

Automatic Keyword Extraction for Database Search

First examiner

Prof. Dr. techn. Dipl.-Ing. Wolfgang Nejdl

Second examiner

Prof. Dr. Heribert Vollmer

Supervisor

MSc. Dipl.-Inf. Elena Demidova

Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 27 Februar 2009

.....
(Iryna Oelze)

Eingegangen am (Datum/Stempel): _____

Abstract

Users often try to assimilate information on a topic of interest from multiple information sources. Sometimes user's information need might be expressed in terms of an available relevant document, rather than a query. This document can result from a web search, but also arrive at user's desktop directly e.g. as an e-mail attachment.

Recently a lot of work was performed towards enabling keyword search in databases. However, database search engines are mostly adapted to the queries manually created by users. In case user's information need is expressed in terms of a document, we need to create algorithms that automatically extract keyword queries from the available data and map them to the database content.

In this work we analyse influence of the selected document and database statistics on effective keyword extraction and disambiguation in order to retrieve relevant results from a database.

We implemented our keyword extraction and disambiguation algorithms on the top of the Okkam entity repository. We evaluated our approach using a real-world dataset containing Wikipedia documents and IMDB data as well as a set of user-defined keyword queries from a Web search engine log.

Table of contents

1 Introduction	5
1.1 Motivation.....	5
1.2 Outline.....	7
2 Problem Analysis	8
2.1 Keyword Extraction.....	8
2.1.1 Existing Approaches.....	8
2.2 Keyword Search.....	11
2.2.1 Keyword Search Approaches.....	11
2.2.2 Entity Repository.....	12
3 Conceptual Design	14
3.1 Automatic Keyword Extraction.....	14
3.2 Keyword Request Processing.....	16
3.2.1 Attribute Ranking Factors.....	16
3.2.2 Query Score.....	19
3.2.3 Query Ranking Algorithm.....	20
4 Software Used	22
5 Datasets Used	23
5.1 Semi-Structured Dataset.....	23
5.2 Document Dataset.....	25
6 Evaluation	26
6.1 Precision.....	26
6.2 Effectiveness.....	28
6.3 Efficiency.....	30
6.4 Relevance.....	33
7 Conclusion	36
8 References	38

1 Introduction

The first chapter begins with the clarification of the motivation for the bachelor thesis, this part will illustrate the actuality and aim of the Bachelor thesis. Finally, the section 1.2 gives a brief overview of the following chapters.

1.1 Motivation

Information is the most powerful weapon in the modern society. Every day we are overflowed with a huge amount of data in form of electronic newspaper articles, e-mails, webpages and search results. Often, information we receive is incomplete, such that further search activities are required to enable correct interpretation and usage of this information. For instance, in an enterprise, given a customer request sent via e-mail, search activities of an employee in the customer support department can include lookups of the information on the related products in the intranet databases, as well as Web- and desktop search.

Keyword search is a usable and powerful tool which enables efficient scanning of large document collections. It frees the user from learning the syntax of a structured query language, like e.g. Boolean query, SQL or XQuery and understanding their complex semantics. Recently, keyword search found application in the databases, where it enables data retrieval in case the schema is unknown to the user, going beyond predefined forms and applications [1, 2, 5, 9, 11, 12, 23, 24, 29]. On the other hand, usability comes at the price of expressiveness. In order to correctly answer a keyword request, database system need to identify intention behind the keywords; this introduces additional query processing cost at the database side.

In case an information need of the user is represented through a document, rather than a manually created keyword request, keyword annotations of this document (as well as other available metadata) can be used to build a keyword query. For instance, scientific articles are often annotated with keywords. Also Web documents, especially multimedia resources, can be already associated with tags. In an electronic magazine, keywords give a clue about the main idea of an article, in a book they quickly lead the reader to the whereabouts of the information sought. On the web, tag annotations help

to find multimedia and other resources. Unfortunately, a large portion of documents on the Web still does not have any keywords assigned. Moreover, creation of manual annotations is time-consuming, such that automatic ways of keyword extraction from the documents are required. In the following we illustrate the necessity of keyword extraction with a short scenario.

A technician Alice supplies customers of an internet hardware sales enterprise with expert information regarding the installation and usage of the products. Every day she receives a lot of e-mails which contain description of the products and usage problems at different level of detail. In order to answer the request, she needs to identify product specifications such as a model, producer, production country, etc. and then search in a database for further product details. In order to answer the request, she first reads the message, trying to identify useful keywords, then retrieves necessary information from the enterprise database using the keyword search interface. However, the manual assignment of high quality keywords is time-consuming. Automatic keyword extraction would enable Alice to immediately identify related information in the enterprise database and essentially reduce response time to the customer.

Many existing algorithms and systems aimed to perform automatic keywords extraction have been proposed [4, 13, 15, 16, 21, 23, 26, 28]. Currently existing solutions for automatic keyword extraction require either domain specific knowledge [13, 21, 23] or training examples [23, 26]. These approaches require human interaction and need to be adapted to the specific application domain. In case when the documents representing user information need are obtained from the Web search or arrive to the user desktop via e-mail, this information is not available.

In this thesis we develop an approach to identify information related to a text document inside a database. We analyse database and document statistics, which are useful for the keyword extraction and develop approach of keyword disambiguation inside the database. We compare performance of the system which uses automatically extracted keywords with the one of user generated queries. We evaluate our approach using entity repository containing data extracted from the Internet Movie Database [14] and subset of Wikipedia pages [25] related to the movie domain.

1.2 Outline

The outline of the thesis is organised as follows:

Chapter 2 analyses the problem area by presenting the existing approaches in the keyword extraction and database search domains. This gives an overview of the related works and explains the choice of our conceptual design.

Chapter 3 specifies the conceptual design and exploited heuristics. The detailed description of the used statistical measures for automatical keyword extraction is provided in the section 3.1. The following section introduces the notion of a structured query and presents the keyword- and attribute-dependent ranking factors for repository request.

Chapter 4 particularises the software used for the implementation.

Chapter 5 illustrates the datasets used for experimental evaluation.

Chapter 6 demonstrates the evaluation results from processing different types of database requests.

Chapter 7 gives a brief summary of the work done and presents some future research directions.

Chapter 8 lists the the related works.

2 Problem Analysis

The focus of our work is in enabling an ordinary user search through the data in the repository, having only a text document. To make this possible, we need to separate this process in two phases: first we need to extract the keywords that describe the document, and then effectively process the keyword query. That's why we divided this section into two parts: the section 2.1 gives the brief insights into the issue of automatic keyword extraction, keyword search problem will be discussed in the section 2.2. Thereby the existing approaches will be presented in order to familiarize the user with the related works and explain the choice of our conceptual design.

2.1 Keyword Extraction

Automatic keyword extraction is the task to identify a small set of words, key phrases, keywords, or key segments from a document that can describe the meaning of the document [13]. It should be done systematically and with either minimal or no human intervention, depending on the model. The goal of automatic extraction is to apply the power and speed of computation to the problems of access and discoverability, adding value to information organization and retrieval without the significant costs and drawbacks associated with human indexers [7].

2.1.1 Existing Approaches

The manual extraction of keywords is slow, expensive and bristling with mistakes. Therefore, most algorithms and systems to help people perform automatic keyword extraction have been proposed.

Existing methods can be divided into four categories: simple statistics, linguistics, machine learning and mixed approaches [7, 28].

Simple Statistics Approaches

These methods are simple, have limited requirements and don't need the training data. They tend to focus on non-linguistic features of the text such as term frequency, inverse document frequency, and position of a keyword. The statistics information of the words can be used to identify the keywords in the document. Cohen uses N-Gram

statistical information to automatically index the document [4]. Other statistical methods include word frequency, TF*IDF, word co-occurrences [16], etc. The benefits of purely statistical methods are their ease of use and the fact that they do generally produce good results.

Linguistics Approaches

These approaches use the linguistic features of the words, sentences and document. Methods which pay attention to linguistic features such as part-of-speech, syntactic structure and semantic qualities tend to add value, functioning sometimes as filters for bad keywords.

Plas *et al.* [21] use for evaluation two lexical resources: the EDR electronic dictionary, and Princeton University's freely available WordNet. Both provide well-populated lexicons including semantic relationships and linking, such as IS-A and PART-OF relations and concept polysemy. During automatic keyword extraction from multiple-party dialogue episodes, the advantages of using the lexical resources are compared to a pure statistical method and relative frequency ratio.

Hulth [13] examines a few different methods of incorporating linguistics into keyword extraction. Terms are vetted as keywords based on three features: document frequency (TF), collection frequency (IDF), relative position of its first occurrence in a document and the term's part of speech tag. The results indicate that the use of linguistic features signify the remarkable improvement of the automatic keyword extraction.

