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, 37]. 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 [1]. The classic example is Wikipedia which is being created by thousands of people around the world. Another example is GoogleBase [15]; 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.

Semantic web technologies like the Resource Description Framework (RDF) 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 are not 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. This makes it possible to automatically create visualizations.

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, but 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. Our examples will simply be based on sets of object instances drawn from a Wikipedia category page.

We compare our technique to 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 attach a display method with each class. For example, most object-oriented programming languages include a method to convert

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

Manuscript received 31 March 2007; accepted 1 August 2007; posted online 2 November 2007.

For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

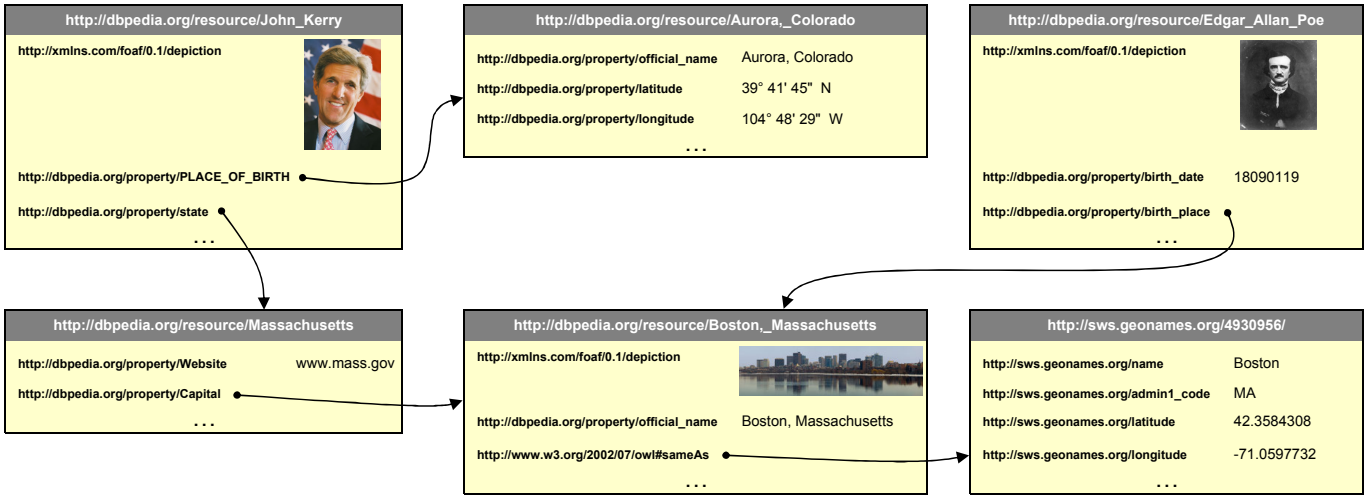

Fig. 1. A small portion of the dbpedia RDF graph illustrating the heterogeneity of representations for people and places. Each box in the diagram depicts an object and several of its literal valued attributes. Associations between objects are shown by arrows.

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 associated with a particular task.

A more flexible method for specifying visualizations can be found in the *Maya Viz* system [2], 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 contexts involving heterogeneous data, it may also make sense to use fields that aren't direct attributes of an object, but rather belong to a related object reached via a sequence of associations. Consider our example in Figure 2, which draws on dbpedia data to depict 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, nor would it make sense for the data object describing a person to have such an attribute. However, if the user has explicitly requested to see senators on a map, it can be reasonable to infer geolocation attributes from the associated object for their home state. Our technique supports automatically inheriting needed attributes from associated objects when the visualization requires it.

As noted, our approach of tightly integrating search into visualization is intended to support casual exploration of unfamiliar data sets. The focus is on providing "best-effort" retrieval of the properties needed for a specific visualization. We propose multiple heuristics to guide this search. Nevertheless, the search is not expected to have perfect precision or recall, so missing or incorrect values must be expected. 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).

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.

