along paths that share intermediate nodes. □

We now formally define the *common-path score*. Consider a candidate matching result $(p_1, \ldots, p_k)$, where $p_1, \ldots, p_k$ are paths from instance $o$. The common-path score is proportional to the number of common intermediate nodes shared by the paths to each attribute. Specifically, let $I_i$ be the set of intermediate nodes in path $p_i$. We denote by $|I_i|$ the size of $I_i$. The common-path score $C$ is defined as follows:

$$C = \frac{\sum_i |I_i| - |\bigcup_i I_i|}{\sum_i |I_i|} + \varepsilon$$

Here, a small $\varepsilon > 0$ ensures that $C > 0$.

**Example 4.2.** Consider our running example and the candidate matching (lastname.*Poe*, birth_place.Boston.sameAS.4930956.latitude.42.3, birth_place.Boston.sameAS.4930956.longitude.−71.1). The first path contains intermediate node *Poe*, the second path includes intermediate nodes {Boston, 4930956}, and the third includes intermediate nodes {Boston, 4930956}. Thus, the common-path score for this candidate matching is $\frac{5-3}{5} + \varepsilon = 0.4 + \varepsilon$. However, for a candidate matching where the latitude and longitude are obtained through different City nodes, the common-path score would only be $\varepsilon$. □

Finally, the overall score for a candidate matching result $(p_1, \ldots, p_k)$ combines the matching scores of the corresponding schema paths and the scores computed according to the two heuristics.

Formally, we define $S = \prod_i [s(p_i)]$, where $s(p_i)$ is the matching score computed during path indexing for the schema path that $p_i$ corresponds to. We define $M = \prod_i [m(p_i)]$ as the overall majority-rule score, where $m(p_i)$ is the majority-rule score for path $p_i$. We define the common-path score $C$ as described above. The final score for a matching result $(p_1, \ldots, p_k)$ is computed as follows:

$$score(p_1, \ldots, p_n) = S \cdot M \cdot C$$

#### 4.2.2 Instance Matching Algorithm

Having outlined the ranking function we use, we now describe the algorithm in detail. Again, consider an instance $o$ of class $C$, and a visualization specification $\{(T_1, N_1, E_1), \ldots, (T_k, N_k, E_k)\}$ We proceed in two steps.

First, we process the fields of the visualization specification sequentially. For each visualization attribute, we consider the ranked list of schema paths generated by path indexing. For each schema path $p$, we query the database for path instances that originate from $o$ and match $p$. We begin with the schema paths with the highest matching scores, and proceed until we have processed schema paths with score $M$, where $M$ is the highest score with which a schema path can retrieve non-empty path instances for $o$. For each retrieved path instance, we compute the majority-rule score. Note that here we consider only the highest score $M$; a possible alternative is to consider the top-$k$ such scores.

Second, for each combination of the path instances, denoted by $\{p_1, \ldots, p_k\}$, we compute the final score and rank these candidate results accordingly. All tuples of matching results that tie for the highest score will be delivered to the visualization, subject to a cap on the maximum number of desired results per instance.

## 5 Experimental Results

The two main evaluation criteria are match quality and running time of the algorithm. We focussed more heavily on generating high quality paths, so there are some cases where our implementation runs slowly.

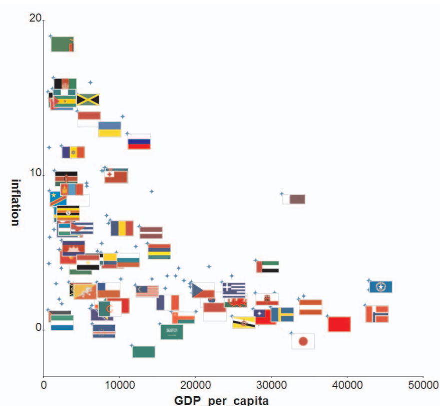


Fig. 4. A scatterplot of inflation versus GDP for countries in the dbpedia data, drawn using the dōjō [1] charting widget.

