Towards Large-Scale Data Discovery [Position Paper]

Raul Castro Fernandez, Ziawasch Abedjan, Samuel Madden, Michael Stonebraker

MIT, CSAIL

raulcf@csail.mit.edu, abedjan@mit.edu, madden@csail.mit.edu, stonebraker@csail.mit.edu

ABSTRACT

With thousands of data sources spread across multiple databases and data lakes, modern organizations face a *data discovery* challenge. Analysts spend more time finding relevant data to answer the questions at hand than analyzing it.

In this paper we introduce a data discovery system that facilitates locating relevant data among thousands of data sources. We represent data sources succinctly through *signatures*, and then create *search paths* that permit quick execution of a set of data discovery primitives used for finding relevant data. We have built a prototype that is being used to solve data discovery challenges of two big organizations.

1. INTRODUCTION

Modern organizations organize their data into thousands of heterogeneous databases and data lakes that are managed by different teams and departments. In such an environment, a data analyst is faced with the problem of finding which datasets contain the data needed to answer their questions. As data processing systems become ever faster, analysts will spend more and more of their time finding *relevant data* to answer the question at hand than the spend actually analyzing it!

Finding relevant data is difficult because no single person in an organization knows about all the data sources [8]. Suppose an analyst wants to answer the question: *what is the monthly sales trend by department?* The analyst knows conceptually what data is needed to answer this question (a table of sales, a table of departments, a table of products sold by each department), but not which specific data sources (relations in a schema or files in an HDFS deployment) contain such data. The typical solution is to 1) ask an expert (if such a person is available), or to perform manual exploration, inspecting datasets one by one (which is time-consuming and prone to missing relevant data sources). We say this analyst is facing a **data discovery challenge**.

Data discovery challenges are widespread in the enterprise and are typically solved with one of the two following top-down approaches: (i) designing a global schema that represents all the data; or (ii) creating metadata to describe the data to enable analysts to search for the data they need. An example of the first case is a data warehouse: database administrators carefully design a schema that integrates different sources fed into the warehouse and makes them usable to other users. An example of the second case is a search service that allows users to specify different search criteria based on characteristics of the original data. For example, a website might provide a search bar to search over the names and departments of the people in an organization.

The problem with these top-down approaches is that neither of them scales to a large number of sources. Designing a global schema of data sources is hard; in fact, most data warehouses integrate only a handful of sources [5]. Assuming complete knowledge of user's discovery needs and designing search filters data prevents ad-hoc queries and is not robust to changes in the discovery needs. Therefore, top-down solutions are brittle to changes in the discovery queries and impossible to use for ad-hoc discovery needs.

In this paper we describe a data discovery system built *bottom-up*. Our system consists of three key components:

1. We first accumulate knowledge of the data sources by summarizing them with **signatures**: domain-dependent, compact representations of the original data.
2. We create **search paths** over the signatures that reflect different relatedness measurements and allows one to navigate the sources, such as similarity measures between data sources.
3. Finally we expose the built knowledge through a set of **data discovery primitives** that can be used interactively to explore and find relevant data.

We propose a set of discovery primitives that can solve data discovery problems at scale. To enable large-scale and low-latency execution of these primitives we present a system that integrates a signature extraction module to create signatures from original data and a search path creation module that allows the discovery primitives to execute efficiently. We handle domain specific data by integrating new signatures that capture domain knowledge. For cases in which the search needs are domain-specific, users can define new primitives and functions based on the provided search paths.

To realize this vision we face a number of technical challenges:

- **Data discovery interface.** What is the minimum number of primitives that will cover most discovery needs? A small number is desirable as each primitive requires a different search path, which is expensive to build.
- **Data signatures.** To cope with the large volumes of data, we need to summarize it. We propose to create *signatures* that capture the underlying data in a compact way. The challenge is to build signatures that are small and sufficiently expressive even in the presence of dirty data, because in the general case it is unrealistic to clean all data beforehand.
- **Large scale** Many of the search paths we envision are represented as graphs with nodes as signatures and edges that represent the

ExploreDB'16, June 26-July 01 2016, San Francisco, CA, USA

DOI: http://dx.doi.org/10.1145/2948674.2948675

different relationships among signatures, such as similarity, overlap, etc. In general, building such graphs requires a pairwise comparison that grows as the square of the attributes and data sources available. The challenge is to find alternative ways to build search paths in a scalable manner.

We recognize that the challenges above are hard to solve in the general case. For example, domain-specific discovery scenarios such as gene databases or large-scale simulations demand tailor-made solutions to capture user needs. We do not aim to provide a general solution for every data discovery problem, but instead a framework for data discovery that can be adapted quickly to specific domains, much as a relational database offers schema management and processing capabilities without constraining the applications that can use it or the specific implementation of the relational operators or search paths.

We have built a prototype system that is being used to solve data discovery problems from different clients (we can demonstrate this prototype at the workshop). We have a collaboration with the MIT data warehouse team that supports around 1TB data divided among approximately 2000 tables, and with a big pharma company with more than 4000 relational databases.

