NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE

DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

DISSERTATION

Keyword search in RDF databases

Charalampos S. Nikolaou

Supervisor: Manolis Koubarakis, Associate Professor N.K.U.A.

ATHENS
NOVEMBER 2010

NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS

SCHOOL OF SCIENCE

DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

DISSERTATION

Keyword search in RDF databases

Charalampos S. Nikolaou

Supervisor: Manolis Koubarakis, Associate Professor N.K.U.A.

ATHENS
NOVEMBER 2010

DISSERTATION

Keyword search in RDF databases

Charalampos S. Nikolaou
R.N.: M953

SUPERVISOR:

Manolis Koubarakis, Associate Professor N.K.U.A.

EXAMINATION COMMITTEE:

Ioannis Ioannidis, Professor N.K.U.A.

Abstract

We present the design and implementation of a keyword-based querying system operating on RDF databases. The system employs a keyword-based query language extended with temporal constructs in the form of all Allen's thirteen temporal relations: before, after, overlaps, etc. The underlying data model is able to capture indefinite temporal information defining the time-limits over which a certain fact is valid. Indefinite information concerning a time point can be given as an interval in which the point must lie. In the case of indefinite information about time intervals, the starting and ending time points of an interval are defined in the same way. We also evaluate our design and implementation commenting on its results. For this, we employ three datasets that exhibit different characteristics. We evaluate our system in terms of efficiency and effectiveness, i.e., scalability, query time response performance, and quality of results. We also study related work on keyword-based search for structured and semi-structured data and present the state-of-the-art on evaluation methodologies for keyword-based search techniques. We position our work in that context and present new challenges that subsequent research attempts should address.

SUBJECT AREA: Semantic Web, Databases, Knowledge Bases

Keywords: keyword search, keywords, query processing, RDF, graphs, temporal information

Dedicated to my loving nan,
1912–2008

Acknowledgements

First and foremost, I would like to thank my supervisor, Manolis Koubarakis, for his patience, trust, and kindness. Then, I would like to thank my colleagues, Vivi, Manolis, Alex, George, and Vissarrion for supporting me both psychologically and spiritually, and for being available to listen to and discuss various matters of this dissertation. Last, I thank Knud Möller, who provided me with the query logs of the Semantic Web Dog Food site.

Contents

List of Figures

List of Tables

Prologue

This dissertation has been conducted during my postgraduate studies in the Department of Informatics & Telecommunications, in the program of Computer Systems Technology, 2009–2010. In this period, I was also involved in the EU funded project Papyrus: Cultural and historical digital libraries dynamically mined from news archives. Papyrus was the context over which this work was developed, applied, tested, and evaluated.

Chapter 1

Introduction

Nowadays, keyword search is the predominant way of searching a data source such as the Web. Using solely *keywords*, i.e., a small number of highly discriminating terms the user anticipates that she will identify the web pages most relevant to her information needs. Keyword search offers a straightforward, intuitive, and yet flexible method of retrieving information. The success of keyword search in the field of Information Retrieval (IR) and the World Wide Web (WWW or just Web) has generated interest in keyword search interfaces to relational databases and similar structured and semi-structured data sources.

This dissertation studies the way keyword search has evolved over time, how it has been adopted by different fields in computer science, such as IR, databases, and semantic web, and surveys the state-of-the-art in keyword search for fields managing structured and semi-structured data. Beyond that, it presents and extensively evaluates the design and implementation of a system operating on RDF data, accepting keyword queries with temporal constraints. Last, it presents the state-of-the-art in evaluation of keyword search systems operating on structured and semi-structured data.

1.1 Keyword search in the Past

In the past, keyword search was studied mainly in the area of IR[1]. This is, mainly, due to its application area, e.g., library systems (bibliographic catalogs, patent collections, text archives, etc.), which motivated the development of the respective technologies. The users of such an application area are non-technical humans with cognitive capabilities and limitations [Wei07]. Consequently, IR systems aim at understanding user queries as approximate, trying, first, to reveal the actual information needs of the user, and then supporting an interactive process of data exploration, query rephrasing, and guidance towards the final results. In this respect, the field of IR views query processing as a ranking task based on statistical models.

[1]In IR's terminology keyword search is known as *free text search* or *full text search*.

On the other hand, the field of databases, while initiated roughly at the same time that IR did, it targeted a very different application area: that of accounting systems, such as online reservations, banking transactions, etc. In contrast to IR, the users of databases consist mainly of technically skilled people, adept at using a specific language to interact with a database system. In turn, a database system expects a user to pose precise queries, aiming at providing exact results in one shot and as fast as possible. In this respect, the field of databases views query processing as a matching task based on testing logical predicates.

However, from a technical point of view, the key difference between IR and databases seems to lie in emphasizing different data types: text in IR, and numbers in databases, or more precisely, unstructured text documents in IR, and structured records with numerical and categorical attributes in databases.

It is evident that the two directions and their respective research communities emphasized very different aspects of information management: text understanding, statistical ranking models, and user satisfaction on the IR side, and data consistency, precise query processing, and efficiency on the databases side.

The picture of these two fields has only started to change in the late 1990s, when there were attempts towards their integration. The most notable of them were the probabilistic Datalog [Fuh95], the probabilistic relational-algebra models [FR97], and the WHIRL approach to similarity joins [Coh98, Coh00], which all brought IR's imprecision to databases. In addition, commercial database systems adopted various IR techniques to provide imprecise text matching and full-text search capabilities in the textual attributes of a relation, and thus paving the way towards enabling keyword search in the field of databases.

1.2 Keyword search in the Present

It is only in the past few years that the need for integration of IR and databases methods are so compelling to efficiently and effectively support keyword search in both fields. This is, at first, due to the emergence of various mission-critical applications, such as Digital Libraries (DL). Digital Libraries are information repositories growing exponentially in size, with documents augmented with metadata and annotations expressed in semi-structured data formats, such as XML, and more recently RDFa[2]. From an IR point of view, there is a need for efficiently storing and querying such data, which is admittedly a characteristic that can be credited to databases. From the perspective of databases, application areas, such as customer support and customer relation management systems (CRM), product and market research, as well as social networks and blogs appeared lately in the Web, require support for structured and textual data, as well as ranking and recommendation in the presence of uncertain information of highly diverse quality.

[2]http://www.w3.org/TR/xhtml-rdfa-primer/

Unfortunately, until the beginning of 2000s keyword search facilities in databases was anything but adequate. The user of a database system had to master a structured query language, know the schema of the database, and then express his information needs in that language. While this way of accessing a database is powerful enough, its learning curve is very steep for ordinary users to adopt. Moreover, imprecise textual matching on specific attributes of a relation is not effective. This is because information needed to answer a keyword query is often split across relations/tuples, due to normalization requirements. These reasons have given rise to a new research community arguing that the logical schema of a database, while being an appropriate abstraction over the underlying physical schema for data organization, it is still at a level too low to serve as the abstraction with which users should interact directly. Instead, a higher level presentation data model abstraction is needed to allow users to structure information in a natural way. Clearly, there is a trend towards making databases more usable [JCE$^+$07].

Keyword search in databases is evolving towards this direction over the last decade (2001–2010). The demand for this need is even higher taking in mind that keyword-based searching for information is an indispensable task of people's lives. Web search engines are widely used for searching textual documents, images, and videos. There are also vast collections of structured and semi-structured data, such as XML and RDF, both on the Web and in relational databases.

To serve its new purpose, keyword search over structured and semi-structured data should automatically assemble relevant pieces of information that are in different locations, but are inter-connected and collectively relevant to the query. Such an approach, apart from addressing the aforementioned requirements, it also allows users to easily access heterogeneous databases. For instance, for websites with various database backends, this approach provides a more flexible search method than the existing solution that uses a fixed set of pre-built template queries. Furthermore, this approach helps to reveal interesting or unexpected relationships among entities. Making databases searchable will substantially increase the information volume that a user can access, have potential to provide search results with better quality compared to keyword search on textual documents and the Web, and thus increase the database usability and make significant impact to people's lives.

1.3 Keyword search in the Future

Still, the current approaches to keyword search over text documents, web pages, structured and semi-structured data, are based on inverted indexes and structural information available through hyperlinks, foreign-primary key relationships, parent-child relationships in XML, and property relationships in RDF.

From the perspective of unstructured data, it is often the case in a web setting that different web pages are ranked very differently, while providing information about the same facts, the same persons, the same events, etc. What makes ranking different is that web pages are authored by different people, with different cultural, cognitive, and technical background, probably speaking different languages, and mastering the subject at different levels. No wonder, there is an increasing need for recognizing named entities and their relationships, as well as place and time attributes expressed in natural-language sentences. These can be made explicit through information extraction techniques, such as pattern matching, statistical learning, and natural language processing. There is also demand in exploiting current state-of-the-art techniques for striking out the particularities and the language of one's writing, keeping only the essence of the writing. Of course, this approach exhibits uncertainty; querying such extracted knowledge entails ranking.

From the structured and semi-structured data point of view, there is a similar problem, but disguised under the veil of integration. Still, the problem of unstructured data co-exists in this setting on textual attributes (databases), or literals (XML/RDF). Here, the norm is that applications access multiple data sources, such as databases, and semi-structured data stored completely as an attribute of a relation, or even separately. Making matters worse, the choice of the data sources to be used for a query often can be made at run-time. Even if each source contains structured data records and comes with an explicit schema, there is no unified global schema unless a breakthrough could be achieved to magically perform perfect, on-the-fly, data integration. So, the application program must be able to cope with the heterogeneity of the underlying schema names, XML tags, and RDF, and queries must be schema-agnostic or at least tolerant to schema relaxation.

According to [Wei07, WKRS09], the solution is in the construction, development, and maintenance of knowledge bases, which have received considerable attention in the recent databases, IR, WWW, and semantic web literature. For example, information-extraction techniques [ERS+09] have been successfully applied to textual as well as semi-structured web sources, such as Wikipedia[3], to build large-scale knowledge repositories, such as DBpedia [ABK+07], Freebase [BEP+08, BCT07], YAGO [SKW08], and also community specific collections, such as DBLife [DCG+08] and Libra [NMS+07]. These repositories typically contain entities, such as people, locations, movies, companies, conferences, organizations, etc., and the relationships between them, such as `bornIn`, `actedIn`, `hasGenre`, `isCEOof`, `isPCMemberOf`, and so on. Such data conceptually forms a large graph with nodes corresponding to (typed) entities and edges denoting (typed and possibly weighted) relationships, and it can be conveniently represented in the form of subject-property-object (SPO) triples of RDF. When triples are extracted from web pages — or even annotated —, they can be associated with a variety of weights, including extraction confidence, the number of times the triple was seen in the corpus, entity extraction confidence, etc. Unquestionably, IR and databases methods could indeed have the potential to play major roles in this endeavor. Furthermore, keyword search should definitely be the prevalent query method to this kind of data, due to its intuitive and simple interface, which has been widely adopted by experienced

[3]`http://en.wikipedia.org/`

and inexperienced users as well.

The construction, development, and maintenance of such knowledge bases has also been dictated both from the emergence of the semantic web technologies and the enormous availability of RDF data in the Web the last few years. This web of data bears enormous potential for supporting web users in accomplishing more complex tasks and ultimately bringing about new possibilities for commercial exploitation. Several initiatives have been started to deal with and to promote this web of data, noticeably the Linking Open Data (LOD) project[4] and the Billion Triple Challenge[5].

1.4 Objectives and contributions of this dissertation

This dissertation follows the spirit of the research work that has been done in keyword search during the last ten years (2001–2010), as discussed above. In line with the trends of this decade and future demands as they have been addressed in [Wei07, WKRS09, CRW05, JCE$^+$07], two different types of data are addressed, structured and semi-structured. For structured data, the prevalent field of computer science is the field of databases; this type of data deals with tuples and relations of the relational data model [AHV95]. For semi-structured data, the prevalent research areas are the fields of Data Management, Data Integration, Data Exchange, the Web, the Semantic Web[6], etc. Consequently, the main representative of these research areas are the Extensible Markup Language (XML)[7], the Resource Description Framework (RDF) [LS99], and the RDF Schema (RDFS) [BG00], from which the works based on the RDF(S) framework have been chosen to be presented.

Although the first objective of this dissertation is to record the related work done in keyword search, it is worth noting that it does not aim at surveying this field. Instead, it aims at giving the full picture of the evolution of keyword searching, initiated by the field of IR, and then adopted by the fields of web and databases, and how each field has contributed to each other and in keyword searching independently. This way, the demands and trends of each research era can be identified and elevated, together with the research problems that each area has faced.

In this respect, for the presentation of the related work in keyword search, a number of categorical dimensions have been identified and extracted, which are considered significant from the perspective of this dissertation, and are orthogonal to the studied data models. As such, they allow for parallelly viewing the way in which keyword search has influenced the above mentioned fields in chronological order, and how the solutions and techniques developed in these different areas — being initially independent to each other — have come to be adopted by all of them. However, for completeness, other dimensions and directions together with references to the respective work have been identified and

[4]http://linkeddata.org/
[5]http://www.cs.vu.nl/pmika/swc/swapplication.html
[6]http://www.w3.org/2001/sw/
[7]http://www.w3.org/XML/

briefly presented.

The second objective of the dissertation is to advance the state-of-the-art research in keyword search over RDF data. To this goal, the contributions of this dissertation lay on the design, implementation, and evaluation of a system supporting keyword searching over RDF data. In particular, the work of this dissertation differentiates itself from related work, because it adds the temporal dimension to keyword-based querying. To this end, first, it introduces a new data model able to capture temporal information. Second, it introduces a keyword-based query language with temporal constructs capturing all Allen's temporal relations [All83]. Third, it improves and extends the keyword search algorithm proposed in [TWRC09] incorporating the temporal dimension. Fourth, it extensively evaluates the proposed keyword querying system both in terms of efficiency and effectiveness, employing three different datasets each one exposing different characteristics. To the best of the author's knowledge, there is no related work providing such rigorous evaluation results. Overall, regarding RDF data as a knowledge base, and taking into account the future trend and demands in keyword search [Wei07, WKRS09], the proposed work is on the cutting edge of the research — thus well targeted —, and can be safely regarded as indispensable.

The third and last objective is to shed light on the evaluation of systems and techniques targeting at keyword search over structured and semi-structured data. Keyword search provide only an approximate specification of the information items to be retrieved. Therefore, the correctness of the retrieval cannot be formally verified, as it is the case with query languages, such as SQL. Instead, retrieval effectiveness is measured by user perception and experience. The empirical assessment of keyword-based systems is therefore imperative. Here, the aim is to study and review the current initiatives and methodologies, which, unquestionably, but justifiably, make this field immature, due to its immense difficulty. Conversely, the area of IR has made a great progress towards the evaluation of its systems and techniques with the establishment of TREC (Text REtrieval Conference)[8], which provides a solid ground and a rigid methodology for evaluation.

1.5 Dissertation Outline

The rest of the dissertation is organized as follows. Chapter 2 discusses related work in the field of keyword search over structured and semi-structured data, and it presents the current status of the evaluation methodologies and techniques of systems that support keyword searching over such data. Besides that, it presents other directions in keyword searching. Then, Chapters 3 and 4 discuss the approach taken by this dissertation to the problem of keyword search over RDF data, presenting the design and implementation of a keyword querying system. Next, Chapter 5 discusses Papyrus, an European Union funded project in the context of which this dissertation has been conducted. Chapter 6 discusses the evaluation methodology followed in this dissertation,

[8]`http://trec.nist.gov/`

and comments on its results. Last, Chapter 7 concludes the dissertation, recapitulating and identifying various directions and open problems for future work.

Chapter 2

Related Work

This chapter discusses related work in the field of keyword search over structured and semi-structured data. First Section 2.1 introduces some basic background knowledge needed for understanding the techniques employed throughout the dissertation. Second, Section 2.2 introduces a number of dimensions, which are then used to categorize related work. This assists the reader to obtain a clear and complete view of related work, identify several aspects playing an important role in keyword-based search, and easily compare related work with this dissertation. Furthermore, for each dimension an approach taken in related work is picked for presentation.

Next, Section 2.3 discusses the current status of the evaluation methodologies and techniques of systems that support keyword searching over structured and semi-structured data. It is pointed out that the current state is far from good, without providing any solid and rigid methodology, as opposed to the respective evaluation process developed in the field of IR. Fortunately, this area has already started to attract the interest of the research community leading to initiatives, such as INEX[1] (INitiative for the Evaluation of XML), and interesting and promising evaluation methodologies.

Last, Section 2.4 discusses other directions taken in keyword searching, which diverge from the path of query processing. Such works focus on the improvement of user interaction with systems providing keyword search interfaces, as well as improving the quality of search results.

2.1 Prerequisite Knowledge

As prerequisite knowledge we define and describe data models (Subsection 2.1.1), the RDF framework (Subsection 2.1.2) and the RDF Schema (Subsection 2.1.3), as well as the Temporal RDF data model (Subsection 2.1.4).

[1] `www.inex.otago.ac.nz`

2.1.1 Data Models

A *data model* [Ull88] is a mathematical formalism with two parts:

1. A notation for describing data, and
2. A set of operations used to manipulate that data.

Examples of such data models in the area of databases are the Relational Data Model [AHV95, Ull88], the Object Relational Model [Ull88], the Network Data Model [Ull88], etc. The relational data model logically organizes data in *relations* composing of *tuples* of *attributes* and includes such operations as `select`, `project`, `join`, etc. The object relational model considers *relations* as *objects* and organizes them in a hierarchy. The network data model is the entity-relationship model with all relationships restricted to be binary, many-one relationships. This restriction allows for using a simple directed graph model for representing the actual data. It organizes data using two fundamental constructs, called records and sets. In this model, data is organized in *records* and *sets* (not to be confused with mathematical sets). Records contain fields (which may be organized hierarchically, as in the programming language COBOL). Sets define one-to-many relationships between records. The operations of the network model are navigational in style: a program maintains a current position, and navigates from one record to another by following the relationships in which the record participates. Records can also be located by supplying key values.

In the area of IR, an example of a data model is the Vector Space Model (VSM) [MRS], which considers *documents* as *vectors/points* in a $n-$dimensional space. Additionally, the RDF framework can be regarded itself as a data model forming a graph of $(subject, predicate, object)$ *triples*. The nodes of that graph are the $subject$ and $object$ parts of these triples. The $predicate$ part correspond to labeled, directed edges emanating from a $subject$ node, and ending in an $object$ node.

2.1.2 The RDF framework

The Resource Description Framework (RDF) [LS99] is a language for representing information about resources in the World Wide Web. Resources are identified using URIs (Uniform Resource Identifiers)[2] and described using triples. A triple consists of three elements: the `subject`, the `predicate`, and the `object`, and it is usually written as follows:

$$(subject,\ predicate,\ object)$$

, where $subject \in U \bigcup B$, $predicate \in \bigcup U$, and $object \in U \bigcup B \bigcup L$. The set U stands for the set of URIs, B for the set of blank nodes (these are resources that cannot be identified exactly with a specific URI), and L for the set of RDF literals. Using the triple

[2]`http://labs.apache.org/webarch/uri/rfc/rfc3986.html` [RFC3986]

notation, a statement concerning the creator of the web page `www.example.com` can be represented as:

$$(\text{www.ex.com/person/p1},\ \text{www.ex.com/creator},\ \text{www.ex.com})$$

A set of RDF triples is called a RDF graph.

2.1.3 The RDF Schema

The RDF Schema (RDFS) [BG00] is a language for defining *vocabularies* that can be used in RDF graphs. These vocabularies specify the *classes* and *properties* that can be used in a domain modeled by an RDF graph. Classes and properties are used for describing groups of related resources and relationships between resources. Classes are sets of resources. Elements of a class (nodes in an RDF graph) are known as *instances* or *individuals* of that class. To state that a resource is an instance of a class, the property `rdf:type` may be used. The following are the most important classes in RDF(S): `rdf:Resource`, `rdfs:Class`, `rdfs:Literal`, `rdfs:Datatype`, `rdf:XMLLiteral`, and `rdf:Property`. Properties are binary relations between subject resources and object resources. The built-in properties of RDF(S) are: `rdfs:range`, `rdfs:domain`, `rdf:type`, `rdfs:subClassOf`, and `rdfs:subPropertyOf`. To avoid distracting the reader, the formal semantics of the RDF Vocabulary are not presented here, except one: all things being described by RDF expressions are called resources, and are considered to be instances of the class `rdfs:Resource`. The class `rdfs:Resource` represents the set called “Resources” in the formal model for RDF(S).