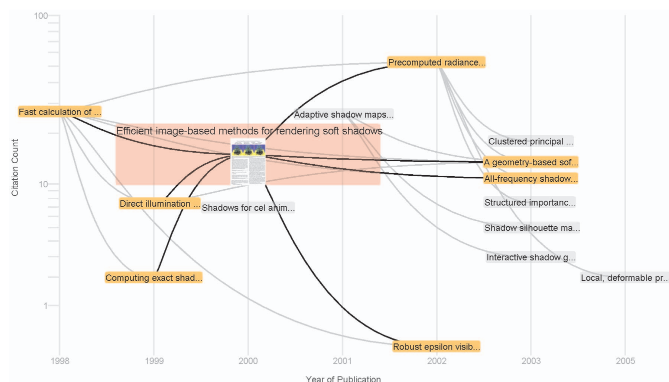


Fig. 5. A node-link diagram depicting the citation relationships among a set of papers. The diagram uses an attribute-based graph layout governed by papers' publication date and citation count.

Fig. 6. A version of the senators map annotated with confidence scores. The bar beneath each item encodes the aggregate score for its fields, with short red bars for low scores and wide green bars for high scores.

| | |
|---|---|
| No heuristics | 42 |
| Favor short paths | 51 |
| Limit branching factor of associations | 54 |
| No short literals as intermediate nodes | 58 |

Table 1. Number of senators for whom correct geocoordinates are obtained as successive heuristics are enabled.

### 5.1 Data and Test Cases

We used the dbpedia data set from March 2007 containing 18.9 million relations (either associations or attributes). It contains 2.3 million objects and 2.8 million distinct literals, with a total of 8,914 distinct types of relations. We were particularly interested in generating visualizations of people, places, and times from this noisy dbpedia data. For example, mapping senators by state, as in Figure 2, placing human visitors to outer space on a timeline of their missions (Figure 3), and plotting economic indicators for countries on a chart (Figure 4). The visualization specifications used to synthesize the map and timeline were given in section 3.2. For the chart, we used the following:

$\{(dollar, \text{GDP per capita}, x),$
$(percent, \text{inflation}, y),$
$(Img, \text{flag}, icon)\}$

The visual encoding for this example is implemented using a javascript scatterplot component from the dōjō toolkit [1]. Each flag image is positioned in $x$ and $y$ according to the GDP per capita and inflation fields, respectively.

In addition to the dbpedia corpus, we also applied our technique to an RDF description of the publication history of the ACM SIGGRAPH conference, which provides an interesting test case for visual analysis of citations. Figure 5 is a node-link diagram depicting the citation relationships among several papers. The visualization specification for this node-link diagram uses two queries, one to retrieve metadata about each paper to create, decorate, and place the nodes in the graph, and one to retrieve the edges connecting them. The input specification for the node query is

$\{(string, \text{title}, text),$
$(integer, \text{year}, x),$
$(integer, \text{citecount}, y),$
$(image, \text{image}, icon)\},$

and the input specification for the edge query is:

$\{(paper, \text{citedpaper}, edge)\}$

The visual encodings for this example are implemented in a Java applet built using prefuse [21]. It uses an attribute-based graph layout, in which year of publication determines a paper's position along the $x$-axis, while total citation count governs position on the $y$-axis. This layout was designed to facilitate visual discovery of the webs of influence embedded within the citation graph. The citation applet provides some brushing/selection interactivity as well. When a paper is selected, its full title and thumbnail image are shown, and the papers it cites or is cited by are highlighted.

Note that in choosing which search results to display, only the single highest scoring result is used for each paper node. Since there are generally multiple citations per paper, all equally high-scoring results for the edge specification are drawn as edges.

### 5.2 Match Quality

We will now more closely examine the paths chosen by our approach.

As Table 2 shows, multiple schema paths were used to match many of the more challenging visualization attributes. Finding and specifying such paths would have been difficult to do manually with any simple query. Among the senators, for example, we find that 7 different paths were used in different cases in order to obtain latitudes.