The rest of the paper provides an overview of an initial set of data discovery uses cases that we have identified by talking to customers and retrieved from the literature (section 2). Then we explain our approach to building bottom-up search paths to assist in discovery tasks in section 3. Finally we contextualize our ideas with respect to the related work in section 4.

2. DATA DISCOVERY INTERFACE

We have demonstrated our prototype data discovery system to many organizations, and have an active collaboration with the the MIT data warehouse (MIT DWH) team and a big pharma company to evaluate the prototype in their environment. Next, we describe the two discovery use cases and the interface we envision for discovery.

2.1 Discovery Use Cases

Discovery problems aim to narrow down the search for relevant data from *many* data sources to a handful of them. These two use cases are examples of such goal:

View Search. The MIT DWH contains data from the entire organization organized into around 2,000 different source tables and a few hundred views. Customers of the DWH pose questions such: "I'd like to know all faculty or staff pay scales at MIT grouped by department and research area". Currently, the managers of the DWH follow a manual process to find tables relevant to answer the question, and offer them to the customer.

Schema Complement. A big pharma company has found a view with a set of attributes that are relevant to a business question they have. They want to find additional attributes that would complement the already existent data in the view. The problem they face is to find those attributes across the 4,000 relational databases they manage.

2.2 Seeds and Primitives

Users interact with the discovery system through *seeds* that are input information of interest to them, invoking *primitives* that use seeds to find other information they do not know.

Seeds. We have identified several different seeds users may use to trigger the discovery process:

- **Value seed.** Users kick off the process with a known value, such as a keyword. This is useful to get a general overview of the data.
- **Attribute seed.** Users know an attribute of interest and they want to find attributes that overlap or that are similar, e.g., find attributes similar to faculty names.
- **Schema seed.** Users want to perform a search over the schema available. This is useful when names are unambiguous or have been annotated with descriptive labels.
- **Table seed.** When users have determined a table of interest, they may use it as a seed to find other related tables—with overlapping schemas or similar attributes—that can be used to augment the information available in the original one.

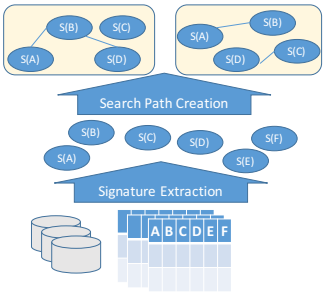

Figure 1: Bottom-up approach to data discovery

Primitives. We have also identified an initial set of primitives useful to solve the use cases we have presented before. These primitives can be used with the previous seeds to find additional useful data.

- **Schema-driven primitives.** These primitives can be used to search the available schemas or to enrich them with additional labels extracted from external systems, e.g., knowledge bases.
- **Data-driven primitives.** These primitives can be used to discover relationships amongst data, e.g., whether two attributes are similar or overlap.
- **Enrichment primitives.** These primitives permit the user to perform entity and schema complement operations, i.e., adding additional records or attributes to a schema.

The idea behind the primitives is to offer basic functionality that can be composed into *discovery functions* that perform more complex operations. For example, a function that computes the join path between two tables, or a function to find all attributes of a given type in the dataset. In addition, we support logical operators to combine results from different primitives and operators, allowing more specialized discovery queries.

3. ARCHITECTURE

A data discovery system must answer queries in human-scale latencies to permit interactive exploration. For this, each primitive uses a search path that is built over signatures. The general system architecture shown in Fig. 1 shows a *signature extraction* and *search path creation* module that together can be used to achieve the low-latency goal. Next we explain both modules using as an example of one of the discovery primitives we have used to solve the use cases of section 2.1.

3.1 Finding relevant data

To solve the use cases presented above we have implemented a primitive called **find similar attributes** that takes an *attribute seed* as input and returns other attributes in the entire dataset that are *similar*. This is a *data-driven primitive*, as it needs to inspect the data values. This primitive is useful when the schema name for two similar attributes is different, such as "Year" and "Fiscal Period"—useful for view search—and to find *similarity join* candidates which is useful for schema complement.

For this primitive to work we need a similarity score across all attributes in the entire dataset. Unfortunately, inspecting every

individual value of each attribute is at odds with our low-latency goal. Instead, we use *signatures* to summarize the values of each attribute first, and *search paths* to speed up the querying process:

Signature extraction. The signature extraction module of Fig. 1 is responsible for computing signatures for each attribute in the dataset. A signature for **finding similar attributes** must: (i) represent the attribute values in a compact way; and (ii) be compatible with a similarity metric. Currently we have primitives for numerical and textual values. For numerical values the module learns the probability distribution of the data. We use non-parametric methods for this, as it is in general impossible to tune the learning process for every attribute in the dataset due to the scale. We use kernel density estimators and we are also exploring one-dimensional support vector machines, which are robust to noise: a desirable property when learning a signature over dirty data. For textual values we first represent each attribute as a bag of words and then compute its TF-IDF vector.