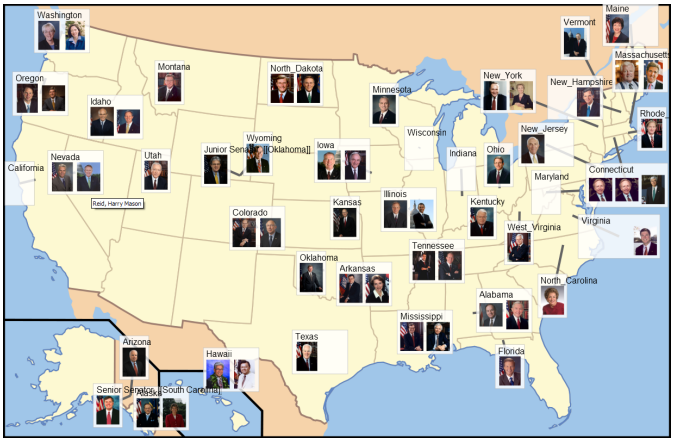

Fig. 2. A map marking states with their senators, based on data in dbpedia.

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.

Figure 1 shows a small portion of the dbpedia RDF graph consisting 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. Note for example that 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, min-

utes, 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.

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 [6] and others [21, 27, 31]. 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 are *x*, *y*, *color*, *size*, etc.

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. The visual encodings we use are generally provided by particular visualization widgets like the SIMILE timeline component [5] or our javascript-based U.S. map component. We will henceforth omit the E_i fields from the visualization specifications shown in the paper, with the understanding that standard toolkits can be used to provide this functionality.

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),
(*decimal*, state longitude),
(*string*, name),
(*Img*, image),
(*string*, state)} □

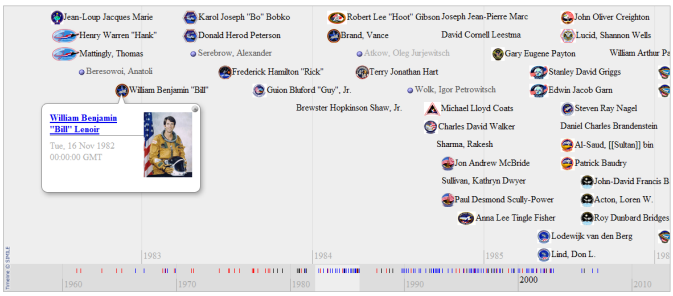

Fig. 3. A timeline of manned spaceflight, shown using the timeline widget from the MIT SIMILE project [5].

Example 3.2 *Next, suppose we want to visualize the space race by plotting astronauts and cosmonauts on a timeline of their respective missions. The fields wanted for the visualization can be specified by:*
{(*date*, mission date),
(*Img*, insignia),
(*string*, name),
(*Img*, image),
(*string*, nationality)}
Figure 3 shows the results obtained by our search algorithm when applying 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 $\{(T_1,N_1),\ldots,(T_k,N_k)\}$, 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 to reach the required input. 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), 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 $\{(T_1,N_1),\ldots,(T_k,N_k)\}$ 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 Tt_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 will fullfill the requirements of the current visualization. This section describes our algorithm, which draws upon techniques from the data integration community.

Schema matching has been studied extensively (see [26] for a survey). However, our problem domain, where visualization requirements motivate the reorganization of semi-structured data, poses several novel challenges. First, the information we have for visualization specifications and that for the source schemas are unbalanced: for attributes in the visualization specifications, we only know their names and types; but for those in the source schemas, we know their names, sometimes their types and constraints, and more importantly, we have a large set of data instances. 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 be found indirectly via associated objects, but significantly increases the number of possible matches. Third, we want to visualize objects from different data sources together. Object instances from different sources may have different types of attributes and associations, and even instances from the same source may have missing attributes and associations. Thus, the matching results will differ from instance to instance.

We propose a two-phase visualization matching algorithm. The *path indexing* phase matches each visualization attribute to a set of candidate attributes in the data sources. The *instance matching* for an object finds an assignment of specific attributes from the database to the requested visualization attributes that best fulfills the visualization specification. The assigned attribute instances can either be direct attributes of the object instance, or attributes of an associated object.

4.1 Path Indexing

Each attribute required for the visualization is specified by a name and a type. For each class in the data sources and each visualization attribute, the path indexing process finds paths through the source schemas that end with the specified type and that semantically match the specified name. Note that paths generated at this stage do not have attribute values from individual object instances. They are indeed "path templates" and at runtime we apply them to each object and retrieve path instances with real attribute values. To distinguish them from paths in the graph, we call such path templates *schema paths*. Indexing proceeds in five steps.