Although our technique does discover many good matches, there are many missing or incorrect attributes. We will examine causes for these issues relative to the senator example. In the map of senators in Figure 2 imperfect precision and recall are clearly evident. Several senators are omitted from the map entirely, usually because no path could be found leading to a latitude and longitude in decimal format. Also, incorrect state names are found for some senators. In these cases, the preferred paths to attributes failed to find any results, and lower ranking paths were used. Since each path used has a confidence score, we can provide a visual indication of the scores. For example, in Figure 6, a bar beneath the images of senators varies in width and color depending on the scores associated with their search results. Appropriate image urls were found for every senator. However, Wikipedia has undergone many updates since the dbpedia data was last collected, and some formerly valid image urls are no longer live.

| **Map of U.S. Senators (101 objects)** | | |
|---|---|---|
| image | dbp:image_name | 101 |
| name | dbp:name | 94 |
| | dbp:NAME | 9 |
| state | dbp:state | 67 |
| | dbp:jr_percent_2Fsr_and_state | 36 |
| state_latitude | dbp:state.r̂dfs:label.dbp:Capital.my:latitude | 31 |
| | dbp:state.ĝeo:name.geo:latitude | 22 |
| | d̂bp:preceded.dbp:state.dbp:LargestCity.my:latitude | 2 |
| | dbp:state.d̂bp:Name.dbp:Capital.my:latitude | 2 |
| | d̂bp:preceded.dbp:state.dbp:Capital.my:latitude | 1 |
| | d̂bp:preceded.dbp:state.dbp:LowestPoint.my:latitude | 1 |
| | dbp:state.d̂bp:Place.dbp:Latitude | 1 |
| state_longitude | dbp:state.r̂dfs:label.dbp:Capital.my:longitude | 32 |
| | dbp:state.ĝeo:name.geo:longitude | 22 |
| | d̂bp:preceded.dbp:state.dbp:LargestCity.my:longitude | 2 |
| | d̂bp:preceded.dbp:state.dbp:Capital.my:longitude | 1 |
| | d̂bp:preceded.dbp:state.dbp:LowestPoint.my:longitude | 1 |
| | dbp:state.d̂bp:Place.dbp:Longitude | 1 |
| | dbp:state.r̂dfs:label.dbp:LargestCity.my:longitude | 1 |
| **Scatterplot of Inflation (255 objects)** | | |
| flag | dbp:image_flag | 248 |
| | dbp:flag | 3 |
| | dbp:flag_p1 | 2 |
| | dbp:flag_s1 | 1 |
| GDP_per_capita | dbp:GDP_PPP_per_capita | 211 |
| | dbp:GDP_nominal_per_capita | 24 |
| | dbp:GDP | 1 |
| | dbp:GDP_per_capita | 1 |
| inflation | dbp:currency.dbp:inflation_rate | 104 |
| | d̂bp:using_countries.dbp:inflation_rate | 63 |
| **Timeline of Astronauts (710 objects)** | | |
| image | dbp:image | 386 |
| | dbp:image_name | 2 |
| mission_insignia | dbp:insignia | 382 |
| mission_launch | dbp:mission.dbp:launch | 317 |
| name | foaf:name | 307 |
| | dbp:name | 212 |
| | dbp:character_name | 12 |
| | dbp:English_name | 1 |
| | dbp:NAME | 1 |
| | dbp:Name | 1 |
| | dbp:full_name | 1 |
| nationality | dbp:nationality | 492 |
| | dborg:birthplace.d̂bp:birth_place.dbp:nationality | 2 |
| | dborg:deathplace.d̂bp:PLACE_OF_DEATH.dbp:nationality | 1 |
| | dborg:deathplace.d̂bp:birth_place.dbp:nationality | 1 |
| | dbp:placeofbirth.d̂bp:birth_place.dbp:nationality | 1 |

Table 2. Valid paths found for each of the dbpedia examples and the number of object instances that used each path.