In fact, some of the linguistic methods are mixed methods, combining some linguistic methods with common statistical measures such as term frequency and inverse document frequency.

Machine Learning Approaches

Keyword extraction can be seen as supervised learning from the examples. The machine learning mechanism works as follows. First a set of training documents is provided to the system, each of which has a range of human-chosen keywords as well. Then the gained knowledge is applied to find keywords from new documents.

The Keyphrase Extraction Algorithm (KEA) [26] uses the machine learning techniques and naive Bayes formula for domain-based extraction of technical keyphrases.

Suzuki *et al.* [23] use spoken language processing techniques to extract keywords

from radio news, using an encyclopedia and newspaper articles as a guide for relevance. The process is separated into two phases: term-weighting and keyword extraction. First, a set of feature vectors is generated from different encyclopedia domains. The same procedure is then performed on a corpus of newspaper articles. The encyclopedia vectors are compared with the article vectors using a similarity calculation so as to separate the latter into different domains, after which they are sorted, producing the final set of feature vectors.

In the second phrase, keyword extraction, a segment is analysed such that the most relevant domain is selected for it using the pre-existing feature vectors. Phoneme recognition software is employed to do the analysis, looking for the best fit between a segment's vectors and that of one of the encyclopedia domains. When the best fitting domain is chosen, its keywords are then assigned to the radio news segment.

Mixed Approaches

Other approaches about keyword extraction mainly combine the methods mentioned above or use some heuristic knowledge in the task of keyword extraction, such as the position, length, layout feature of the words, html tags around of the words, etc [15].

The overview of the related works reveals that the automatic keyword extraction is faster and less expensive than human intervention. Moreover the authors claim that it achieves the precision of the human indexers. However, currently existing solutions for automatic keyword extraction require either training examples or domain specific knowledge. Our approach, on the contrary, doesn't have this additional information. We apply the statistical measures to the automatical keyword extraction as they are domain-independent and have limited requirements. Moreover, in our work we want to analyse how the database context can be exploited in order to automatically extract representative keywords from a document.

2.2 Keyword Search

Keyword search enables user to process his query without any or only little knowledge of the database schema.

2.2.1 Keyword Search Approaches

Many approaches try to satisfy the need for efficient information retrieval over structured and semi-structured data. BANKS [2] models the database as a directed graph where the tuples are the weighted nodes and a foreign-key relationships between the tuples are the directed edges. An answer to a query is then a subgraph connecting nodes matching the keywords. Similarly, Hristidis et al. [12] view a XML database as a graph of segments, where the nodes correspond to labelled XML elements. The aim of this method is to find connections between them that contain all the query keywords. DBXplorer [1] creates auxiliary tables during a preprocessing phase and DISCOVER [11] generates and evaluates networks of tuples. XRANK[9] proposes a PageRank-style ranking for the XML result trees, which combines the scores of the individual nodes of the result tree.

As a potential result, all these techniques return a list of tuple trees that contain *all* the keywords of the query. The main difference between them lies in the ranking function for ordering the results. In our case, for a query consisting of extracted keywords we will not always find an exact match in a database, so we are also interested in partial correlation.

Several systems have been also proposed, that implicitly or explicitly integrate structure-free components into structured queries, and allow a user to specify queries in a loose fashion on XML data. Meaningful Summary Query (MSQ) [5] permits users to write complex queries using only a summary of the schema. But the complexity of writing an MSQ query is comparable to XQuery and far from the simplicity of keyword queries. Florescu et al. [6] extend an existing XML query language in order to support keyword search.

With these methods it is easier for a user to formulate a query. But our user, that have no knowledge of the repository schema, shouldn't care about the learning of query language.

Furthermore, instead of first using structural information of the database schema and then ranking answers, new approaches have been proposed, that translate the keyword query into the correct structured query. SQAK[24] generates the network of ranked structured queries. The results are then obtained by exploiting the fact that keyword query can be answered with just the few most relevant high-scored structured queries. SUITS[29] proposes a framework for efficient constructing relational database query from keywords. The process is split into two phases. During the first pre-processing phase the templates (information about the primary-and foreign-key relations between the tuples) are created. In the second phase SUITS checks for all occurrences of the query terms in database tables and attributes. Then it combines the gained information with the pre-computed query templates and transforms a user's keyword query into structured queries. In the last step the system ranks the structured queries according to their likelihood of matching the user's intent and returns the results from top-k queries.

In our work we use the idea of constructing a structured query with well-defined semantics, but apply it to another kind of repository, represented in the following section. As it differs from a relational database, we introduce several keyword- and attribute-dependent ranking factors, described in detail in the chapter 3.

2.2.2 Entity Repository

. The aim of the Okkam project [19] is to provide a basic set of entity name system (ENS) functionality, it is designed to enable a web-scale system for assigning and managing unique, global identifiers to entities in the WWW[20]. A main aim of an ENS is to provide means for searching for the identifier of an entity. The Figure 2.1 [3] shows the implementation of a single node providing entity identifiers across the system boundaries. While processing a query, the system has to decide whether an entity is already in repository and return a unique entity identifier or whether a new entity should be created.

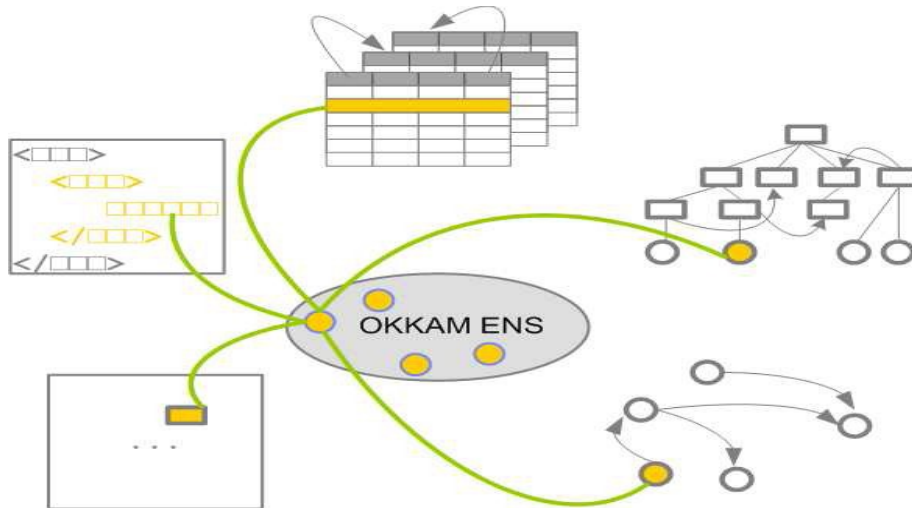


Figure 2.1 The ENS functionality of OKKAM

The Okkam entity repository is a large-scale structured directory, where entity IDs along with some small amount of descriptive semi-structured information for each entity are stored. This description is represented as key-value pairs encoded in XML. The data in repository is partly de-normalized, i.e. an entity can contain some key-value pairs of the referred entities. All entities in the repository are indexed using an inverted index, which includes attribute specific statistics.

The purpose of storing this information is to use it for discriminating among entities.

3 Conceptual Design

This chapter describes in detail the techniques used for automatic keyword extraction in the section 3.1 and our approach for keyword search in the section 3.2.

3.1 Automatic Keyword Extraction

The task of automatic keyword extraction is to identify a set of words, representative for a document. To achieve this we use a simple statistical approach. Thereby, as we intend to exploit the properties of a document and of a repository, we need to find the comparable measures.

One of the simple weighting is TF*IDF. The TF part intends to give a higher score to a document that has more occurrences of a term, while the IDF part is to penalize words that are popular in the whole collection. The further factors such as position of the word in a document or the length of a document are not comparable, as the database entries are much more shorter.

Due to the type of extraction, we divide the automatic keyword extraction into 3 groups:

- Text – Based
- Database – Based
- Text – and Database – Based

Text – Based Extraction

The keyword extraction is conducted exploiting the TF*IDF weight of the term. It is calculated according to the formula:

$$TF * IDF (term) = TF (term) * \log \left(1 + \frac{N}{DF (term)} \right)$$

where $TF(term)$ is the frequency of a term in the given document, N is the total number of documents in the collection, $DF(term)$ is number of documents, that contain the term.

Database – Based Extraction

In this type of extraction we use the database specific statistical information.

As to the entity representation considered in this work, its attribute values tend to

contain in average about 2 words, so we can assume that the tf score equals 1. As term occurrences are usually distributed sparsely in a database, there can be more than one attribute it appears in. We build the average TF*IDF score over all the attributes, that contain the given term.

$$avg (TF*IDF(term,attribute)) = \frac{\sum_{attributes} TF * IDF (term , attribute)}{n}$$

where n is the number of attributes that contain the given term and attribute-specific TF*IDF score of a term is computed as follows:

$$TF * IDF (term, attribute) = 1 * \log \left(1 + \frac{DF (attribute)}{DF (term , attribute)} \right)$$

thereby $DF(attribute)$ is number of entities containing the given attribute and $DF(term,attribute)$ is number of entities where the given term appears in the given attribute.