2.1.4 Temporal RDF

The RDF(S) data model fails to capture the different semantics that arise with the description of statements containing temporal information. The first works that proposed temporal features in RDF were by Gutierrez and colleagues [GHV05, HV06, GHV07]. In their proposal, a framework to incorporate valid time in RDF is introduced. Extending the concept of RDF triple, a *temporal triple* is an RDF triple with an additional temporal label (a natural number). For example, $(s, p, o)[t]$ is a temporal triple which denotes the fact that the triple (s, p, o) is valid at time t. Triples valid at time intervals are then defined by sets of triples valid at time points. Finally, a *temporal RDF graph* is defined as a set of temporal RDF triples. The works in [GHV05, HV06, GHV07] study the semantics of the proposed extension to RDF, define appropriate query languages for the extension and present results on the complexity of query answering. The representation and reasoning about incomplete temporal information in RDF has also been studied in [HV06]. The authors of [HV06] treat indefinite temporal information (i.e., anonymous time) as blank nodes incorporating, also, temporal constraints based on Allen’s interval algebra [All83]. The resulting graphs are called *c-temporal graphs* for which they provide semantics by extending the semantics of temporal RDF graphs defined in their earlier works [GHV05, GHV07].

2.2 Categorization of keyword-based search approaches

Here, the dimensions used for categorization of related work on keyword-based search are introduced. These include the employed data model (Subsection 2.2.1), the structure of the answer (Subsection 2.2.2), the underlying exploration algorithm (Subsection 2.2.3), the ranking/scoring functions of answers (Subsection 2.2.4), and indexing techniques (Subsection 2.2.5).

2.2.1 Data Models

Clearly, the various data models proposed in the area of databases deal with the same kind of data, i.e., numerical and categorical, but each of them provides a different view over them employing different kind of operations on that data. This specific view is tightly associated with the respective *query language* used to access the stored data. The query language is the interface between the database system and the user that needs access to the data, and is designed to reflect the structure and properties of the underlying data model.

From the perspective of this dissertation a query is simply a set of keywords. In the databases setting, the problem of answering such a query amounts to finding tuples having attributes with values *matching* a keyword from that set (according to some matching criteria), and then trying to *connect* them following a primary-foreign key relationship. This sense of connection is the one making the graph-based (tree-based) data model the most appropriate to handle such queries. That is, the query language and the peculiarities of the relational data model (e.g., the organization of closely related information across different relations) dictated the usage of another data model on top of the relational one. Indeed, all related work discussed here from the area of databases use such a graph data model on top of the relational one. This is due to the wide spread and tremendous development of technologies that are based on the relation data model. In this respect, the proposed research works are interoperable and can be directly applied to such systems.

Similarly, for semi-structured data, and especially for RDF data, the data model is a graph. Consequently, all approaches to keyword search employ a graph data model, either undirected or directed for representing the underlying data and their inter-relationships. It is also possible that weights can be associated with each graph element, that is, node or edge, to reflect its importance in relation to the other graph elements. However, the key differentiation of these approaches is whether they take into account the schema of the underlying data (schema-aware) or not (schema-agnostic). In relational databases this schema is that of the database. Likewise, in RDF, it is the RDF Schema (RDFS) [BG00].

2.2.1.1 Schema-aware

Schema-aware approaches model the underlying data as a graph based on their schema definition. In databases, such an approach maps database relations to nodes, and the edges represent relationships between two relations, such as primary-foreign key dependencies [ACD02, HP02, HGP03, LYMC06, LLZ07, MYP07, QYCT09, SKI08, QYC09]. In RDF, this approach maps RDF classes to nodes, and the properties that should appear between the instances of two classes are mapped to edges. The edges are labeled with the name of the property [LUM06, TCRS07, TWRC09].

These approaches frequently involve two phases for the generation of the answer to a keyword query. First, they take as input the schema graph and any elements matching a keyword of the query, and then try to derive the schema of every possible answer[3]. From the databases side, such matching elements might be a relation, either because it matched against a keyword, or because a tuple in that relation matched against it. From the RDF side, it might be a class, or a property, that is, an edge between two class nodes, or even a node/edge of the data graph. Second, given these answer schemas, appropriate queries are constructed to retrieve the actual tuples from the database (or entities in the case of RDF).

In the literature, schema-aware approaches can also be found as *schema-level*, or *schema graph*.

2.2.1.2 Schema-agnostic

Schema-agnostic approaches model the underlying data as a data graph. In databases, such an approach maps tuples to nodes, and edges to primary-foreign key dependencies [HBN$^+$01, BHN$^+$02, BHP04, KPC$^+$05, HWY07, LOF$^+$08]. In RDF, the data graph is precisely the graph that is implicitly formed from the RDF triples (see Subsection 2.2.1) [HWY07, LOF$^+$08, TLR, WLP$^+$09, LT10].

These approaches involve one phase for the answer generation, where the answer schema extraction and the tuple/entity retrieval task are interleaved: the system explores the data graph trying to build trees/graphs connecting all the keywords of the query, in the case of AND semantics, or some of them, in case of OR semantics.

It is worth mentioning that the works based on a schema-aware data model in a databases setting are significantly more than those based on a schema-agnostic one[4]. In contrast to databases, in a semi-structured setting, the works based on a schema-aware data model are significantly fewer than those based on a schema-agnostic one. This observation is justified taking in mind that the data stored in relational databases must adhere to a specific schema, which is not the case for semi-structured data.

[3]This phase can be found in the literature as *keyword query translation*.

[4]Works such as [HWY07] and [LOF$^+$08] have to be excluded from the schema-agnostic citation list, due to the fact that they are proposed for general graph structured data, and not especially for the area of databases.

Accordingly, in the literature, schema-agnostic approaches can also be found as *tuple-level*, or *data graph*.

2.2.2 Structure of the answer

Currently, posing a keyword query at a web search engine, the result is a ranked set of web pages on account of containing these keywords. Comparing this approach with the approaches taken for handling keywords queries on structured and semi-structured data, the result is generally a structured item to reflect apart from the plain information contained in the keywords, the way this information is connected, defining, in this way, a possible interpretation of the answer. Consequently, such an item might be a *tree* [HBN+01, BHN+02, HP02, ACD02, HGP03, KPC+05, LYMC06, LLZ07, MYP07, HWY07, QYCT09, QYC09], a *graph* [LOF+08, SKI08], or just a *node/entity* [ACD+03, BHP04, LUM06, WLP+09, LT10] of these structures.

Other approaches, especially in RDF, stop at the stage of query computation letting the user pick the query close to his information needs [TCRS07, TWRC09]. They compute the queries corresponding to all possible interpretations of the user keyword query, and then present them in a graph structure to the user. Last, there are works, towards providing the interpretation of a structured answer in natural language [SKAI08, KSI10].

2.2.3 Exploration Algorithms

Many approaches to keyword search that generate tree or graph structured answers deal with the problem of finding substructures connecting the elements containing the keywords of the query. General approaches that rely on tree and graph structured data are mainly *Backward Expansion* [HBN+01, BHN+02], Bidirectional Search [KPC+05] and their extensions [HWY07, TWRC09]. In the following, backward expansion and bidirectional search are presented.

Backward Expansion The Backward Expansion algorithm [HBN+01, BHN+02] first identifies the nodes of the graph containing a keyword of the query, called *keyword nodes*, and then performs an iterative traversal along incoming edges of visited nodes, until a node is found, called *answer root*, with the special property of being a node connected to all keyword nodes through a path of visited outgoing edges. This iterative traversal is performed initiating a Dijkstra single source shortest path algorithm for each keyword node. If a keyword is contained in more than one nodes, then these nodes form a cluster for which a single copy of the Dijkstra algorithm is instantiated. This cluster is called *Initial Keyword Cluster*, and is denoted as IKC_i, on account of containing nodes matching with the i-th keyword.

At each iteration of the algorithm there are two strategies for choosing the direction (i.e., the node) to follow (i.e., to visit) next. Before proceeding to their presentation, some terminology is introduced. The keyword query is a set of keywords $K = \{k_1, k_2, \ldots, k_n\}$. A *Keyword Cluster*, KC_i, is the set of nodes that have been visited and can reach keyword

k_i, i.e., $IKC_i \cap KC_i = IKC_i$. Initially, each such cluster contains all the keyword nodes for each k_i, i.e., $KC_i = IKC_i$. The expansion strategies are the following:

Equi-distance expansion in each keyword cluster: Having picked a keyword cluster, KC_i, this strategy decides which node to visit next. Intuitively, this strategy expands a keyword cluster by visiting nodes in order of increasing distance from the initial keyword cluster. Formally, the node to visit next for a keyword cluster KC_i is the node (among all nodes not in KC_i) with the shortest distance to a node in the set IKC_i.

Distance-balanced expansion across keyword clusters: This strategy decides the frontier of which keyword cluster will be expanded. Intuitively, the algorithm attempts to balance the distance between each initial keyword cluster and its frontier across all clusters. Formally, let (u, KC_i) be the node-cluster pair such that $u \notin KC_i$ and the distance from u to IKC_i is the shortest possible. Then, the cluster to expand next is KC_i.

Bidirectional Expansion The same authors proposed the Bidirectional Expansion algorithm [KPC$^+$05] to overcome the poor performance that Backward Expansion exhibits in certain types of graphs. The Bidirectional Expansion algorithm is an extension of the Backward Expansion algorithm that has the option of exploring the graph by following forward edges as well. The rationale is to identify a root node much faster. To control the expansion order, the algorithm prioritize nodes by heuristic *activation factors*, which intuitively estimate how likely nodes can be answer roots. While this strategy is shown to perform well in multiple scenarios, it is difficult to provide any worst-case performance guarantee. The reason is that activation factors are heuristic measures derived from general graph topology and parts of the graph already visited; they may not accurately reflect the likelihood of reaching keyword nodes through an unexplored region of the graph within a reasonable distance.

2.2.4 Ranking/Scoring of Answers

Keyword searches are inherently ambiguous and not all query results are equally relevant to a user. Various ranking schemes have been proposed to order the query results into a sorted list so that users can focus on the top ones, which are hopefully the most relevant ones. Various ranking schemes are used in existing work, which consider both the properties of data nodes (e.g., TFIDF and complex measures adopted from IR, node/edge weight, ranking in the style of page rank) and the properties of the whole query result (e.g., path length, number of nodes/edges, weights of nodes/edges, size normalization) [GKS08, HWY07, LOF$^+$08, LYMC06, LLZ07, HBN$^+$01, BHN$^+$02, KPC$^+$05, ACD02, HP02, HGP03, TWRC09, WLP$^+$09].

In particular, in [HBN$^+$01, BHN$^+$02] answers are ranked using a notion of proximity coupled with a notion of prestige of nodes based on incoming links, similar to techniques developed for web search. In addition, the overall ranking scheme includes weighting of edges, and it is the additive or multiplicative combination of edge and node scores. Representing the overall node and edge score with N_{score} and E_{score} respectively, then the overall score of an answer is given as follows:

$$\begin{aligned}\text{Additive} &: (1-\lambda) * E_{score} + \lambda * N_{score} \\ \text{Multiplicative} &: E_{score} * N_{score}^{\lambda}\end{aligned}$$

Considering answer trees, [HBN+01, BHN+02] compute the overall node score as the average of the scores of the root and leaf nodes, i.e., omitting any intermediate nodes. The score of a node u is determined by the number of incoming edges $Pr(u)$, conveying this way the notion of prestige. This score is normalized using the maximum prestige score of the nodes of the underlying graph:

$$Score(u) = \frac{Pr(u)}{\max_{u \in V(G)}\{Pr(u)\}}$$

Works such as [HGP03, LLZ07, LYMC06, LOF+08, TWRC09] employ a number of IR measures to rank answers. In [HGP03, LLZ07] they exploit the IR techniques built-in in RDBMs, while in [LYMC06] they extend them providing novel and more fine grained scoring measures (tree normalization, document length normalization, document frequency normalization, inter-document frequency normalization, different scoring schemes for schema and value matching keyword term). In [LOF+08] they propose a novel index which materializes TFIDF-based IR rankings. In contrast, the work [TWRC09] employs the Lucene index to textual ranking.

Apart from ranking answers based on textual similarity, other factors are also considered, such as structural compactness, completeness factor, and popularity measures (i.e., simplified variations of page rank ranking). All these factors can be used independently or combined in a additive or multiplicative way as mentioned previously (or following variations of these basic schemes). The structural compactness of answers is determined by the sum of the path lengths present in the answer as in [TWRC09] or a more sophisticated combination of the path lengths between any two nodes n_i, n_j of the answer as in [LOF+08]:

$$Score(n_i, n_j) = \sum_{P_{ij}} \frac{1}{(|P_{ij}|+1)^2}$$

where P_{ij} is any path between nodes n_i and n_j, and $|P_{ij}|$ is the length of such a path.

The authors of [LLZ07] consider answers that are relevant to any subset of query keywords. To quantify the factor of the number of matching query keywords, they employ the completeness factor, which has been recognized as significant by IR researchers in the face of short queries.

Another aspect differentiating approaches to scoring is whether the score of an answer is computed from the combination of the scores of its components. Most approaches use monotonic functions, in which the score of an answer reflects an aggregation function over the scores of its components. Approaches such as [LLZ07, LYMC06] use non-monotonic functions, in which the score of an answer is independent to its components. Especially for top-k query processing approaches, non-monotonic functions have to be paired with a respective monotonic upper bounding function to efficiently determine the top-k answers.

2.2.5 Indexing

Subsection 2.2.1.2 made an observation that schema-aware works in databases are more than those concerning semi-structured data. And this was due to the fact that it is very difficult to find semi-structured data adhering to a specific schema, as opposed to structured data. In fact, most of the publicly available semi-structured data, even implying a specific schema, come without explicitly defining one, in order to be flexible for future updates, and able to include data containing incomplete and indefinite information, which otherwise would require revisiting the schema and modifying the portion of data adhering to the old schema, possibly introducing further incomplete and indefinite information. Taking also into consideration the increasing amount of heterogeneous and schema-less data, as well as the need for their efficient storing, databases are employed extensively to this mean. Consequently, this imposing need for schema-agnostic techniques, apart from having been reflected on databases also, it has led to approaches leveraging special graph indexes to handle their increasing size and process keyword queries more efficiently [HWY07, LOF$^+$08, TLR, WLP$^+$09].

For emphasizing the drift of keyword search approaches to indexing, it is worth mentioning the BANKS approach [HBN$^+$01, BHN$^+$02], which employs a schema-agnostic data model by mapping a tuple to a graph node and a primary-foreign key dependency to a graph edge. Specifically, the authors had assumed that such a graph can fit in main memory [BHN$^+$02]. The verbatim argument was the following:

> We assume that the graph fits in memory. This is not unreasonable, even for moderately large databases, because the in-memory node representation need not store any attribute of the corresponding tuple other than the RID. The only other in-memory structure is an index to map RIDs to the graph nodes. Indices to map keywords to RIDs can be disk resident. As a result the graphs of even large databases with millions of nodes and edges can fit in modest amounts of memory.

Unquestionably, this argument has been invalidated by the enormous amount of today's data availability.

Another motivation towards employing special indexes is the low performance on the computation of the top-k answers to a keyword query. Until 2007, approaches were employing indexes only for the identification of nodes containing query keywords; finding substructures connecting these nodes relies on graph exploration (see Subsection 2.2.3). For a system supporting a large, ongoing workload of keyword queries, it is natural and critical to exploit indexes that provide graph connectivity information to speed up searches. Lack of this feature can be attributed in part to the difficulty in indexing connectivity for general graphs, because a naive index would have an unacceptably high (quadratic) storage requirement. As a result, such approaches pre-compute various quantities (such as node distances, path lengths, etc.), which are not affected by the query keywords, and are employed in scoring functions. This direction is taken in [HWY07, LOF$^+$08].

Bi-level Index

The approach followed in [HWY07] is the construction of a bi-level index. The index partitions a graph into multiple subgraphs, or alternatively, blocks. The first level of the index is called *top-level block index* (TB-index) and is used for storing mappings between keyword and nodes to blocks. The second level of the index is called *intra-block index* (IB-index) and is used for storing detailed information about a block. The functionality of each kind of index is described in the following.

The TB-index is needed for initiating the exploration algorithm on these blocks of the graph that contain nodes matching with a keyword of the query. Furthermore, it is advised to direct the exploration process across blocks. To do so, it keeps keyword-block lists ($L_{KB}(w)$), which denote the blocks containing nodes matching with keyword w, and portal-block lists ($L_{PB}(p)$), which denote the blocks in which node p is contained and has an outgoing edge to a node contained in another block, i.e., p is an *out-portal* node[5].

The IB-index is exploited to guide the exploration algorithm inside a block. At each step of the exploration process the TB-index is consulted to determine whether it is time for the exploration process to be continued in another block. The IB-index employs the following datastructures:

Intra-block keyword-node list, $L_{KN}(b, w)$. For each block b and keyword w it keeps the list of nodes (sorted by distance) connected to a node matching with w inside block b. This data structure is employed for backward search, as described in Subsection 2.2.3, implementing the equi-distance expansion in each keyword cluster strategy.

Intra-block node-keyword map, $M_{NK}(b, u, w)$. For each block b, node u, and keyword w it keeps the shortest distance of u to the node matching with w inside block b. This data structure is employed for forward search, as described in Subsection 2.2.3, implementing the heuristic evaluation function used in bidirectional search.

Intra-block portal-node lists, $L_{PN}(b, p)$. For each block b and node p it keeps the list of nodes (sorted by distance) connected to a portal node, i.e., a node of block b, which has either an outgoing or incoming edge from another block, different from b. This data structure supports backward expansion search to continue in a different block, because of the fact that an answer might cover more than one

[5]Likewise, an *in-portal* node is one that has an incoming edge from a node contained in another block.

blocks. Indeed, the previous two data structures guarantee the finding of the local shortest distance of two nodes, but this distance is not guaranteed to be the shortest globally, consider all blocks.

Intra-block node-portal distance map, $D_{NP}(b, u)$. For each block b and node u it keeps the shortest distance from u to an out-portal node of b. This data structure determines a lower bound of the shortest distance between a node and a node matching with a keyword, and is used to prune the search space.

2.3 Evaluation Methodologies for keyword search

This section discusses the current status of the evaluation methodologies and techniques of systems that support keyword searching over structured and semi-structured data, and is compared to that of IR. It is pointed out that the current state is far from good, without providing any solid and rigid methodology, as opposed to the respective evaluation process developed in the field of IR [Web10]. Fortunately, this area has already started to attract the interest of the research community leading to initiatives, such as INEX (INitiative for the Evaluation of XML)[6], and interesting and promising evaluation methodologies.

2.3.1 Evaluation in unstructured data

Unstructured data is unquestionably a form of data that has been exhaustively studied and processed in the field of IR. From its early days, the most powerful way to effective retrieval was the hierarchical indexing scheme. However, experiments conducted by the librarian Cyril Cleverdon, at the library of the Cranfield Aeronautical College, in England, in the late 1950s and early 1960s, showed that such indexing techniques were weak in terms of retrieval quality. In this same position was the technique for indexing documents by plain keywords. The experiments resulted in the fact that what mattered was the process of retrieval itself.

The experimental methodology developed in the Cranfield tests has been highly influential. This methodology consists of three components, which are the following:

Corpus This consists of the set of documents to index and retrieve.

Topics These are the request statements, which is part of the information requests that have to be processed against the corpus.

qrels These constitute the assessments of which documents in the corpus are relevant to each request.

[6]`www.inex.otago.ac.nz`

These three components together form a *test collection*. The use of such a test collection in an evaluation process is often termed the “Cranfield methodology” or even the “Cranfield paradigm” [Voo01].

The test collection methodology is ideally suited to automation and computerization. Relevance assessments are made in advance and are reusable, so experiments can be performed automatically and cheaply. The first such computerized retrieval systems were developed in the early 1960s, the most famous being that of the SMART project at Cornell University [Sal81]. Early progress was brisk, driven by the fast turnaround of collection-based evaluation, and the foundations of statistical information retrieval were laid down within a decade, including term weighting, query expansion, and result ranking. Over time, however, the field suffered from a lack of consolidation of results, due, in part, to the small ageing test collections employed. The credibility of experimental findings was undermined, impeding the adoption of research technologies in operational systems.

The second great impetus to empirical IR research came with the institution of TREC[7] (Text REtrieval Conferences) in 1992. TREC produces large-scale, up-to-date test collections, encourages collaborative experiments upon them, and provides a venue for publishing and discussing results. The first collection used at TREC, known as TIPSTER[8], contained around $750,000$ documents. The impact that the TREC effort had upon the effectiveness of retrieval systems was tremendous. Even though SMART had been under active development for three decades, the first five years of TREC saw its retrieval effectiveness almost double, as measured by mean average precision (MAP) [MRS], one common evaluation metric. A number of important innovations were made during these early years of TREC, from new similarity metrics such as Okapi BM25 or just BM25 [RWJ+96, RW94] — now the standard retrieval formula —, to smaller refinements that, nevertheless, had significant impacts, such as document length normalization. However, few of these innovations were revolutionary in their nature; rather, the existence of a large and standard test environment allowed existing ideas to be extended, refined, and tuned. Figure 2.1 depicts how much TREC affected and improved the retrieval effectiveness of documents since its establishment, and how this improvement rate of effectiveness started reaching the limits of improvement of current approaches.

