Keyword Search over Data Service Integration for Accurate Results

Vidmantas Zemleris[1], Valentin Y Kuznetsov[2] and Peter Kreuzer[3]

[1]Faculty of Mathematics and Informatics, Vilnius University, Lithuania
[2]Cornell University, USA
[3]Rheinisch-Westfaelische Tech. Hoch., Germany

E-mail: [1]vidmantas.zemleris@cern.ch

Abstract. Virtual data integration aims at providing a coherent interface for querying heterogeneous data sources (e.g., web services, web forms, proprietary systems) with minimum upfront integration effort. Data is usually accessed through structured queries, such as SQL, requiring to learn the language and to get acquainted with data organization, which may pose problems even to proficient users. We present a keyword search system, which proposes a ranked list of structured queries along with their explanations. It operates mainly on the metadata, such as the constraints on inputs accepted by services. It was developed as an integral part of the CMS data discovery service, and is currently available as open source.

1. Introduction

Virtual Data Integration (VDI) is a lightweight approach to integrating data from "multiple sources without having to first load data into a central warehouse"[1, page 1]. Instead, data physically stays at its origin, and is requested on demand. The query is parsed, transformed and sent to relevant services. Then, the service responses are transformed including elimination of inconsistencies in data formats or in entity naming, and finally combined. Still, that forces its users to learn the query language and to get acquainted with data organization, which is often not straight-forward when where is no direct access to the data at the data-services. In this work, we explore keyword search, as a more intuitive alternative to writing structured queries from scratch. This subject in fact, received relatively little attention in the field of data integration[2].

Unfortunately/

We will present our implementation of a keyword search system, which proposes a ranked list of structured queries. Query suggestion operates mainly on the metadata[1], such as the constraints on inputs accepted by services. It was developed at the *CMS Experiment, CERN* where it makes part of an open-source data integration **tool** called *Data Aggregation System (DAS)*[3, 4]. DAS integrates a **dozen** of services with the largest storing over 700GB of relational data. DAS has no predefined schema, thus only minimal mappings are needed to describe differences among the services. It uses simple structured queries formed of an entity to be retrieved and some selection criteria. The results could be further filtered, sorted or aggregated:

system re-peats!

[1] without contacting any of services before user has chosen the correct query suggestion

As seen in the example above, the query language closely correspond to the physical execution flow[2], which allows users to explicitly specify the operations to be performed and is motivated by large data volumes managed by services.

2. Problem definition

Given a keyword query, KWQ=$(kw_1, kw_2, .., kw_n)$, we are interested in translating it into a ranked list of best matching structured queries. We are given this metadata:

- *schema terms*: entities and their attributes (*inputs* to the services or their *output* fields)
- *value terms*: for some fields a list of values, but for most only *constraints* on data-service inputs (mandatory inputs, regular expressions defining values accepted).

3. Mapping keywords into structured query suggestions

3.1. Overview of our implementation

Firstly, the query is *tokenized* cleaning up the query and identifying any explicit quoted phrase tokens, operators, or other structural patterns.

Then, employing a number of entity matching techniques, the "*entry points*" are identified: for each keyword, we obtain a listing of schema and value terms it may correspond to along with a rough estimate of the likelihood.

Lastly, different permutations of *entry points* are evaluated combining the scores of individual keywords and applying the contextual rules to boost scores of *query interpretations* that "respect" the likely dependencies between the nearby keywords. During the same step, the *interpretations* not compatible with the data integration system are filtered out. Then these are interpreted as structured queries, disambiguating the keyword matchings between: result types, projections, selections (filters on service inputs or outputs), or some simple operators.

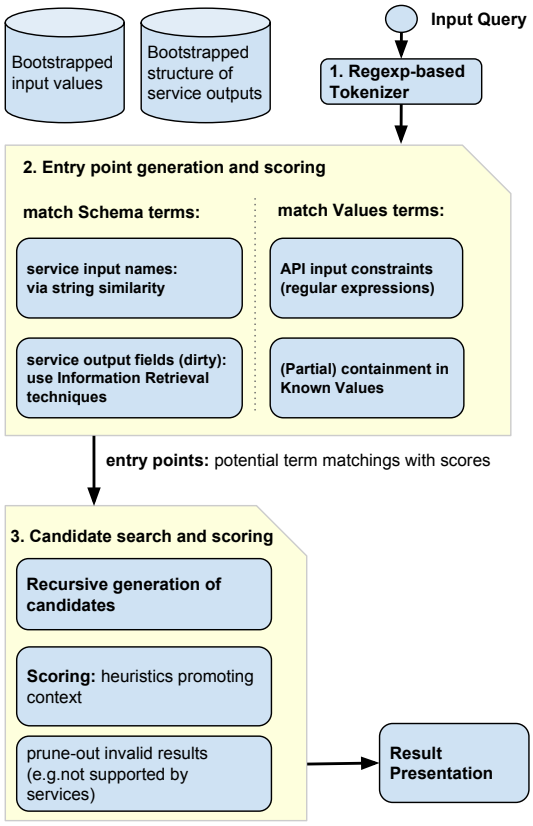

Figure 1. Keyword query processing

Example. Consider this query: `RelVal 'number of events > 1000'`. Tokenizer would return these tokens: `'RelVal'; 'number of events>1000'`. Then, each token result in some entry points:

```
'RelVal' → (1.0, input-value:  group=RelVal)
'RelVal' → (0.7, input-value:  dataset=*RelVal*)
'number of events>100' → (0.93, output-filter:  dataset.nevents>100)
'number of events>100' → (0.93, output-filter:  file.nevents>100)
and more with lower scores...
```