Text – and Database – Based Extraction

A combined TF*IDF score is then a product of document- and database-specific scores : $TF*IDF (term) * avg(TF*IDF(term,attribute))$.

3.2 Keyword Request Processing

In order to construct the structured keyword request for an entity (i.e. individual, instance, “thing”), we first need to identify the attributes in which each keyword appears. This is performed in one step using an inverted index available in the entity repository. Then the score is computed for every subquery q , which is a combination of an attribute a and a keyword k so that $q = “k \text{ occurs in } a”$. In our work we evaluate several attribute ranking approaches. In the next step, possible structured queries, each being a conjunction of subqueries, are constructed. Finally, these queries are ranked using query ranking criteria discussed in the following in Section 3.2.3 and executed against the entity repository.

3.2.1 Attribute Ranking Factors

The Okkam entity is represented by a set of (attributeName = attributeValue) pairs. As our keyword request is a bundle of terms without the specification of attribute names, our first task is an identification of the attributes where each keyword appears in the repository. Then a specific score is computed for each attribute/keyword pair.

The three intuitive and desirable constraints that any reasonable retrieval formula should satisfy are: term frequency tf , inverse term frequency idf and document length normalization dl . Applied to our attribute-specific approach, the tf heuristic intends to assign a higher score to an attribute of a single entity that has more occurrences of a query term. By intuition, in a collection, the more entities a term appears in a certain attribute, the worse discriminator it is, and it should be assigned a smaller idf weight. The attribute length normalization is to avoid favouring long attributes, as long attributes generally have more chances to match a query term simply because they contain more words.

As to the entity representation considered in this work, its attribute values tend to contain in average about 2 words, so that the tf and dl score will have no effect, as the term usually appears only once pro attribute value and all attributes are approximately of the same length.

Because of this as a basis for our score computing we use only the attribute-specific idf weight of a keyword, which is computed as follows :

Attribute specific IDF score (IDF):

$$IDF(\text{keyword}, \text{attribute}) = \log \left(1 + \frac{DF(\text{attribute})}{DF(\text{keyword}, \text{attribute})} \right)$$

where $DF(\text{attribute})$ is number of entities containing the given attribute and $DF(\text{keyword}, \text{attribute})$ is number of entities where the given keyword appears in the given attribute.

Attribute specific DF score (DF):

Opposite to *idf*, we propose the method that is based on the probability of keyword match in an attribute. The core idea of the *df* score is that probability of the match increases with increasing spreading of the keyword over the attributes. If the keyword appears in the given attribute more frequently than in other attributes than this attribute/keyword combination becomes the higher score than the others. The spread score is calculated according to the formula:

$$DF(\text{keyword}, \text{attribute}) = \frac{DF(\text{keyword}, \text{attribute})}{\sum_{\text{attributes}} DF(\text{keyword}, \text{attribute})}$$

where $DF(\text{keyword}, \text{attribute})$ is number of documents, where the keyword appears in the given attribute. The sum of the *df* scores of different attributes is 1.

The *idf* and *df* scores are keyword-dependent, but what about the attribute itself? How do attributes influence the quality of keyword search, is it helpful to exploit them in our retrieval?

In this thesis we represent two attribute-dependent ranking factors: collection attribute frequency, which reflects the importance of an attribute in the collection and average document attribute frequency, that expresses the cardinality of an attribute in a document.

Collection attribute frequency (CAF):

By intuition, the more documents have the attribute, the higher is the general

importance of this attribute. The collection attribute frequency is computed as follows:

$$CAF(attribute) = \frac{DF(attribute)}{N}$$

where $DF(attribute)$ is number of documents containing the given attribute and N is the total number of documents in the collection.

Average document attribute frequency (avg(DAF)):

In the context of this thesis, the cardinality of the attribute describes the relationship of an attribute with its related values in a single entity. The possible values of connectivity are "one-to-one" or "one-to-many".

A *one-to-one* (1:1) relationship is when an entity attribute has only one value. For example, a movie entity has one title or single production year.

A *one-to-many* (1:N) relationship exists in a case when an entity has pairs attribute_i = value_i, attribute_j = value_j, so that attribute_i = attribute_j and value_i ≠ value_j. An example of a 1:N relationships is actors or different shooting spots.

The 1:1 relations are more descriptive than 1:N, that's why the smaller the number of attribute values per document is, the higher is the relevance of this attribute. The avg(DAF) is calculated according to the formula :

$$avg(DAF(attribute)) = \frac{1}{\log \left(1 + \frac{\sum_{documents} DAF(attribute)}{DF(attribute)} \right)}$$

where $DAF(attribute)$ is the number of times an entity contains the given attribute and $DF(attribute)$ is number of entities containing the given attribute.

Attribute Rank (ARank):

The total global rank of the attribute is then a combination of keyword independent attribute ranking factors.

$$ARank(attribute) = CAF(attribute) * avg(DAF(attribute))$$

3.2.2 Query Score

After obtaining the attribute-specific score for each attribute/keyword combination, our next step lies in constructing the structured query for further request processing. The key idea is that a structured query is composed from subqueries using the *and*-semantics, corresponding to the “and” operator of the boolean model.

Let q_1, \dots, q_n be a set of subqueries that represent the attribute/keyword combinations, a **structured query** Q is then defined as the conjunction of the subqueries $q_1 \circ \dots \circ q_m$, $m \leq n$.

The possible conjunctions of subqueries q_1, q_2, q_3 are presented in a Figure 3.1.

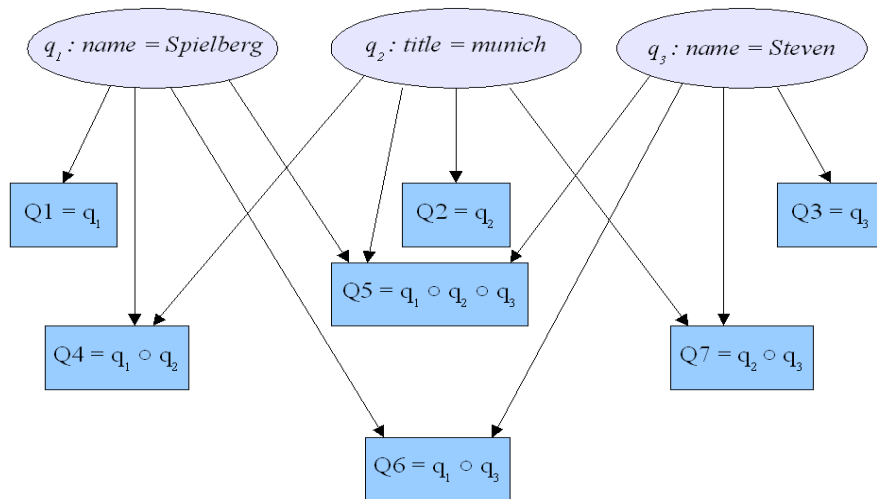


Figure 3.1 Construction of structured query

The relevance of the whole query is represented as a sum of the scores of all subqueries.

$$\text{Score}(\text{query}) = \sum \text{Score}(\text{subquery } q)$$

where $\text{Score}(\text{subquery } q)$ can be defined using a combination of the above attribute

ranking factors. Typical combinations are: IDF, DF, IDF*avg(DAF), IDF*CAF, ARank, IDF*ARank.

3.2.3 Query Ranking Algorithm

The aim of the query ranking procedure is to identify the structured query which delivers possibly precise results to the keyword entity requests. But the number of possible structured queries increases exponentially with the growing number of keywords and attributes in the repository. As for instance Figure 3.1 shows, we become 7 structured queries from only 3 subqueries. For that reason the construction and processing of all intended entity requests will be a very expensive and time-consuming operation.

The first native solution is to construct all possible queries, rank them before execution and process only the high-scored conjunctions. But typically, the number of queries is too high, such that it is infeasible to build and score all possible combinations. Therefore we developed the following optimization algorithm to iteratively calculate the highly scored requests.

Given a sorted subquery list $\{q_{n1} \dots q_{nk}\}$ for all occurrences of a keyword n in different attributes, we build a set $S = \{\{q_{11} \dots q_{1k}\}, \dots, \{q_{n1} \dots q_{nk}\}\}$ for all keywords from 1 to n .