2.3.2 Evaluation in structured and semi-structured data

From the perspective of keyword search on structured and semi-structured data, the earliest works laid the burden of research on performing it in an efficient manner. At this stage, for structured approaches, any and only tuples that contained all of the query keywords were considered correct matches of the query [ACD02, HP02]. Clearly, the evaluation of effectiveness was not considered. As techniques improved, researches became interested in the quality of results also. Proper ranking functions of the rel-

[7]http://trec.nist.gov/
[8]http://www.itl.nist.gov/iaui/894.02/related_projects/tipster/

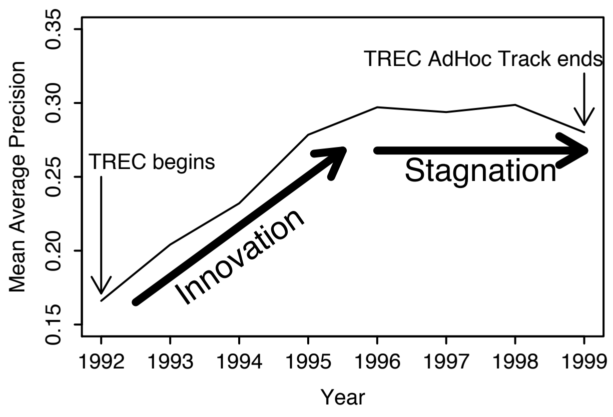

Figure 2.1: Improvement course of TREC's retrieval effectiveness across the first eight TREC collections [BW99] (image taken from [Web10]).

evance of the answers to the query were introduced. Some of these were specific to the nature of structured and semi-structured data, and specifically, to their tree/graph structure [BHN$^+$02, HWY07]. Other ranking functions were adopted from IR, treating individual attributes or whole answers as virtual documents [HGP03, LLZ07]. The most fruitful of all were those which combined both a structural and a full-text ranking component [HGP03, HWY07, LOF$^+$08, TWRC09].

With the development of ranking functions came the need to assess the quality of the results. For this, a number of different test datasets have been employed. The most widely used are the IMDB[9] movie database[10] and the DBLP[11] database of academic publications for use in a database[12], XML[13], or RDF[14] context. Conversely, the wide acceptance of these datasets cannot be found on the query sets. Queries are generally formulated by the authors themselves, rather than taken from a query log, or written by independent third parties. Self-authored queries have a strong potential for bias. Making matters worse, query sets have not been re-used between experiments and experimenters, making comparison of results, if not applicable, extremely difficult, unless the researcher provides that comparison directly through re-implementing or re-using existing approaches as baselines — an important practice that is rarely followed by researchers. Moreover, query sets are often quite small, rarely more than 20 per dataset. Compared to the query sets employed in TREC collections, which are often sampled from the log of a commercial search engine and can reach the number of 50 — a number regarded as insufficient by many researchers [WMZ08] —, the poorness in the quality of

[9]http://www.imdb.com/

[10]http://www.imdb.com/interfaces

[11]http://www.informatik.uni-trier.de/~ley/db/

[12]http://dblp.l3s.de/dblp++.php

[13]http://dblp.uni-trier.de/xml/

[14]http://dblp.l3s.de/dblp++.php

the first queries is unquestionable.

In the same style follows the relevance assessment of the results, which is generally performed by the authors themselves or their colleagues. Like self-authoring, self-assessment is prone to biasing. The suspicion that such bias has occurred is strongest where abnormally high effectiveness scores are achieved. For instance, top-end scores at TREC for mean reciprocal rank (MRR) — an admittedly unstable metric —, are around 0.8, but [LLZ07] achieve the rather astonishing, perfect MRR score of 1. Similarly, the best fully automatic TREC participant systems achieve scores of around 0.25 under the top-100 precision metric, but [LOF$^+$08] reports around 0.9.

Given the disparity of query sets, corpora, and assessment methodologies, it is not straightforward to determine how retrieval effectiveness in keyword search over structured and semi-structured data has progressed. The only and safe method is to follow the chain of baselines. It seems that keyword search over structured data is at roughly the same stage that IR was before the "TREC era". What is needed is a standard method, combined with large-scale, independently curated test collections [Web10].

2.3.2.1 XML Evaluation

Fortunately, from the perspective of semi-structured data, there is an increasing mobility towards this direction. For the evaluation of XML the INEX initiative has been established, which aims at providing an infrastructure for evaluating the effectiveness of content-oriented XML retrieval. The main goal of INEX is to promote the evaluation of *focused retrieval*[15] by providing large test collections of semi-structured documents, uniform evaluation measures, and a forum for organizations to compare their results. The objective of the evaluation in INEX, based on the ad-hoc task, is to assess a system's retrieval effectiveness, where effectiveness is measured as a system's ability to satisfy both content and structural aspects of a user's information need and retrieve the most specific relevant document components, which are exhaustive to the topic of request and match its structural constraints [GK02].

In this direction, and in line with the methodology employed in TREC, in 2002 INEX licensed a collection of IEEE articles for use in XML element retrieval experiments. In 2005 this collection was expanded with more IEEE articles. In 2006 the IEEE collection was complemented with an XML dump of the Wikipedia, which was itself updated in 2009. The Lonely Planet Guide[16] has also been used, and since 2007 a collection of scanned books (licensed from Microsoft) has also been made available for book retrieval experiments.

The methodoly that INEX follows for test collection construction is the following: each year INEX subscribers (participants) provide sample queries (called topics) they believe are suitable for experimental purposes. These are collected, verified, and de-duplicated

[15]The process of focused retrieval is oriented towards satisfying information need based not only on the document level, but also on the component level that comprises a document.

[16]`http://www.lonelyplanet.com/`

by INEX before being distributed back to the participants as a new set of topics. Participants then run the topics through their search engines[17] and submit back to INEX their results. It is important to be noted here, that the results are not only document relevances, but also component (i.e., XML element) relevances of these documents. From the submitted result sets, using a technique known as pooling, a set of documents are chosen for evaluation. These documents are then distributed back to the original authors of the topics to make judgments as to which document/elements are relevant and which are not for each topic, providing also a ranked list of documents/components. In this way the relevance of a document to a query is not known before the participant submits their runs, and no one person is responsible for creating the set of topics or making the decision as to which documents are relevant to which topics.

The assessements of document/components are based on the following two dimensions:

Topical relevance , which reflects the extent to which the information contained in a document component satisfies the information need.

Component coverage , which reflects the extent to which a document component is focused on the information need, while being an informative unit.

Both these dimensions are measured using graded scales. For document relevance the following four-point scale is used [KJ02]:

Irrelevant (0): The document component does not contain any information about the topic of request.

Marginally relevant (1): The document component mentions the topic of request, but only in passing.

Fairly relevant (2): The document component contains more information than the topic description, but this information is not exhaustive. In the case of multifaceted topics, only some of the sub-themes or viewpoints are discussed.

Highly relevant (3): The document component discusses the topic of request exhaustively. In the case of multifaceted topics, all or most sub-themes or viewpoints are discussed.

Component coverage is selected from the following four categories [Sch98]:

No coverage (N) : The topic or an aspect of the topic is not a theme of the document component.

Too large (L): The topic or an aspect of the topic is only a minor theme of the document component.

[17]This is in contrast to TREC, in which the participants are provided with a reference retrieval system to assess the relevance of a document to a topic.

Too small (S): The topic or an aspect of the topic is the main or only theme of the document component, but the component is too small to act as a meaningful unit of information.

Exact coverage (E): The topic or an aspect of the topic is the main or only theme of the document component, and the component acts as a meaningful unit of information.

Due to the nature of XML retrieval, metrics from traditional evaluation initiatives like TREC could not be applied in INEX without modification. Therefore, it was necessary for INEX to develop new evaluation procedures, such as implicit relevance assessements, quantisation of relevance and coverage, and recall/precision metrics. More details can be found in [GK02].

2.3.2.2 Semantic Web Evaluation

From the perspective of the RDF framework and Semantic Web technologies in general, there are a few experimental works proposing interesting evaluation methodologies, the most notable of which are [PAAG$^+$10, PAAG$^+$, FLS$^+$09]. Except for these, the SemSearch Workshop[18] of the WWW conference, established in 2009, focuses (apart from others) around semantic search in general, and specifically in proposing evaluation methodologies, and constructing standard datasets and benchmarks for semantic search.

INEX+DBpedia The authors of [PAAG$^+$10, PAAG$^+$], first, attempt to built a new corpus from the intersection of the DBpedia[19] knowledge base and the INEX-Wikipedia[20] collection, which results in a set of $2,233,718$ documents. With the new corpus, the INEX 2009 topics and assessments are adapted to this new corpus, producing 68 topics and a properly modified assessments file. Such an approach is justifiable, taking in mind the common structure of XML and RDF. Second, they establish metrics to evaluate retrieval performance exploiting the TREC-eval software[21], which implements state-of-the-art IR metrics used for search engine performance evaluation [MRS]. These metrics are generally oriented around two main directions: the ability to retrieve relevant documents and the ability to sort them properly. Some of the used metrics are the following:

Mean Average Precision (MAP), which is the average of the precision values measured at different recall levels,

[18]SemSearch 2009-2010: `http://km.aifb.kit.edu/ws/semsearch09`, `http://km.aifb.kit.edu/ws/semsearch10/`

[19]`http://DBPedia.org/About`

[20]A collection of XML documents obtained from Wikipedia, which are offered through INEX, and can be adapted to meets the needs of Semantic Web technologies, such as RDF.

[21]`http://trec.nist.gov/trec_eval`

Geometric Mean Average Precision (GMAP), which is a variant of MAP that uses a geometric mean,

Precision after X documents (P@X), which measures the precision after X documents have been retrieved, and

R-Precision , which measures precision after R documents have been retrieved, where R is the total number of relevant documents for a query.

The proposed evaluation framework was used to evaluate two ranking functions, BM25, and Lucene[22]. The results showed that BM25 outperforms Lucene, especially when dealing with structured documents, e.g., XML documents. It is worth noting that the evaluation setting was similar in style to an evaluation based on the TREC standards. Specifically, the set of queries was taken from the INEX 2009 contest, which provides judgments for 68 queries. Each query consists of three different versions of the same topic: *title*, *description*, and *narrative*.

TREC in Ontology-based setting The authors of [FLS$^+$09] propose a benchmark based on an adaptation of the Cranfield paradigm (see Subsection 2.3.1) to evaluate ontology-based search approaches and compare them against baseline IR models. According to the standards of a test collection, the proposed benchmark comprises the following components:

1. a text document collection of size 10 GB,
2. a set of 20 queries,
3. the document relevance judgments for these queries,
4. a set of ontologies and knowledge bases covering the query topics, and
5. a set of annotations for the ontologies and knowledge bases.

The text document collection as well as the set of queries and the document relevance judgments are obtained from one of the most widely used datasets in the IR community, the TREC WEB track, and specifically, the TREC 9 and TREC 2001 WEB track. Concerning the ontologies, the authors faced the problem of the sparsity and incompleteness of semantic data available on the Web. Due to this problem, a number of 40 publicly available ontologies were chosen, together with another 100 repositories of semantic data, which covered a subset of the test collection and queries.

As far as knowledge bases are concerned, the problem of sparsity and availability is much more overt. Current publicly available ontologies contain significant structural information in the form of classes and relations, but conversely are barely populated

[22]`http://lucene.apache.org/`

or not at all. As a result, the available KBs are still not enough to perform significant large-scale experiments. To overcome this limitation, some of the selected ontologies have been semi-automatically populated using Wikipedia as an information source.

Last, from the side of annotations, the goal is to generate weighted annotations about the documents of the collection for each semantic entity of each ontology and knowledge base. The rationale behind this approach lies in the attempt of the authors towards bridging the semantic gap between a document and an ontology. Provided that all ontology entities are associated to one or more documents via these annotations, the answers of a keyword-based search algorithm can be mapped to the documents of the collection, and then evaluated using the standard IR methodology.

The authors applied the proposed benchmark on a use case example to compare a real ontology-based keyword search system [FLS$^+$08] against the following systems:

Keyword search (KS): a conventional keyword-based retrieval approach, using the Jakarta Lucene library[23].

Best TREC automatic search (BTA): the approach used by the best TREC search engine that uses as query just the title section.

Best TREC manual search (BTM): the approach used by the best TREC search engine, which manually generates the queries using information from the title, description, and the narrative.

The evaluation conducted using two metrics, MAP and P@10. The results show that the ontology-based approach outperforms KS, mainly due to the annotation generation, which results in entities been annotated with documents not containing keywords describing that entity. As far as BTA is concerned, the results are comparable. Regarding BTM, it outperforms all systems. To justify this last result, the authors argue that there are two reasons to blame: first, the annotation process, the quality of which affects the quality of the answers, and second, the fact that many documents in the answer of the the ontology-based approach were not judged in the TREC collection, and thus considered as irrelevant, whereas close inspection by the authors showed that a significant portion, 31.5%, were in fact relevant.

2.4 Other Directions to Keyword Search

The research interest around keyword search does not focus on query processing only. Besides, a lot of interest lays on other aspects of keyword search that have to do with the improvement of user interaction with systems providing keyword search interfaces, as well as improving the quality of search results. In this respect, the proposed works aim at providing more intuitive interfaces and visualizations for exploring both the results of keyword searching and the content that the underlying data capture, as

[23]`http://lucene.apache.org`

well as techniques for bridging the semantic gap between the keywords of the query and the content of the underlying data. The most notable approaches involve browsing, faceted search, result snippets, result clustering, and query cleaning. Although all of them have been successfully used in text search, they pose new challenges in the context of keyword searching on structured and semi-structured data. In the following, a brief introduction is given for each one.

Browsing. Approaches dealing with browsing provide zero-effort web publishing of the underlying data, which would otherwise require a lot of effort or remain dormant. Furthermore, they provide rich interfaces for exploring search results in the form of hierarchical, graphical or faceted views, which is very helpful in identifying the in-between connections of the underlying data and refining the initial keyword query [ACD$^+$03, HBN$^+$01, BHN$^+$02, WLP$^+$09, TMH10, BW07].

Result Snippets. To compensate the inaccuracy of ranking functions, result snippets should be generated [HLC08b, HLC08a]. The principle of result snippets is orthogonal to that of ranking functions: let users quickly judge the relevance of query results by providing a brief quotable passage of each query result, so that users can choose and explore relevant ones among many results.

Result Clustering. In face of query ambiguity, instead of displaying a mixture of query results of different semantics, it is more desirable to cluster query results based on their similarity, so that the user can quickly browse all possible interpretations of query semantics and choose the sets of results that are relevant [HKPS06, KZGM09, WPZ$^+$06].

Query Cleaning. Query cleaning involves semantic linkage and spelling corrections of database-relevant query keywords, followed by segmentation of nearby query keywords so that each segment corresponds to a high quality data term. Compared to query cleaning on textual documents, query cleaning for structured data brings great potentials with new challenges [PY08].

2.5 Conclusions

In this chapter related work in the field of keyword search over structured and semi-structured data was discussed. To assist presentation of related work, a number of dimensions were introduced, such as the data model, structure of the answer, exploration algorithms, ranking/scoring of answers, and indexing. Related work, then, was categorized according to these dimensions.

The next contribution of this chapter was the presentation of the current status of evaluation methodologies and techniques of systems that support keyword searching over structured and semi-structured data. It was pointed out that the current state is far from good, without providing any solid and rigid methodology, as opposed to the respective evaluation process developed in the field of IR.

Last, the chapter discussed other directions taken in keyword searching, which diverge from the path of query processing aiming at improving user interaction and quality of results.

Chapter 3

Our approach to keyword-based search

This chapter presents our approach to keyword-based search. The approach is based on and extends the approach presented in [TWRC09]. In [TWRC09], the authors present an approach for keyword search on graph-structured data, and particularly RDF. Concepts from the field of Information Retrieval are employed to support an imprecise matching that incorporates syntactic and semantic similarities between a user keyword and the content of the queried RDF data. As a result, the user does not need to know the labels of the data elements when doing keyword search. From the perspective of query answering, the user keywords are interpreted as elements of structured queries letting the user select, in an additional step, one of the top-k computed queries to retrieve all its answers. The authors of [TWRC09] have devised a new algorithm for subgraph exploration guaranteeing that the computed results have the k best scores. Last, a strategy for graph summarization is employed that can substantially reduce the search space. In effect, the exploration of subgraphs does not operate on the entire data graph, but a summary one containing only the elements that are necessary to compute the queries.

The rest of the chapter is organized as follows. Section 3.1 discusses the contributions of this dissertation to keyword-based search. Next, in Section 3.2, the data model and query language employed in the query processor are presented. Section 3.3 presents the algorithm of the keyword-based search. Last, Section 3.4 concludes the chapter.

3.1 Contributions to keyword-based Search

This dissertation adopts and improves several of the techniques presented in [TWRC09]. The contributions of the work of this dissertation are summarized in the following:

- A key point that differentiates our approach is that the answer to a keyword query is a sorted list of entities that are relevant to the conjunction of the keywords of the query. From the perspective of an information search system, these entities try to capture the meaning of the keywords and provide relevant information with a view to satisfying the information needs of the user that posed such a query. Conversely, the authors of [TWRC09] are interested in presenting structured queries to the user, which correspond to descriptions of the answers. Thus, the keyword search process contains an additional, interactive, step, namely the presentation of these queries. We feel that such an interactive step is not efficient and useful, because, first, it adds an extra interaction between the user and the system, and, second, it does not clearly separate the answer's space. It is very common, that different structured queries are very similar, each one providing a highly overlapping set of answers at such a level that the user feels they should have been merged.

- A significant extension is the addition of a temporal dimension to a keyword query. To this end, the keyword-based query language is extended with temporal constructs (i.e., "`before` 15/05/1985") (see Subsection 3.2.2), which define temporal constraints on the given keywords. Further, the graph exploration algorithm has been extended to operate on such temporal constraints.
- The graph exploration algorithm has been improved appropriately (see Subsection 3.3.3) resulting in better time performance and better quality in the results.
- Finally, in contrast to [TWRC09], the top-k interpretations of the query keywords are taken into account sacrificing performance against better quality of results and satisfaction of the user information needs.

3.2 Data Model and Query Language

The data model of the query processor that does keyword search is built on the data models of RDF(S) and temporal RDF (see subsection 2.2.1 for more details). The query language is keyword-based, i.e., a query is just a set of keywords, and has been extended with temporal constraints. In the following the data model and the query language are presented in detail.

3.2.1 Data Model

The data model adopts the temporal model and representation of [GHV05, GHV07] concerning the time during which a statement (i.e., a triple) is valid, but also extends it appropriately in order to associate also a class or an individual with the time during which it is valid, or in other words, define its lifetime. This is done using the semantics of RDF(S) and particularly the fact that every resource is an instance of class `rdfs:Resource`. Hence, the lifetime of a class `c` is a temporal triple of the form `(c, rdf:subClassOf, rdfs:Resource)[t]` and in the case of an individual `i`, it is a temporal triple of the form `(i, rdf:type, rdfs:Resource)[t]`.

Definition 1 *The lifetime of a resource is given by the mapping $\lambda : U \to I$ which maps a class c and an individual i to $t \in I$ if `(c, rdfs:subClassOf, rdfs:Resource)[t]` or `(i, rdf:type, rdfs:Resource)[t]` exists respectively.*

In contrast to works [GHV05, GHV07], which deal with definite temporal information, we deal also with indefinite temporal information. Indefinite information concerning a time point can be given as an interval in which the point must lie. In the case of indefinite information about time intervals, the starting and ending time points of an interval are defined in the same way. This way, an interval expression is enough both for time points and intervals (a time point is an time interval whose start and end points are the same).

Definition 2 *A time interval is a quadruple $(s_1, s_2, e_1, e_2) \in I$, where $I \subseteq N \times N \times N \times N$ is the set of time intervals and N is the set of natural numbers.*

Likewise, a temporal RDF triple is a temporal RDF triple as defined in [GHV05, GHV07], that is, $(s, p, o)[t]$, where $t \in I$. An example of such an interval is this: $(19850501, 19850520, 20100301, 20100304)$, which may be used to denote that a statement was valid during the period that started sometime between days 01/05/1985 and 20/05/1985 and ended some time during days 01/03/2010 and 04/03/2010. For readability purposes, such an interval can be given in the form of $[19850501 - 19850520, 20100301 - 20100304]$. Using this second form, an interval having a definite start time point can be given as $[19850501, 20100301 - 20100304]$ and an interval having both a definite start and end time point can be given simply as $[198505015, 20100302]$.

