Indexing Heterogeneous Datasets for Open Question Answering

Harsh Thakkar, and others

Enterprise Information Systems Lab,
University of Bonn, Germany

hthakkar@uni-bonn.de

Abstract. Abstract will appear here. *When it's time!*

1 Introduction

Introduction will appear here.

1.1 Early inception

Computers or earlier referred to as autonomous computing/calculating machines had started being used for the purpose of searching through texts from the the early 1950s's. It was then, the field of Information Retrieval (IR) begun to emerge as a distinguished research domain. The main application of IR was for a group of librarians who wanted to automate the process of referencing and searching through huge amount of texts. IR emerged in this with key challenges such as: How to *index* these texts or documents and then how to *retrieve* them later for referencing and/or searching.

Searching in the early 1950-1960's included approaches which were as simple as using regular keywords from the text for indexing, to which later on moved to great reforms incorporating the foundation of probability and likelihood of occurrence of a keyword in the text. This probability based approach also included the ranking of the search results based on their relevance (*i.e.* the documents retrieved were sorted based on the words used in the query). The work from 1960 and onwards, laid a crisp foundation for the development of a variety of algorithms for ranking and retrieval of content, established the use of computers as an ultimate apparatus for search.

In the table below we present an overview of the decade-wise developments in the field of information retrieval and search for a historic understanding.

Year	Progress
1950's	First text indexing algorithm developed, using keyword matching
1960's	Ranked retrieval based on query terms is introduced
1970's	TF-IDF based approached developed (Luhn's TF weights, S. Jones's IDF)
1980's	Probability theory amalgamates with ranked retrieval and search
1990's	TF-IDF, Vector space model based variants developed (LSI)
2000 onwards	BIM, OKapi-BM25 based complex statistical ranking approaches allowing full text indexing and searching capabilities developed

1.2 Motivation

1.3 Paper selection methodology or Range of years covered?

2 Types of datasources

2.1 Semi OR Un-structured datasources

HTML pages (Web crawls)

Text dumps

2.2 Hybrid datasources

Triple stores

SQL-ORDBMS - OpenLink Virtuso

find out and list

3 Types of queries

3.1 Structure aware queries

Predicate queries

SPARQL queries

XQuery queries

SQL queries

3.2 Structure unaware *or* partially aware queries

Boolean queries

Keyword queries

– Wildcard queries

Natural language queries

4 Indexing support for QA

With the explosion of data and in-experienced user in the current time, the challenges of information retrieval have also increased by leaps and bounds. System which were based on primarily keywords matching, now-a-days are observed to return very poor quality results due to the out burst of data. An efficient search engine is therefore required to make use of a variety of additional factors for improving performance such as heuristics, linguistic resources and also statistical models.

A typical Question Answering (QA) system is empirically only as good as the performance of its indexing module. The performance of indexing servers as an upper bound to the overall output of the QA, since it is can process only that much data which is being presented/served to it in the indexes. The precision and recall of the system may be good, but if all or most the top relevant documents are not indexed in the system, the user's needs have to suffer.

Many researchers have compared the effectiveness between manual and automatic indexing techniques. Manual indices were often presumed to be better than machine generated indices. However, it has been demonstrated that both indexing techniques are equally effective for text retrieval [Salton, 1989]. The retrieval performance were also showed positive improvement if both techniques were combined compared to individual indexing technique [Callan et al., 1993, Rajashekar & Croft, 1993].

4.1 Indexing techniques and underlying data structures

needs major revision
where to add the manual, semi-automatic, automatic indexing?

In this section we discuss in detail the wide range of indexing techniques which have been categorized into three predominant categories namely: Manual, Semi-automatic and automatic. We also discuss the reasoning of a wide spectrum of underlying fundamental datastructures which are used in these techniques respectively.

1. **Indexing approaches for unstructured data:**

(a) **Manual Indexing:**

(b) **Semi-automatic Indexing:**

(c) **Automatic Indexing:**
here goes to

(d) *Inverted Indexing:*
In this method, the words are mapped to their respective position in the document. A sparse matrix structure is used with columns representing dimensions like document numbers (the identification number of each document where the term is present), Term, and Term ID. Whereas, the rows are the list of different terms to be indexed. Since the documents are searched using the terms present in them, it is called *inverted index*. This method tells us whether a word is in a particular document or not, but nothing about its frequency of occurrence. This makes the naive inverted indexing approach less useful for evaluating the relevance of a document according to the user's query. In their work [ref_here] propose variants of inverted lists to overcome this challenge.