Numerical signatures can then be compared in one of two different ways. One is with statistical distances such as Kolmogorov-Smirnov, that determine the distance between two different sample distributions. An additional method we are currently testing consists of transforming probability distributions into vectors of sample values that can be represented in an Euclidean space, where other non-statistical distance metrics are defined. For textual values we use cosine similarity that yields good results due to the sparsity of the TF-IDF vectors.

Search path creation. Despite the existence of signatures, we would still need to compute the similarity between the seed attribute and all the others. To avoid this, we rely on search paths that facilitate such similarity search. The search path for the primitive we consider is represented as a graph where nodes are signatures and there exists an edge between every pair of signatures weighted with the similarity score of the attributes.

Although this search path creates a logically complete graph, in practice it is possible to define a minimum similarity threshold to prune edges. Executing the primitive means finding the node that represents the seed attribute in the graph (we use an indexed representation of the graph for this) and returning its neighbors, that can be ranked by their weight score. We can additionally pass a parameter that defines a new threshold to further reduce the returned results on runtime.

We have found that search paths based on graphs support many of the primitives we have implemented so far. In particular, those primitives that require finding some relatedness among attributes.

3.2 Additional Challenges

Support for domain-specific data. Certain data can be represented with domain-specific signatures. Gene sequences may be better represented with their descriptors than with the original sequences. Highly structured text can be represented with regular expressions. The challenge is to incorporate an appropriate *feature engineering* engine. Our current approach is to specify a signature per primitive and data type, along with a signature extractor that is included into the signature extractor module.

Taming the scale. Building the graph search path requires a pair-wise operation among all attributes in the entire dataset, a N^2 operation. To scale this process, we are introducing approximate nearest neighbor techniques based on locality sensitive hashing [4] that helps to scale the process. In addition, we have built a distributed prototype that can ingest data and compute signatures and search paths in parallel.

Continuous operation. As new data sources appear, we want to add them incrementally to the search paths, so that the discovery system can include them in the answers to discovery primitives. A related challenge is to detect when the values of a given attribute have changed enough to justify recomputing the signature, or alternatively maintaining the signatures incrementally.

4. RELATED WORK

Data discovery as IR. Data discovery over large scale repositories is related to the central problem of information retrieval: how to find a ranked list of *objects* relevant to a search query. Previous work has recognized this analogy. In [6], the authors describe a retrieval-style ranked search system for oceanographic data. The system relies on metadata specific to the domain, such as initial and end observation time of the data, and proposes a similarity metric to compare user-input criteria with this metadata. An analogous method is used for genomic data search in [7]. These approaches aim to find relevant data within a corpus of domain-specific datasets. Unlike them our vision is a data discovery system that helps analysts to find relevant data among large number of heterogeneous datasets. By following a bottom-up approach we incorporate knowledge from the original data sources through our *signature extraction* module and use it to broaden the search possibilities.

Finding relevant data. The task of finding relevant data is present in multiple scenarios. In the context of webtables [2], researchers have proposed ways of locating tables related to the ones provided by users (table seed as described in this paper) [3]. Other research has focused on searching with numbers [1], which resembles our *find similar attributes* primitive with a numerical attribute seed Unlike these approaches, our focus is on providing a platform that integrates different seeds and primitives that users need to solve a broad set of discovery use cases. This means that our platform strives for generality, although many of the techniques presented in previous work can be incorporated into our platform.

5. CONCLUSIONS

We have described a bottom-up approach to data discovery, a challenge found in every middle and large size organization. By defining signatures on the source data and creating search paths that structure the signatures, users can execute efficiently a set of data discovery primitives that permit them to find relevant data.

As part of our ongoing effort we are: (i) incorporating signatures for domain-specific data; (ii) additional primitives to generate metadata automatically from data; (iii) primitives to capture structural similarity of different attributes across the dataset; and (iv) adding support for unstructured data, among others.

6. REFERENCES

[1] R. Agrawal and R. Srikant. Searching with Numbers. In *WWW*, 2002.

[2] M. J. Cafarella, A. Halevy, et al. WebTables: Exploring the Power of Tables on the Web. *VLDB*, 2008.

[3] A. Das Sarma, L. Fang, et al. Finding Related Tables. In *SIGMOD*, 2012.

[4] M. Datar, N. Immorlica, et al. Locality-sensitive Hashing Scheme Based on P-stable Distributions. In *SCG*, 2004.

[5] A. Halevy, A. Rajaraman, et al. Data Integration: The Teenage Years. In *VLDB*, 2006.

[6] V. M. Megler and D. Maier. Are Data Sets Like Documents?: Evaluating Similarity-Based Ranked Search over Scientific Data. In *TKDE*, 2015.

[7] V. M. Megler, D. Maier, et al. Data like this: Ranked search of genomic data vision paper. In *ExploreDB*, 2015.

[8] I. Terrizano, P. Schwarz, et al. Data Wrangling: The Challenging Journey from the Wild to the Lake. In *CIDR*, 2015.