In terms of data representation, a time interval is represented with four natural numbers. These numbers encode the usual data notation year-month-day as done in ISO 8601 encoding[1]. The first two numbers denote the interval of the starting point and the last two numbers denote the interval of the ending point. If all numbers are equal, then the interval is a definite time point. If the first two numbers are equal, then the interval has a definite starting time. The same applies to the ending time point. In effect, time intervals may be definite or have indefinite start or end points for which a time interval estimate is known, in which case the information is indefinite.

3.2.2 Query Language

The query language is keyword-based, i.e., a query is just a set of keywords. To query data with temporal information the keyword query language is extended with temporal constraints. Temporal constraints define a temporal relation between two intervals. Temporal relations may be any of the thirteen temporal relations defined by the work of Allen [All81], such as "a `before` b", "a `meets` b", "a `overlaps` b", "a `starts` b", "a `finishes` b", "a `during` b", and "a `cotemporal` b". The other six can be derived swapping the intervals of the first six relations. All these temporal relations have been implemented as temporal operators on time intervals and extended to take indefinite information into account. In effect, having indefinite time intervals introduces uncertainty in the answer of whether a time interval is related to another in any of these thirteen relations. For example, consider a definite time interval a, $[1985/05/15, 1985/05/29]$ and an indefinite one b, $[1984/05/23, 1985/05/10 - 1985/05/25]$.

[1]`http://en.wikipedia.org/wiki/ISO_8601`

Then, it is uncertain whether "b `meets` a", or "b `overlaps` a", or "b `before` a", but it is certain that "a `starts` b", "a `finishes` b", "a `cotemporal` b", etc. do not hold. This kind of uncertainty is analogous to the one captured by the uncertainty operator of the work in[Kou94].

Formally, a query is of the following form:

$$KL \quad \phi$$

The expression KL is a list of keywords and ϕ is a finite conjunction of formulas of the form $R \quad c$, where R is one of the thirteen Allen relations and $c \in I$, i.e., it is a time interval. Given that a temporal RDF graph in the proposed data model can contain indefinite temporal information, the usual issues known from work on indefinite information in the relational model arise [Gra91]. Thus, a query can have possible answers, certain answers, or answers under conditions. Querying temporal data is orthogonal to the approach taken for keyword querying, in the sense that a temporal constraint, given together with a user keyword, implies the execution of the keyword search algorithm applying the temporal constraints in every stage of the exploration process (see Subsection 3.3.3). This means that during exploration, an element of the graph model is not explored if does not satisfy the given temporal constraints.

3.3 The keyword-based Search Algorithm

In this section, the query processing algorithm that does keyword search on top of RDF data is presented. Before proceeding to its presentation, the employed data structures are discussed.

3.3.1 Employed Data Structures

The keyword search algorithm can operate on graph data models representing the underlying RDF data. Currently, three different graph data structures can be used, each one providing a different view on the underlying data and affecting the space and time complexity of the algorithm in a different way. Using the terminology of [TWRC09], these graph data structures are the *data graph*, *summary graph*, and *augmented graph*. The data graph is an identical view of the underlying RDF graph. The summary graph summarizes the data graph in the sense that it contains structural (schema) elements only, such as classes and properties between classes. The augmented graph is a super graph of the summary graph containing also specific elements of the data graph; those that are not present in the summary graph and match with a keyword in the query, that is, those that are present in the instance level. In the following, formal definitions of the data, summary, and augmented graphs are given.

Definition 3 *A data graph G is a tuple (V, L, E), where the following apply:*

Table 3.1: A set of RDF triples

Subject	Predicate	Object	Subject	Predicate	Object
pro_1	$type$	$Project$	res_2	$name$	$Yannis\ Ioannidis$
pro_2	$type$	$Project$	res_1	$name$	$Manolis\ Koubarakis$
pro_1	$name$	$Papyrus$	res_1	$worksAt$	$univ_1$
pub_1	$type$	$Publication$	$univ_1$	$type$	$University$
pub_1	$author$	res_1	$univ_2$	$type$	$University$
pub_1	$author$	res_2	$University$	$subclass$	$Agent$
pub_1	$year$	2010	$Researcher$	$subclass$	$Person$
pub_2	$type$	$Publication$	$Person$	$subclass$	$Agent$
res_1	$type$	$Researcher$	$Agent$	$subclass$	$Resource$
res_2	$type$	$Researcher$	res_3	$type$	$Researcher$
$univ_1$	$name$	$DI\&T$	pub_1	$hasProject$	pro_1

- *The set V is a finite set of vertices, which is the disjoint union $V_E \uplus V_C \uplus V_V$. The set V_E is called E-vertices and represents the set of RDF entities, the set V_C is called C-vertices and represents the set of RDF classes, while the set V_V is called V-vertices and represents the set of RDF literals.*

- *The set L is a finite set of edge labels. subdivided by $L = L_R \uplus L_A \uplus \{type, subclass\}$, where L_R represents the labels of edges between two entities and L_A represents the labels of edges between an entity and a literal. The labels $type$ and $subclass$ have the same meaning as in the RDF framework.*

- *The set E is a finite set of edges of the form $e(v_1, v_2)$ with $v_1, v_2 \in V$ and $e \in L$. Moreover, the following restrictions apply:*
  - *$e \in L_R$ if and only if $v_1, v_2 \in V_E$,*
  - *$e \in L_A$ if and only if $v_1 \in V_E$ and $v_2 \in V_V$,*
  - *$e = type$ if and only if $v_1 \in V_E$ and $v_2 \in V_C$, and*
  - *$e = subclass$ if and only if $v_1, v_2 \in V_C$.*

For example, the data graph representing the triples of Table 3.1 is shown in Figure 3.1. Note that nodes representing entities are depicted in a circles, classes are depicted in ellipses, while typed literals are depicted in squared circles.

Definition 4 *A summary graph G^s of a data graph $G = (V, L, E)$ is a tuple (V', L', E') with vertices $V' = V_C \cup \{Resource\}$, edge labels $L' = L_R \uplus \{subclass\}$, and edges E' of type $e(v_1, v_2)$ with $v_1, v_2 \in V'$ and $e \in L'$. In particular, every vertex $v' \in V_C$ represents an aggregation of all the vertices $v \in V$ having the type v', i.e., $[\![v']\!] := \{v | type(v, v') \in E\}$ and $Resource$ represents the aggregation of all the vertices $v \in V$ with no given type, i.e., $[\![Resource]\!] = \{v | \neg\exists c \in V_C \text{ with } type(v, c) \in E\}$. Accordingly, we have $e(v_1', v_2') \in E'$ if and only if there is an edge $e(v_1, v_2) \in E$ for some $v_1 \in [\![v_1']\!]$ and $v_2 \in [\![v_2']\!]$.*

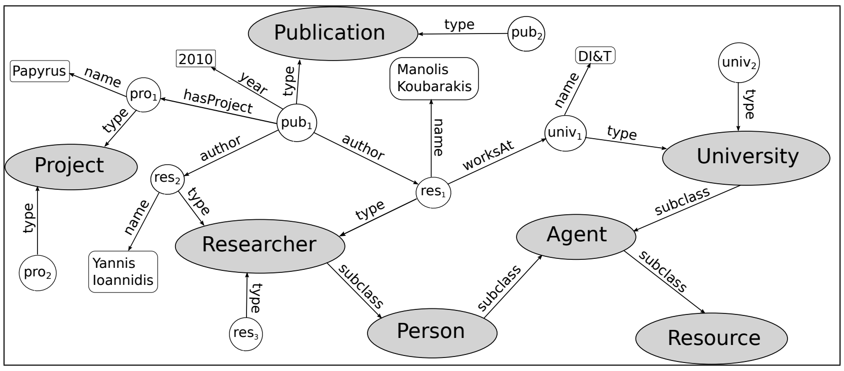

Figure 3.1: The data graph of the RDF triples of Table 3.1

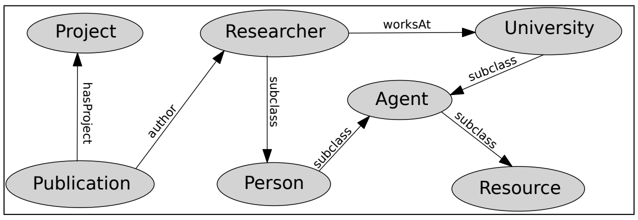

Figure 3.2: The summary graph of the data graph of Figure 3.1

For example, the summary graph corresponding to the data graph of Figure 3.1 is shown in Figure 3.2.

Definition 5 *Given a set K of keywords, the augmented graph G^s_K of a data graph G consists of G's summary graph G^s additionally containing the following:*

- $e(v', v_k)$ *for any keyword matching element* v_k*, where* G *contains* $e(v, v_k)$ *and* $type(v, v')$*, and*
- $e_k(v', value)$ *for any keyword matching element* e_k*, where* G *contains* $e_k(v, \tilde{v})$ *and* $type(v, v')$*, and* $\tilde{v}$ *is not a keyword matching element. Thereby,* $value$ *is an new artificial node.*

For example, the augmented graph corresponding to the data graph of Figure 3.1 and the keyword query `2010 koubarakis publications` is shown in Figure 3.3. Note that the keyword elements that have been matched with the user keywords are highlighted in orange. These keyword elements are the starting points of the exploration process described in subsection 3.3.3.

The keyword search algorithm conducts an exploration algorithm only on the augmented graph upon submission of a user query, and uses the summary graph to succinctly represent the underlying data. In effect, a user query corresponds to an augmented graph, that is, the summary graph augmented with data from the data graph.

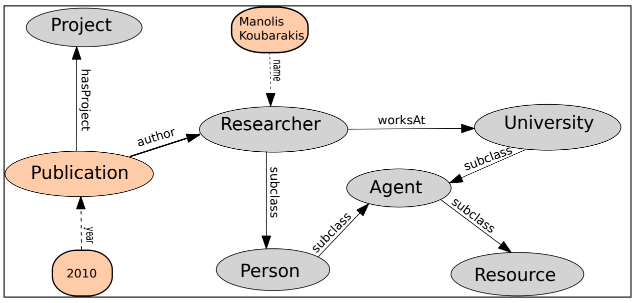

Figure 3.3: The augmented graph of the data graph of Figure 3.1 and keyword query `2010 koubarakis publications`

Clearly, the use of summary and augmented graphs plays a significant role on the data and time performance of the algorithm. In the case of data performance, the algorithm uses only the structural information from the data graph, which is embedded in the summary and augmented graph making the keyword search algorithm capable of working mostly in memory.

At this point, it has to be noted that it is absolutely realistic to assume that the summary and augmented graph structures can fit in memory. For example, as of the November 2009, the DBpedia dataset describes 2.9 million "things" [sic] (i.e., resources) with 479 million "facts" [sic] (i.e., triples). From these resources, 205 are classes which are inter-connected with 1200 properties, 1.170.000 are individuals, and the rest 479 million resources identify links to external images, web pages, and other datasets. Taking into account that DBpedia is a snapshot of Wikipedia, which is rapidly evolving, mostly in terms of data entries and not in terms of new knowledge that affects its schema, it is realistic to assume that the aforementioned classes and properties, which comprise the summary graph, can fit in main memory, and that do not incur any significant space overhead to the algorithm.

In the case of time performance, the worst case scenario for the algorithm is to explore the whole augmented graph. While this does not incur any significant overhead, it is avoided because it leads to poor quality answers. So, again the time performance is of no concern. The only case in which there is a time performance issue, is when a keyword may have a great number of interpretations, that is, when it could correspond to different elements of the data graph. In such cases, which are the norm for such search applications, the time overhead is significant, because keyword interpretation becomes a combinatorial problem, since we have to take into account all possible combinations of the interpretations of the user keywords.

Details concerning the construction of these graph data structures are given in Chapter 4, where the components of the query processor are presented in detail.

3.3.2 Scoring of graphs

The computation process can result in many queries all corresponding to possible interpretations of the query keywords. In the following, several scoring functions are introduced that aim to assess the relevance of the computed queries. It is worth noting that while this subsection talks about scoring, the scoring functions employed are in fact cost functions, i.e., they measure the badness (high cost, low score) and not the goodness (low cost, high score) of the answers. This should not confuse the reader.

Another point worth noting, is that queries are seen as graphs. These graphs are constructed from a set of paths P. The score of such a graph is defined as a monotonic aggregation of its path costs.

$$C_G = \sum_{p_i \in P} C_{p_i}$$

In general, the cost of a path is computed from the cost of its elements.

$$C_{p_i} = \sum_{n \in p_i} c(n)$$

Path Length

The path length is commonly used as a basic metric for ranking answer trees/graphs in recent approaches to keyword queries. This is based on the assumption that the information need of the user can be modelled in terms of entities, which are closely related. Thus, a shorter path between two entities should be preferred. In effect, compact graphs or alternatively graphs with low diameter are preferred. For computing path length, the general cost function for paths given above can be casted as $C_{p_i} = \sum_{n \in p_i} 1$, i.e., the cost of an element in any path is simply one, $c_p(n) = 1$. Accordingly, the cost of a graph can be computed via

$$C_{G_{\text{path}}} = \sum_{p_i \in P} \sum_{n \in p_i} c_p(n)$$

Another, more strict way of defining the cost of an element knowing a-priori the diameter of the graph $diam$, is to use $c_p(n) = 1/diam$. In this respect, the cost of a path with length equals to the diameter of the graph, would be 1, i.e., the maximum cost.

Keyword Matching

The keyword matching cost function assigns to every element of a graph a cost in the interval $[0, 1]$ according to whether it is a keyword element or not. Suppose that $sim(\cdot)$ is a score function determining the textual similarity of a query keyword to that of a keyword present in the index. Then, the cost for a graph element n is given by the following expression:

$$c_{km}(n) = \begin{cases} \epsilon > 0 & \text{, if } 1 - sim(n) = 0 \\ 1 - sim(n) & \text{, otherwise} \end{cases}$$

That is, keyword elements have lower cost, than those that were not matched against a keyword of the query, which have a cost of one[2]. Accordingly, the cost of a graph can be computed via

$$C_{G_{\text{km}}} = \sum_{p_i \in P} \sum_{n \in p_i} (1 - sim(n))$$

Popularity

The purpose of the popularity function is to measure how popular is a graph taking into account the popularity of its elements. For class nodes, popularity is determined by the number of entities that have as type that class node. For entity nodes, popularity is determined by the number of incoming edges (triples in which they are objects). For edges, popularity is determined by the number of pair of entities connected with that edge. The higher the popularity of an element, the lower should its contribution be to the cost of a path. Accordingly, if $|v_{agg}|$ is the number of E-vertices that have been clustered to a C-vertex v, $|e_{agg}|$ is the number of R-edges that have been clustered to a corresponding R-edge e of the summary graph, $|V|$ and $|E|$ is the number of nodes and edges, respectively, in the data graph, and $|v_{inc}|$ is the number of incoming edges for node v, then the cost for a graph element can be computed by the following expression:

$$c_{pop}(n) = \begin{cases} 1 - \frac{|n_{agg}|}{|V|} & \text{, if } n \in V_C \\ 1 - \frac{|n_{inc}|}{|V|} & \text{, if } n \in V_E \\ 1 - \frac{|n_{agg}|}{|E|} & \text{, if } n \in L_R \end{cases}$$

As before, the cost of a graph can be computed via

$$C_{G_{\text{pop}}} = \sum_{p_i \in P} \sum_{n \in p_i} c_{pop}(n)$$

Combine

The combine function aims at combining the former defined functions, to derive a score function taking in mind the succinctness of the answer (path length), the level of matching of the query keywords with the answer (keyword matching), and also the popularity of each element in the answer relative to the whole graph (popularity). The combination of these functions can be computed via

$$C_{G_{\text{comb}}} = \sum_{p_i \in P} \sum_{n \in p_i} (c_p(n) * c_{kw}(n) * c_{pop}(n))$$

[2]It is required that c_{km} be positive so as to be combined effectively with other score functions.

It is worth mentioning that while the path length and the popularity scores can be computed off-line, the matching scores are specific to the query and are thus computed and associated with elements of the summary graph only during query computation.

3.3.3 Overview of the algorithm

The process of querying RDF graph data using keywords consists of the following phases:

Keyword interpretation. During this phase each keyword of the query is interpreted as an element of the RDF data (i.e., class, property, or individual) and, subsequently, as an element of the data graph. To this end, a keyword index is employed, which indexes all string literals. This way, imprecise syntactic matching of keywords to RDF entities is supported. Next, these RDF entities are mapped to entities of the data and summary graph constructing the augmented graph upon which the subsequent phases operate.

Graph Exploration. This phase explores the augmented graph. Specifically, the interpreted keywords from the previous stage serve as starting points of the graph exploration. The aim is to find the k best subgraphs that connect the keywords with each other and satisfy the temporal constraints (i.e., all entities of the subgraphs have a lifetime that satisfy the temporal relations in the query).

Query Mapping. During this phase, the subgraphs of the previous phase are mapped to SPARQL queries, which are, then, evaluated against the RDF store of the RDF data deriving new entities.

Entity Transformation. The last phase constructs the answer of the keyword query in the form of a sorted list of RDF entities. For this, each subgraph is traversed to extract its entities. All properties (i.e., edges) and literals (i.e., value nodes) are discarded. Thereby, the derived entities are only classes or entities.

In the following, the aforementioned phases are described in more detail.

Keyword Interpretation

This phase of the algorithm interprets the user keywords to elements of the RDF data, which can be either a class, a property, or an individual. To do that, a keyword index is constructed which indexes all string literals of the RDF data. For each keyword, first, the index of RDF literals is examined taking the matching elements. These matching elements comprise different interpretations of a single user keyword in the context of the underlying data. Each such interpretation, i.e., each element of the data graph, is mapped to an element of the summary graph. Because of the fact that the summary graph contains only structural information, some interpretations will not be mapped to any elements of the summary graph. Those elements are added to the summary

Table 3.2: An example of the interpretations for two keywords.

kw_1	**score**	kw_2	**score**
a_1	0.9	b_1	0.9
a_2	0.8	b_2	0.85
a_3	0.72	b_3	0.7
a_4	0.6	b_4	0.2
a_5	0.5	b_5	0.15
a_6	0.4	b_6	0.1
a_7	0.2	b_7	0.05

graph, comprising the augmented graph. After this stage, all these elements are named *keyword elements*. This interpretation process is repeated for each user keyword and, at a final step, the top-k' possible combinations of interpretations are calculated. It is mentioned that the augmented graph is constructed using a single interpretation from every keyword in the user query. In other words, every combination of keyword interpretations is mapped to a different augmented graph.

The calculation of all possible combinations of keyword interpretations is an issue that is not tackled in keyword search works in general due to its high time complexity. Most of the works simply ignore this problem. Others, opt for taking it into account in the design of the keyword search algorithms, but ignore it in the respective implementation (see for example [TWRC09]). We believe that this limitation is very restrictive and its support is of great importance due to the fact that the user might have no knowledge about the domain of the underlying data. Furthermore, using techniques from information retrieval such as stemming, lemmatization, and imprecise syntactic matching, it is evident that the best result of the interpretation process may not satisfy the user needs. In contrast to the work in [TWRC09], we calculate the top-k' possible combinations of keyword interpretations and for each such combination a graph exploration is initiated on the respective augmented graph. It is worth noting, that the design of the keyword search algorithm in [TWRC09] enforces the addition of all keyword interpretations in the summary graph. We strongly believe that such an approach is irrational taking into account the design of the algorithm: the exploration process of the resultant augmented graph reports a subgraph, i.e., a candidate answer, only if the subgraph is composed of paths the origin of which is a keyword element. That is, such an answer would embody all keyword interpretations, which is rather impossible.

The calculation of the top-k' possible combinations of keyword interpretations is done using Algorithm 1. Formally, for n keywords, and n sorted lists in descending score order (interpretations per keyword), the problem amounts to computing the top-k' n-ary tuples, that is, the n tuples with the highest sum of scores of items from each sorted list. As an example, consider two keywords, kw_1 and kw_2, and their respective interpretations with scores as shown in Table 3.2. Then, the top-2, top-3, top-4, top-5, and top-

6 combinations are the following: (a_1, b_2) with $score = 1.75$, (a_2, b_1) with $score = 1.7$, (a_2, b_2) with $score = 1.65$, (a_3, b_1) with $score = 1.62$, and (a_1, b_3) with $score = 1.6$. Note, also, that in every combination it must be the case that one and only one item is present from each sorted list.

```
Algorithm 1: Compute top-$k'$ Combinations
  Input: $n$ sorted lists in descending order, $k'$: the number of top-$k'$ combinations
  Output: $Topk$: the top-$k'$ combinations (vectors)
  // initialize heap to the best combination
1 $Heap.add((1, 1, \ldots, 1))$;
2 while $Heap.size() > 0$ and $Topk.size() < k'$ do
3     $v \leftarrow Heap.pop()$;
4     $Topk.add(v)$;
5     foreach valid descendant $u$ of $v$ do
6         $Heap.add(u)$;
7 return $Topk$;
```

The algorithm employs a MAX-Heap and is initialized so that it contains the first, best n-ary tuple from the first items of the n sorted lists. We will refer to a tuple composed of the i_1-th, i_2-th, . . . , i_n-th item of the first, second, . . . , n-th sorted list, respectively, with the vector $(i_1, i_2, \ldots, i_n)$. Clearly, the best tuple is $v = (1, 1, \ldots, 1)$, with n co-ordinates and $score = \sum_{i=1}^{n} v(i)$. At each step, the algorithm pops the root of the heap, which is appended to the top-k' combinations, and adds its descendants into the heap (lines 2–6). A descendant of a vector $(i_1, i_2, \ldots, i_n)$ is a vector which has one and only one of its co-ordinates increased by 1. A descendant is valid only if all co-ordinates are less or equal than the respective lengths of the n lists. The process terminates when k' vectors have been computed or the heap has become empty.

