Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2013) 000–000

Procedia
Computer Science

www.elsevier.com/locate/procedia

Keyword Search over Data Services

Vidmantas Zemleris, Valentin Kuznetsov, Robert Gwadera

Abstract

Data integration provides a centralized solution for locating and combining data from multiple sources. The complexity of writing structured-queries is impacting not only simple users who are forced to learn the schema and the query language, but also the tech-savvy users who may have only a vague idea of where exactly to find the data they need.

In this paper we explore more intuitive alternatives such as keyword and natural language search, which, in fact, received fairly little attention in the field of data services integration. First, we review the state-of-the-art of approaches to searching data services including ways how these systems could self-adapt to users' needs by taking into account historical queries and their results. Then, a keyword search system over enterprise data integration is presented that, given a keyword query, proposes top-k most probable structured queries which could be later answered using current data integration infrastructure. The system was developed and evaluated in the setting of a scientific collaboration of over 3,000 physicists at the CMS Experiment, CERN. **TODO: abstract shall be shorter: 50-100 words**

Keywords: Keyword-based Search; Natural-Language Search; Data Services; Enterprise Information Integration

(this page only for the thesis report)

Contents

1. Introduction

Enterprise Information Integration provides a centralized solution for locating and combining data from multiple sources. The complexity of writing structured-queries is impacting not only simple users who are forced to learn the schema and the query language, but also the tech-savvy users who may have only a vague idea of where exactly to find the data they need.

In this work we explore more intuitive alternatives such as keyword and natural language search, which, in fact, received fairly little attention in the field of data services integration. First, we review the state-of-the-art of approaches to searching data services including ways how these systems could self-adapt to users' needs by taking into account historical queries and their results. Then, a keyword search system over enterprise data integration is presented that, given a keyword query, proposes top-k most probable structured queries which could be later answered using current data integration infrastructure. The system was developed and evaluated in the setting of a scientific collaboration of over 3,000 physicists at the CMS Experiment, CERN.

TODO: introduce the paper; add motivation: usefulness of KWS/NLs, contradicting results; lack of research in the field of KW/NL Search over Web Services; needs of CMS.

TODO: present the problem in general, setting at CMS and our solution

Major problems:

- what is better suitable for querying information: keywords, full sentence, or restricted structured language
- keyword search is too much ambiguous (for searching structured data)
- full sentences are long to write, and often hard to process as there are many ways of expressing same idea
- structured language must be learned, including schema terms and its "grammar"

1.1. Intro, Usability issues

TODO: discuss differences between Structured, Keyword and NL Search

refer to Paper...

1.2. Our work

2. Problem definition

The problem in more generic sense is satisfying users' information need expressed in any of many possible ways, either as natural language (NL), keyword (KW), structured query or other interfaces such as predefined Query Forms.

2.1. Communicating User's intent

Figuring out what is the best suited for **communicating user's intent** is another sub-problem, where recent studies are reported disagreement in their results. Some users prefer the complete sentences (or even speech) being more natural and **expressive**[2, 4], or the shorter keyword queries that are more ambiguous, while others prefer the structured languages that are **easiest** to process by a machine, but requires learning the grammar and "lexicon" (for instance, the ones who already know the language).

"The research suggests people prefer to state their information need rather than use keywords. § But after first using a search engine they quickly learned that full questions resulted in failure." http://people.ischool.berkeley.edu/~hearst/talks/upitt.p

2.2. From keywords to queries

We believe that supporting a combination of these is closest the the optimal solution. An input that is not conforming to the requirements of some structured query language, **could be processed as input in natural language if it conforms to its grammar or as a keyword query otherwise.**

Then we define our problem as translating user's query (expressed either as sequence of keywords or as a sentence in a natural language) into a structured query that is referring to data-services (or their composition).