Step 1: We begin by clustering attributes in the data sources into attribute groups, which we call *attribute concepts*, such that within each group the attributes are semantically related to each other. To obtain this goal, we can apply existing techniques that match a corpus of schemas, such as [22], and then cluster attributes that match each other. We denote by $con = \{a_1, \ldots, a_m\}$ an attribute concept that contains attributes $a_1, \ldots, a_m$. For example, birth_place and PLACE_OF_BIRTH from the sample dbpedia biographical entries (see Figure 1) can be clustered into an attribute concept.

Step 2: We generate schema paths for each data attribute with respect to each class in the schema. Specifically, consider an attribute a and a class C. If a is an attribute of C, the path is a direct one and written as $C.a$. For example, the path of birth_date w.r.t. Person is Person.birth_date. If a is an attribute of a different class C_n and C_n is directly or indirectly associated with C, then the path is an indirect one and written as $C.A_1.C_1.\ldots.A_n.C_n.a$, where $A_1, \ldots, A_n$ are associations, $C_1, \ldots, C_n$ are classes, and $C.A_1.C_1.\ldots.A_n.C_n$ is an *association chain* from C to C_n. For example, one possible path of latitude w.r.t. 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. We consider only those paths that do not contain loops and are shorter than a specified bounded length.

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. 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.

Recall that we allow paths to traverse edges in the RDF graph in either direction. In particular, we permit paths that use literals as intermediate nodes, like:
Person.dbp:name.string.∧foaf:name.birth_place
Allowing paths of this type can be very effective for discovering associations between multiple objects (potentially from different sources) that correspond to the same entity. However, there is also a potential to introduce large numbers of spurious paths if the literal nodes occuring as intermediates are not sufficiently discriminating. Multiple objects referencing the literal value "Edgar Allan Poe" may be related – but the fact that multiple objects reference the numeral "1" is unlikely to be significant. For our implementation, we only allow strings with a length greater than 4 characters to serve as intermediate nodes on a path. Shorter strings, numeric values, and dates are only allowed as terminal attributes.

Step 3: We now index each attribute concept with respect to a class. Specifically, given a class C and an attribute concept $con = \{a_1, \cdots, a_m\}$, we generate the bag of words for con w.r.t. C, denoted as $B(con, C)$, as follows. For each $a_i, i \in [1, m]$, if the schema path of a_i w.r.t. C is not empty, we add every term on the paths from a_i to C, except the name of C itself, into B. We thus have an index of (con, C) pairs, each of which is indexed on words in $B(con, C)$. In our example, we index the concept {birth_place, PLACE_OF_BIRTH} w.r.t. class Person as a bag of words {birth birth place place of}, and index the concept {latitude} w.r.t. class Person as a bag of words containing {birth place city same as geoname latitude}, plus words from other schema paths connecting Persons to latitudes. In many cases attribute names may contain misspellings or abbreviations. To be more tolerant of such cases, we index the n-grams of each term on the path. For example, if we index 3-grams, the concept {birth_place} w.r.t. class Person is indexed on the bag of words {bir irt rth th_ h_p _pl pla lac ace}. Note that a concept can be indexed multiple times, each corresponding to a specific class, and because the attribute paths with respect to these classes are different, the concept is indexed on different bags of words.

Step 4: For each class C and each visualization attribute v, we use the index from Step 3 to generate a list of schema paths as matching options. To do this we first match the visualization attribute to a set of attribute concepts by looking up in the index the terms of v (or the n-grams of the terms if we index n-grams). For example, for the visualization attribute (location, birth) and the class Person, we look up "birth" in the index (or "bir irt rth" if we index 3-grams). Among the concepts returned, we only consider those that are paired with C. For each attribute in such a concept con, if the type of the attribute matches the type of v, we include all paths of the attribute w.r.t. C in the returned list of paths.

Step 5: Finally, we rank schema paths based on two measures: TF/IDF score [29] and the length. Specifically, for each attribute path p in concept con, we compute the matching score as follows:

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

Here, S is the TF/IDF score of the concept con. To favor short paths, we multiply S with the factor $\frac{l \cdot \alpha + 1}{l \cdot \alpha}$, where l is the length of the path p and α is a constant that decides the weight of the length.

Not surprisingly, path indexing provides the best result when the attribute concepts generated in Step 1 are of high quality. However, using our algorithm we can often generate matching plans that are meaningful, and for data sources that are important, the users can manually refine the results.

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:

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

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 alternatives. The score computed for each schema path by the path indexing algorithm is the main ranking criterion, but we now describe two additional heuristics that have a major impact on the quality of our results.