Our task is to limit the number of queries to be constructed, as we are only interested in a few *top-k* highly scored queries. For this purpose we introduce two bounds for the score of the query Q_{top-k} . The upper bound corresponds to the score of the query Q_k , that consists of the subqueries q at k position. The lower bound is the sum of the scores of elements at the $(k + 1)$ -th position in each list. For a query Q_k is true:

$$\text{score}(Q_{k-1}) > \text{score}(Q_k) > \text{score}(Q_{k+1})$$

The intermediate scores are obtained due to the fact that some of highly scored elements at k position can build a number of highly scored combinations with the other lower scored elements in the lists. Due to this fact, a list of queries is constructed with the participation of the subqueries at the position k . The query Q is called *top-k* query

when its score satisfies the condition:

$$\text{score}(Q_k) \geq \text{score}(Q_{top-k}) > \text{score}(Q_{k+1})$$

In the Figure 3.2 we present an example for constructing the top-1 queries. The list of constructed queries consists of 40 queries, but only 11 of them satisfy the score bounds and are considered as top-1 queries. With the native solution there would be 70 possible queries.

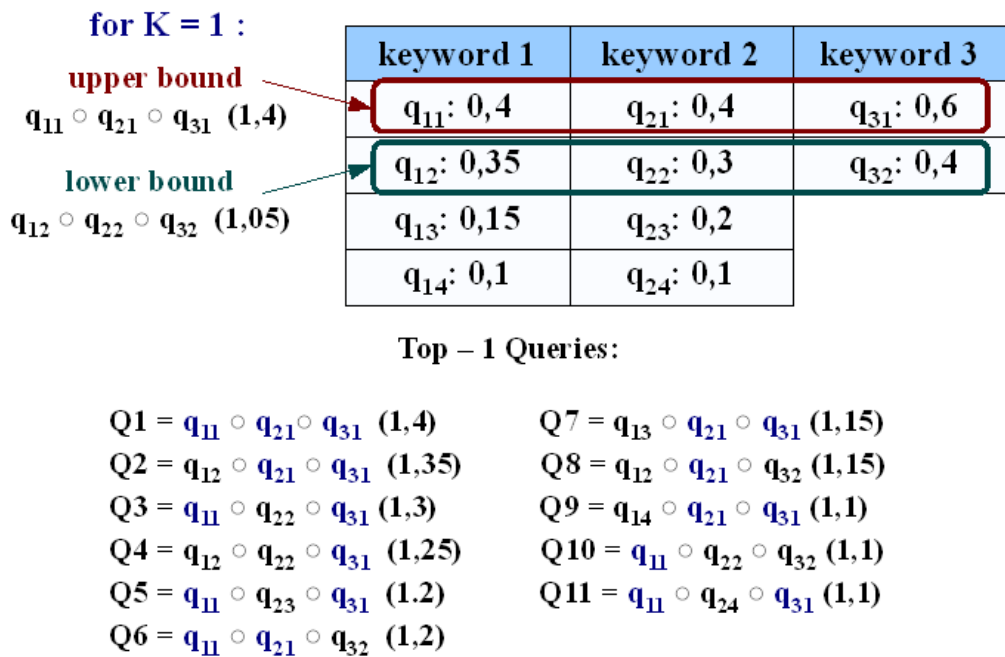


Figure 3.2 An Example for Top-1 Queries.

The requests with the highest scores are then executed till we obtain the intended minimum number of results. Algorithmically this method gives an advantage, especially if the length of the lists (number of attributes) is big.

4 Software Used

We implemented our keyword extraction and disambiguation algorithms on the top of the Okkam entity repository.

The entity IDs along with some small amount of descriptive semi-structured information for each entity are stored using the hbase (version 0.1.2) [10] - the Hadoop database that can manage very large tables. The description is represented as key-value pairs encoded in XML. All entities in the repository are indexed using an inverted index, which includes attribute specific statistics. This capacity is provided by lucene (version 2.3.2).

Lucene [8] is a high-performance, scalable Information Retrieval library. It is a mature, free, open-source project implemented in java. Lucene can index and make searchable any data that can be converted to text format.

The algorithms were implemented using JDK 1.6, all experiments were conducted on the Linux server of the L3S Research Center.

5 Datasets Used

This chapter describes the datasets that we use for experimental evaluation.


5.1 Semi-Structured Dataset

In our work we use the IMDB [14] dataset that consists of 2.347.778 entities, describing persons and 1.263.756 movie entities. Each entity has a unique identifier, called okkamid (oid) and a short description. This description of the entity is represented as key-value pairs encoded in XML. The data in repository is partly de-normalized, i.e. an entity can contain some key-value pairs of the referred entities. All entities are indexed using an inverted index, which includes attribute specific statistics. In total there are 46 different attributes.

Below we present examples of entities, describing person and movie:

Person Entity

```
<?xml version='1.0' encoding='UTF-8'?>
<entity xmlns='http://www.okkam.org/schemas/entitySchema.xsd'>
<oid>http://www.okkam.org/entity/okec9035c7-6cf0-491a-86ac-0aba5fc58d28</oid>
<profile>
<semanticType> movie </semanticType>
<attributes>
  <attribute>
    <name> name </name>
    <value> Aaltonen, Remu </value>
  </attribute>
  <attribute>
    <name> birth notes </name>
    <value> Helsinki, Finland </value>
  </attribute>
  <attribute>
    <name> birth name </name>
    <value> Aaltonen, Henry Olavi </value>
  </attribute>
  <attribute>
    <name> birth date </name>
    <value> 10 January 1948 </value>
  </attribute>
</attributes>
</profile>
</entity>
```



Movie Entity

```
<?xml version='1.0' encoding='UTF-8'?>
<entity xmlns='http://www.okkam.org/schemas/entitySchema.xsd'>
<oid> http://www.okkam.org/entity/ok6c9ea0ff-838f-4c98-bdad-0e458843546c </oid>
<profile>
<semanticType> movie </semanticType>
<attributes>
  <attribute>
    <name> title </name>
    <value> $5000 Reward, Dead or Alive </value>
  </attribute>
  <attribute>
    <name> kind </name>
    <value> movie </value>
  </attribute>
  <attribute>
    <name> production_year </name>
    <value> 1911 </value>
  </attribute>
  <attribute>
    <name> genres </name>
    <value> Short </value>
  </attribute>
  <attribute>
    <name> genres </name>
    <value> Western </value>
  </attribute>
  <attribute>
    <name> release_dates </name>
    <value> USA:8 June 1911 </value>
  </attribute>
  <attribute>
    <name> actor </name>
    <value> Kerrigan, J. Warren </value>
    <veid> http://www.okkam.org/entity/ok61fd1f8d-64c3-4c20-b4c9-1cbf5f4842a8 </veid>
  </attribute>
  <attribute>
    <name> actress </name>
    <value> Bush, Pauline </value>
    <veid> http://www.okkam.org/entity/ok0b509300-407b-4ba8-849a-2ba5314619e6 </veid>
  </attribute>
  <attribute>
    <name> director </name>
    <value> Dwan, Allan </value>
    <veid> http://www.okkam.org/entity/ok45276a61-85cd-43d8-9a68-e920bc3f670d </veid>
  </attribute>
</attributes>
</profile>
</entity>
```

1:N Relationship

Decomposition

5.2 Document Dataset

Wikipedia is a multilingual, web-based, free-content encyclopedia project. It is written by volunteers from all around the world and contains now more than 10 millions articles in more than 260 languages. Today it is one of the largest and most visited sites on the web. For this reason it is often used for as a source for information retrieval.

Wikipedia articles are organised as follows:

- Each subject in the encyclopedia is covered by one article and is identifiable by the article title.
- Articles can also belong to one or more categories, pre-existing or created by the author manually. Encyclopedia users can access the knowledge base by exploring the articles within a category.
- Article can link to other articles, so that the users can navigate following the links.
- Articles may contain an infobox (a relational concise summary of an article: a set of attribute / value pairs describing the article's subject).

Many researches use the Wikipedia's categorisation structure and links to other articles whether to build a thesaurus[18] or to automatically cross-reference the documents and enrich them with links to the appropriate Wikipedia articles[17]. The others[27] benefit from the extraction of Wikipedia infobox attribute values.

Schönhofen (2006) [22] exploits only the titles and categories of Wikipedia articles in order to determine the most characteristic category of a document. The algorithm identifies and ranks all Wikipedia categories supposedly related to the document by matching Wikipedia article titles with words of the document.

In our work we exploit the informativity of Wikipedia articles and use them for conducting the experiments..

As an open source project, the entire content of Wikipedia is easily obtainable. The version used in this study was released in 2006. The full content and revision history at this point occupy 40 GB of compressed data. We consider only the part of the articles, that belong to a category "Film", in total 45086 documents.

6 Evaluation

In this chapter we compare different scoring methods for structured query ranking with respect to their ability to correctly disambiguate keyword query through systematic experiments. We analyse the following ranking functions: IDF, $IDF \cdot \text{avg}(DAF)$, $IDF \cdot CAF$, ARank, $IDF \cdot ARank$ and DF.

