Keyword Search over Data Services [Early Draft]

Vidmantas Zemleris

14th January 2013

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 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: abstract shall be shorter: 50-100 words**

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

Contents

shall we put the solutions here or somewhere lower in the paper after describing Keyword Search?

List of Symbols and Abbreviations

CMS	Compact Moun Selenoid Experiment at the European Organization for Nuclear Research (CERN)
EII	Enterprise Information Integration
HMM	Hidden Markov Model
KWQ	keyword query
schema	by schema we refer to the integration schema (virtual schema based of entities exposed by the services)
schema terms	entities of integration schema and their attributes
schema values	values of entity attributes (that could be input parameters of data services, or form part of their results)

1 Introduction

crap! even CMS case-study version has better introduction

Enterprise Information Integration (EII) 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: add a motivating example in the beginning!
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...

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

1.2 Our work / contributions

- case analysis at the CMS Experiment
  - query log analysis
  - user's survey
  - suggestions for data providers on how to improving the performance
- literature review
- choosing a precise the topic to focus on, has took a considerable amount of resources
- implementation of open-source keyword search engine

2 Problem statement

The problem in global sense is about satisfying user's information need. This need has to be expressed in some way, e.g. as natural language (NL) query, keyword (KW) query, a structured query or through any other interfaces such as predefined Query Forms.

2.1 Communicating User's Intent

Deciding on what is the best suited for **communicating user's intent** is a problem, where recent studies reported disagreement in their results. Some users prefer composing queries as complete sentences (or even natural speech) that is more natural [Hea11, KB07] than the shorter to type keyword queries that are even more ambiguous. Others prefer the structured languages that are **the easiest** to process by a machine, but incurs a steeper learning curve (grammar and "lexicon" has to be learned).

"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.pdf

2.2 From keywords to structured queries

We believe that in many cases supporting a combination of these approaches is a good compromise: the input (or subset of it) that is not conforming to the rules of given structured language, could be processed as natural language if it conforms to its grammar or as a keyword query otherwise.

Given an enterprise information integration system that is able to answer structured queries [1] and an *integration schema*[2], we are interested in translating user's query (a list of keywords or a full sentence) into the corresponding structured query. The services could be also composed, but this is not yet supported by DAS.

composing services is of secondary importance

For example, a keyword query "configuration Zmm" shall be mapped into: `config dataset=*Zmm*`[3]. While a more complex full sentence query "what are the average and total sizes of Zmm datesets containing more than 1000 events" shall be converted into:

```
dataset dataset=*Zmm* | grep dataset.nevents > 1000 | avg(dataset.size), sum(dataset.size)
```

More formally, an *integration schema* consists of:

Better terminology?

- schema vocabulary: entities of *integration schema* and their attributes
- values (domain vocabulary): values of entity attributes (that could be input parameters of data services, or form part of their results)

We assume a keyword query KWQ to be an ordered k-tuple of keywords $(k_1, k_2, .., k_l)$. Answering a keyword query means *interpreting* the keyword query in terms of it's semantics over the *integration schema.* In this work, we the structured queries that we consider are

[1]e.g. conjunctive queries for DAS at CERN (our main focus); SQL-like queries for YQLhttp://developer.yahoo.com/yql/

[2]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; The integration schema is referring to data service interfaces (accepted inputs and outputs provided)

[3]i.e. configuration of datasets containing Zmm in their name; at CERN, a dataset is a collection of files describing physics events and meta-data about them

conjunctive queries augmented with simple aggregation functions without grouping (that would correspond to select-project-join in SQL or YQL, where selections can be only conjunctions (AND) of predicates).

2.3 Something in between? A UI?

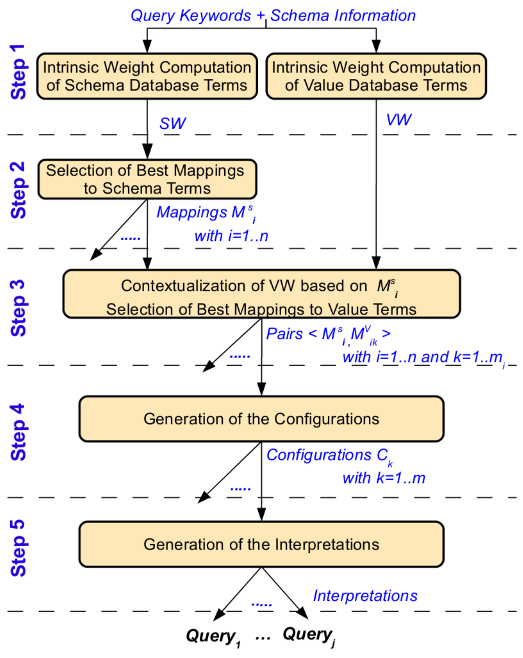
3 Non-structured Search over Data Services

The approaches below are applicable then there is no possibility to index the data terms, e.g. then a DB is behind a wrapper (e.g. accessible only through a *Web form* in "Hidden Web" or a *web-service*) then crawling is generally not possible.