(e) *Latent Semantic Indexing (LSI):* As the name suggests, "latent" means hidden, and "semantic"

(f) *Probabilistic LSI:*

2. **Indexing on semi-structured [confirm with other papers about semi or full structured data] data:**

   (a) *XML Indexing techniques*

BITMAP index Bitmap Indices A bitmap index is a special type of index that is tailored for being efficient for key columns with relatively few distinct values, i.e. low cardinality key columns. A bitmap index is created with the normal create index statement by putting the bitmap keyword in front of index, as follows:

create table customer (c_id int primary key, c_state char (2), c_gender char (1), ...);

create bitmap index c_gender on customer (c_gender);
create bitmap index c_state on customer (c_state);

Bitmap indices offer space savings of up to 1000 x in some cases and specially for large tables the savings in I/O can be very significant.

A bitmap index can only be used on tables with an integer primary key or in other situations where the last effective key part of the index is an integer. This is understandable since a bitmap index uses a bitmap for representing the values of the last key part, thus having one bitmap for each distinct combination of leading key parts. In the above example, the customer table has an integer c_id column as primary key and a character for the customer's gender and a 2 character field for the state where the customer is located. Thus, to count all the male customers in Massachusetts, one will take the males bitmap and the MA bitmap and perform a bitwise AND of the two. This will have a 1 bit corresponding to the c_id of each male customer in Massachusetts.

We also note that in order to do the count, the customer table itself does not even have to be referenced, as the bitmaps hold all the information. Even if the table did have to be referenced, for example for adding up the outstanding credit of all male customers in Massachusetts, the bitwise AND could be done first and only the relevant rows would have to be retrieved from the table itself.

Virtuoso's implementation of bitmap indices is designed to work efficiently even when the leading key parts have relatively high cardinality, i.e. many distinct values, causing there to be a large number of mostly empty bitmaps. Of course, if each bitmap has only one bit set, for example if every customer is in a different state, there is no benefit to bitmap indices. On the other hand, there is also almost no penalty, only 6 bytes more per index entry than for a regular index. Therefore, for any non-unique key where bitmap indices are applicable, even if there are only a few repeated values, bitmap indices are a safe choice. If there are at least 2 times more rows than distinct values of the keys, space savings are certain.

A bitmap index may have any number of key parts of any type, provided that the last effective part is an integer or and IRI id. The last key parts of an index are those primary key parts that do not occur elsewhere in the key. Thus, if the primary key is a single integer, bitmap indices are always applicable. However, supposing the primary key were an integer plus a string, it would be possible to make a bitmap index where the string were first, followed by the integer. This would make sense and save space if the string were not unique by itself. As another example, the RDF_QUAD system table, the default location of the Virtuoso RDF triple store, has the columns P, G, O and S, where all are IRI ID's, except for O which is ANY. Thus, The primary key is the concatenation of all columns, by default in the order GSPO. There is another key in the order PGOS which can be implemented as a bitmap index because S is an IRI ID, hence integer-like for purposes of bitmap indices.

9.31.1. Bitmap Indices and Transactions The minimum locking unit is the row. In the case of a bitmap index, one row holds a bitmap which most often refers to many rows. Locking is therefore less granular than with regular indices. Thus, if multiple threads insert rows with bitmap indices, more waits may occur than if the index were not bitmapped. A single row of a bitmap index references maximally 8192 other rows, most often however the count is much less. In all other respects, locking and transactional behavior are identical with other indices.

9.31.2. Performance Implications The main advantage of a bitmap index is more compact size, reflected in less I/O. Inserting an entry takes on the average 10% longer than for another type of index, likewise for random lookups with exact key values. Sequential access is usually faster. Space savings and thereby improved working set behavior can produce dramatic gains for large tables.

9.31.3. Physical Structure and Overheads Bitmap indices divide the range of signed 64 bit integer values into ranges holding 8192 (8K) values. Each such range where at least one bit is set is represented by a compression entry (CE). Multiple CE's can be on the same row. CE's having one bit set take 4 bytes, CE's with 512 or less bits set take 4 bytes plus 2 bytes per bit, CE's with over 512 bits set take 1K byte regardless of how many bits are set. A bitmap index where the bitmap holds only one bit takes 6 bytes more than the corresponding non-bitmap index entry. A second value, if it falls in the same 8K range adds 2 bytes, 4 bytes if it does not fall within in the same 8K range. If more than 512 values fall within the same 8K range, the bits are represented as a 1K byte bitmap and adding subsequent values takes no extra space.

5 Question Answering systems

QA systems which have been developed for addressing the need of the users based on a variety of of structured and unstructured datasources. We present our efforts to review almost all of the major advancements in the scenario in context.

5.1 SWOOGLE (2004)

5.2 AQUALOG (2005)

5.3 Power Aqua (2006)

Lopez et al. introduce PowerAqua, another open source system, which is agnostic of the underlying yet heterogeneous sets of knowledge bases. It detects on-the-fly the needed ontologies to answer a certain question, maps the users query to Semantic Web vocabulary and composes the retrieved (fragment-)information to an answer. However, PowerAqua is outperformed by TBSL (see below) in terms of accuracy w.r.t. the state-of-the-art QALD 3 benchmark.