In section 4, we proposed multiple heuristics for ranking the quality of paths. We now examine the utility of these heuristics by quantifying their effect on the accuracy of the geocoordinate values, as shown in Table 1 With all heuristics enabled, 58 of the senators were correctly assigned to a location in their home state.

### 5.3 Execution Time

Table 3 shows a summary of the execution times for the path indexing and instance matching phases. Searching for the long paths needed in the senator example is currently slow because we enumerate and then rank paths separately. The size of a complete enumeration grows exponentially with path length. Ideally, enumeration and ranking should be done together using a technique like an A* search, where low rankings predicted from partial paths can prevent further enumeration.

Other than the astronaut timeline, instance matching performs reasonably. However, for the astronaut case, there are many alternative paths for each field. Performance suffers here because we use a naive Cartesian product of the attribute matches from each field to evaluate the instance matching heuristics.

## 6 Related Work

User interfaces for databases have been approached from two main directions: helping users precisely and conveniently express their information needs, and helping users visualize query results.

| | enumerate paths (s) | rank paths (s) | match instances (s) |
|---|---|---|---|
| U.S. Senator map | 300 | 160 | 2.6 |
| inflation scatterplot | 18 | 4.9 | 0.43 |
| astronaut timeline | 20 | 18 | 131 |

Table 3. Execution times for each matching phase on the dbpedia test cases.

The first category focuses on interactive interfaces for query formulation and query refinement. VIQING [30] provides a visual interface for querying relational data. Lore [18] uses Dataguides to walk users through XML schemas for composition of XML queries. Other authors have considered keyword-based search over XML data [26, 22, 8]. CLIDE [31] offers an interactive interface for users to pose queries in a data-integration environment. Several papers [40, 37, 45] have considered how to provide integrative refinement of queries to view XML data, RDF data, and images. VisTrails [11] streamlines the process of creating multiple related visualizations by modeling workflows. Finally, Polyviou et al. [32] propose a visual query language that allows complex queries over heterogeneous data.

Our work falls in the second category, which studies the visualization of information. The visualization of unstructured data, such as documents, webpages, and tags, is studied in several papers [14, 17, 13, 15]. For structured data, Catarci et al. [12] surveyed a large number of systems that visualize relational data. Visionary [39], Visage [35] and DEVise [27] display query results on a canvas and provide powerful visualization features such as zooming, panning and distortion. Keim [24] surveyed the visualization of data from those that are one-dimensional and those that are multidimensional. XmdvTool [42], XGobi [10], and VisDB [25] studied how to visualize query results when they correspond to high-dimensional data. NUITS [41] displays the results of keyword search on relational data as a tree of tuples connected by foreign keys.

Of particular importance are formal models of visualizations. Much of the early work in this area was inspired by Bertin's *Semiology of Graphics* [9]. Bertin is credited with specifying visualization as relations and encodings. Mackinlay [28] developed APT, which formalized Bertin's methodology and showed how to optimize the choice of visual encodings. Sage [34] is a more recent system that develops a more elaborate data model for designing visualizations. DEVise [27] extends the formal approach using relational algebra and in particular develops methods for linking multiple views. Wilkinson's *Grammar of Graphics* [43] presents a very thoughtful and detailed design of a system for formalism the specification of visualizations. The VizQL language [20] further extends the method for specifying visualizations by including support for data cubes and table-based visualizations.

Finally, a number of systems have been designed to allow users to interactively define visualization of information. For example, in Polaris [38] users can specify the visualization of a data warehouse. In Haystack [23] users can specify the display of personal information (stored as RDF data) using RDF. However, these systems all make the assumption that the schema of the data is known *a priori*.

Our work is different from the above in that we consider the visualization of loosely-coupled heterogeneous data. Our system treats visualizations as first-class citizens and the visualizations are specified independently of the data sources. To display a set of objects, possibly from disparate data sources, we perform run-time schema matching to select attributes that best match the visualization specifications. Schema matching has been studied extensively (see Rahm and Bernstein [33] for a survey). However, our problem domain, where visualization requirements motivate the reorganization of semi-structured data, poses several novel challenges.