Graph Exploration

The second phase of the algorithm is the exploration of the augmented graph for finding the top-k subgraphs, and is shown in Algorithms 2 and 3. The input to the algorithm comprises the summary graph, G_s and the keyword elements, $K = (k_1, \ldots, k_n)$, which correspond to one interpretation from the set of the top-k'. Further, k denotes the number of the subgraphs to be computed. The maximum distance d_{max} is provided to constrain the exploration to neighbors that are within a given diameter. In order to keep track of the visited paths during exploration the concept of *cursor* is employed. A cursor is represented as $c(n, k, p, d, w)$, where n is the graph element just visited, k is a keyword element representing the origin of the path captured by c, and p is the parent cursor of c. Besides, the cost w and the distance d are stored for the path. In order to keep track of information related to a graph element n and the different paths discovered for n during the exploration, a data structure of the form $(w, (C_1, \ldots, C_n))$ is employed, where w is the cost of n as discussed in Subsection 3.3.2 and C_i is a sorted list of cursors representing paths form k_i to n.

Algorithm 2: Explore Augmented Graph

```
   Input: $k$, $d_{max}$, $G^s$, $K = (k_1, \dots, k_m)$, $TC$
   Output: the top-$k$ subgraphs
   // add cursor for each keyword element/interpretation to
      $Q_i \in LQ$
1  foreach $k \in K$ do
2      $Q_i.add(new\ Cursor(k, k, \emptyset, 0, k.w))$;
3  while not all queues $Q_i \in LQ$ are empty do
4      $c \leftarrow minCostCursor(LQ)$;
5      $n \leftarrow c.n$;
6      if $c.d < d_{max}$ then
7          $n.addCursor(c)$;
           // do not expand keyword elements further
8          if $n \notin K$ then
               // get all neighbors except parent element of $c$
9              $nbrs \leftarrow neighbors(n) \setminus (c.p).n$;
10             if $nbrs \neq \emptyset$ then
11                 foreach $n \in nbrs$ do
                       // check for cyclic path and temporal
                          satisfaction
12                     if $n \notin parents(c)$ and $satisfies(n, TC)$ then
                           // add new cursor to respective queue
13                         $Q_i.add(new\ Cursor(n, c.k, c.n, c.d + 1, c.w + n.w))$;
14         $Q_i.pop(c)$;
15         $topk \leftarrow Topk(n, SG, LQ, k)$;
16         if $topk \neq \emptyset$ then
17             return $topk$;
   // top-k results failed to be computed, return the current
      best
18 return $SG$;
```

The first step of the algorithm is the construction of the augmented graph. This graph is constructed at query time augmenting the information present in the summary graph. The exploration process uses the keyword elements derived from the keyword interpretation as its starting elements ($K = (k_1, \dots, k_n)$), constructing also the respective cursors (lines 1–2). All generated cursors are kept into priority queues, $Q_i \in LQ$. Each queue Q_i keeps the cursors having as origin the keyword element k_i. Each starting element forms a point of an independent exploration of the augmented graph. In effect, during exploration and for each starting element, many different paths are explored and for each of them a different cursor is used. At each step of the exploration process a cursor (i.e., a path) with the lowest cost is chosen for expansion (according to a cost function) from a queue of cursors (line 4). If the current visited element has also been explored by other cursors emanating from every other starting element, then a subgraph has been found, and the current visited element constitutes a *connecting element* (line 1 of Algorithm 3). This subgraph, which is produced by the combination of the paths emanating from all starting elements and end at the connecting element, is inserted into a list of candidates subgraphs (line 2 of Algorithm 3), SG, from which only the top-k will be selected at the end. The exploration process terminates when one of the following conditions are true:

1. The exploration depth has reached an upper limit, d_{max} (line 6 of Algorithm 2). In this case, all regions with radius d_{max} around each starting element have been explored. It is justified that the exploration process have to stop, because it is unlikely to lead to relevant information.

2. All top-k subgraphs have been produced. This happens only when the highest cost of a subgraph from the list of candidate subgraphs (worst subgraph) becomes less than the lowest cost of newly created cursors (current best path) (line 7 of Algorithm 3). Because of the fact that at each step of the exploration, the cursor with the globally lowest cost is selected and expanded, it is certain that paths with the lowest cost are explored at first place. So, new subgraphs will have cost greater than this cost (given that a subgraph is composed of many paths). In the case that this cost becomes greater than the highest cost of all produced subgraphs, the top-k subgraphs are the k subgraphs with the lowest cost.

3. All graph elements have been explored.

The above description of the exploration process is very similar to the one employed in [TWRC09]. We have extended this process in the following ways:

Temporal dimension. The exploration algorithm has been extended to handle temporal data. More specifically, each element of the data graph (either edge or node) is associated with a time period denoting its validity time/lifetime. Upon submission of a user keyword query with a temporal constraint, TC, the exploration process, as described above, expands only those elements that satisfy TC (line 12 of Algorithm 2). Because of the fact that indefinite temporal information can be queried, elements that possibly satisfy the temporal constraints are visited and expanded, but their contribution to the overall cost is higher, so they are ranked lower in the answer.

Exploration termination. Apart from the termination conditions listed above, we have enforced another one that terminates the exploration process of a cursor emanating

```
Algorithm 3: Compute top-$k$ Subgraphs
  Input: $n$, $SG$, $LQ$, $k$
  Output: $\emptyset$ or topk subgraphs
1 if $n$ is a connecting element then
      // generate possible subgraphs
2     $SG.add(gensubgraphs(n))$;
3 $SG_{best} \leftarrow k\text{-}best(SG)$;
4 $lowestCost \leftarrow minCostCursor(LQ).w$;
5 $highestCost \leftarrow k\text{-}ranked(SG_{best})$;
6 if $highestCost < lowestCost$ then
      // top-k subgraphs have been computed
7     return $SG_{best}$;
8 return $\emptyset$;
```

from a specific starting element. According to this, when the exploration process is about to expand a cursor to an element that is a starting element itself, it is visited, but a cursor is not produced for that element (line 8 of Algorithm 2). The rationale behind this is that if the expansion continued for this element, then the cursors that would be produced beyond that would have already been produced by the exploration emanating from this starting element itself. This extension is considered significant, because it reduces both the exploration time and space.

Synchronous exploration for each combination of keyword interpretation . This algorithm has been extended to allow synchronous execution of a number of exploration processes sharing a number of resources and data structures.

Query Mapping

During the phase of query mapping, the subgraphs that have been produced during the exploration process are mapped to SPARQL queries, which are evaluated on top of the RDF store.

A complete mapping of such a subgraph to a conjunctive query can be obtained as follows:

Processing of graph nodes. The labels of nodes might be used as constants. Thus, nodes are associated with their labels. Also, nodes might stand for variables. Every such a node is therefore also associated with a distinct variable. To support this two functions are defined, $constant(n)$ and $var(n)$ that return either the label of a node or a variable.

Mapping of relation edges. Relation edges are edges between two nodes that are entities. In the augmented graph, these nodes denote classes. So, each such edge

$e(n_1, n_2)$ is mapped to three RDF triples of the following form:

$$\begin{aligned}(var(n_1), type, constant(n_1))\\(var(n_2), type, constant(n_2))\\(var(n_1), e, var(n_2))\end{aligned}$$

The first two triples express the fact that n_1 and n_2 are instances of the respective classes, while the third triple express the fact that e is the property that relates these two entities.

Mapping of attribute edges. Attribute edges are edges between two nodes of which the first is an individual and the second is a typed literal value. Such an edge $e(n_1, n_2)$ is mapped to two RDF triples of the following form:

$$\begin{aligned}(var(n_1), type, constant(n_1))\\(var(n_1), e, constant(n_2))\end{aligned}$$

By traversing the subgraph and by the exhaustive application of these mapping rules, a subgraph can be translated to a query. The query is simply a conjunction of all the triples generated for a given subgraph.

Entity Transformation

During the phase of entity transformation, all subgraphs generated by the phase of graph exploration, together with the entities derived by the phase of query mapping are processed to derive all distinct entities and assign them a cost. These entities form the answer to the user. This is in contrast to [TWRC09], in which they stop at the previous phase, involving the user to chose the preferred query (subgraph) to be evaluated.

To transform a subgraph to a set of entities, the subgraph is traversed keeping only the nodes. We will denote the set of entities derived from a subgraph SG_i as SGE_{SG_i} and their union as $SGE \equiv \bigcup_{SG_i} SGE_{SG_i}$. Similarly, the set of entities derived from the phase of query mapping for a subgraph SG_i is given as QE_{SG_i} and their union as $QE \equiv \bigcup_{SG_i} QE_{SG_i}$.

The next step is to assign costs to the entities of both sets SGE and QE. For this purpose, we keep the minimum cost of all subgraphs in $minCost$, and the set of subgraphs in which an entity from each set appears in $SG_{SGE}(e)$ and $SG_{QE}(e)$. Furthermore, the cost of an entity or a subgraph can be derived from the cost function $C_{cf}(\cdot)$, where cf is any cost function mentioned in subsection 3.3.2..

Then, the cost of an entity derived from all subgraphs, i.e., $e \in SGE$, is given by the following formula:

$$Cost(e, S) = \frac{C_{cf}(e) * minSGCost}{|SG_S(e)|} \sum_{SG_i \in SG_S(e)} \frac{1}{C_{cf}(SG_i)}$$

The previous formulae resemble a weighted average of the cost of an entity over the subgraphs in which it appears. The weight for an entity cost in a specific subgraph is getting lower as the subgraph containing it is ranked lower. If an entity belongs to both sets SGE and QE, then the final cost is the average of the costs $Cost(e, SGE)$ and $Cost(e, QE)$. Thus, the final cost of an entity $e \in SGE \cup QE$ is given by the following expression:

$$Cost(e) = \begin{cases} \frac{Cost(e,SGE)+Cost(e,QE)}{2} & \text{, if } e \in SGE \cap QE \\ Cost(e, SGE) & \text{, if } e \in SGE \text{ and } e \notin QE \\ Cost(e, QE) & \text{, if } e \in QE \text{ and } e \notin SGE \end{cases}$$

3.4 Conclusions

This chapter presented our approach to keyword-based search. In particular, it discussed our contributions to keyword-based search over RDF data and how it differentiates itself with the work it has been based on, namely, [TWRC09]. It presented the data model and the query language, which exhibit temporal constructs. Last, it presented the keyword-based search algorithm emphasizing on the extensions of the respective algorithm discussed in [TWRC09].

Chapter 4

Implementation

In this chapter the implementation details concerning the work discussed in Chapter 3 is presented. For the rest of the document, the implementation of this work is referenced as *keyword querying system* or simply *system*, when there is no possibility for confusion. The keyword-based query system can be seen as a system accepting keyword-based queries, and specifically, queries following the form of the query language introduced in Chapter 3.

The contributions of our implementation are the following:

- An indexing mechanism is employed (see Subsection 4.2.2) which is able to index RDF literals while the algorithm is running as opposed to [TWRC09], in which the indexing takes place offline. This is achieved, because the algorithm has been designed to be independent from the storage layer, which can be updated independently.

- Similar to that is the construction of the graph data structures used by the keyword search algorithm, which are constructed online without affecting its time performance (see Subsection 4.2.1).

The rest of the chapter is organized as follows: First, in Section 4.1 the architecture and the role of the system in a more abstract way are presented. Second, Section 4.2 discusses the various components of the system in detail and how they coordinate with each other. Third, Section 4.3 discusses the technical details of the system, such as the programming language and the environment on which it was developed. Last, Section 4.4 concludes the chapter.

4.1 The Architecture of the Keyword Querying System

The keyword querying system functions on top of a RDF store providing a keyword-based search service. Any application interested in keyword-based search on RDF data can easily integrate the keyword querying system in its architectural design. The keyword querying system expects as input a query expressed in the query language introduced in Chapter 3. In this respect, it expects a list of keywords and a set of temporal constraints. The output is an ordered list of RDF entities ranked according to how much they reflect the user information needs as conveyed by the query.

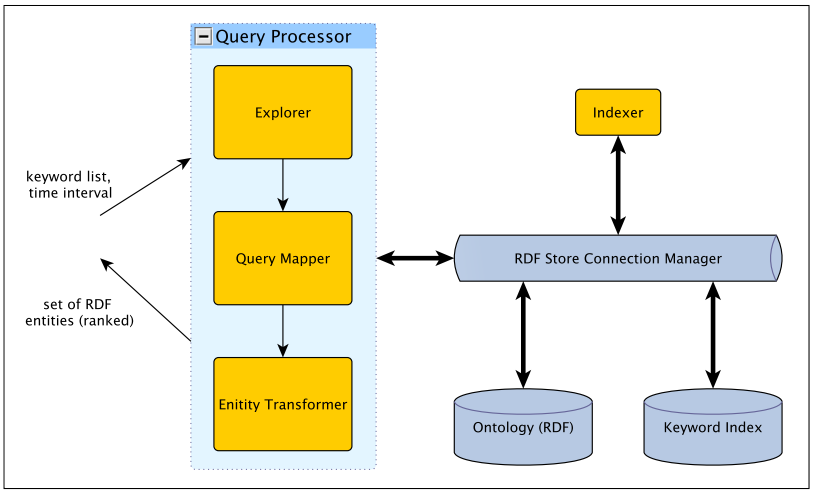

Figure 4.1: The architecture of the keyword querying system.

In Figure 4.1 the architecture of the system is depicted, together with the user interaction. Upon submission of a user query, the keywords are processed by the *Query Processor* component and mapped to entities of the underlying RDF data, using the index that has been constructed by the *Indexer* component (see Subsection 4.2.2). These entities are the starting points of the exploration process of the augmented graph of the underlying data (see Chapter 3 for details). The exploration process computes subgraphs of the augmented graph that match the user keywords and the temporal constraints. At a next step, each subgraph is mapped to a SPARQL query that is evaluated in the RDF store, whose results, i.e., entities, are added to the subgraph. All such subgraphs are then processed and transformed to an ordered list of RDF entities with scores, which form the answer to the user query.

In the next section we present the various components of our query processing module in detail.

4.2 Components

The query processing module (Figure 4.1) consists of three first-level components, the Query Processor (Subsection 4.2.1), the Indexer (Subsection 4.2.2), and the RDFStore Connection Manager (Subsection 4.2.3).

4.2.1 Query Processor

The Query Processor is the main component of the querying system. This processor receives the input (in the form of a string) and then invokes and controls the rest of the first-level components. The most important part of the Query Processor is the Graph Index which encapsulates the data model of the system. In fact, Graph Index contains a view of the underlying RDF data, employing the graph data structures data, summary, and augmented graphs as described in Chapter 3. The Query Processor component uses the Query Parser in order to parse the input and break it in keywords and temporal constraints, eliminating duplicate keywords and special characters (such as *, \, ?, ", ’, ˜, etc.). These keywords are then mapped to entities by invoking the Indexer component. These entities, which can be nodes or edges of the underlying RDF graph, are obtained from the Indexer in the form of URIs. At a second step these URIs are mapped to elements of the data model using the Graph Index producing keyword elements. It is crucial here to draw a distinction between the URIs of the Indexer that refer to URIs of the underlying RDF graph and the respective elements of the URIs in the data model. Clearly, the Query Processor orchestrates the invocation of the other components and their in-between interaction.

Graph Index

The Graph Index embodies the data model of the keyword querying system. It makes use of the three graph datastructures, namely, data graph, summary graph, and augmented graph. Graph Index constructs a summary graph view of the underlying RDF data upon loading the RDF data in the RDF store, and persists it in the RDF store. The augmented graph is constructed at the time of query processing. To achieve that, the summary graph is augmented with the keyword elements obtained from both the Indexer and the Graph Index forming the starting points of the exploration process executed by the Explorer component.

Note that the summary graph is loaded only once at the initialization stage of the Query Processor component, whereas the augmented graph is constructed only when a query is submitted. The exploration process takes place on the augmented graph only. The reason behind this is that, first, we are interested in computing queries and not answers to queries; we want to derive the query structure of the computed subgraphs and then have them evaluated over the RDF store. This also leads to a very good performance since the augmented graph is much more compact and smaller than the actual data graph. Finally, upon construction of these views over the underlying RDF data, the Query Processor does not interact further with the RDF store.

Construction of the summary graph The construction of the summary graph is data-driven meaning that it is constructed taking into account only the instances of the underlying RDF graph and it represents the schema that the instance data conforms to. The construction is done at the time of loading the RDF data in the RDF store. In contrast to this approach, other approaches, such as [TWRC09], first, load the data into a storage backend, and then have the indices constructed offline.

According to our method, the construction of the summary graph does not incurs any significant extra time cost than just loading a RDF dataset. This is because the *inference mechanism*, provided from almost all RDF stores, is utilized. An inference mechanism is one through which new statements can be derived based on the triggering of a set of predefined rules when the premise is satisfied. There are two prominent algorithms for inference, *forward chaining* and *backward chaining*. Forward chaining runs proactively during the insertion of new statements adding inferred statements to the existing dataset, while backward chaining runs upon query submission inferring statements on the fly without persisting them. Each method has the advantages and disadvantages. In our setting, the most appropriate is forward chaining. The construction process is as follows: when a property between two instances is encountered, an edge between the respective classes of these instances is inferred. To do so, the property `rdf:type` of these instances has to be available. In other words, two RDF triples concerning the types of these instances have to be already present in the RDF store.

Comparing this approach to the one in [TWRC09], it is shown in Chapter 6 that the load time of an RDF dataset and the construction of the respective summary graph is much more efficient than executing these two tasks sequentially.

Summary graph statistics Besides the inference process described above, statistics are kept for the nodes and edges of the summary graph. In the case of nodes, these statistics reflect the number of instances of each class. In the case of edges, they reflect the number of properties that are between instances of the respective class nodes of the edge. Both numbers represent the support of each element of the graph by the data graph and can be used as a measure for their significance in the data graph. Such statistics are employed when the popularity or combine score functions are used (see Subsection 3.3.2).

Explorer

The Explorer component is invoked after the keywords have been mapped to elements of the summary and data Graph, and the augmented graph has been constructed. From these keyword elements, the augmented graph is then explored to find a connecting element, i.e., a particular type of a graph element (either edge or node) that is connected to all keyword elements. The paths between the connecting element and a keyword element are combined to construct a matching subgraph. The process continues until the top-k queries have been computed[1]. Finally, the resulting subgraphs are returned to the Query Processor, who then invokes the Query Mapper component to construct a SPARQL query for each such subgraph, and evaluate it on the RDF store.

[1]The process of exploration is described in detail in Chapter 3.

Query Mapper

The Query Mapper component is responsible for constructing SPARQL queries out of matching subgraphs of the explored augmented graph. A complete mapping of such a subgraph to a conjunctive query is computed using the rules described in detail in Chapter 3. The step of translating such a conjunctive query to graph patterns of a SPARQL query is then straightforward.

Entity Transformer

The Entity Transformer component takes over the process of transforming the subgraphs produced during exploration to RDF entities. The implementation is straightforward conforming to the description of the entity transformation as described in Chapter 3.

4.2.2 Indexer

The Indexer component is responsible for indexing the literals of the RDF graph. This index is created and stored in the RDF store. The purpose of this index is to facilitate the mapping of user keywords to URIs of the RDF graph. For the indexing, the LuceneSail software[2] is used which has been proposed in [MSG+08]. LuceneSail employs full-text search functionality over RDF by simply combining two well-known established systems: Sesame[3] and Lucene. It employs pure Lucene queries within pure RDF queries (using SPARQL), taking full advantage of the expressiveness of each of them. From the design perspective, LuceneSail can be incorporated into a Sesame system and accessed in a uniform and transparent way. This is achieved without any modifications of the syntax of the SPARQL query language according to [MSG+08], and because it has been designed and implemented as a SAIL stack. LuceneSail has excellent performance characteristics, while keeping requirements of resources low. It implements many common IR features, which are required in our case, such as:

- stemming and lemmatization,
- phrase, wildcard, fuzzy, proximity and range queries,
- boolean operators and term boosting.