We are given: the users query, an integration schema (consisting of entity names, their attributes, regular expressions defining the types of values accepted, and in some cases a list of possible input/output values), query log (in the beginning only for structured queries; but after the service is deployed we will have query log for keyword queries as well), as well as a number of services with their Interface definitions (or a structured QL that could access that – DAS, YQL) that could be composed if needed.

composing services is of secondary importance

3. Possible approaches (?Preliminaries: Searching Data Services)

3.1. Keyword Search: Heuristic based

(mainly based on meta-data i.e. without access to the instance (that could include web-services)

3.2. Keyword Search: HMM based

Bergamaschi et al.[1] approached the same problem through Hidden Markov Model (HMM) and allow taking into account the query logs. Keywords that are near to each other are expected to represent interrelated concepts, which motivates the application of an HMM as it can efficiently model such kind of interdependences [1]. Their approach was designed for Relational Databases (then access to the instance is limited), but it could be also adapted for data integration over data services.

An HMM models a stochastic process that could not be observed directly (it is hidden, in our case, it is mapping to schema entities), but observable indirectly through observations symbols (keywords in our case) produced by another stochastic process.

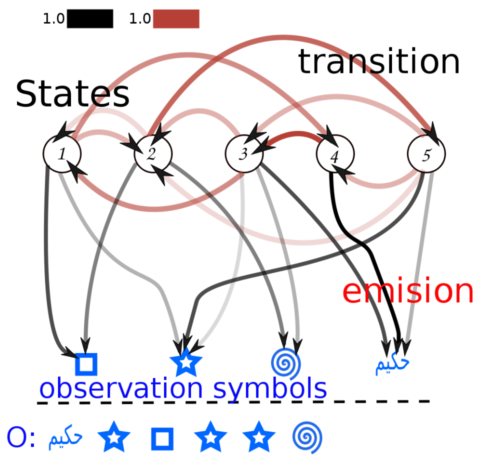

Figure 1. HMM

In this approach, HMM is used as described below:

- keywords are represented as a sequence of *observations*

  $O = \{o_t\}$, from the set of observation symbols $V = \left\{v_j\right\}$

- a set of *states* $S = \{s_i\}$, each representing mapping of keywords into a schema **term** (entity, their attributes, operators). These are “hidden” and have to be decoded by HMM into $Q = \{q_i\}$.
- *transition* probabilities, $a_{ij} = P(q_{t+1} = s_j \mid q_t = s_i)$
  - use heuristics taking into account semantic relationships between database terms (aggregation, generalization, inclusion)
  - Initially: with the goal to foster transition between terms belonging to the same table or tables connected via FK.
  - Our: with the goal to foster transition between:
    - terms belonging to the same entity (entity and it’s attribute, attribute and it’s value)
    - terms that are commonly referred by same API call (as inputs or outputs)
    - terms belonging to requested answer type **(not possible with HMM, but maybe possible with CRF)**

operators c
in DAS

- *emission* probabilities - prob. of observing keyword v_m given schema term q_t: $b_i(m) = P(o_t = v_m \mid q_t = s_i)$
  - computed on the basis of similarity measures
  - edit-distance between keywords and each term (and it's **synonyms**) in schema vocabulary
  - domain vocabulary: domain compatibilities and regular expressions
  - then use calculated similarity as **an estimate** for conditional probability $P(q_t = s_i|o_t = v_m)$, then using Bayes theorem calculate emission probability is
  - **are these calculated live or stored, or combination of the two !? how expensive would be the storage then?**
- *initial state* probabilities - in initial implementation HITS algorithm is used to estimate the "authority" of each entity (~how much valuable information it contains) based on number of attributes and links (foreign keys) to other tables [1, p.150]. **This do not seem not so relevant without modification for data services, where better measures could be:**
  - how often an entity is accessed by users
  - what's the probability that it could start a query? e.g. if that's
  - **TODO: what are the initial state probs defining?**
- **TODO: how does it work? It hopefully doesn't need instantiating all schema terms?**

Initialization: Setting HMM parameters

This approach could be also used even when query log is not yet available by estimating the HMM parameters by applying similar heuristics as in the earlier chapter (Heuristic based KWS).

In Action: Decoding HMM

A List Viterbi could provide a list of most probable mappings (in contrast to standard Viterbi returning only the most probable result).

The limitation of HMM is that decision for the current label may only depend on a fixed number of earlier labels and input keywords.

CRF?