5.4 Eyphra (2006)

Schlaefer et al. describe Ephyra, an open-source question answering system and its extension with factoid and list questions via semantic technologies. Using Wordnet as well as a answer type classifier to combine statistical, fuzzy models and previously developed, manually refined rules. The disadvantage of this system lies in the handcoded answer type hierarchy which prohibits its multi-lingual use.

5.5 YARS2 (2007)

5.6 ESTER (2007)

5.7 SINDICE (2008)

5.8 QAST (2009)

5.9 SIG.MA (2010)

5.10 WATSON (2011)

5.11 TBSL (2011)

5.12 MAYA (2011)

5.13 QAKiS (2012)

Cabrio et al. present a demo of QAKiS, an agnostic QA system grounded in ontology-relation matches. The relation matches are based on surface forms extracted from Wikipedia to enforce a wide variety of context matches, e.g., a relation birthplace(person, place) can be explicated by X was born in Y or Y is the birthplace of X. Unfortunately, QAKiS matches only one relation per query and moreover relies on basic heuristics which do not account for the variety of natural language in general

5.14 SINA (2013)

Shekarpour et al. have developed SINA, a keyword and natural language query search engine which is aware of the underlying semantics of a keyword query. The system is based on Hidden Markov Models for choosing the correct dataset to query. Due to the costly Hidden Markov Models SINAs answer time (on average 3.9s) is above enduser expectations.

5.15 FReyA (2013)

Damljanovic et al. present their work FREyA, to tackle ambiguity problems when using natural language interfaces. Many ontologies in the Semantic Web contain hard to map relations, e.g., questions starting with How long. . . can be disambiguated to a time or a distance. By incorporating user feedback and syntactic analysis FREyA is able to learn the users query formulation preferences increasing the systems question answering precision.

5.16 QUADS (2014)

5.17 HAWK (2014)

Usbek et al. present their work HAWK, which is claimed as the first hybrid QA system. In this paper they showcase the capability of HAWK by making use of a generic approach in order to generate SPARQL queries from predicate arguments. They auothors report HAWK to achieve a F-measure score above 0.61 based on the QALD-4 hybrid query benchmark. HAWK is portrayed more towards serving as a plug and play architecture for other domain-specific QA systems. However, in doing so, the inherent components of HAWK system are computationally extremely complex. This work also puts forward a couple of open questions to be addressed in context of advancing the current state of ranking algorithms. This will serve towards bridging the semantic gap between the predicate arguments to an interpretation in the present context of the query in a generic QA system, to aim a more competitive score.

5.18 MEANS (2015)

In order to thoroughly compare the performance of the indexing techniques, many researchers tested various query formulation methods, or query types, on each of the techniques. Previous studies with English texts showed that phrase and Boolean queries perform better than natural language queries [Croft et.al., 1991; Belkin et.al., 1993]. The results of the studies also show that adding word-proximity operators to the Boolean and phrase queries, resulting in structured queries, can further improve the queries effectiveness for large collections.

6 Influence of indexing on system performance

parameters go here.........

1. Datasource selection
2. POS tagging & NER
3. Indexing parameters
4. Ranking model

In this paper we only focus on the indexing module of the system, and thus we discuss the indexing parameters in detail with respect to their influence on overall system performance. The performance of an indexing module of a QA system can be enhanced by the following measures.

– Weighted indexing: Applying weights to certain terms in the constructed index with respect to the users information need, allows the system to vary the depth of the search or exhaustiveness of the search.

– linking of terms : This refers to the process of associating entities ,i.e. terms in our context. Associations (relationships) are created between the terms that are relatively close in the meaning or are often synonyms of each other. The linking of terms eradicates the retrieval of false or irrelevant information by reducing the span of index search.
– Role indicators – need data here
– Subheadings: This reduces the ambiguity in search of the electronic index database. Sub headings are should be used widely where ever possible in the post co-ordinate (thesauri based) indexing and retrieval systems, since they provide a crisp focus to the domain of search.
– Index language device: This refers to the domain specific components in the indexing mechanism of terms. For instance, if the data to be indexed if of medical domain, specific medical vocabularies and ontologies such as MeSH, PubMed and etc are used to boost a certain choice of words as compared to the others. This helps in the overall performance of the system.

All the tables of comparison and summary go here.. **Table:**
listing each component of all survey systems.
i.e. Graph based systems, SPARQL used, Type of Model used (HMM, Naive bayes, ANN, etc). and a column for notes/special remarks
check mark every component used to build it.

Chart:
Papers published on different applications or research areas that make use of indexing.
Histogram of year wise papers published in such areas?

Other sections are here.

7 Conclusion

Conclusions are here.

Acknowledgments

This project is supported by funding received from the European Unions Horizon 2020 research and innovation program under the Marie Sklodowska-Curie grant agreement No 642795.

References

1. A. Einstein, On the movement of small particles suspended in stationary liquids required by the molecular-kinetic theory of heat, Annalen der Physik 17, pp. 549-560, 1905.