3.1 Keyword Search: Heuristics and exhaustive search

Taking inspiration from Keymantic [BDG+10] (that is focusing on relational databases), a keyword query may be processed as follows (see Fig. 3.1):

First, we use a number of similarity metrics and matching techniques[4] to identify the schema terms (or their values) to which each keyword may correspond to, along with related likelihoods (Step 1). Then, all keywords that correspond to meta-data items (e.g., field names) are extracted (Step 2). Third, the remaining keywords are considered as possible Value Terms (parameters to the input fields of web services; or value-based filters on their results). Then, a number of combinations (configurations) are considered and ranked, permuting different combinations of assignment of keywords to schema terms or their values. Each such interpretation is assigned a score based on a number of heuristics:



Query Keywords + Schema Information
Step 1
Intrinsic Weight Computation of Schema Database Terms
Intrinsic Weight Computation of Value Database Terms
SW
VW
Step 2
Selection of Best Mappings to Schema Terms
Mappings M^s_i with i=1..n
.....
Step 3
Contextualization of VW based on M^s_i
Selection of Best Mappings to Value Terms
Pairs $< M^s_i, M^V_{ik} >$ with i=1..n and k=1..m$_i$
.....
Step 4
Generation of the Configurations
Configurations C_k with k=1..m
.....
Step 5
Generation of the Interpretations
Interpretations
.....
Query$_1$ ... Query$_j$

Figure 3.1: Keyword query translation process in Keymantic (source:[BDG+10, p. 1639])

- promoting such combinations where nearby keywords refer to related schema terms (e.g. entity name and it's value)
- promoting such combinations that match most of the keywords;
- if requested entity and a filter condition is the same (small increase)
- data service constraints must be satisfied: we could also include identifying interpretations that achieve high rank even if they do not satisfy some constraint (e.g. a mandatory filter is missing), and informing the user

Finally the configurations are interpreted into actual queries (for databases the join order is important; in our case (services): applying any operators and how services are combined). The queries could be further re-ranked by executing those of the best queries that are estimated to be less heavy than some threshold (based on service statistics, see Section 4.1).

Because of limited no access to the actual data, results of this method were reported considerably worse on queries containing data terms, even if all meta-data (e.g. business ontology) is given[BJK+12], therefore it is helpful to have some insight on the data behind a web-service, but has limitations.

[4]including lexical (edit-distance), semantic word similarity and regular expressions defining allowed inputs (not always informative enough) for generating entry points; see Section 3.4 for more details

3.2 Keyword Search: Machine Learning

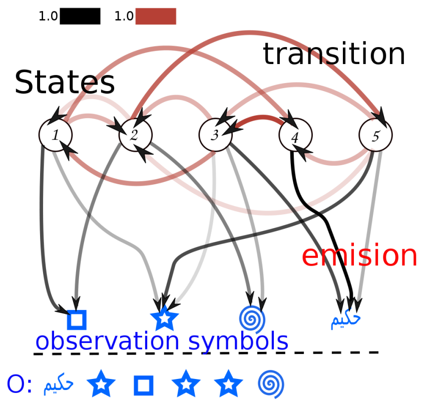


Figure 3.2: HMM consituents

Bergamaschi et al. 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 [BDG+12]. 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 is a probabilistic *sequence labeler/classifier* which task is to assign a label to every unit in a sequence[JM09]. It models a stochastic process not observable directly (in our case, the labels are keyword correspondence to schema entities) that is observable only indirectly through observations produced by another stochastic process (keywords, in our case), assuming that the probability of a label of any sequence unit depend only on a fixed number of that go before it (the Markov Assumption).

In this approach, HMM is used as described below:

- keywords are represented as a sequence of T *observations*

  $O = \{o_1 o_2 ... o_T\}$, each from observation vocabulary $V = \{v_1, ..., v_M\}$

- a set of N *states* $S = \{s_1, s_2, ..., s_N\}$, each representing mapping of keywords into a schema **term** (entity, their attributes, operators). These are "hidden" and have to be decoded by HMM into $Q = \{q_1 q_2 ... q_T\}$.

operators only in DAS

aren't the symbols here messed up from the standard notation?

