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

Abstract. Virtual data integration provides a coherent interface for querying heterogeneous data sources (e.g., web services, proprietary systems) with minimum upfront effort. Still, this require its users to learn the query 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[1] approach to integrating heterogeneous data sources where data physically stays at its origin, and is requested only on demand. Queries are interpreted and sent to relevant services, those responses are integrated eliminating inconsistencies in data formats and entity namings, 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, especially in data-services case - without direct access to the data.

In this work, we present a keyword search system which proposes a ranked list of structured queries with their explanations. This operates "offline" using metadata such as 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)*[1, 2]. DAS integrates a **dozen** of services, where the largest stores 700GB of relational data. DAS has no predefined schema, thus only minimal service 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. Optionally, the results can be further filtered, sorted or aggregated, e.g:

DAS query language closely correspond to the physical execution flow allowing users to be aware of it and explicitly to specify the operations to be performed and is largely motivated by vast data volumes managed by services. Keyword search relaxes the need to know internal details, but still allows to be aware of it while reviewing the query suggestions.

[1] i.e. publish-subscribe is not applicable to proprietary (reluctant to change) systems, data-warehousing is too complex when large portions of data are volatile or when only limited interfaces are provided 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 (or their combination), we obtain a list of schema and value terms it may correspond to and 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. In the same step, the *interpretations* not compatible with the data integration system are pruned out as early as possible.