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 a conference-publication data source, where each* Paper *is associated with multiple* AuthorshipNode*s, each having attributes* authorName *and* institution*. Suppose we query this data source with the visualization specification:*

$$\{(\mathsf{string},\mathsf{name}),(\mathsf{string},\mathsf{institution})\}$$

The paths that might be used for these fields are:

Paper.author.AuthorshipNode.authorName
Paper.author.AuthorshipNode.institution

Note that the paths leading to both authorName *and* institution *pass through an* AuthorshipNode*. 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:*

Paper104.author.AuthorshipNode297.authorName="Joe Smith"
Paper104.author.AuthorshipNode297.institution="Oxbridge"
Paper104.author.AuthorshipNode298.authorName="Jane Williams"
Paper104.author.AuthorshipNode298.institution="Camford"

Then, the tuples ("Joe Smith", "Oxbridge") *and* ("Jane Williams", "Camford") *would be preferred to those which spuriously pair each author with the other's institution. 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* ε*.* □

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:

$$score(p_1,\ldots,p_n) = S \cdot M \cdot C^{\beta}$$

Here, β is a parameter that controls the importance of the majority-rule heuristic relative to the common-path heuristic.

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

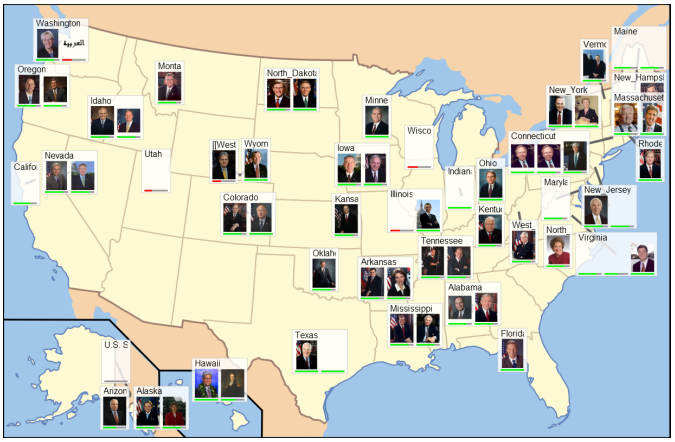

Fig. 4. A version of the senators map annotated with confidence scores. The bar beneath each item encodes the aggregate score for its fields, with short red bars for low scores and wide green bars for high scores. The path scores tend to match our subjective assessment: clearly misplaced or mislabelled entries have low scores, while correct entries have high scores.

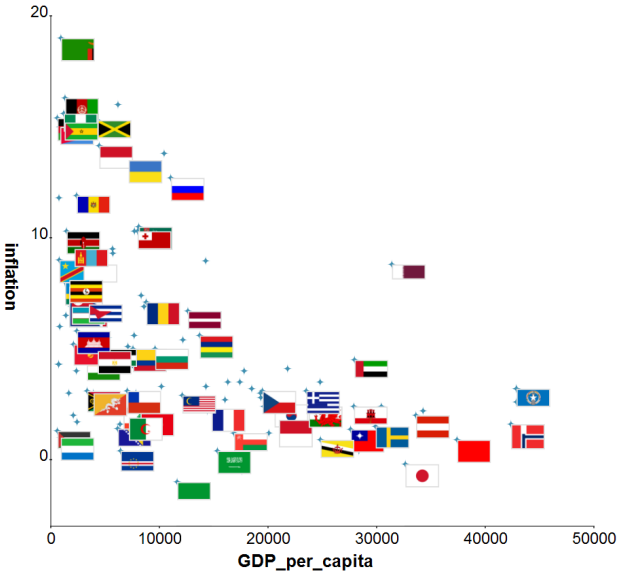

Fig. 5. A scatterplot of inflation versus GDP for countries in the dbpedia data, drawn using the dōjō [3] charting widget.

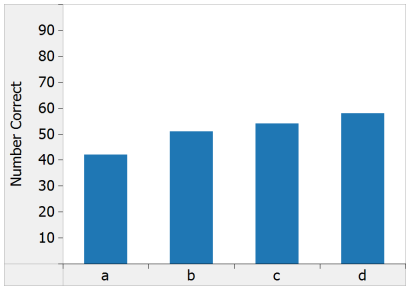

Fig. 6. Graphs of the number of senators for whom correct geocoordinates are obtained as successive heuristics are enabled. A. No heuristics. B. Favor short paths. C. Limit branching factor of associations. D. Don't allow short literals as intermediate nodes.

