Keyword Search over Data Service Integration for Accurate Results

Vidmantas Zemleris[1] **and Valentin Y Kuznetsov**[2]

[1]Faculty of Mathematics and Informatics, Vilnius University, Lithuania
[2]Cornell University, USA

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

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.

still/Unfortunately

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].

2. DAS - a tool for virtual data integration

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.

As seen in figure 1, DAS query language closely correspond to the physical execution flow allowing users to be aware of it that is largely motivated by vast large volumes managed by data 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.

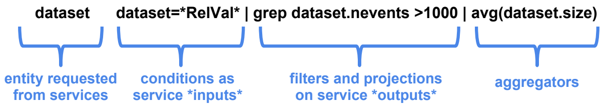

dataset
dataset=*RelVal* | grep dataset.nevents >1000 | avg(dataset.size)
entity requested from services
conditions as service *inputs*
filters and projections on service *outputs*
aggregators

Figure 1. A DAS query: get average size of datasets matching *RelVal* and having nevents>1000

3. 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).

4. Overview of our solution

4.1. From keywords to structured query suggestions

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

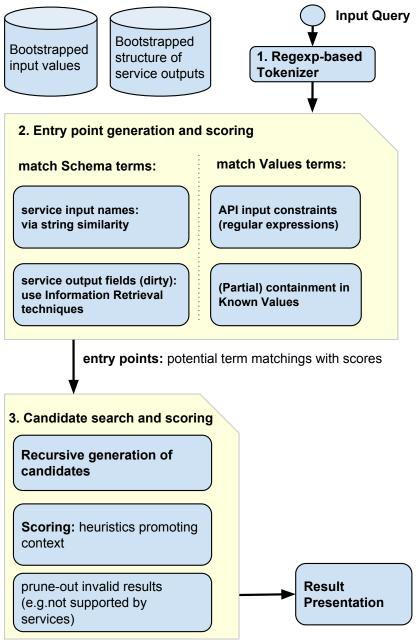
Then, employing a number of entity matching techniques, the "*entry points*" are identified: for each keyword (or their combination), we obtain a list of schema and value terms it may correspond to and a rough estimate of the likelihood.

Lastly, different permutations of *entry points* are evaluated and ranked by combining the scores of individual keywords. In the same step, the *interpretations* not compatible with the data integration system are pruned out.

Example. Consider the following keyword query: *RelVal 'number of events'> 100*. Tokenization results in: *'RelVal'; 'number of events>100'*. Then, each token may yield some entry points:

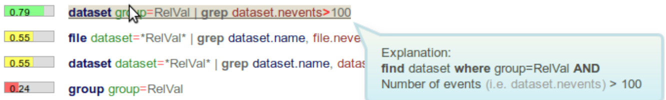
```
'RelVal' → (1.0, input-value:  group=RelVal)
'RelVal' → (0.7, input-value:  dataset=*RelVal*)
'number of events>100'→(0.93, filter: dataset.nevents>100)
'number of events>100'→(0.93, filter: file.nevents>100)
...
```

It can be seen that both *RelVal* and *'number of events'* are ambiguous. The final results obtained in step 3, where entry point scores are combined, are displayed in figure 3.


Bootstrapped input values
Bootstrapped structure of service outputs
Input Query
1. Regexp-based Tokenizer
2. Entry point generation and scoring
match Schema terms:
match Values terms:
service input names: via string similarity
API input constraints (regular expressions)
service output fields (dirty): use Information Retrieval techniques
(Partial) containment in Known Values
entry points: potential term matchings with scores
3. Candidate search and scoring
Recursive generation of candidates
Scoring: heuristics promoting context
prune-out invalid results (e.g.not supported by services)
Result Presentation

make shorter?

Figure 2. Components of our keyword search system



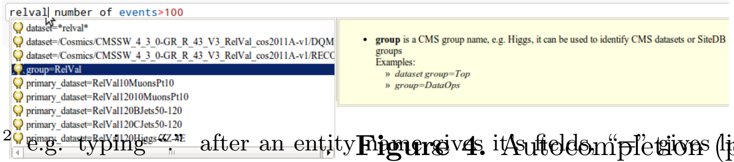
0.79 dataset group=RelVal | grep dataset.nevents>100
0.55 file dataset=*RelVal* | grep dataset.name, file.neve
0.55 dataset dataset=*RelVal* | grep dataset.name, datas
0.24 group group=RelVal

Explanation:
find dataset **where** group=RelVal **AND**
Number of events (i.e. dataset.nevents) > 100

Figure 3. Results of keyword search: structured query suggestions

4.2. Helping users to type queries: autocompletion prototype

To aids users in typing the queries we try parsing the query while being typed and provide live context dependent suggestions[2] (see figure 4). This is implemented on **top of CodeMirror's** Javascript "source-code editor" library by writing a custom parser, and implementing some autocompletion routines.



```
relval number of events>100
```

dataset=*relval*
dataset=/Cosmics/CMSSW_4_3_0-GR_R_43_V3_RelVal_cos2011A-v1/DQM
dataset=/Cosmics/CMSSW_4_3_0-GR_R_43_V3_RelVal_cos2011A-v1/RECC
group=RelVal
primary_dataset=RelVal10MuonsPt10
primary_dataset=RelVal12010MuonsPt10
primary_dataset=RelVal120BJets50-120
primary_dataset=RelVal120CJets50-120
primary_dataset=RelVal120HiggsZZ4E

- **group** is a CMS group name, e.g. Higgs, it can be used to identify CMS datasets or SiteDB groups
  Examples:
  » *dataset group=Top*
  » *group=DataOps*

Figure 4. Autocompletion (prototype)

[2] e.g. typing "." after an entity name gives its fields, "=" gives list of available values, while typing arbitrary keywords provide multiple implementations (as value, entity name, attribute name, etc)

5. Implementation details of Keyword Search

5.1. *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.

shorten down!