4.2.3 RDFStore Connection Manager

The RDFStore Connection Manager is the component that takes over the interaction between the rest of the components (i.e., Query Processor and Indexer) and the RDF store (in our case Sesame). Every component that needs access to the RDF store has

[2]LuceneSail is available at `https://dev.nepomuk.semanticdesktop.org/wiki/LuceneSail`.
[3]`www.openrdf.org`

to invoke an appropriate method of this component. It has to be mentioned here that the RDFStore Connection Manager operates both locally and remotely because it uses the HTTP communication protocol of Sesame 2.0 and communicate directly with the Sesame Server[4]. Currently, the only type of queries that the RDFStore Connection Manager supports and are needed by the query module is SPARQL query evaluation and RDF data loading. Using the first type, the connection manager sends a SPARQL query for evaluation to the repository and returns back the matched resources. With the second type, the connection manager loads RDF data in the repository; the data may come from a file or a URI.

4.3 Technical Details

This section contains details concerning the implementation of the developed keyword querying system and dependencies to other systems that were utilized. The keyword querying system has been implemented in the Java programming language using the Java Development Kit 1.6[5]. For the development, the Ganymede[6] release of the Eclipse IDE[7] was used. The development took place on a Linux system with the following characteristics: Linux 2.6.28-13-generic #45-Ubuntu SMP i686 GNU/Linux. It has already been tested in a Windows system running the XP Professional operation system. The query processing module depends on the LuceneSail 1.2.0[8] full-text indexing tool of RDF data and Lucene 2.3.2[9] search engine. The RDF store functionality is offered using the Sesame2 RDF store[10].

4.4 Conclusions

In this chapter the architecture of the keyword querying system was presented and the various components that comprise it were described in detail. Finally, the technical details and requirements of the keyword querying system were discussed.

[4]`http://www.openrdf.org/doc/sesame2/system/ch08.html`
[5]`http://java.sun.com/javase/downloads/index.jsp`
[6]`www.eclipse.org/ganymede/`
[7]`www.eclipse.org/`
[8]`https://dev.nepomuk.semanticdesktop.org/wiki/LuceneSail`
[9]`http://lucene.apache.org/`
[10]`www.openrdf.org`

Chapter 5

The Papyrus Platform

Papyrus[1] (grant agreement number, 215874) is an European Union (EU) funded research project of the Information and Communication Technologies (ICT) Work Programme[2] under the 7th Framework Programme[3] (FP7-ICT-2007-1). Papyrus addresses Challenge 4[4] (Objective ICT-2007.4.1) of ICT Work Programme, i.e., Digital libraries and content and specifically Call 3[5], i.e., "Digital libraries and technology-enhanced learning".

The full title of Papyrus is "Cultural and historical digital libraries dynamically mined from news archives". Papyrus is a dynamic Digital Library (DL) aiming at providing many user communities with electronic access to available information of their discipline. In this respect, Papyrus follows the line of past and existing work for digital recapturing and preservation of European cultural and scientific heritage. These works consume significant effort and resources for the digitization, characterization, and classification of available content. What has never been targeted, however, is a digital library that makes available the content of one discipline to the user community of another.

Papyrus approaches this need by introducing the concept of a cross-discipline digital library engine. Papyrus is a dynamic digital library that processes user queries in the context of a specific discipline, searches for content in another domain, possible unrelated to that discipline, and presents the results in a way useful and comprehensive to the user. Papyrus showcases this approach with a specific pair of disciplines, which can be illustrated as an apparent need and may prove to be an immediate exploitation opportunity even on its own.

This proposed use case is the recovery of history from news digital content. The rational behind this selection is that vast amounts of digital news content exist in huge archives, which, although being of incredible value, are underused as they are not easily searchable and do not have a significant value if seen as individual news items. News

[1]`http://www.ict-papyrus.eu`
[2]`http://cordis.europa.eu/fp7/ict/programme/home_en.html`
[3]`http://cordis.europa.eu/fp7/home_en.html`
[4]`http://cordis.europa.eu/fp7/ict/programme/overview4_en.html`
[5]`http://cordis.europa.eu/fp7/ict/programme/challenge4_en.html`

organizations have been recording history as it developed at each point in time and have been doing this since their foundation, which for many news publishers or agencies dates back to the previous centuries. Furthermore, the content found in these news archives addresses the widest sense of cultural and scientific heritage, covering disciplines like the history of politics, sciences, and entertainment, allowing for the potential of cross examination of events in all these domains.

Apart from offering an interesting case study of Semantic Web technologies in action, Papyrus advances the current state-of-the-art of particular Semantic Web technologies in several ways. First, through a detailed user study, Papyrus has captured user needs in relation to ontologies for historical research and has implemented them in its two ontologies. Second, Papyrus has implemented a Web-based ontology browser suitable for users that are neither ontology, nor computer experts. Furthermore, it has implemented a keyword search tool that is tailored to the needs of historians taking into account entity evolution and time information.

The rest of the chapter is organized as follows. In Section 5.1 the user requirements, which set the goals of Papyrus, are presented. Next, in Section 5.2, an overview of the Papyrus platform is presented and is shown how it addresses the user requirements, and how it is related to this dissertation. Then, in Section 5.3, the News and History ontologies are presented that represent the two disciplines of Papyrus, namely, news journalism and history research. Last, Section 5.4 concludes the chapter.

5.1 User Requirements

Historians (or, alternatively, history researchers) constitute the users of Papyrus. At its first stages, Papyrus put the burden of work on conducting extensive user needs studies. A dimension that was particularly stressed during that time is the educational value of the Papyrus platform [KST09b]. The combination of existing historical research results — in the form of essays and terminology definitions — with archival material is considered of particular importance for the education of history students and the training of new researchers.

In the following subsections, the most challenging needs are presented.

5.1.1 Accessing archival content

The first relevant user need identified has been the ways that history researchers prefer to search and explore archival content. The usual way for a historian to proceed when searching for relevant material is to break down the research topic into groups of keywords, and then try to find material related to these keywords. Through our study, it was evident that history researchers feel comfortable searching with keywords, which, as a result, constitutes their main method for retrieving content from an archive. However, most of the researchers pointed out the deficiencies of existing keyword search tools for archives, in terms of both precision and recall. As a result, it is important for them to be able to have an effective keyword-based search tool to support archival research. The historians also emphasized the need for additional search facilities that would allow better navigation through search results and refinement based on various criteria. Another important requirement is the one for providing efficient ways to browse vocabularies and catalogues related to their historical research.

5.1.2 Historical research method

An important step in understanding user needs within Papyrus has been the study of representative topics and questions for history research. An example, which we will utilize in the rest, is the following:

> I am interested in the history of the discipline of Biotechnology, and in particular the history of Cloning and stem-cell research, and the controversies and ethical issues related to these.

Historians proceed in specific steps when attempting to gather the material needed to investigate a specific topic like the aforementioned. These steps are the following:

- Collecting *secondary material*, which includes essays of other history researchers on related subjects. This material is typically available in conventional or digital libraries and comes with a set of common vocabulary used by historians to refer to the topics covered by particular essays. This could contain historiographical issues, like “Controversies and Disputes”, “Discipline formation”, or “Change in science”, as well as general concepts like that of “Research”, or “Ethics”.

- Collecting *primary material*, e.g., news archive content related to the research subject. This material usually comes with another, different vocabulary, the one prominent during the time of the creation of the archive documents.

To collect the appropriate secondary material, users need to formulate their research topic in terms of the historiographical issues and concepts involved. To collect the necessary archival content, users need to identify the appropriate vocabulary used in archives to annotate this content. This means that history researchers should be able to effectively express their research topics in the vocabulary of the archive. However, as already mentioned, this is a challenging task due to the fact that archival material covers large time-spans and may even be different in different time periods. Thus, inexperienced researchers may easily miss important information that is present in the archive due to the lack of vocabulary knowledge of the right terminology to use.

In our example, an inexperienced user may search the news archive with keywords, such as “history of biotechnology”, “cloning and ethics”, “stem-cell controversies”, which is in fact the vocabulary of the secondary sources. As a result, such an approach would not return many relevant news items. A more experienced researcher would “translate” the query into the vocabulary of the news archive, by searching for the term “biotechnology”, along with its related terms in the past, like “biomedical chemistry”, “biotechnics”, “fermentation”, etc. Also, to identify news items related to ethical issues and Cloning, she would use keywords like “bioethics committee” or even “Roman Catholic Church”. However, this ability to transform a query to those terms, which may help extend the search results, greatly depends on the experience of the researcher in the particular domain.

5.1.3 Concept evolution

A very important issue for history researchers is the evolution of concepts with the passage of time, which may include changes in their name, or subtle changes in their definition. Sometimes a search may result in a scarce or irrelevant set of documents, returned due to such misinterpretations of a modern term against its past usage.

Returning back to our example, the history of the concept “Biotechnology” has changed in meaning and name many times within the 20th century. Biotechnology as a concept and scientific discipline has progressed from food technology and fermentation, to genetics and biomedical engineering [Bud91], whereas the concept of Cloning has had several important breakthroughs and changes in its techniques over the last century. Time is also important to historians for capturing the evolution of concepts. In fact, many kinds of time are important: absolute (e.g., the year 1963), symbolic (e.g., the year of Dolly’s creation), relative (e.g., before 1963), and indefinite (e.g., three to five years after 1963).

5.1.4 Multilingualism

The issue of multilingualism in the context of a digital repository providing access to archival content of different countries and in different languages is particularly important for historical research. One dimension of the problem is related to the fact that a concept may have been introduced in different time points in different languages. In our example, the concept “Biotechnology” has undergone different development paths in German-speaking and English-speaking countries[Bud91]. A second, more complicated dimension of the problem is related to the fact that a term used as the name of the concept during the same time period could mean different things in different languages. Taking, for example, the development of the concept Biotechnology in the German-speaking countries, there were two terms with different connotations used to refer to the same concept of Biotechnology in English, i.e., “biotechnik” (biology-based technology) and “biontotechnologie” (microbiology and fermentation). Based on such cases present in historical essays it was recorded that concepts can be described by different terms depending on the context of use. For example, the concepts and the terms that describe them may change not only due to the passage of time, but also due to the place they are used, the viewpoint of the person using them, etc.

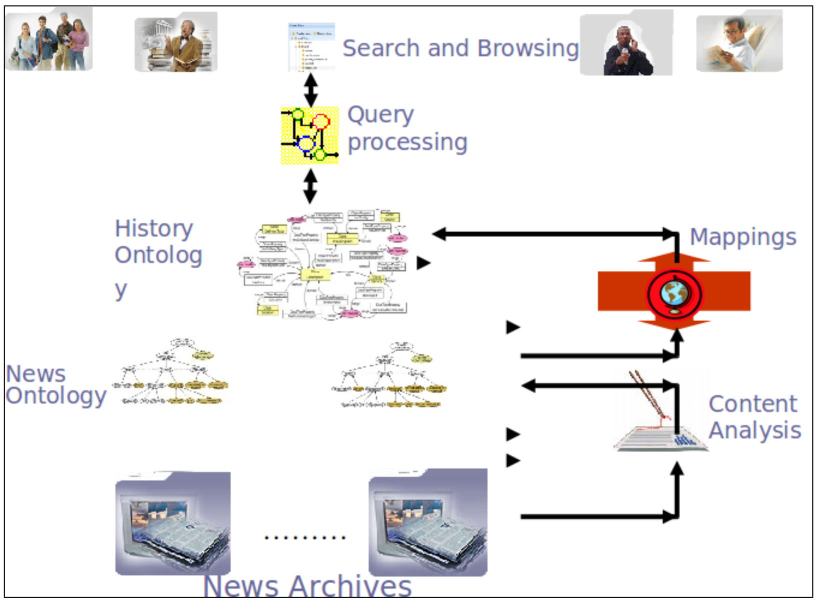

Figure 5.1: The Papyrus Platform

5.2 The Papyrus Platform

Taking into account the user needs identified in Section 5.1, Papyrus attempts to provide an effective way to support the history researcher in the process of accessing historical archives to retrieve useful material. To realize this objective, Papyrus has applied and extended existing Semantic Web technologies. The current working prototype is already available[6] and its evaluation has been concluded and is presented in Chapter 6.

The Papyrus platform, shown in Figure 5.1, is designed with the History and News ontologies at its core, which model the History and News domains respectively. The two ontologies have been created as extensions of existing standards with the cooperation of the corresponding domain experts, historians, and journalists. Details about these ontologies are given in Section 5.3. The Papyrus platform allows historians to carry out their research by utilizing news archives as primary sources. In the current implementation, these archives consist of subsets of the news archives of Agence France Press[7] (AFP) and Deutsche Welle[8] (DW), which contain articles, images, and videos specializing in the area of Biotechnology and Renewable Energy, areas of particular interest to historians and journalists. News articles are represented in XML and stored in a relational database. Through specialized content analysis algorithms and tools developed in the context of the project [FCI10, PPT10], archival content has been extensively annotated

[6]http://hades.atc.gr/CMS_Papyrus_1_1/
[7]http://www.afp.com/afpcom/en/
[8]http://www.dw-world.de/

with a rich set of metadata based on the entities of the News ontology. These analysis and annotation processes are out of the scope of this dissertation and will not be presented any further.

The News ontology is mapped to the History ontology by taking advantage of the historical research method to retrieve information on specific historical topics. These mappings are discussed in Subsection 5.3.3.

For the visualization and browsing of the ontologies and the news archive, the Papyrus platform offers a specialized browser which, together with the keyword search and the mapping mechanisms of the platform, enables users to navigate from History ontology entities to News ontology entities and achieve effective access to the primary material in the archives. The Papyrus platform offers, also, a number of Web tools to allow for distributed multiuser ontology editing, creation of mappings between the two ontologies, and management of news content and analysis results [BKTV10]. The description of these tools is not included in this dissertation.

5.3 News and History Ontologies

This section, first, describes the ontologies used to express the primary and secondary domain of Papyrus, i.e., the news archive and history respectively. Second, it discusses the role of mappings, and how they are employed to bridge the gap present in these ontologies.

5.3.1 The News Ontology

The News ontology [Kiy] was developed with the help of news professionals working in AFP and is intended to describe the structure and the semantics of the news content. The ontology was constructed based on the NewsML-G2 XML standard[9], designed by the IPTC[10] [Tro08] for conveying and annotating news content. The purpose of this standard is to provide a model for the description of news items, and their related topics and keywords. It is used by major news providers like EBU[11] and Reuters Media[12]. For the needs of the Papyrus project, there was a need for integrating two different parts in the ontology:

(a) the modeling of the format in which news items are produced by the main news agencies, i.e., the constructs adopted from NewsML- G2, and

(b) the modeling of concepts present in the news items and relevant to the Papyrus domains, i.e., Biotechnology and Renewable Energy. These include named entities, concepts to accommodate domain-specific information, and instances.

[9]`http://www.iptc.org/cms/site/index.html?channel=CH0111`
[10]International Press Telecommunications Council, `http://www.iptc.org`
[11]European Broadcasting Union, `http://www.ebu.ch/`
[12]`http://www.reuters.com/`

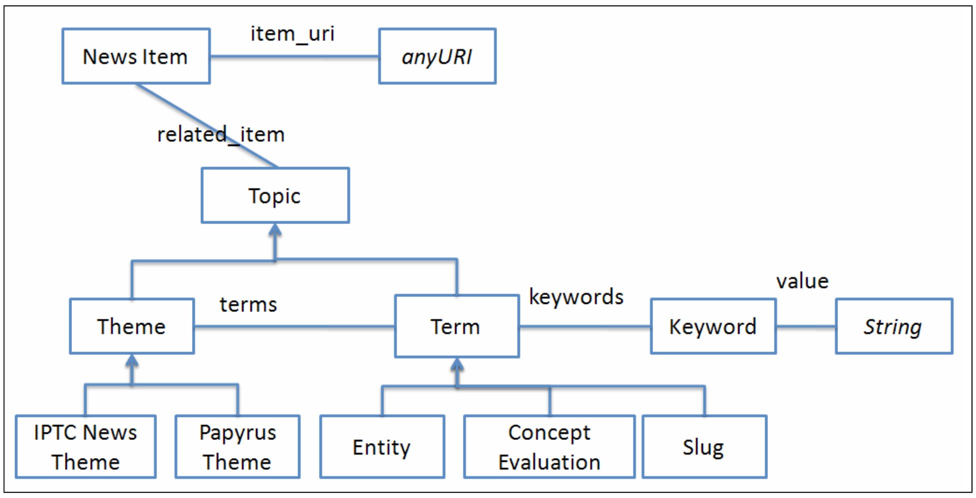

Figure 5.2: News ontology structure

The part of the News ontology corresponding to the NewsML-G2 XML standard, represents 'syntactic' metadata related to news items packaging and exchange, and it is omitted from the presentation. More information can be found in the NewsML-G2 web site. The basic structure of the Papyrus News ontology is illustrated in Figure 5.2; arrows represent `is-a` relations and named arrows roles. Each news item is identified by its URI and has a list of related topics that may be *themes* — those established by the IPTC categorization[13] to be used by the news agencies when annotating their news content —, or *terms* — such as *named entities*, *concepts*, or *slugs*, i.e., terms related to the IPTC subjects. In turn, each term can be described by a set of keywords.

Within Papyrus, the ontology has been largely populated with named entities, domain-specific concepts and their related keywords specified in three languages, English, French, and German. The populated concepts are those related to the two main Papyrus domains, Biotechnology and Renewable Energy, and were retrieved from a selected training document set by using semi-automatic mining techniques on the news archive content [FGM05, McC96].

5.3.2 The History Ontology

In contrast to the News ontology, which attempts to capture the "here and now" of everyday events, the History ontology [Kiy] models the history perspective on the events and issues covered by the news. The History ontology was developed as an extension to the CIDOC Concept Reference Model[14] (CRM) that embraces several good practices of modeling information in different domains. This ontology focuses on the comprehensive

[13]http://www.iptc.org/NewsCodes/index.php
[14]http://cidoc.ics.forth.gr/

representation of the world knowledge, and includes such concepts as Thing, Physical Man-made Object, Conceptual Object, Primitive Value, and others.

To model the history domain, the CIDOC CRM was extended with domain-specific knowledge and abstract concepts important for the needs of historians. Three different groups of concepts were identified in [KST09a], which are presented in the following:

- General historiographical issues that express history research topics, like "Change in Science", or "Public opinion". These have been modeled as instances of the sub-concept "Historiographical Issue" of CRM's concept "Type", and their hierarchy is expressed by "narrower-broader" CRM's relations. The main issues that form the upper level of the hierarchy are: "Change in Science and Technology", "Controversies and Disputes", "Ethics", "Institutions", "Popularization" and "Research and Development". The source for this categorization has been the journals of the Society for the History of Technology[15] and the History of Science Society[16], Technology and Culture[17], and ISIS[18], respectively.

- Specific domains of science and technology, like "Biotechnology" or "Renewable Energy". The domains have been modeled as instances of the sub-concept "Domain" of CRM's concept "Type".

- Specific entities that may be the subject of the research in these domains, like "Stem cell" or "Wind mill", or general concepts like "Researcher". These have been added as sub-concepts or instances under the most appropriate corresponding CIDOC sub-concept of "Persistent Item", "Place", or "Temporal Entity".

To fully tailor the ontology to the needs of history, the three aspects mentioned in Section 5.1 had to be appropriately modeled: time periods, evolution, and multilingualism. This effort was the most challenging aspect of the modeling process. The modeling of concept and instance change in the ontology is addressed by assigning periods to all entities and by representing changes from one to the other. Assigned periods allow the representation of fuzzy time intervals (i.e., the start and end point are intervals) with days as the minimum granularity [Kiy]. The evolution schema allows the expression of change through the use of properties like "split", "merge", "evolve", "detach", and "join". For the example of change in Biotechnology, a set of evolution operators which connect the concept "Biotechnology" with other concepts that preceded or superseded it have been defined: "Fermentation joined Conventional Biotech", "Conventional Biotech became Biotechnology", "Cloning joined Biotechnology", and others (see detailed example in [RVMB09]). Once these evolution operators are defined, they can be used to formulate and run historical queries about concept changes. More information on the evolution framework can be found in [RVMB09].

[15]`http://www.historyoftechnology.org/`
[16]`http://www.hssonline.org/`
[17]`http://etc.technologyandculture.net`
[18]`http://www.journals.uchicago.edu/toc/isis/current`

For multilingualism we have developed a context-based approach [TVKM10]. In particular, our approach allows associating terms and concepts under specific contexts (essentially under different conditions), thus specifying which terms are valid interpretations of other terms in specific contexts. A *Term* is associated with a *Concept* through a *TermAssociation*. A *TermAssociation* specifies the context under which this association is valid. The context is characterized by its related time, place, language, dialect, domain, historiographical issues, viewpoint, formality and diatype. In addition, a *TermAssociation* has a *Confidence value* and may carry a *Definition*.