For our evaluation we use 50 manually created user's requests and 50 keyword requests, automatically extracted from randomly selected Wikipedia articles.

The automatically extracted requests contain 5 keywords (in this case they can be compared to user's requests, which contain 3-5 words) and are divided into :

- (a) text-based (TF-IDF in the document)
- (b) database-based (average TF-IDF in the database)
- (c) text- and database-based (a product of TF-IDF in the document and average TF-IDF in the database)

For each request exists exactly one entity in the repository, so we execute the keyword search till we find the relevant entity.

The further evaluation is based on 4 aspects :

1. *Precision* (number of answered requests)
2. *Effectiveness* (rank of a relevant result)
3. *Efficiency* (rank of the query, that returns a relevant result)
4. *Relevance* (the number of retrieved results in top-k requests)

6.1 Precision

The quality of a entity request denotes that the intended entity was found. As we were mostly interested how precise the automatically extracted queries are, we executed all structured queries. Table 6.1 shows the evaluation results for both user's and extracted processed entity requests and presents how many intended entities were found.

The high precision of user's requests shows that users can plausibly describe their information need and almost always find the potential keywords. Pleasant is the fact, that in 47 cases a relevant entity was found with both text-based and with a mix of text-based and database-based requests. The similarity of results while using these extracted requests can be explained by the fact, that the extracted keyword queries were alike. As

a consequence, the average idf score from repository adds little to keywords retrieval, it functions only as a filter for the words that don't occur in the repository. Only 10 of extracted database-based requests returned a relevant entity. It means that keywords are not representative for the document. But current results also indicate the importance of the context.

	User's Requests	Text-based Extraction	Database-based Extraction	Text- and Database-based Extraction
Successful Search	50	47	10	47

Table 6.1 Number of successfully answered requests

In our next experiment we've tested the precision of the top-k queries. For this purpose, we proceed with the construction and execution of top-k structured queries till we obtain the first results. The precision of the ranking factor is then calculated as an amount of found relevant entities proportional to the total number of relevant entities. The results are presented in table 6.2, the highest precision is highlighted in red.

	IDF	IDF*avg(DAF)	IDF*CAF	ARank	IDF*ARank	DF
User's Requests	0,89	0,9	0,72	0,92	0,74	0,5
Text-based Extraction	0,54	0,62	0,68	0,72	0,68	0,42
Database-based Extraction	0,16	0,08	0,06	0,04	0,02	0,06
Text- and Database-based Extraction	0,56	0,6	0,6	0,7	0,7	0,4

Table 6.2 Precision of results from Top-k Queries

The results show that ARank has the highest precision for user's requests as well

as for automatically extracted requests. On the whole, we've noticed that the use of attribute-independent ranking factors for scoring extracted requests increases the precision of results compared to pure IDF factor. Interesting is also the fact, that though DF has the highest probability of match in the repository, the returned results lack on precision.

6.2 Effectiveness

The aim of the keyword query is to reveal the most relevant results first. That's why given a keyword query, we execute the structured queries till we found the relevant entity and then assess how the different factors rank the proper result.

The results are illustrated in the Figures 6.1-6.4.

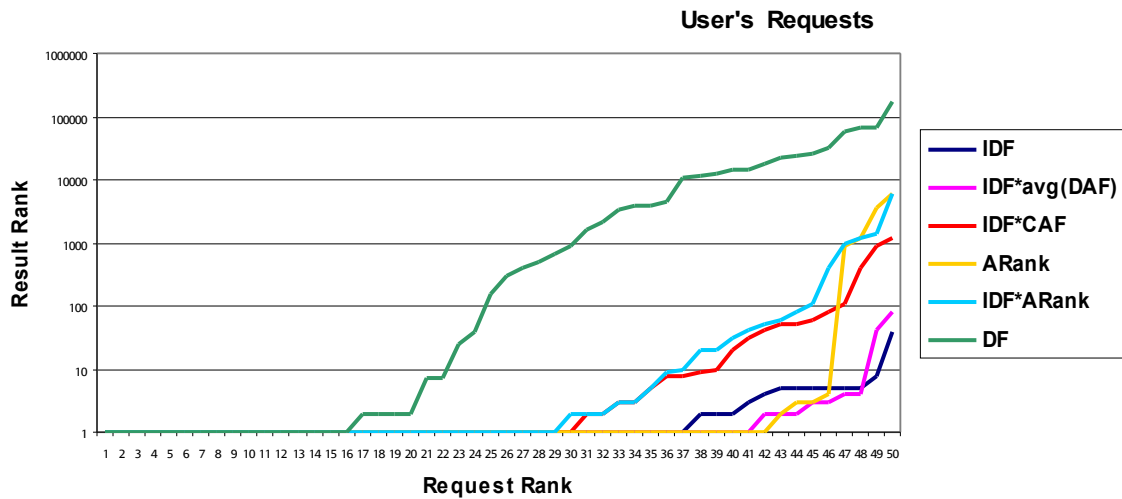


Figure 6.1 Effectiveness of ranking factors in user's requests

We observed different ranking behaviours for the tested approaches. IDF ranking factor functions well for each type of requests. In the case of user's requests it is overtopped only by IDF*avg(DAF). This shows that the use of cardinality factor is important for predicting of the structured query, intended by a user. ARank was able to rank the proper result in the first position for 41 queries, but it is not stable enough. The worst performance was achieved with DF factor.

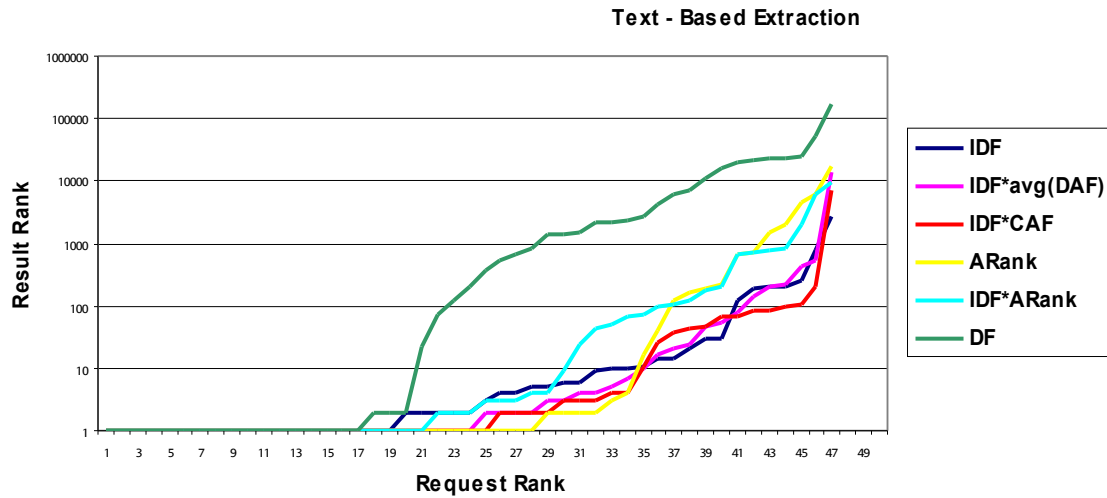


Figure 6.2 Effectiveness of ranking factors in text-based extracted requests

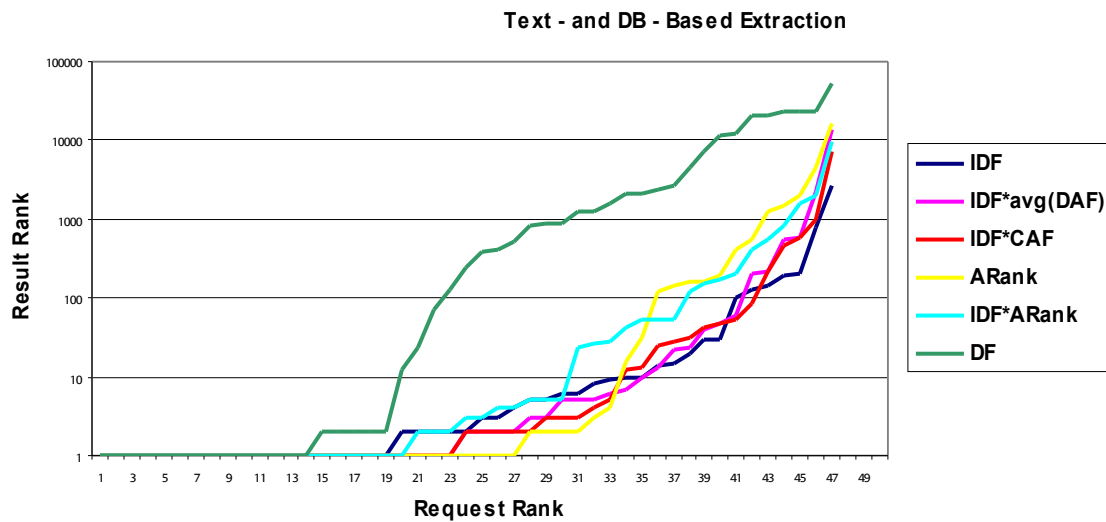


Figure 6.3 Effectiveness of ranking factors in text- and database-based extracted requests

Concerning the text-based and a mix of text- and database-based extracted requests, we've noticed that a combination of IDF and attribute-dependent ranking factors avg(DAF) and CAF improves the effectiveness of the entity request. ARank behaviour is here the same as in user's requests. The text-based extracted queries were more effective than in a combination with the database statistics, it once more proves the importance of the context.