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.

- down-rank rare use-cases: prevent schema terms to be mapped to values with high score, as this is unlikely (e.g. dataset names include 'dataset')

Matching the schema terms We found that basic string similarity metrics, such as the *Levenshtein* edit-distance (where inserts, edits, and mutations are equal), introduced too many false-positives (e.g. 'file' for 'site' with high score). Instead, we use a combination of more trustful metrics: full, lemma, stem matches, and a stem match within a 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 names of fields in service outputs We also need to identify multi-word chunks of keywords corresponding to names of fields in service results: many of them are unclean, machine-readable field-names, with irrelevant and frequent terms (as obtained from JSON or XML responses). Thus, we employed an existing information retrieval library *whoosh*[3], where for each outputs field we create a "fake multi-fielded document" containing:

- fully-qualified technical field-name (e.g. block.replica.creation_time)
- tokenized&stemmed of technical field-name **(e.g. creation time)** and it's context - tokenized&stemmed technical field's parent name (e.g. block replica)
- human readable title, if any (e.g. "Creation time of block's replica", but often this do not include the context: "Creation time", or it does not exist)

TODO: mention paper from XML retrieval

To find the matches, we query the IR library, both for phrase and single term matches of up to k-nearby keywords (we use maximum of 4, which both provides sufficient context, and is short enough to be computationally inexpensive). The ranker uses BM25F scoring function (fields are assigned different weights and full phrase matches are also scored higer). Currently we directly use the IR score normalized between [0..1]. The scoring could be improved tuning the scoring function, but in our case it works already not so bad.

normalization: threshold based on weights signifying a good score + (maybe smoothing function)

5.2. *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 combining the scores of individual keywords. We experimented with two scoring functions: 1) averaging the scores, as used in *Keymantic*[3], and 2) summing the log-likelihoods

[3] Even if Apache *Lucene* is assumed as the most mature of the open-source libraries, it requires Java and has large footprint. Even if that may impact the results slightly, we use *whoosh,* a python library which has no dependencies.

(scores are rough estimations of likelihood). 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.

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

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

Above, $score_{tag_i|kw_i}$ - likelihood of matching an individual keyword kw_i as tag_i (entry point); $h_j(tag_i|kw_i; tag_{i-1,..,1})$ - score boost returned by contextualization heuristic h_j given earlier tags.

Heuristics and contextual rules in ranking:

- if nearby keywords refer to related schema terms (e.g. entity name and it's value)
- minor score-boost if retrieving entity by it's "primary key", e.g. dataset dataset=*Zmm*

do we use it?

5.2.1. Implementation details Currently the ranker is implemented as exhaustive search with early branch pruning to remove the suggestions not supported by the services. Implemented in *cython* it already gives the performance in most cases dominated by the entry points step.

There are other alternatives, but the simplest one was chosen which allows early pruning, unlimited contextualization and returning top-k optimal (but not approximate) results. See Discussion for more details.

6. Related works

6.1. 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 [5] attempted to incorporate users feedback by training an Hidden Markov Model's (HMM) tagger taking keywords as its input. It uses the List-Viterbi [6] 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 [7]. According to [5] the accuracy of the later system didn't differ much from Keymantic.

Also, [8] attempts to process open-domain full-sentence natural language queries over web-services. It uses focus extraction to find the main 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 (also tries recognizing the intent modifiers [e.g. adjectives] as parameters to services). This is farther from our work as we were focusing on closed domain querying with minimal preparatory work, and most importantly we didn't want to be limited to full-sentence questions. A combination of the two approaches would be interesting to look at in open-domain setup..

(too ambitious/not-mature; open domain, real natural language questions)

6.2. String and entity matching

From the fields of information retrieval, entity, schema and string matching, vast amounts of works exist, including various methods for calculating string, word and phrase similarities: string-edit distances, learned string distances [9], and frameworks for semantic similarity.

6.3. 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 *[10]* vs suggesting structured queries as SQL [11], performance optimization, user feedback mechanisms, keyword searching over distributed sources, up to lightweight exploratory[4] probabilistic data integration based on users-feedback that minimize the upfront human effort required [12, ch.16]. On the other extreme, the *SODA [11]* 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.

also mention: NL querying over services is this relevant enough?!

what in addition to schema mappings for item-based ranking?

7. 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?

8. 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[5] 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.

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

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

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

[7] 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

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

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

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

[11] 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`

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