The advantage of this approach is that history researchers that specify queries using certain terms will retrieve news items which are valid interpretations under the history context. For example, a query that uses the English term "biotechnology", will also retrieve the news items containing the German terms "biotechnologie" and "biotechnik", on account of being different terms representing the same concept, that of Biotechnology.

Both ontologies have been modeled in RDF and are available for browsing through the Ontology Browser view of the Papyrus platform. More information can be found in [Kat, Kiy].

5.3.3 Mapping the History and News Ontologies

The mappings are the tools for bridging information across the two domains. The Papyrus mapping framework adopts an entity-based data model, which is based on a dataspace data model like the one in [DKP$^+$09], making the entity the primitive mapped unit. This facilitates the formulation of the information modeling and of the mappings, since it is conceptually closer to the way humans are thinking. In terms of the mapping language, Papyrus adopts an entity-based language that is similar, in spirit, to logical languages like Datalog [AHV95]. The semantics of the language are, however, fundamentally different from Datalog in that it allows the definition of mappings even in the absence of any schema information. More information on the mapping framework may be found in [BKTV10].

An example of a simple mapping is the following:

```
'history:Cloning'(), 'history:Ethics'() ->
                        'news:Concept_Cloning_00085'()
```

The domain expert, in this case, has defined that when the user in interested in the historiographical issue "Ethics" in relation to "Cloning", one of the related news ontology concepts to be retrieved will be "Bioethics committee" (`Concept_Cloning_00085`). To support the construction of complex expressions as mappings, Papyrus offers a graphical mapping interface in which the ontology information is presented as a set of entities [BKTV10].

5.4 Conclusions

This chapter presented an overview of the progress made so far within the on-going EU-funded project Papyrus in relation to providing end user tools and services to support cross-discipline access to archives, and more specifically, historical research in news archives. Several Semantic Web concepts, techniques, and tools were presented, that Papyrus applies and extends to achieve its goals. Papyrus allows the history researcher to explore both primary and secondary sources which have been structured and unified through their respective domain ontologies. This coupling addresses a very important user need, that of bringing together these two different sources.

Papyrus is envisioned as a platform to be deployed on top of existing digital libraries and archives in conjunctions with platforms like Europeana[19], in order to offer advanced and unified exploration and search functionality to researchers with advanced needs for information retrieval.

[19]`http://www.europeana.eu/portal/`

Chapter 6

Evaluation

This chapter discusses the evaluation methodology followed in this dissertation and comments on its results. Chapter 2 showed that the current status of the evaluation methodologies are far from good without providing any solid and rigid methodology, as opposed to the respective evaluation process developed in the field of IR. However, this dissertation does not contribute in any way to this field. Instead, it follows the footsteps of related work, but conversely tries to extensively evaluate its work both in terms of efficiency and effectiveness. To this end, it uses three different datasets of various sizes, which exhibit different qualitative characteristics. Then, based on these datasets, it measures the performance of the algorithms and the scalability of the keyword search system, as well as the quality of the results, employing a variety of IR metrics.

The rest of the chapter is organized as follows. Section 6.1 presents the employed datasets over which the evaluation of the keyword querying system took place. Section 6.2 focuses on efficiency; it introduces the dimensions used in the experiments and presents the respective evaluation results. Section 6.3 is oriented towards effectiveness; it introduces the employed IR metrics, and presents the respective evaluation results for the case of the History ontology only[1]. Last, Section 6.4 concludes the chapter.

6.1 Datasets

6.1.1 History ontology

The History ontology dataset[2] is an extension of the CIDOC Concept Reference Model[3] (CRM) that embraces several good practices of modeling information in different domains. It was developed in the context of the Papyrus project (see Chapter 5) and more details about its content can be found in Subsection 5.3.2.

[1]This is due to lack in experienced users that are needed to provide relevance judgements. For the case of History ontology, the historians of the Papyrus project were consulted.

[2]It is available for browsing at `http://igg.di.uoa.gr/PapyrusOntoBrowser` using the “Ontology Browser” tab.

[3]`http://cidoc.ics.forth.gr/`

Table 6.1: Structural statistics for the History ontology dataset.

#Triples	#Classes	#Properties	#Instances	Avg. #Instances/Class
8,327	367	727	1,405	4

Table 6.1 shows various statistics concerning its structure. As far as the number of triples is concerned, it is a rather small ontology. Compared with the number of classes and properties, it contains a lot of structural information, that is, it contains a great many classes connected with each other with several properties[4]. Last, judging by the number of instances, it is sparsely populated. Such characteristics have a significant impact on the exploration algorithm, because they lead to a dense summary graph. On the other hand, its small size allows for a small size in the full-text index, and thus, the process of keyword interpretation is very fast.

The queries posed to the History ontology dataset have been compiled by expert historians and can be found in Appendix A.1.

6.1.2 Semantic Web Dog Food Corpus

The Semantic Web Dog Food dataset[5] is a corpus containing information on papers that were presented, people who attended, and, in general, information that has to do with the main conferences and workshops in the area of Semantic Web research since 2006. The snapshot used in this evaluation covers the period 2006–2010 (until month September) and was obtained in RDF format[6].

Table 6.2: Structural statistics for the Semantic Web Dog Food dataset.

#Triples	#Classes	#Properties	#Instances	Avg. #Instances/Class
88,996	96	369	8,580	89

Table 6.2 shows various statistics concerning its structure. As far as the number of triples is concerned, it is a medium sized ontology. Compared with the number of classes and properties, it contains a lot of structural information, that is, it contains a great many classes connected with each other with several properties. Last, judging by the number of instances, it is well populated. Such characteristics have a significant

[4]Note that the number of properties reflects those properties that have as domain and range instances of classes, that is, it does not include properties that have as range literals or other datatypes. This applies, also, to the tables that follow.

[5]`http://data.semanticweb.org/`

[6]`http://data.semanticweb.org/dumps/`

impact on the exploration algorithm, because they lead to a dense summary graph. On the other hand, its medium size allows for a medium size in the full-text index, and thus, the process of keyword interpretation is quite fast.

The queries posed to the Semantic Web Dog Food dataset can be found in Appendix A.2.

6.1.3 DBLP

The DBLP dataset[7] contains the computer science bibliography as published in various conferences, journals, and books. The dataset used in this dissertation is the snapshot of 14 May, 2010, and was obtained in N-Triples RDF format[8].

Table 6.3: Structural statistics for the DBLP dataset.

#Triples	#Classes	#Properties	#Instances	Avg. #Instances/Class
55,364,046	12	20	3,609,294	300,775

Table 6.3 shows various statistics concerning its structure. As far as the number of triples is concerned, it is a big dataset. Compared with the number of classes and properties, it is very simple in structure, and conversely it is highly populated. Such characteristics is not expected to have a significant impact on the exploration algorithm, because they lead to a small summary graph. On the other hand, its big size allows for a big size in the full-text index, and thus, the process of keyword interpretation is expected to be time-consuming. Last, due to the fact that the dataset covers a specific field — which in turn leads to a high number of the same keywords appearing in many different places in the dataset —, it is highly probable that the small schema graph will grow in size at query time.

The queries posed to the DBLP dataset can be found in Appendix A.3.

6.2 Measuring Efficiency

The efficiency of the keyword querying system is measured[9] both in terms of scalability and performance. For the case of scalability, it is measured against several dimensions, such as load, service, data, geographic, and domain. For the case of performance, first, the construction and load times of the indexes is measured, and then the query processing performance, as well as the performance of each query processing task (i.e.,

[7]`http://www.informatik.uni-trier.de/~ley/db/`

[8]`http://dblp.l3s.de/dblp++.php`

[9]It is worth noting, that all measurements reflect the average of the measurements of 5 different runs of the experiments for each case (either for measuring efficiency, or effectiveness).

keyword interpretation, graph exploration, query mapping, and entity transformation) are measured.

6.2.1 Scalability

In the following, load, data, geographic, and domain scalability are analyzed. It is of great importance to mention here that the RDF store is not considered as part of the keyword querying system, but instead as a separate data storage backend system providing access to the RDF data. In this respect, all scalability measures considered are not presented for the case of the RDF store. The interested reader shall consult the documents concerning its implementation, namely, the Sesame RDF store.

Load scalability

The keyword system has been designed and implemented to be scalable in load. The keyword search algorithm is multithreaded keeping the memory consumption for each query issue at low levels and restricting the number of critical segments to the minimum.

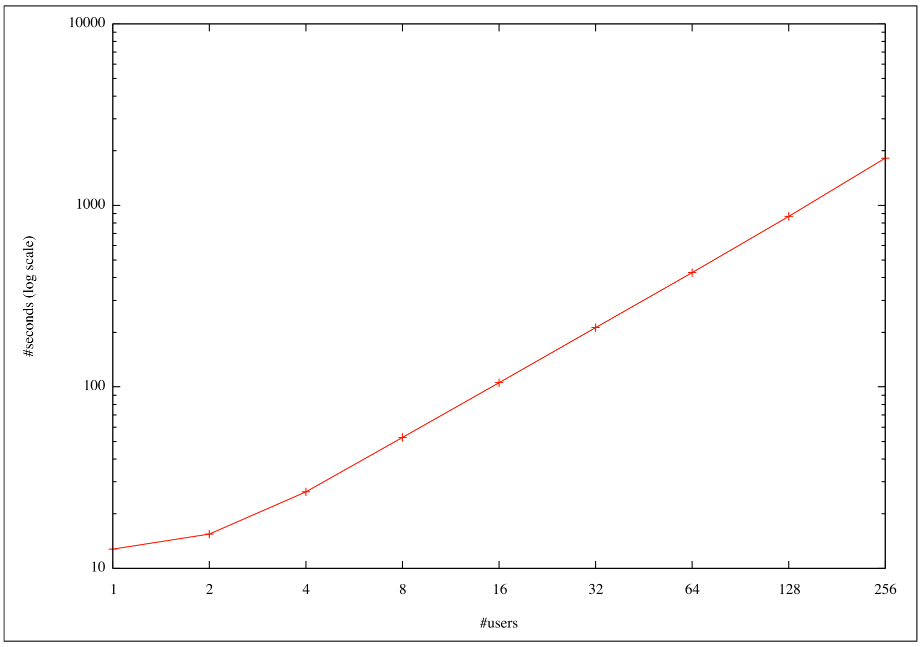

Figure 6.1: Load scalability for the History ontology

To measure load scalability, the keyword system is exposed to an increasing number of users, each one posing the same set of queries, which are 20 in the number. This set of queries differs from dataset to dataset and can be found in the Appendix A.

The results for each dataset are shown in Figures 6.1, 6.2, and 6.3. In all cases, the y axis corresponds to the time (in seconds) each group of users posed the set of

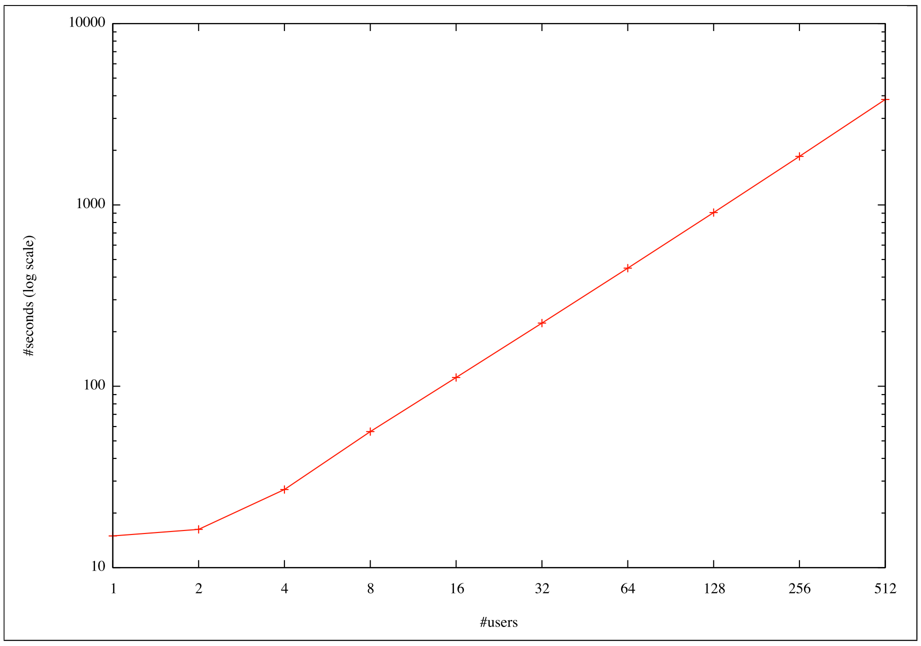

Figure 6.2: Load scalability for the Semantic Web Dog Food dataset

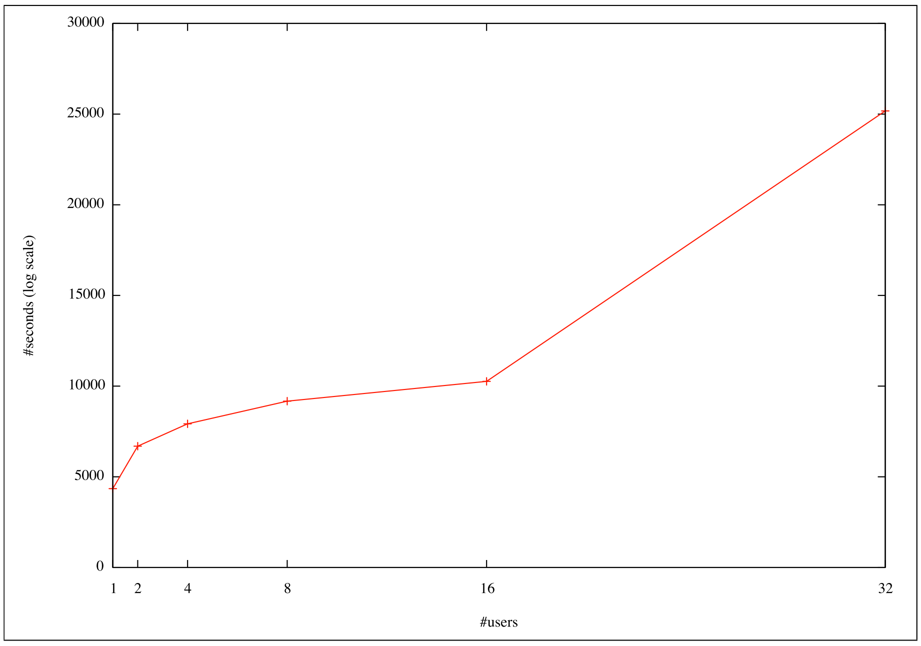

Figure 6.3: Load scalability for the DBLP dataset

queries and received the respected results. Both axes are drawn in logarithmic scale. Judging by these figures, the results are very promising; the keyword querying system is very scalable in terms of load, even for 512 users. Of course, there is no denying that the number of queries posed is very small, howbeit, the results are encouraging. In the special case of the DBLP dataset[10], the system does not present a stable scaling. In particular, the increase in query response time from 1 user to 2 users is very high, while from 2 to 16 users the system exhibits excellent scaling. Last, from 16 to 32 users the system exhibits a scale of 1.25, namely, it performs 0.25 worse than the ideal.

Service scalability

From the perspective of the keyword querying system, service scalability amounts to load scalability. This is because the load for this system corresponds to the number of its concurrent users. In fact, the load amounts to the number of concurrent queries issued and, as such, a user is viewed as a query. Therefore, a session is embodied in a single query.

Data scalability

From the perspective of the keyword querying system, data scalability heavily depends on the memory/CPU requirements of the underlying keyword search algorithm, as well as the performance characteristics and data scalability of the employed RDF store. As mentioned before, the scalability of the RDF store should not burden the keyword system. The latter has been designed with the requirement of storage abstraction and thus different RDF stores can be employed for storing the RDF data.

The keyword search algorithm employed has been designed and implemented to meet low memory requirements. The algorithm operates on the schema of the underlying ontology and thus the memory requirements are insignificant even for very large ontologies. For example, as of the November 2009, the DBpedia dataset describes 2.9 million "things" [sic] (i.e., resources) with 479 million "facts" [sic] (i.e., triples). From these resources, 205 are classes which are inter-connected with 1200 properties, 1.170.000 are individuals, and the rest 479 million resources identify links to external images, web pages, and other datasets. Taking into account that DBpedia is a snapshot of Wikipedia, which is rapidly evolving, mostly in terms of data entries and not in terms of new knowledge that affects its schema, it is realistic to assume that the aforementioned classes and properties, which comprise the summary graph, can fit in main memory, and that do not incur any significant space overhead to the algorithm.

From the perspective of response time, the keyword search algorithm has to be viewed from the perspective of several tasks taking place upon issuing of a keyword query. These tasks are presented in great detail in Chapter 3 and are the following:

[10]Measurements are presented for up to 32 users, because of lack in memory, which is due to the bad performance of the keyword index.

1. Derive the entities that syntactically match the query keywords. This task depends on the scalability of the keyword index employed in the keyword querying system (details can be found in Chapter 4).

2. Place the matched entities on the summary graph representing deriving the augmented graph, and then perform an exploration of this graph to compute the top-k subgraphs. This task operates on the schema of the underlying ontology and thus even for very large ontologies of thousands number of classes, it takes up to a small number of seconds. For hundreds of classes and properties, it takes some milliseconds.

3. Map the derived subgraphs to SPARQL queries and evaluate them on the RDF store. This task depends on the scalability of the RDF store and the type of indexes that have been created.

4. Transform the subgraphs and the entities derived from task 3 to a list of ranked entities. This task is very easy to be computed.

Geographic scalability

The keyword querying system has been designed to operate in a centralized environment. This means that the RDF data are stored in a single, central place (locally or remotely to the system). Likewise, the keyword search algorithm implemented has been designed to operate in a centralized environment too. The effort for extending the algorithm and adapting it to a distributed setting is foreseen to be long in time and high in development costs, due to significant architectural changes needed in the keyword system.

Domain scalability

The keyword querying system is adaptable to any domain and RDF datasets apart from the datasets employed in this dissertation. The keyword search algorithm operates on top of an RDF graph, and is not dependent on the domain knowledge that it covers.

6.2.2 Performance

Index Performance

The keyword querying system employs two indexes: keyword, and summary graph. Both of these indexes are constructed at the time of loading data in Sesame RDF store. Sesame RDF store provides the native index, which indexes RDF data in files on the file system and is enabled in every case. To measure the performance of the keyword and summary graph indexes, we compare their construction times, having as a basis the construction time of the native index only. We have measured their performance employing the following combinations of indexes:

Native. It uses the native index only.

Lucene. It uses the native and the keyword index, which is implemented by LuceneSail.

Inference. It uses the native and the summary graph index.

LuceneInf. It uses the native plus the keyword and summary graph indexes.

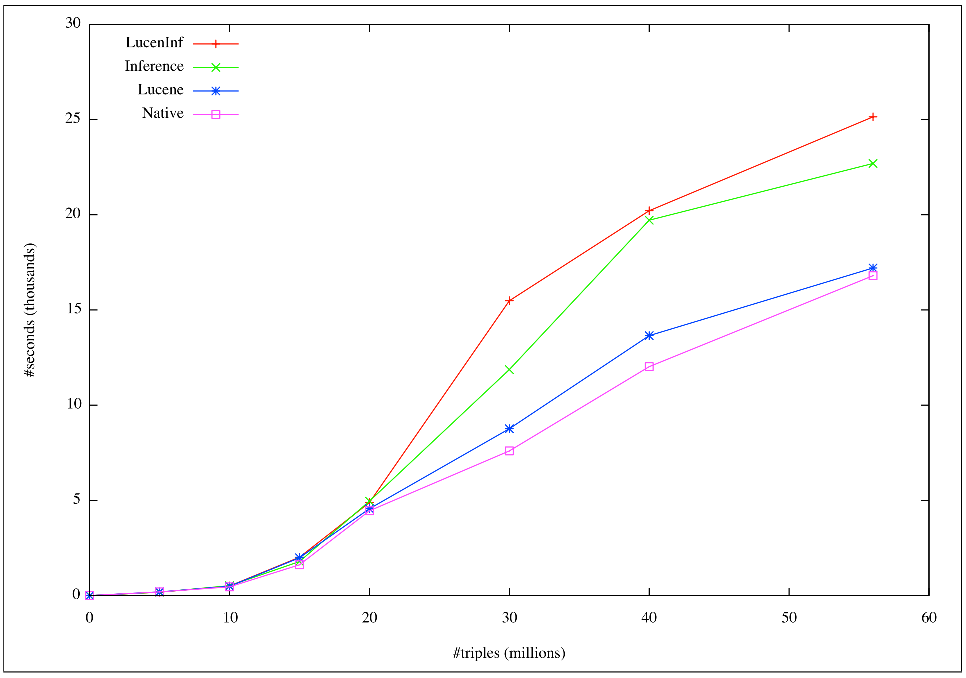

Figure 6.4: Comparison of the load times of different index types against various number of triples (DBLP dataset)

In all above cases, the time performance of building the indexes should be greater or equal to the time performance of the native index. The choice of measuring the performance of all these types of indexes is based on comparing their performance not only against the native index, but also against different combinations of them. It is noted that the space performance has not been evaluated, because native and keyword indexes are implemented by others, and summary graph space performance has been discussed in Chapters 3 and in Section 6.1.