First, we perform matching between two different kinds of attributes. Visualization attributes always have names and types; Attribute paths from the database always have names but may be missing types, as in the dbpedia dataset. More importantly, attribute paths have a large set of corresponding data instances, while visualization at-

tributes usually have no such examples. Second, we choose to match each visualization attribute to an *attribute path*, generated by following a sequence of associations. This allows desired data to be found indirectly via associated objects, but significantly increases the number of possible matches. Third, we match at the instance level. Object instances may have different types of attributes and associations, thus the matching results will also differ.

Many techniques for improving schema matching can be used with our approach In particular, a corpus-based approach [29] can handle synonyms in addition to exact word matches.

## 7 Conclusions and Future Work

Visualization and data management are interrelated. When data comes from multiple sources and is highly heterogeneous, much of the initial interaction with it will be exploratory. Users need to see the data in order to even formulate appropriate queries. We described techniques for deeply integrating automatic searching within a visualization pipeline. This is a new way to approach the problem of visualizing heterogeneous data. We introduced a mechanism for describing visualizations independently of the data in the sources, and an algorithm for retrieving the appropriate data for a given visualization. Our initial experiments have shown that our system is able to find the appropriate data often enough to be useful as an exploratory tool, while sheltering users from schema heterogeneity.

We note two ways this work might be extended. The first is to automatically select the visualization that is most appropriate for a given set of objects from a library of alternatives. The second is to integrate visualization and querying—using the query to give us additional hints for selecting appropriate visualizations and using the visualization to help the user formulate the next query. Such interactive query formulation would work well with a more flexible query specification, which could include preferred paths or path components.

## Acknowledgements

This material is based on work supported by SRI International/DARPA contract #55-000679 TO-5, Battelle Memorial Institute/US DOE under contract #15597-3, and Boeing under contract #SPO 35845-1.