It can be seen that the *'RelVal'* keyword is ambiguous , while *'number of events'* could be attribute of *dataset, file* or *block*. Combining the entry point scores yields the shown results in fig. 2.

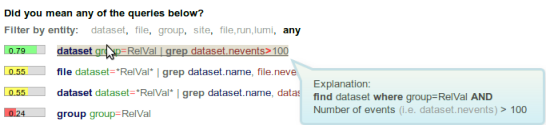

Figure 2. Results - query suggestions

[2] based on the requested entity and the conditions on service inputs, DAS decides the set of services to be queried. Then, after retrieving, processing and merging the service responses, the filters, projections, and aggregators are applied and the results are cached for subsequent uses.

3.2. Scoring individual keywords (step 2)

Matching the value terms DAS mappings contain regular expressions (regexp) that describe the input values accepted by services. As some of regexp's could be loosely defined, this can result in false positives, thus, the regexp matches are scored lower than others, distinguishing different levels of regexp's accuracy. For some schema terms, a list of possible values, that were obtained by bootstrapping them through respective data service interfaces. Different cases of matches are distinguished: full match, partial match, and matches of keywords containing wildcards (in order of decreasing score). If a value matches a regexp, but is not contained in the known values list and the values listing of the given field is considered to be changing not often, we exclude this very likely false match to reduce the false positives.

shorten down!

Matching the schema terms: TODO This also includes identifying keyword chunks corresponding to multi-word terms (output fields in service results): many of them are unclean, machine-readable field-names, with irrelevant and frequent terms, motivating the use of IDF-based information retrieval techniques.

TODO: describe in more details the crappy stuff we have

3.3. The ranker and scoring functions (step 3)

As the last step, different combinations of the entry points have to be explored combining the scores of individual keywords in some way (e.g. summing of log likelihoods, averaging, described below). We experimented with two scoring functions: 1) first one averaging the scores, as used by Keymantic[5], and 2) summing the log likelihoods which is closer to probabilistic [**thinking**]. In the beginning two methods seemed to perform almost equally well**,** with the probabilistic approach being more sensitive to **inaccuracies** in scoring of the entry points, but it became clearly better than *averaging* when entry points accuracy was slightly improved[3].

$$averaging\ score(tags, kwq) := \frac{\sum_{kw_i \subset kwq} \left(score_{tag_i|kw_i} + \sum_{h_j \in H} h_j(tag_i|kw_i; tag_{i-1,..,1}) \right)}{N\_non\_stopword} \quad (1)$$

$$likelihood\ score(tags, kwq) := \sum_{kw_i \subset kwq} \left(\log\left(score_{tag_i|kw_i}\right) + \sum_{h_j \in H} h_j(tag_i|kw_i; tag_{i-1,..,1}) \right) \quad (2)$$

Notation: $score_{tag_i|kw_i}$ is the likelihood of matching an individual keyword kw_i as tag_i (an entry point); $h_j(tag_i|kw_i; tag_{i-1,..,1})$ denotes the score boost returned by contextualization heuristic h_j given a concrete tagging so far (or last few tags). The *probabilistic* approach uses a set of "fake" tags with predefined scores: unmapped, unmapped stop word.

Heuristics and contextual rules in ranking. As part of entry points:

- balance between taking the keyword or leaving it out (the one that we are unsure about)
- rare use-cases: prevent schema terms to be mapped to values with high score, as this is unlikely (e.g. dataset names include 'dataset')
- boost important keywords (different parts of speech are of different importance, e.g. stop-words are less useful than nouns)

Keymantic assumed all keyword have interpretation! only stop-words are distinguished

[3] the scores are just estimates of our confidence, not real probabilities; the results improved with improvements to accuracy of string matching functions and removed the unreliable semantic matching

Contextualization rules:

- promoting such combinations where nearby keywords refer to related schema terms (e.g. entity name and it's value)

3.4. Miscellanea

Automatically identifying the qualities of data services The data integration system, DAS, uses only minimal mappings - they describe services, their input parameters, and mappings between inconsistently named output fields. Any other information, such as the complete listing of fields in the service outputs, or their types are identified by processing results of historical queries. To get satisfactory coverage immediately, a list of bootstrapping queries is used to initialize the most important field and value listings.