3.3. Natural Language (NL) Processing and Full sentence Search

semantic distance, string dist WSD (word sense d..,) focus extraction (1 NP or last NP)
E.g. what is the Total size of all Zmm datasets
what is the size of /Zmm/.,X datasets — parsing of the query file Zmm
figuring out between KWS and NL search

3.4. Subtask: Matching keywords to entities (string matching)

- String similarity (edit-distance)
  - need to alter the weights...
    - ∗ cheap removing from the end
    - ∗ expensive mutations, removals from inside
  - matching the entity and attribute names
  - values with small edit-distance (spelling-correction)
- Semantic distance
  - could also match the possible values
- Regular expressions
- Matching keyword into an sample of values (guessing which is the best attribute without having all of it's values)

3.5. TODO: Other

- also Keyword Search combining services?
- try bootstrapping the results (but they count be different)
- understanding relationships between services (e.g. through existing queries?)
- evaluation of degradation of ranking quality with expansion of search space (operators: aggression, min, max, post-filtering through result fields)
- interaction with the user e.g. creation time/ modification time dataset taken on 1/Jan/2013 dataset - F size, type, I n_events decrease false positives via machine learning?
- dateset with more than 100 event avg, sum
- Limitations of the approach: control statements? controlled statements: top 10, rank by, aggregate by etc

Examples our domain? different domain? Evaluation/usability study by end-users

4. Our contribution

4.1. The use-case of CMS Experiment at CERN: Problem definition

amount of data; description of services; contstraints; DAS; targeting simplification of simple-users; why new system is hard; why structured language is hard; powerful users vs simple users why IR is not applicable here. number and it's meaning?

feedback from Users

consider relationships between services explicit (PK dataset.name) implicit - even more ambiguity and complexity

loose coupling of proprietary services / systems

Current implementation / Solution developed?

4.2. a Hybrid approach

4.3. Incorporating User Feedback

promote/demote query suggestions

4.4. Personalizing to Users' Needs

First the most common queries may be promoted in the results, maximizing the .

Information need among different user roles (department, function) differs a lot. Most of the time many users are interested only in a few types of queries or entities that they are interacting with the most.

Prioritization may promote queries related to what user (or his group?) has previously used.

4.5. Improving the Performance

Estimating query running time

Tracking of the execution time of each data-service, have been implemented, that allows a) informing user of long lasting queries, and running them only with his confirmation b) pre-running the speedy queries even before user has explicitly selected them (e.g. the top few queries proposed).

It has been chosen to track the mean of execution time, and it's standard deviation. Knuth has shown that the standard deviation can be efficiently computed in an online fashion without need to store each individual value, nor recomputing everything from scratch [5, p. 232].

Because input parameters passed to the service may heavily impact the service performance, we differentiate between these parameter types: 1) some specific value, 2) a value with wild-card (presumably returning more results than specific value as it may match multiple values), 3) not provided (matches all values). So we store only four values per data-service input parameter's combination (Algorithm1).

Algorithm 1 Calculating the Standard-Deviation in online fashion

```
from math import *
# stored in DB
n = 0
mean = 0
stddev = 0
M2 = 0  # intermediate result, the sum of squares of differences from the (current) mean

def update_stddev(x):
    global n, mean, stddev, M2
    n += 1
    delta = x - mean
    mean += delta/n
    M2 += delta*(x - mean)
    variance = M2/n
    stddev = sqrt(variance)

    return stddev
```

Proposals for service providers

Pagination & Ordering
Incremental view maintenance
? Bloomjoin ?

5. Evaluation

The evaluation was performed at the CMS Experiment described earlier.
......

5.0.1. Usability

5.0.2. Usefulness of KWS / NL / DAS QL

KWS vs NL vs Structured language
TODO: evaluation strategy

6. Related work / Literature Review

TODO: plug in old and update the literature review

6.1. DAS at CERN

6.2. KWS based on metadata

i.e. without access to the instance (that could include webservices)

Heuristic based

HMM

CRF?

6.3. NL Search over Webservices

6.4. Question Answering (QA) and Natural Language Processing

In early days of question answering these systems were conceived as natural language interfaces to (relational) Databases such as **[1-2]**. 'These early QA systems worked by translating the natural-language question into a formal structured query and issuing it against a pre-compiled database of knowledge in order to arrive at the answer.' Kalyanpur et al. [3]. There, the translation is usually based on predefined rules / templates representing semantic relations between concepts in the question (called Relation Extraction). Similarly more recent works also employs predefined rules for QA over Databases **[TODO...].**

The IBM Watson that currently is a state-of-the-art open-domain QA system, also employs Relation Extraction, but in addition to structured sources it heavily relies on using text passages as an evidence for answering the questions (which is inevitable for answering open domain questions in a self-contained way).