- *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)**

- *emission* probabilities - probability of observing keyword v_m from a given state (i.e. 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 [BDG+12, 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?**

Computing answers through decoding an HMM

A List Viterbi algorithm[SS94] provides a rank ordered list of top-K (globally best) candidate mappings of keywords into the HMM states (i.e. schema terms) . These can later be turned into structured queries (with some additional ambiguity to be handled).

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).

Semi-supervised learning: incorporating users feedback

TODO: this was not (?) well specified in any of their papers
TODO: The standard procedures Baum-Welch algorithm... [JM09, sec. 6.5. HMM training]

—-

The limitation of HMM is that decision for the current label may only depend on a fixed number of earlier labels and input keywords, this may be true for keyword search, but is more complex for NL search.

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 Incorporating User Feedback

- influence keyword to schema term matching
  - similarity metrics and their weights
  - allow users to add new: [this is the explicit feedback, that is more valuable than implicit]
    - values for schema entities
    - synonyms for schema terms
- promote/demote query suggestions

3.5.1 Personalizing to Users' Needs

First the most common queries may be promoted in the results, maximizing the probability of getting results right.

Information need among different user roles (department, function) differs a lot. At least at the CMS Experiment, CERN, users are often interested only in a few types of queries or entities that they are interacting with the most.

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

shall we distinguish links generated by DAS, e.g. for dataset: config, files, etc?

4 A case study of the CMS Experiment at CERN

At large scientific collaborations like the CMS Experiment at CERN's Large Hadron Collider, that includes more than 3000 collaborators, data usually resides on a fair number of autonomous and heterogeneous proprietary systems each serving it's own purpose [5]. As data stored on one system may be related to data residing on others[6], users are in need of a centralized and easy-to-use solution for locating and combining data from all these multiple services.

Using a highly structured language like SQL is problematic because users need to know not only the language but also where to find the information and also lots of technical details like schema. An enterprise information integration system based on simple structured queries is already in place. Various improvements had to be researched: adding support for more complex queries and for less restricted keyword search, improvements to system's usability and performance.

A System for Virtual Data Integration

The *CMS Data Aggregation System (DAS)*[KEM10, BKEM11] was created which allows integrated access to a number of proprietary data-sources by processing user's queries on demand - it determines data-sources are able to answer[7], queries them, merges the results and caches them for subsequent use. DAS uses *Boolean retrieval model* as users are often interested in retrieving ALL the items matching their query.

Currently the queries specify what entity the user is interested in (dataset, file, etc) and provide selection criteria (attribute=value, name BETWEEN [v1, v2]). The combined query results could be later 'piped' for further filtering, sorting or aggregation (min, max, avg, sum, count, median), e.g.:

```
# returns average and median dataset sizes of ones containing RelVal in their name having more than 1000 events.
dataset=*RelVal* | grep dataset.nevents >1000 | avg(dataset.size), median(dataset.size)
```

Queries are executed either from web browser or through a command line interface where the results could be fed into another application (e.g. program doing physics analysis or automatic software release validation).

TODO:

- 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

[5]For instance, at CERN, due to many reasons (e.g. research-orientedness and need of freedom, politics of institutes involved) software projects usually evolve in independent fashion resulting in fair number of proprietary systems[KPGM00]. Further high turnover makes it harder to extend these systems

[6]For example, datasets containing physics events are registered at DBS, while the physical location of files is tracked by Phedex which also takes care of their transfers within the grid storage worldwide

[7]This is done by a mapping between flat mediated schema entities ('DAS keys') into web-service methods and their arguments. Then system queries all services that could provide a result of expected type with given parameters

- **loose coupling of proprietary services / systems**
- Current implementation / Solution developed?

4.1 Improving the Performance

After initial problem analysis, it was found that most performance problems were due to large data amounts the data providers are processing, and some of the work that is unnecessarily being repeated (or requested). Below are the proposals:

Proposals for service providers

Pagination & Ordering
Incremental view maintenance
? Bloomjoin ?

Estimating query running time

not yet implemented

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 [Knu98, 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.

4.2 Improving the Search Interface

shall we put the solutions here or somewhere lower in the paper after describing Keyword Search?

....