5 Experimental Results

The techniques described in this paper are intended to be applicable to a variety of RDF datasets. Because of the emphasis on coping with missing data and schema heterogeneity, we choose to perform most of our evaluations using the dbpedia data, which exhibits these qualities. The portion of dbpedia we used consisted of 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. Searching for attribute names containing the string "birth" reveals at least 12 different attribute names all apparently describing dates of birth. There is also great variability in which attributes are available. There are 70 different kinds of relations observed that originate with one of the U.S. senators, but each senator uses only 27 of them, on average, with some using as few as 15 and some as many as 41.

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 5). The visualization specifications used to synthesize the map and timeline were given in section 3.2. For the chart, we used the following:
{(*dollar*, GDP per capita),
(*percent*, inflation),
(*Img*, flag)}

Shortcomings in precision and recall are evident in both diagrams. 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 photos and 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 have the visualizations reflect the quality of the search results by providing visual indicators of the scores. For example, in Figure 4, a bar beneath the images of senators varies in width and color depending on the scores associated with their search results.

With all heuristics enabled, 58 of the senators were correctly assigned to a location in their home state. 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 assigned by our algorithm are no longer live. Among the correctly-placed senators, we find that 7 different paths were used in different cases in order to obtain latitudes. In section 4, we described a number of heuristics for choosing and ranking paths. To evaluate their effect on the quality of results, we show the number of correct geolocations found for the 100 senators.

In addition to the dbpedia corpus, we also applied our technique to an RDF description of the publication history of the ACM SIGGRAPH conference. Figure 7 is a node-link diagram depicting the citation relationships among papers. The node-link diagram applet needed two input specifications, one to retrieve metadata about each paper to decorate the nodes in the graph, and one to retrieve the edges connecting them. The input specification for nodes is:
{(string,title),
(integer,year),
(integer,citecount),
(image,image)}
and the input for edges is:
{(paper,citedpaper)}
Only the single highest scoring result is used for each paper node. Since there are multiple citations, all equally high-scoring results for the edge specification are drawn as edges.

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 effectively visualize and understand query results.

The first category focuses on interactive interfaces for query formulation and query refinement. VIQING [23] provides a visual in-

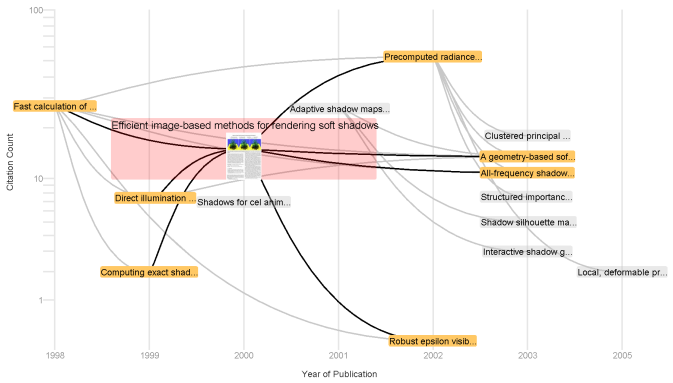

Fig. 7. A node-link diagram depicting the citation relationships between a set of papers. The data used for this example was an RDF description of all publications in the ACM SIGGRAPH conference.

terface for querying relational data. Lore [14] uses Dataguides to walk users through XML schemas for composition of XML queries. CLIDE [24] offers an interactive interface for users to pose queries in a data-integration environment. In [33, 30, 38] the authors studied how to provide integrative refinement of queries to view XML data, RDF data, and images. VisTrails [8] streamlines the process of creating multiple related visualizations by modeling workflows. Finally, [25] proposes a visual query language that allows the formulation of 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 [11, 13, 10, 12]. For structured data, Catarci et al. [9] surveyed a large number of systems that visualize relational data. Visionary [32], Visage [28] and DEVise [20] display query results on a canvas and provide powerful visualization features such as zooming, panning and distortion. Kaim [18] surveyed the visualization of data from those that are one-dimensional and those that are multidimensional. XmdvTool [35], XGobi [7], and VisDB [19] studied how to visualize query results when they correspond to high-dimensional data. NUITS [34] 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* [6]. Bertin is credited with specifying visualization as relations and encodings. Mackinlay [21] developed APT, which formalized Bertin's methodology and showed how to optimize the choice of visual encodings. Sage[27] is a more recent system that develops a more elaborate data model for designing visualizations. DEVise [20] extends the formal approach using relational algebra and in particular develops methods for linking multiple views. Wilkinson's *Grammar of Graphics* [36] presents a very thoughtful and detailed design of a system for formalism the specification of visualizations. The VizQL language [16] 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 [31] users can specify the visualization of a data warehouse. In Haystack [17] 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 work 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.