Example. Consider the following query:
`RelVal 'number of events'> 1000.`
First, tokenizing results in these tokens: `'RelVal'; 'number of events>1000'`. Then, each token or their combination may 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, filter: dataset.nevents>100)
'number of events>100'→(0.93, filter: file.nevents>100)
...
```

It can be seen that both *'RelVal'* and *'number of events'* keywords are ambiguous. Combining the entry point scores yields the results in fig.2.

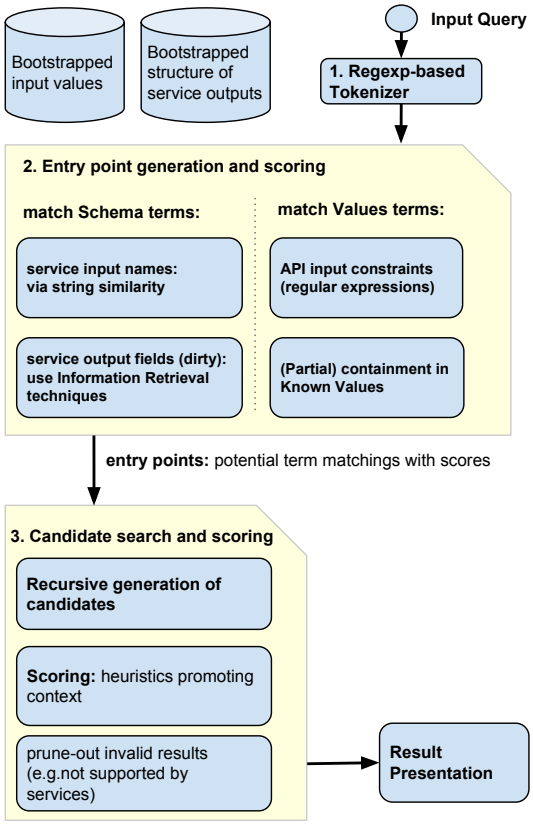

Figure 1. Keyword query processing

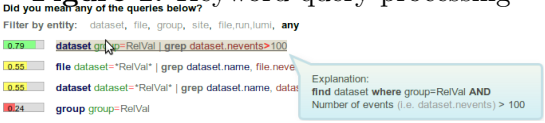

Figure 2. The query suggestions

3.2. Scoring individual keywords (step 2)

In this step possible keyword interpretations are identified and assigned scores between $[0, 1]$.

Matching the value terms DAS service 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 multiple levels of regexp's accuracy.

For some fields, a list of possible values is available. Different cases are distinguished: full match, partial match, and matches containing wildcards. If a value matches a regexp, but is not contained in the known values list and the field is considered to be changing not often, this likely false match is excluded.

Matching the schema terms Our experience is that basic string-edit distance metrics, such as the standard *Levenshtein* edit-distance (where inserts, edits, and mutations are equal) being designed for general matching tasks, do not perform well in our case, introducing too many false-positives (e.g. 'file' for 'site'). Instead, we use a simple combination of more trustful metrics: full match, lemma match, stem-match, and finally a stem match within a very small edit distance.

$$sim(A,B) = \begin{cases} 1, & if\ A = B \\ 0.9, & if\ lemma(A) = lemma(B) \\ 0.7, & if\ stem(A) = stem(B) \\ 0.6 \cdot dist(stem(A), stem(B)), & otherwise \end{cases}$$

Above *dist* is a tight string distance function (e.g. max 1-3 characters differing in beginning or end, max 1 mutation/transposion).

Matching the names of fields in service outputs We also identify chunks of keywords 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)

In this step, different combinations of the entry points are explored and ranked. The final score is obtained by combining the individual keywords scores in some way. We experimented with two scoring functions: 1) averaging the scores, as used in *Keymantic*[3], and 2) summing the log-likelihoods. At first the two methods seemed to perform almost equally well**,** with the probabilistic approach being more sensitive to **inaccuracies** in entry points scoring, but it became clearly better than *averaging* when entry points accuracy was slightly improved[2].

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

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

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)

[2] the scores are just estimates of our confidence, not real probabilities; the results improved with improvements to accuracy of string matching functions and removal of 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. 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 queries. They are ambiguous and often underspecified[4], thus the standard approach is to produce a ranked list of most-likely structured query suggestions[5, 3, 6]. Second, even if keyword queries have no clear syntactic structure, keywords referring to related concepts usually come close to each other [7, 8], this can be used to improve the ranking.

4.1. Keyword search over data-services

Keymantic [4, 3] 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 *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" (of keywords into their tags) problem 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.

KEYRY [6] attempted to incorporate users feedback by training an Hidden Markov Model's (HMM) tagger taking keywords as its input. It uses the List-Viterbi [9] 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 [10]. According to [6] the accuracy of the later system didn't differ much from Keymantic.

4.2. String and entity matching

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

String and Entity Matching provides the possible interpretations of individual keywords, as entry points for further processing. From the fields of information retrieval, entity and string matching, vast amounts of works exist, including various methods for calculating string, word and phrase similarities: string-edit distances, learned string distances [11], and frameworks for semantic similarity.

NL interfaces to Data Services: [12] attempts to process multi-domain full-sentence natural language queries over web-services. It uses focus extraction to find the focus entity, splits the query into constituents (sub-questions), classifies the domain of each constituent, and then tries to combine and resolve these constituents over the data service interfaces (tries recognizing the intent modifiers [e.g. adjectives] as parameters to services). (too ambitious/not-mature; open domain, real natural language questions - a bit farther from our focus, our domain is very specific.).

4.3. Searching structured DBs

The problem of keyword search over relational and other structured databases received a significant attention within the last decade. It was explored from a number of perspectives: returning top-k ranked data-tuples *[13]* vs suggesting structured queries as SQL [5], performance optimization, user feedback mechanisms, keyword searching over distributed sources, up to lightweight exploratory[3] probabilistic data integration based on users-feedback that minimize the upfront human effort required [14, ch.16]. On the other extreme, the *SODA [5]* system has proved that if enough meta-data is in place, even quite complex queries given in business terms could be answered over a large and complex warehouse.

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

TODO: Global usefulness. YQL, etc. see conclusions of MSc report.

The availability of public, corporate and governmental services is increasing as well as the popularity of data service repositories and tools[4] for combining them. Whereas, the lack of user-friendly interfaces is becoming an important issue, not only within the corporate environments.

An implementation of keyword search over dataservices has been presented discussing the implementation details, some real-world issues and ways to solving them. The implemented system do not impose any constraints on the input query , and it is able to profit from any structure available in the query (phrases, selections through auto-completion).

Acknowledgments

Part of this work was conducted as a master thesis project between *École Polytechnique Fédérale de Lausanne* and *CMS Experiment, CERN*. The main author is thankful to his supervisors for their valuable support.

[3] because of probabilistic nature of schema mappings, it do not provide 100% result exactness

[4] *such as the YQL* or the *"Google Fusion Tables"*

References

[1] 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

[2] 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

[3] 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

[4] 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

[5] 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

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

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

[8] 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

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

[10] 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

[11] McCallum A, Bellare K and Pereira F 2012 *arXiv preprint arXiv:1207.1406*

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

[13] Luo Y, Wang W, Lin X, Zhou X, Wang J and Li K 2011 *Knowledge and Data Engineering, IEEE Transactions on* **23** 1763–1780

[14] Anhai Doan Alon Halevy Z I 2012 *Principles of data integration* 9780124160446 (Morgan Kaufmann) 497p.