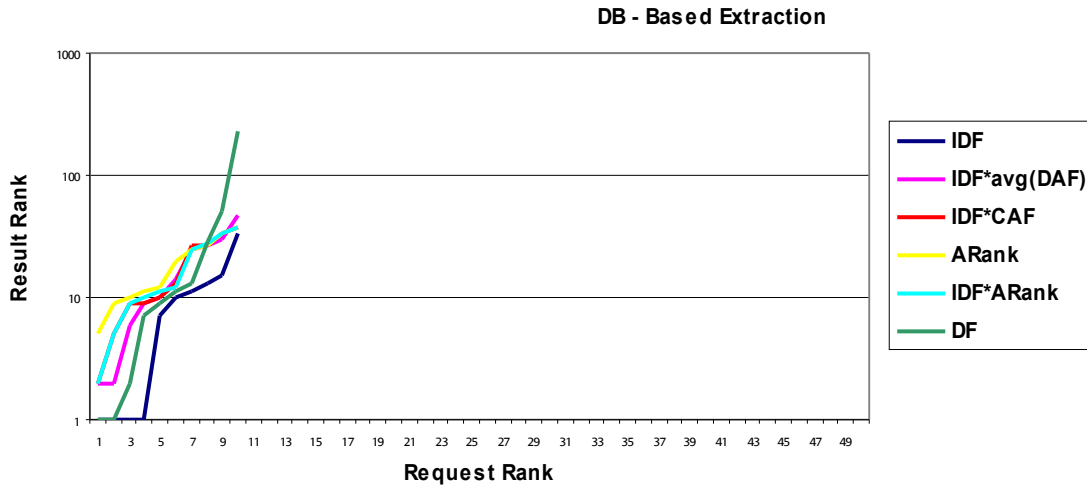


Figure 6.3 Effectiveness of ranking factors in database-based extracted requests

As to the database-based extracted request, where we concern the TF*IDF of the words in the repository, the best performance is achieved by IDF. The other ranking factors slightly decrease the effectiveness of the requests.

6.3 Efficiency

The purpose of ranking function is to disambiguate the structured queries in order to quickly find a proper query for retrieving a relevant entity. The next aspect of our evaluation is the number of queries, that need to be execute for obtaining the intended answer from the repository. The figures 6.5-6.8 illustrate the efficiency of the ranking factors in requests of different type.

Not surprisingly is the fact that the entity search lengthens while using IDF factor. Here we deal with the terms that occur rarely in the collection and a probability for a successful conjunction of subqueries is thus very low. As to the user's requests, the best performance is achieved with a combination with an attribute-dependent ranking factor CAF.

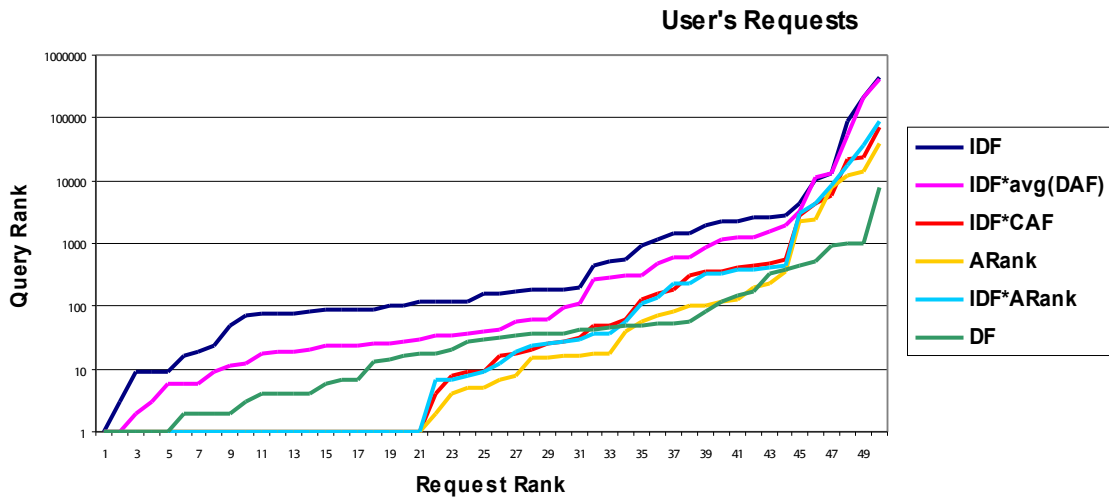


Figure 6.5 Efficiency of ranking factors in user's requests

The attribute-dependent ranking factors increase the efficiency of the extracted requests compared to IDF factor, as shown in the Figure 6.6 and 6.7, but this effect is not so strong as in user's requests.

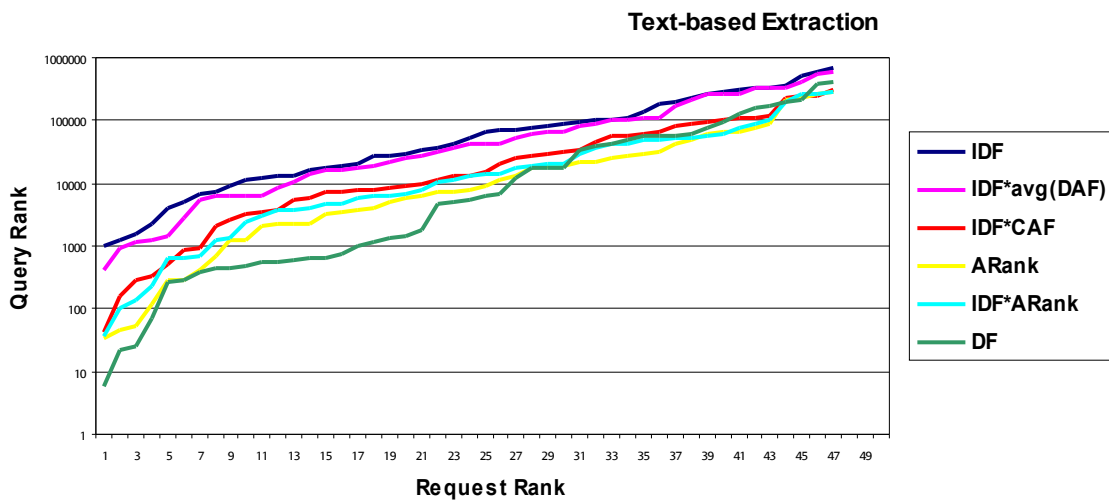


Figure 6.6 Efficiency of ranking factors in text-based extracted requests

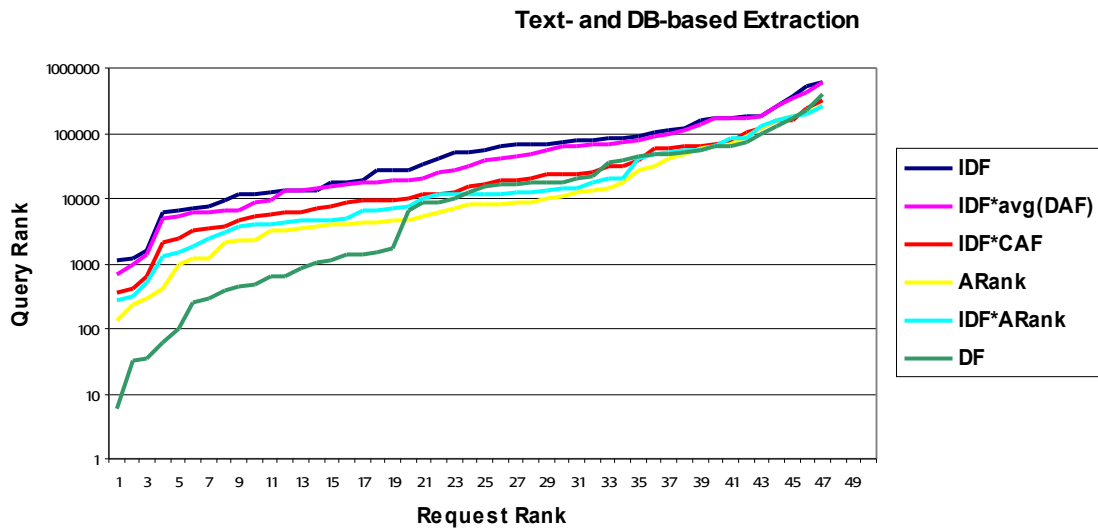


Figure 6.7 Efficiency of ranking factors in text- and database-based extracted requests

The quality of the keywords extracted only with the use of database statistics were not significant enough to make coherent evaluation of the efficiency aspect. This can be possibly because of the average TF*IDF score we use, as it to some extent corrupts the attribute-specific score.