5 Technical Implementation details

packages, tool-kits: uncertainties, nltk, wordnet

6 Evaluation

The evaluation was performed at the CMS Experiment described earlier.

......

6.1 Usability

6.2 Usefulness of KWS / NL / DAS QL

- KWS vs NL vs Structured language
- TODO: evaluation strategy

7 Related work / Literature Review

TODO: plug in old and update the literature review

7.1 DAS at CERN

7.2 KWS based on metadata

7.3 NL Search over Webservices

7.4 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 precompiled database of knowledge in order to arrive at the answer.' **(author?)** [KBP+12]. 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

7.5 NL S over Ontologies and Semantic Web / Semantic DBs

7.6 KWS over Databases / Semantic KWS over large 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;

7.7 Keyword query cleaning

7.8 Usability aspects of QA, KWS, Structured search

7.9 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:

.....

(author?) [BDG+12]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).

8 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**.

9 Other applications?

10 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 Computing group of the Compact Muon Solenoid Experiment at the 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

[BDG+10] Sonia Bergamaschi, Elton Domnori, Francesco Guerra, Mirko Orsini, Raquel Trillo Lado, and Yannis Velegrakis. Keymantic: semantic keyword-based searching in data integration systems. *Proc. VLDB Endow.*, 3(1-2):1637–1640, September 2010.

[BDG+12] 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.

[BJK+12] Lukas Blunschi, Claudio Jossen, Donald Kossmann, Magdalini Mori, and Kurt Stockinger. Soda: generating sql for business users. *Proc. VLDB Endow.*, 5(10):932–943, June 2012.

[BKEM11] G Ball, V Kuznetsov, D Evans, and S Metson. Data aggregation system - a system for information retrieval on demand over relational and non-relational distributed data sources. *Journal of Physics: Conference Series*, 331(4):042029, 2011.

[Hea11] M.A. Hearst. 'natural'search user interfaces. *Communications of the ACM*, 54(11):60–67, 2011.

[JM09] Daniel Jurafsky and James H. Martin. *Speech and Language Processing (2nd Edition) (Prentice Hall Series in Artificial Intelligence)*. Prentice Hall, 2 edition, May 2009.

[KB07] 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.

[KBP+12] 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.

[KEM10] Valentin Kuznetsov, Dave Evans, and Simon Metson. The cms data aggregation system. *Procedia Computer Science*, 1(1):1535 – 1543, 2010. ICCS 2010.

[Knu98] Donald E. Knuth. *The Art of Computer Programming, Volume 2: Semi numerical Algorithms.* Number 9788177583359. Addison-Wesley Longman, Inc, 1998.

[KPGM00] Christoph Koch, Paolo Petta, Jean-Marie Le Goff, and Richard McCatchey. On information integration in large scientific collaborations, 2000.

[SS94] N. Seshadri and C.E.W. Sundberg. List viterbi decoding algorithms with applications. *Communications, IEEE Transactions on*, 42(234):313–323, 1994.

Appendices

Appendix: Logs Analysis (thesis only?)
Appendix: Kinds of Queries at CMS

TODO / Notes / etc

- Showing results in natural language too!!!
- Faceted navigation and/or Refinement of User's Intent?!
- Real-time suggestions
- also Keyword Search combining services?
- try loading the results online (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)
- Limitations of the approach: control statements? controlled statements: top 10, rank by, aggregate by etc
- Examples in our domain? in different domain? Evaluation/usability study by end-users
- CRF implementations that can output n-best results
  - http://crfpp.googlecode.com/svn/trunk/doc/index.html
  - http://wapiti.limsi.fr/
  - mallet is Java library (not prefferd because of Java)
- **TO Check: Natural Language Querying over Databases Using Cascaded CRFs**
- Training Conditional Random Fields using Virtual Evidence Boosting
  - http://www.ijcai.org/Past%20Proceedings/IJCAI-2007/PDF/IJCAI07-407.pdf
- **About converting HMM into CRF:**
  - http://www.cs.ucf.edu/~gitars/cap6938/vail07conditional.pdf
  - **http://knight.cis.temple.edu/~yates/cis8538/sp11/slides/conditional-random-fields.ppt**
    - http://knight.cis.temple.edu/~yates/cis8538/sp11/

Another thing which could useful for the users at CMS Experiment, CERN, is search system that allows a) combing keyword-based and structured search (even in the same query), and that b) do not require the queries to be in a format that depends on physical service implementation (inputs vs outputs), while the current *DAS Query Language* does.