The stages of QA that are relevant to our system includes:

- Deep Parsing
- **Predicate Argument Structure**, that normalizes and simplifies the results removing semantically irrelevant details such changing passive voice in active ("it was bought by CERN" into "CERN bought it").
- Answer Typing - figuring out the requested Answer type
- Relation Extraction (Rule-based with handcrafted rules; or Statistical of lower precision)
- Keyword Extraction – the words extracted as "keywords" would be given higher score than others
- Entity disambiguation and Matching

TODO: QA systems over Databses that use rule-based mappings to structured Q

6.5. NL S over Ontologies and Semantic Web / Semantic DBs

6.6. KWS over Databases / Semantic KWS over Datawarehouses + Ontology

SODA and other; SODA includes user feedback. Instead of trying to interpret natural language, SODA uses predefined patterns for more complex operations (returning top-k results, aggregation) as there are many ways of expressing these;

6.7. Keyword query cleaning

6.8. Other approaches:

6.8.1. Showing results in natural language too!!!

6.8.2. Faceted navig.

6.8.3. Refinement of User's Intent?!

6.8.4. Real-time suggestions

6.9. Usability aspects of QA, KWS, Structured search

6.10. User's Feedback

SODA uses Like/So So/Wrong feedback next to each result (a proposed structured query). They incorporate it by using the feedback as a component of score used for ranking result candidates [Add citation: SODA master thesis ethz]. The formula used is:

.....

Bergamaschi et al. [1]proposed using an HMM, that could adapt to query logs (while the HMM parameters could be be 'bootstrapped' by applying a couple of heuristics even if the logs are unavailable).

7. Future work

Because of the project was fairly short and the author had to follow some of priorities of the CMS Experiment, some parts of the research work had to be excluded, including: **TODO**.

8. Other applications?

9. Conclusion

NL and KW Search over Web Services is still lacking attention from research community.

In addition to QL this could help in learning and get results more quickly (immediately) proprietary systems with . . . rules of changing them; would ease adoption of Struct L and fit naturally with existing contrains.

TODO: try to find out few cases of mapping NL to other existing services. (e.g. proprietary + public services)

Acknowledgements

This work was financially supported by the CMS Computing group, at the Compact Muon Experiment at European Organization for Nuclear Research (CERN). The authors of this paper are thankful to Robert Gwadera and prof. Karl Aberer who were supervising the Master Thesis project of the first author at the Swiss Federal Institute of Technology in Lausanne.

References

[1] Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Silvia Rota, Raquel Trillo Lado, and Yannis Velegrakis. Understanding the semantics of keyword queries on relational data without accessing the instance. In Roberto Virgilio, Francesco Guerra, Yannis Velegrakis, M. J. Carey, and S. Ceri, editors, *Semantic Search over the Web*, Data-Centric Systems and Applications, pages 131–158. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-25008-8. URL `http://dx.doi.org/10.1007/978-3-642-25008-8_6`.

[2] M.A. Hearst. 'natural'search user interfaces. *Communications of the ACM*, 54(11):60–67, 2011.

[3] A. Kalyanpur, BK Boguraev, S. Patwardhan, JW Murdock, A. Lally, C. Welty, JM Prager, B. Coppola, A. Fokoue-Nkoutche, L. Zhang, et al. Structured data and inference in deepqa. *IBM Journal of Research and Development*, 56(3.4):10–1, 2012.

[4] E. Kaufmann and A. Bernstein. How useful are natural language interfaces to the semantic web for casual end-users? *The Semantic Web*, pages 281–294, 2007.

[5] Donald E. Knuth. *The Art of Computer Programming, Volume 2: Semi numerical Algorithms*. Number 9788177583359. Addison-Wesley Longman, Inc, 1998.

Appendices

Appendix: Logs Analysis (thesis only?)

Appendix: Proposed solutions for CMS?