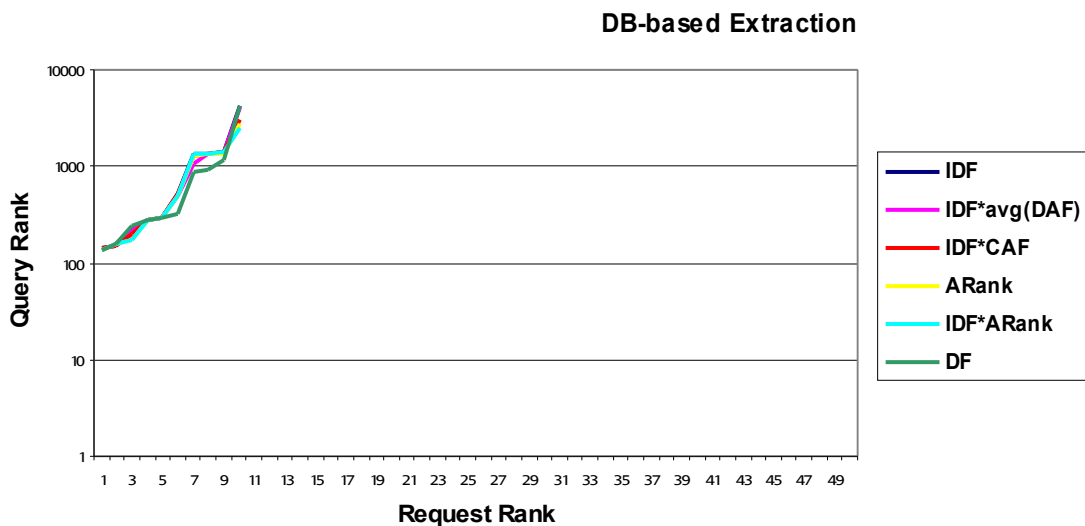


Figure 6.8 Efficiency of ranking factors in database-based extracted requests

6.4 Relevance

An efficient query should return only relevant results. That's why our next aspect of investigation is the total number of results returned from top-k queries. The evaluation results are presented in Figure 6.8-6.12.

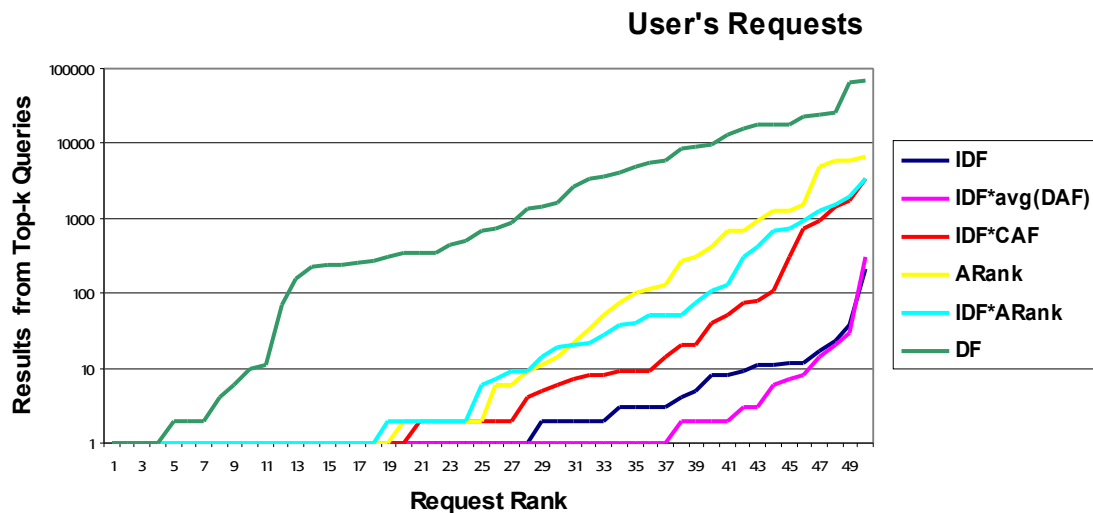


Figure 6.9 Relevance in user's requests

As to the user's requests, the least amount of entities is returned with $IDF \cdot avg(DAF)$ ranking factor, followed by IDF . So we can state that $avg(DAF)$ filters the results returned by IDF . Furthermore, the both factors have a similar behaviour in almost all types of requests.

DF factor returns the greatest number of results, this can be crucial when a keyword query is very specific. Concerning the extracted keyword queries, we notice the better performance of DF factor as compared to user's requests.

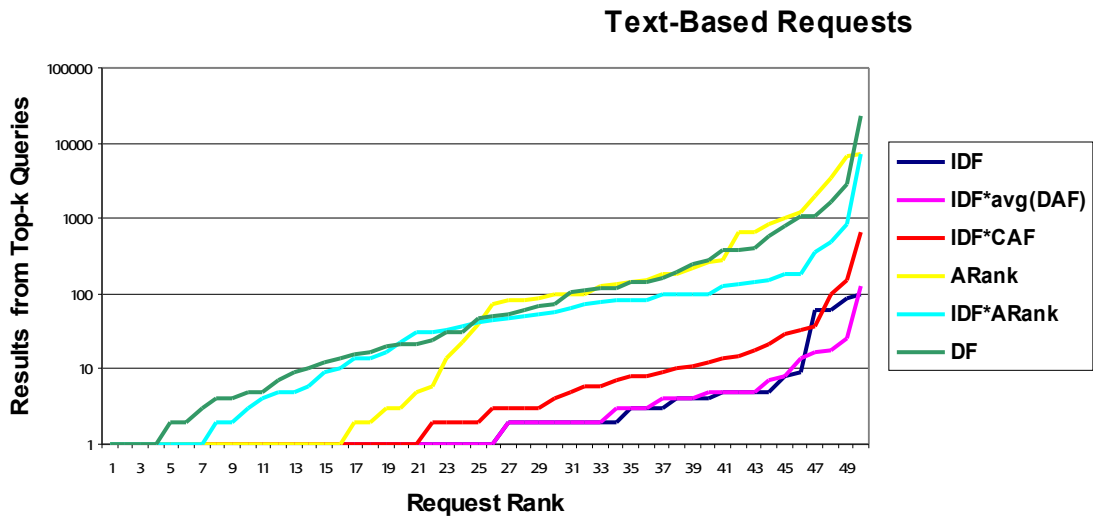


Figure 6.10 Relevance in text-based extracted requests

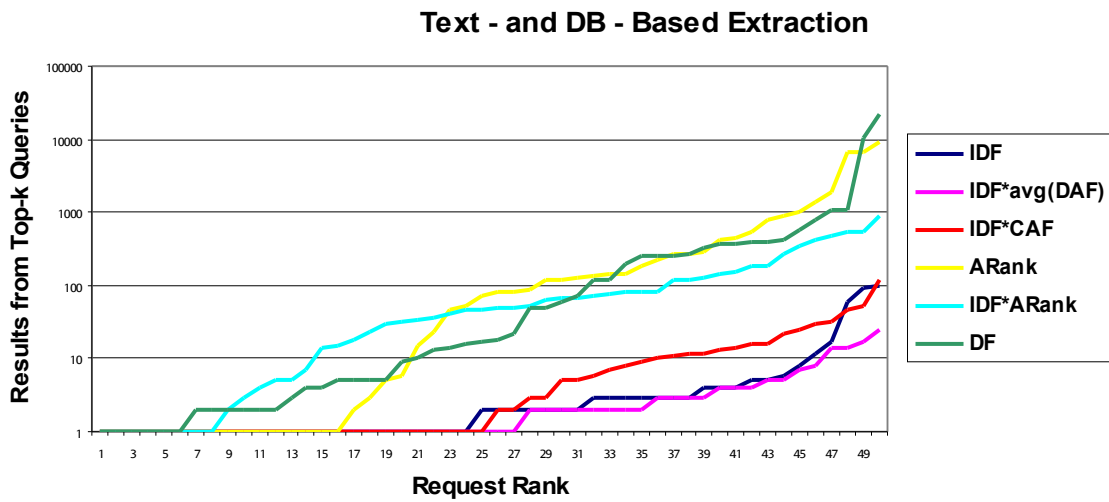


Figure 6.11 Relevance in text- and database-based extracted requests

The relatively small number of returned entities for database-based extracted requests, as shown in Figure 6.12, is due to the rareness of the keywords in the repository. But as we already mentioned, the possible reason is the average TF*IDF factor is error-prone.