## References

[1] dōjō javascript toolkit. `http://dojotoolkit.org/`.
[2] Information commons. `http://www.maya.com/infocommons/`.
[3] Maya Viz. `http://www.mayaviz.com/web/concepts/downloads/viz_comotion_overview.pdf`.
[4] Programmableweb. `http://www.programmableweb.com/`.
[5] Resource description framework (rdf). `http://www.w3.org/RDF/`.
[6] Simile timeline. `http://simile.mit.edu/timeline/`.
[7] S. Auer, C. Bizer, R. Cyganiak, O. Erling, K. Idehen, G. Kobilarov, J. Lehmann, and J. Schüppel. `http://dbpedia.org/`.
[8] A. Balmin, V. Hristidis, N. Koudas, Y. Papakonstantinou, D. Srivastava, and T. Wang. A system for keyword search on xml databases. In *VLDB (demo)*, pages 1069–1072, 2003.
[9] J. Bertin. *The Semiology of Graphics*. Univ. of Wisconsin Press, 1984.
[10] A. Buja, D. Cook, and D. F. Swayne. Interactive high-dimensional data visualization. *Computational and Graphical Statistics*, 5(1):78–99, 1996.
[11] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. VisTrails: Visualization meets data management. In *Sigmod*, pages 745–747, 2006.
[12] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. Visual query systems for databases: a survey. *Journal of Visual Languages and Computing*, 8(2):215–260, 1997.
[13] J. Chen, L. Sun, O. R. Zaiane, and R. Goebel. Visualizing and discovering web navigational patterns. In *WebDB*, pages 13–18, 2004.
[14] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. In *SIGIR*, pages 318–329, 1992.
[15] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. Visualizing tags over time. In *WWW*, pages 193–202, 2006.
[16] M. Franklin, A. Halevy, and D. Maier. From databases to dataspaces: A new abstraction for information management. *Sigmod Record*, 34(4):27–33, 2005.
[17] G. W. Furnas and S. J. Rauch. Considerations for information environments and the NaviQue workspace. *INEX Workshop*, pages 79–88, 2003.
[18] R. Goldman and J. Widom. Interactive query and search in semistructured databases. In *WebDB*, pages 52–62, 1998.
[19] GoogleBase. `http://base.google.com/`, 2005.
[20] P. Hanrahan. VizQL: A language for query, analysis and visualization. In *Sigmod*, page 721, 2006.
[21] J. Heer, S. K. Card, and J. A. Landay. prefuse: a toolkit for interactive information visualization. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430, 2005.
[22] V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on xml graphs. In *International Conference on Data Engineering*, pages 367–378, 2003.
[23] D. R. Karger, K. Bakshi, D. Huynh, D. Quan, and V. Sinha. Haystack: A general-purpose information management tool for end users of semistructured data. In *CIDR*, pages 13–26, 2005.
[24] D. A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 7(1):1–8, 2002.
[25] D. A. Keim and H.-P. Kriegel. VisDB: Database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications*, 14(5):40–49, 1994.
[26] Y. Li, C. Yu, and H. V. Jagadish. Schema-free xquery. In *VLDB*, pages 72–83, 2004.
[27] M. Livny, R. Ramakrishnan, K. S. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and R. K. Wenger. DEVise: Integrated querying and visualization of large datasets. In *Sigmod*, pages 301–312, 1997.
[28] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, 1986.
[29] J. Madhavan, P. A. Bernstein, A. Doan, and A. Halevy. Corpus-based schema matching. In *ICDE*, pages 57–68, 2005.
[30] C. Olston, M. Stonebraker, A. Aiken, and J. M. Hellerstein. VIQING: Visual interactive querying. In *VL*, pages 162–169, 1998.
[31] M. Petropoulos, A. Deutsch, and Y. Papakonstantinou. Interactive query formulation over web service-accessed sources. In *Sigmod*, pages 253–264, 2006.
[32] S. Polyviou, G. Samaras, and P. Evripidou. A relationally complete visual query language for heterogeneous data sources and pervasive querying. In *ICDE*, pages 471–482, 2005.
[33] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.
[34] S. F. Roth, J. Kolojejchick, J. Mattis, and J. Goldstein. Interactive graphic design using automatic presentation knowledge. In *CHI '94: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 112–117, New York, NY, USA, 1994. ACM Press.
[35] S. F. Roth, P. Lucas, J. A. Senn, C. C. Gomberg, M. B. Burks, P. J. Stroffolino, J. A. Kolojejchick, and C. Dunmire. Visage: A user interface environment for exploring information. In *Information Visualization*, pages 3–16, 1996.
[36] G. Salton, editor. *The SMART Retrieval System—Experiments in Automatic Document Retrieval*. Prentice Hall, Englewood Cliffs, NJ, 1971.
[37] V. Sinha and D. R. Karger. Magnet: Supporting navigation in semistructured data environments. In *Sigmod*, pages 97–106, 2005.
[38] C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis and visualization of multidimensional relational databases. *IEEE Transaction on Visualization and Computer Graphics*, 8(1):52–65, 2002.
[39] M. Stonebraker. Visionary: A next generation visualization system for data bases. In *Sigmod*, page 635, 2003.
[40] A. Trigoni. Interactive query formulation in semistructured databases. In *FQAS*, pages 356–369, 2002.
[41] S. Wang, Z. Peng, J. Zhang, L. Qin, S. Wang, J. X. Yu, and B. Ding. NUITS: A novel user interface for efficient keyword search over databases. In *VLDB*, pages 1143–1146, 2006.
[42] M. O. Ward. XmdvTool: Integrating multiple methods for visualzing multivariate data. In *Visualization*, pages 326–333, 1994.
[43] L. Wilkinson and G. Wills. *The Grammar of Graphics*. Springer, 2005.
[44] J. Wong and J. I. Hong. Making mashups with marmite: Towards end-user programming for the web. In *CHI '07: Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1435–1444. ACM Press, 2007.
[45] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *CHI*, pages 401–408, 2003.