4. Related works

Keyword Querying. Keyword queries are ambiguous and often underspecified[6], thus the standard approach is to produce a ranked list of most-likely structured query suggestions[7, 5, 8] for the user to choose from[4]. Second, even if keyword queries have no clear syntactic structure, keywords referring to related concepts usually come close to each other [9, 10], this can be used to improve the ranking.

Keymantic [6, 5] answers keyword queries over relational databases with limited access to the data instances (including data integration). First, based on meta-data, individual keywords are scored as potential matches to *schema terms* (using various entity matching techniques) or as potential *value* matches (by checking any available constraints, such as the regular expressions imposed by the database or data-services). Next, to obtain the **global ranking,** they consider the "min-cost weighted bipartite matching" problem (of keywords into their tags) extended with weight contextualizations (i.e. conditional increase of scores for those keyword mappings where the nearby keywords have obtained related labels). Finally, these labels are interpreted as SQL queries. To cope with contextualization the internal steps of Munkres algorithm have been modified, the presumptive implications of this change are discussed in the following section.

[[[**Note: Their assumptions:** "keyword can be mapped to only one database term; no two keywords can be mapped into the same database term [**how about multi-keyword-terms?**]. **every keyword plays some role in the query, i.e., there are no unjustified keywords (!)**" **it seems it was summing the scores (no log()!) Weighted-bipartite matching as optimization; still exponencial because of first step?**]]]

KEYRY [8] attempted to incorporate users feedback by training an Hidden Markov Model's (HMM) tagger taking keywords as its input. It uses the List-Viterbi [11] algorithm to produce the top-k most probable tagging sequences (where tags represent the "meaning" of each keyword). This is interpreted as SQL queries and presented to the users. The HMM is first initialized through the supervised training, but even if no training data is available, the initial HMM probability distributions can be estimated through a number of heuristic rules (e.g. promoting related tags). Later, user's feedback can be used for supervised training, while even the keyword queries itself can serve for unsupervised training [12]. According to [8] the accuracy of the later system didn't differ much from Keymantic.

also mention: NL querying over services

String and entity matching

- string distances; schema matching
- other matching: google distance,
- methods - "relevant values"

[4] as query execution can be quite expensive, the suggestions are shown to the user, instead executing immediately.

5. Discussion and Future work

TODO: Discuss differneses from our implementation

Keymantic which is the closest work

HMM and what's modelled

On weighted bipartite matching extended with contextualizations blah blah

Our proposed algorithm?

6. Conclusions

References

[1] Halevy A Y, Ashish N, Bitton D, Carey M, Draper D, Pollock J, Rosenthal A and Sikka V 2005 Enterprise information integration: successes, challenges and controversies *Proceedings of the 2005 ACM SIGMOD international conference on Management of data* SIGMOD '05 (New York, NY, USA: ACM) pp 778–787 ISBN 1-59593-060-4 URL http://doi.acm.org/10.1145/1066157.1066246

[2] Guerrisi V, La Torre P and Quarteroni S 2012 *Search Computing* 82–97

[3] Kuznetsov V, Evans D and Metson S 2010 *Procedia Computer Science* **1** 1535 – 1543 ISSN 1877-0509 iCCS 2010 URL http://www.sciencedirect.com/science/article/pii/S1877050910001730

[4] Ball G, Kuznetsov V, Evans D and Metson S 2011 *Journal of Physics: Conference Series* **331** 042029 URL http://stacks.iop.org/1742-6596/331/i=4/a=042029

[5] Bergamaschi S, Domnori E, Guerra F, Orsini M, Lado R T and Velegrakis Y 2010 *Proc. VLDB Endow.* **3** 1637–1640 ISSN 2150-8097 URL http://dl.acm.org/citation.cfm?id=1920841.1921059

[6] Bergamaschi S, Domnori E, Guerra F, Trillo Lado R and Velegrakis Y 2011 Keyword search over relational databases: a metadata approach *Proceedings of the 2011 international conference on Management of data* (ACM) pp 565–576 URL http://dl.acm.org/citation.cfm?id=1989383

[7] Blunschi L, Jossen C, Kossmann D, Mori M and Stockinger K 2012 *Proc. VLDB Endow.* **5** 932–943 ISSN 2150-8097 URL http://dl.acm.org/citation.cfm?id=2336664.2336667

[8] Bergamaschi S, Guerra F, Rota S and Velegrakis Y 2011 *Conceptual Modeling–ER 2011* 411–420

[9] Kumar R and Tomkins A 2009 *IEEE Data Engineering Bulletin* **32** 3–11

[10] Bergamaschi S, Domnori E, Guerra F, Lado R T and Velegrakis Y 2011 Keyword search over relational databases: a metadata approach *Proceedings of the 2011 international conference on Management of data* (ACM) pp 565–576

[11] Seshadri N and Sundberg C 1994 *Communications, IEEE Transactions on* **42** 313–323

[12] Rota S, Bergamaschi S and Guerra F 2011 The list viterbi training algorithm and its application to keyword search over databases *Proceedings of the 20th ACM international conference on Information and knowledge management* (ACM) pp 1601–1606