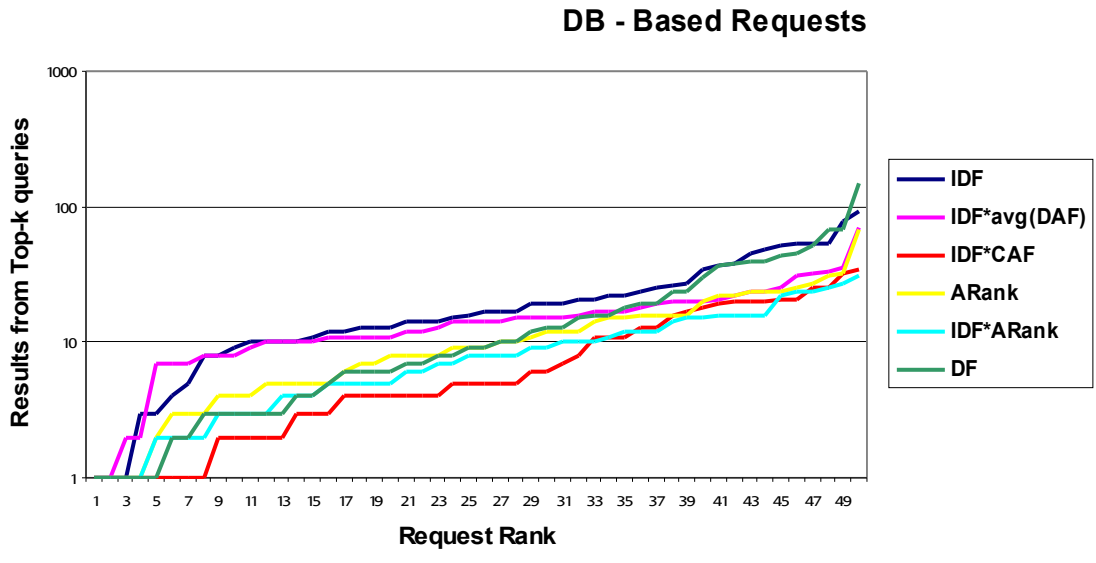


Figure 6.12 Relevance in database-based extracted requests

7 Conclusion

Keyword search interfaces, which are popular by the human users due to their simplicity and usability, provide as well a convenient way for application developers to address the problem of incompleteness of the automatically extracted database requests. As a consequence, keyword-based requests, either manually created, or automatically extracted from unstructured documents, are typically underspecified. Keyword requests lack expressiveness, such that further disambiguation is required in order to precisely match them against the content of the database.

In this work we investigated how the database and document context can be exploited in order to automatically extract representative keywords from a document and retrieve information related to the original document from the database. We analysed the influence of the statistical information either provided by the document collection or the target database on the quality of the extracted keywords and corresponding search results.

During the query answering process we performed disambiguation of the keywords. Thereby the keyword request was translated into a ranked set of structured queries each having well-defined semantics. We compared the factors, which were important for the keyword request disambiguation for user- and automatically extracted requests. We employed statistical metadata about the database content to create and rank such structured queries and analysed how alternative query ranking factors increase effectiveness and efficiency of the search. We identified that the number of possible interpretations of a keyword request over the database grows exponentially with the number of keywords. We introduced an optimization algorithm, which enabled to reduce the number of interpretations, needed to be evaluated by the database.

We implemented our keyword extraction and disambiguation algorithms on the top of the Okkam entity repository. We evaluated our approach using 50 movie related Wikipedia documents and IMDB dataset imported in the Okkam entity repository as well as a set of 50 movie related user requests from a log of a Web search engine. The evaluation of the automatic keyword extraction techniques showed the importance of the context information, like e.g. document or collection content. Database statistics such as average keyword frequency across the database attributes was found to have little influence on the effective keyword extraction. We identified differences in the

ranking factors, which have positive effects on disambiguation of user- vs. automatically extracted keyword requests. Our experiments have shown, that attribute specific selectivity plays an important role in answering both, user- as well as automatically extracted requests. Concerning the user requests, a combination of the selectivity factor with a keyword independent attribute ranking factor $\text{avg}(\text{DAF})$ produced the best effectiveness, whereas the other keyword independent factor, CAF, played a more important role in answering automatically extracted requests. The ranking factors are described in Section 3.2.1 in detail.

This work opens many interesting future research directions. First of all, experiments performed in this work can be repeated on a more heterogeneous dataset, which can, for example, be contained in an entity repository like Okkam. Furthermore, on the one hand, our experiments have shown that keyword extraction based on the document context performs well, whereas on the other hand database statistics we used so far did not enable satisfactory search results. We can further look at the useful database statistics, like attribute specific selectivity to improve these results. Finally, attribute selectivity based query ranking, which was shown to be a very effective ranking factor, produced precise search results at the price of efficiency, as many queries with empty results become the highest ranks. In the future work we can look at the optimisation algorithms to reduce the number of executed structured queries and thus response time of the system.

8. References

1. S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A System for Keyword-Based Search over Relational Databases. In ICDE, 2002
2. G. Bhalotia, A. Hulgeri, C. Nakhey, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In ICDE, 2002
3. P. Bouquet, H. Stoermer, D. Cordioli, G. Tummarello. An entity name system for linking semantic web data. 2008
4. J. D. Cohen. Language and domain-independent automatic indexing terms for abstracting. Journal of the American Society for Information Science, 1995
5. Yu Cong, H.V. Jagadish. Querying complex structured databases. In VLDB, 2007
6. Daniela Florescu, Donald Kossmann, Ioana Manolescu. Integrating keyword search into XML query processing. In WWW9, 2000
7. Michael J. Giarlo. A comparative analysis of keyword extraction techniques. Rutgers, The State University of New Jersey
8. Otis Gospodnetić, Erik Hatcher. Lucene in action, 2005
9. L. Guo, F. Shao, C. Botev, J. Shanmugandaram. XRANK: ranked keyword search over XML documents. SIGMOD, 2003
10. Hbase <http://hadoop.apache.org/hbase/>
11. V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In VLDB, 2002
12. V. Hristidis, Y. Papakonstantinou, and A. Balmin. Keyword proximity search on XML graphs. In ICDE, 2003
13. A. Hulth. Improved automatic keyword extraction given more linguistic knowledge. In Proceedings of the 2003 Conference on Empirical Methods in Natural Language Processing, Sapporo, Japan, 2003
14. Internet Movie Database <http://www.imdb.com/>
15. J. B. Keith Humphreys. Phraserate: An HTML keyphrase extractor. Technical Report. 2002

16. Y. Matsuo, M. Ishizuka. Keyword extraction from a single document using word co-occurrence statistical information. International Journal on Artificial Intelligence Tools, 2004
17. David Milne, Ian H. Witten. Learning to link with Wikipedia. In CIKM, 2008
18. David Milne, Olena Medelyan, Ian H. Witten. Mining domain-specific thesauri from Wikipedia : A case study. In WI, 2006
19. Okkam Project <http://www.okkam.org/>
20. T.Palpanas, J. Chaudhry, P. Andritsos, Y. Velegrakis. Entity Management in OKKAM. 2008
21. L. Plas, V.Pallotta, M.Rajman, H.Ghorbel. Automatic keyword extraction from spoken text. A comparison of two lexical resources: the EDR and WordNet. Proceedings of the 4th International Language Resources and Evaluation, European Language Resource Association, 2004
22. Peter Schönhofen. Identifying document topics using the Wikipedia category network. In WI, 2006
23. Y. Suzuki, F. Fukumoto, Y. Sekiguchi. Keyword extraction of radio news using term weighting with an encyclopedia and newspaper articles. SIGIR, 1998.
24. Sandeep Tata, Guy M. Lohman. SQAK: Doing more with keywords. SIGMOD, June 9–12, 2008
25. Wikipedia <http://www.wikipedia.org/>
26. I. Witten, G. Paynte, E. Frank, C. Gutwin, C. Nevill-Manning. KEA: practical automatic keyphrase extraction. In Proceedings of the 4th ACM Conference on Digital Library, 1999
27. Fei Wu, Raphael Hoffmann, Daniel S. Weld. Information extraction from Wikipedia: moving down the long tail. In KDD'08, 2008
28. Chengzhi Zhang, Huilin Wang, Yao Liu, Dan Wu, Yi Liao, Bo Wang. Automatic Keyword Extraction from Documents Using Conditional Random Fields. Journal of Computational Information Systems, 2008
29. Xuan Zhou, Gideon Zenz, Elena Demidova, Wolfgang Nejdl. SUITS: structuring user's intent in search. In EDBT, 2009