7 Conclusions and Future Work

Visualization and data management are interrelated. When data comes from multiple sources and is highly heterogeneous, much of the intial 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 and automatically filling in some incomplete data.

We note several ways this work might be expanded upon in the future. The first is to automatically select from a visualization library the visualization that is most appropriate for a given set of objects. The second is to better integrate visualization and querying—using the visualization to help formulate the next query, and using the query to give us additional hints for selecting appropriate visualizations.

References

[1] Information commons. `http://www.maya.com/infocommons/`, 2005.

[2] Maya Viz. `http://www.mayaviz.com/web/concepts/downloads/viz_comotion_overview.pdf`, 2006.

[3] dōjō javascript toolkit. `http://dojotoolkit.org/`, 2007.

[4] Programmableweb. `http://www.programmableweb.com/`, 2007.

[5] Simile timeline. `http://simile.mit.edu/timeline/`, 2007.

[6] J. Bertin. *The Semiology of Graphics*. Univ. of Wisconsin Press, 1984.

[7] A. Buja, D. Cook, and D. F. Swayne. Interactive high-dimensional data visualization. *Computational and Graphical Statistics*, 5(1), 1996.

[8] S. P. Callahan, J. Freire, E. Santos, C. E. Scheidegger, C. T. Silva, and H. T. Vo. VisTrails: Visualiztion meets data management. In *Sigmod*, 2006.

[9] 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.

[10] J. Chen, L. Sun, O. R. Zaiane, and R. Goebel. Visualizing and discovering web navigational patterns. In *WebDB*, 2004.

[11] 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.

[12] M. Dubinko, R. Kumar, J. Magnani, J. Novak, P. Raghavan, and A. Tomkins. Visualizing tags over time. In *WWW*, 2006.

[13] G. W. Furnas and S. J. Rauch. Considerations for information environments and the NaviQue workspace. In *INEX Workshop*, 2003.

[14] R. Goldman and J. Widom. Interactive query and search in semistructured databases. In *WebDB*, 1998.

[15] GoogleBase. http://base.google.com/, 2005.

[16] P. Hanrahan. VizQL: A language for query, analysis and visualization. In *Sigmod*, 2006.

[17] 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*, 2005.

[18] D. A. Keim. Information visualization and visual data mining. *IEEE Transactions on Visualization and Computer Graphics*, 7(1), 2002.

[19] D. A. Keim and H.-P. Kriegel. VisDB: Database exploration using multidimensional visualization. *IEEE Computer Graphics and Applications*, 14(5):40–49, 1994.

[20] 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.

[21] J. Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, 1986.

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

[23] C. Olston, M. Stonebraker, A. Aiken, and J. M. Hellerstein. VIQING: Visual interactive querying. In *VL*, pages 162–169, 1998.

[24] M. Petropoulos, A. Deutsch, and Y. Papakonstantinou. Interactive query formulation over web service-accessed sources. In *Sigmod*, 2006.

[25] 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.

[26] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[27] 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.

[28] 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.

[29] G. Salton, editor. *The SMART Retrieval System—Experiments in Automatic Document Retrieval*. Prentice Hall Inc., Englewood Cliffs, NJ, 1971.

[30] V. Sinha and D. R. Karger. Magnet: Support navigation in semistructured data environments. In *Sigmod*, 2005.

[31] 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), 2002.

[32] M. Stonebraker. Visionary: A next generation visualization system for data bases. In *Sigmod*, 2003.

[33] A. Trigoni. Interactive query formulation in semistructured databases. In *FQAS*, pages 356–369, 2002.

[34] 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*, 2006.

[35] M. O. Ward. XmdvTool: Integrating multiple methods for visualzing multivariate data. In *Visualization*, pages 326–333, 1994.

[36] L. Wilkinson and G. Wills. *The Grammar of Graphics*. Springer, 2005.

[37] 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*. ACM Press, 2007.

[38] K.-P. Yee, K. Swearingen, K. Li, and M. Hearst. Faceted metadata for image search and browsing. In *CHI*, 2003.