Figure 6.4 compares how the load time of each index is affected by the number of triples for the DBLP dataset. We have provided such a figure only for the DBLP dataset, because it is a quite big dataset, and we could partition it in parts of 10 millions of triples in size. It is evident, that all indexes behave the same up to 20 millions of triples, and beyond this number they differentiate themselves. Lucene behaves very well, close to the performance of the Native index, while the Inference index shows a sharper differentiation. This is logical, because of the time spent in forward chaining. Last, LuceneInf exposes slightly worse behaviour than the Inference index because it has to build the keyword index in addition to forward chaining. Overall, the LuceneInf index, which is based in a great extend on the Sesame's forward chaining implementation, is worse up to a grade of 0.63, i.e.,

$$\frac{\text{Performance of Lucene}}{\text{Performance of LuceneInf}} = 0.63$$

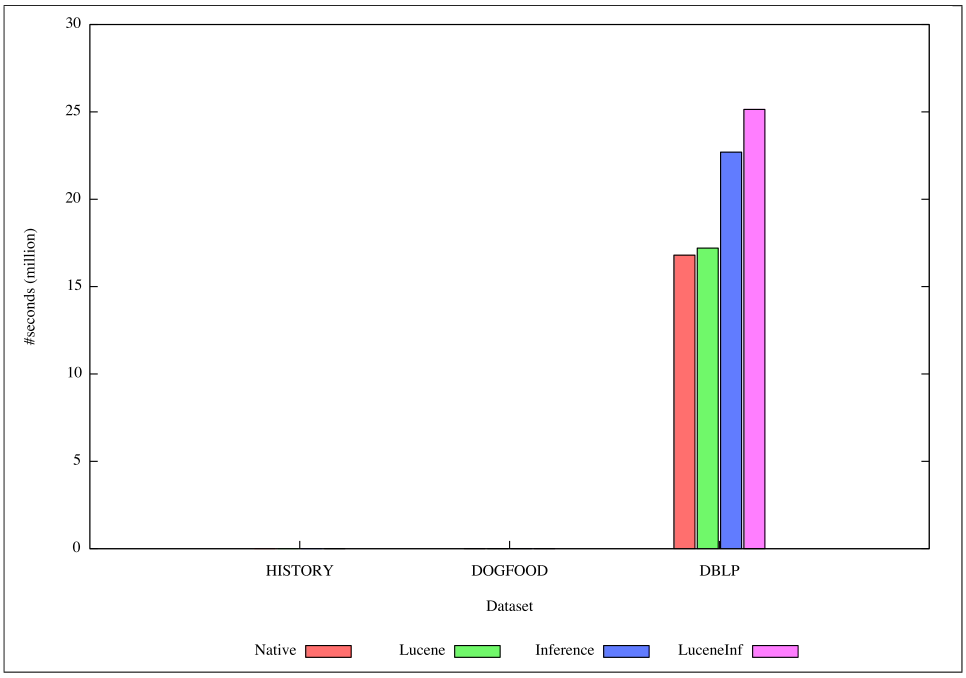

Figure 6.5: Construction times for each index type for each dataset.

The construction times for each index type and for each dataset are shown in Figure 6.5. Because, DBLP is a quite big dataset, its index time construction is much more higher than that of History and Web Dog Food. To have a more detailed view in these two datasets we have used them together in Figure 6.6.

Besides evaluating the time needed for the index construction, we measure, also, the time needed to load the summary graph, when the keyword querying system is set into operation. Figure 6.7 shows such times for our three datasets.

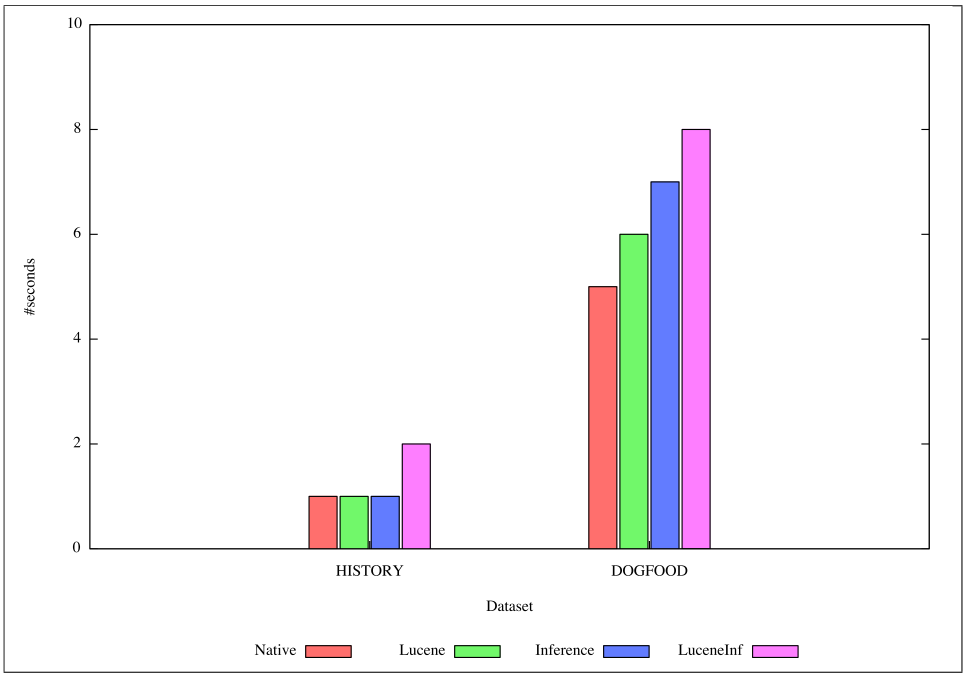

Figure 6.6: Construction times for each index type for History and Semantic Web Dog Food datasets.

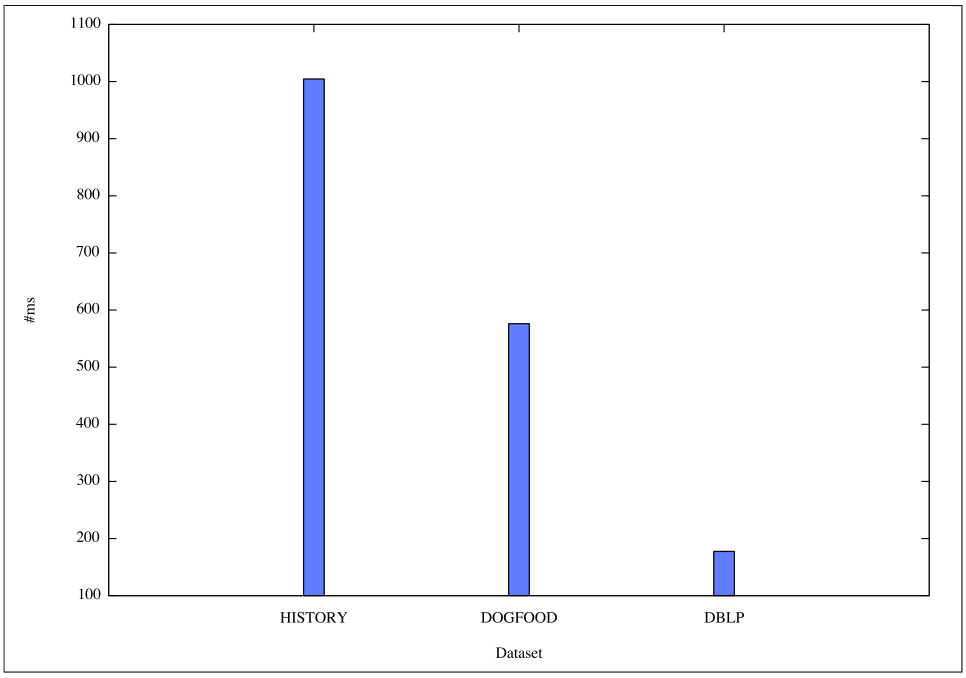

Figure 6.7: Load times of the summary graph index for each dataset.

Query time performance

In this subsection, the query time performance is measured, that is, the time that the keyword querying system takes to answer a query. First, query time performance is measured along different scoring functions, i.e., the ones mentioned in Subsection 3.3.2. Then, we compare the time that the query system spends on a query task for a specific query, and last, we measure the contribution of each query processing task to the overall query processing time.

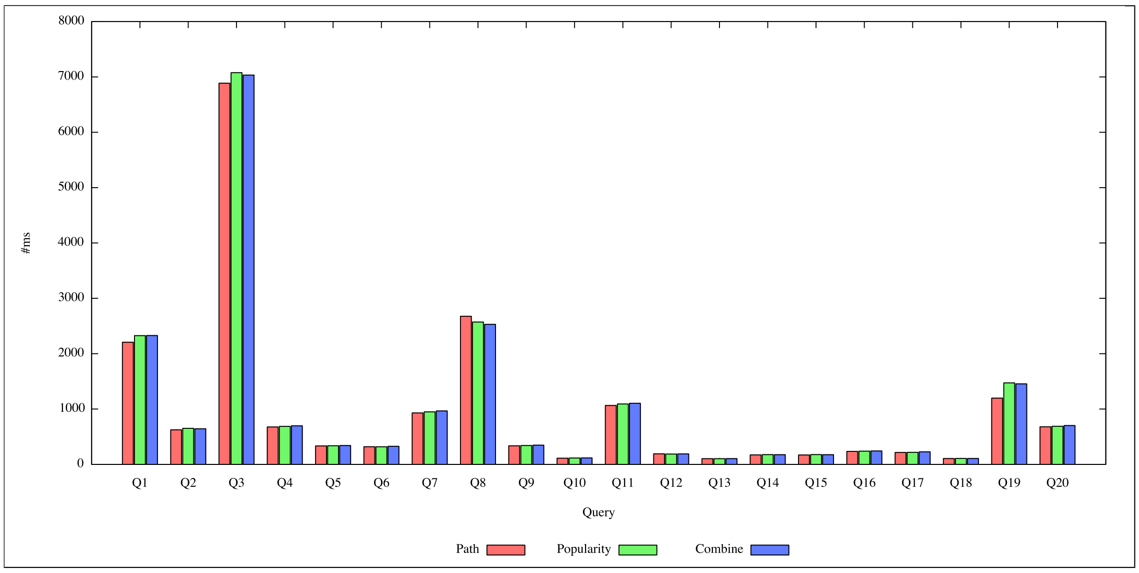

Figure 6.8: Query time performance for different scoring functions for the History ontology.

For the History ontology, the query time performance for different scoring functions is shown in Figure 6.8. It is evident that all three scoring functions expose similar behaviour. Next, Figure 6.9 shows the performance of the query processing tasks for each query and in combination to the overall query processing performance. We observe that the graph exploration task is the one which presided over the overall performance of query processing. This is justified by the characteristics that the History ontology exhibits (see Subsection 6.1.1): it is a very complex ontology in terms of schema, while its triples are not of significant number.

Last, Figure 6.10 shows the percentage contribution of each query task to the average query processing time. This percentage is the average contribution over all queries and summarizes better Figure 6.9.

For the Semantic Web Dog Food dataset, the query time performance for different scoring functions is shown in Figure 6.11. It is evident that all three scoring functions expose similar behaviour, except for the case of the 7th query in which the popularity score function exhibits poor performance. We claim that this result is an outlier, because

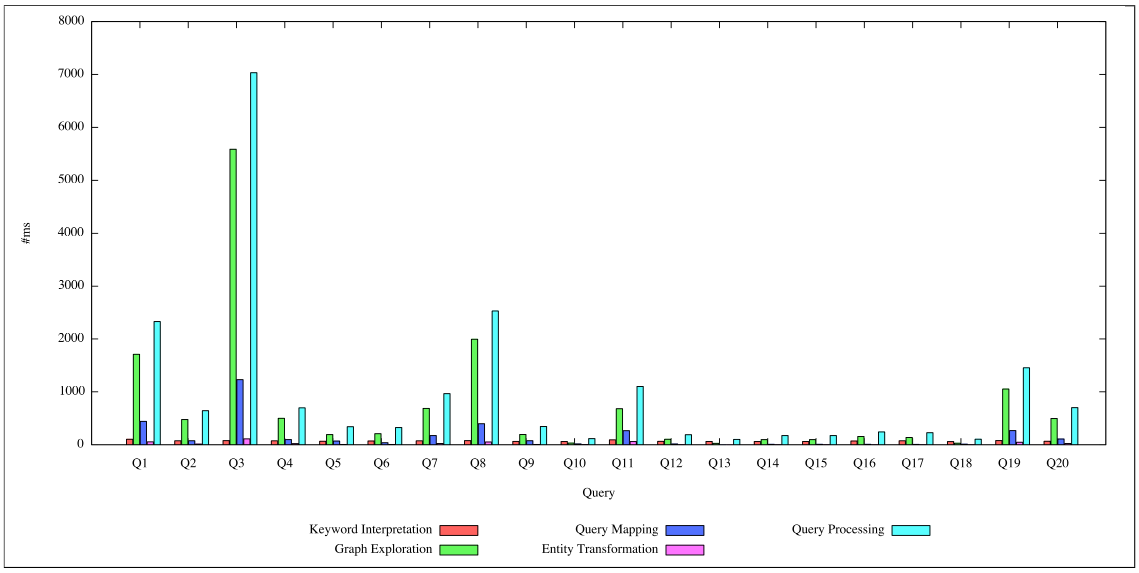

Figure 6.9: Performance of query tasks for different queries for the History ontology.

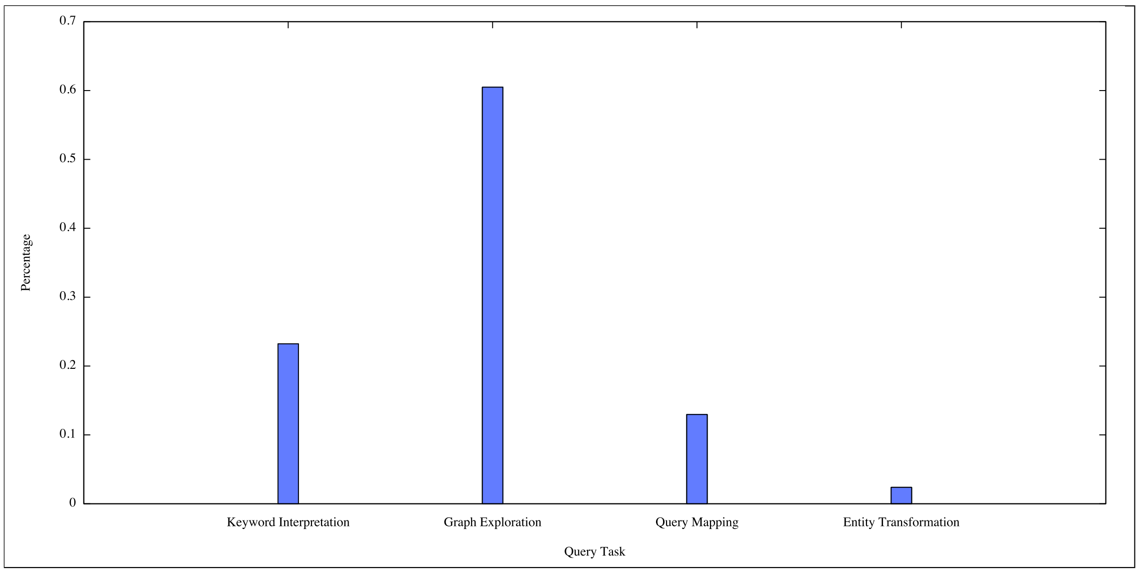

Figure 6.10: Percentage of query tasks to overall query processing for the History ontology.

in practice, popularity (the same applies for the case of the path scoring function) must perform at most as bad as the combine scoring function; the last one is the combination of the path, keyword, and popularity scoring functions.

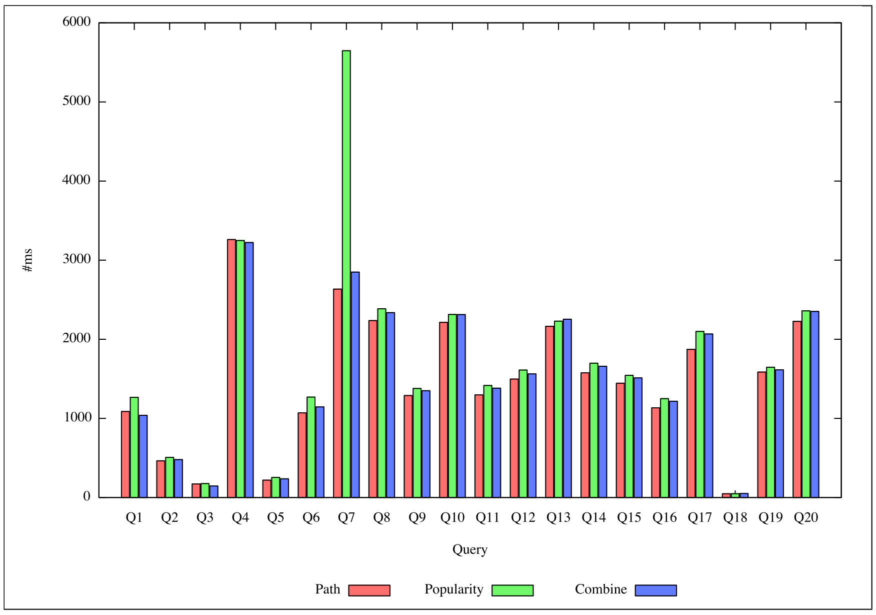

Figure 6.11: Query time performance for different scoring functions for the Semantic Web Dog Food dataset.

Next, Figure 6.12 shows the performance of the query processing tasks for each query and in combination to the overall query processing performance. The main observation here is that the task of keyword interpretation competes that of graph exploration, and there are cases in which it behaves worse. This is justified by the characteristics of the Semantic Web Dog Food dataset, as mentioned in Subsection 6.1.2; indeed, the dataset contains many more triples than those of the History ontology, and the summary graph is of medium size, but much more simpler than that of the History ontology.

Last, Figure 6.13 shows the percentage contribution of each query task to the average query processing time. This percentage is the average contribution over all queries. Query mapping and entity transformation preserve their low contribution to the overall query processing time. In contrast, the contribution of the keyword interpretation's task has become greater, while that of the graph exploration's has become lower. This is in line with the previous observations.

For the DBLP dataset, the query time performance for different scoring functions is shown in Figure 6.14. It is evident that all three scoring functions expose similar behaviour.

Next, Figure 6.15 shows the performance of the query processing tasks for each query and in combination to the overall query processing performance. The main observation here is that the task of keyword interpretation is the only one responsible for the query

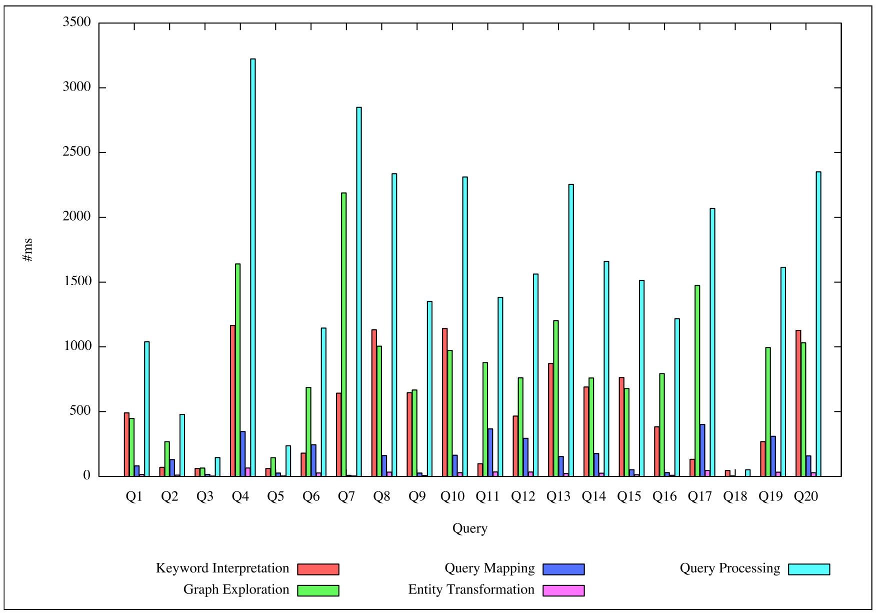

Figure 6.12: Performance of query tasks for different queries for the Semantic Web Dog Food dataset.

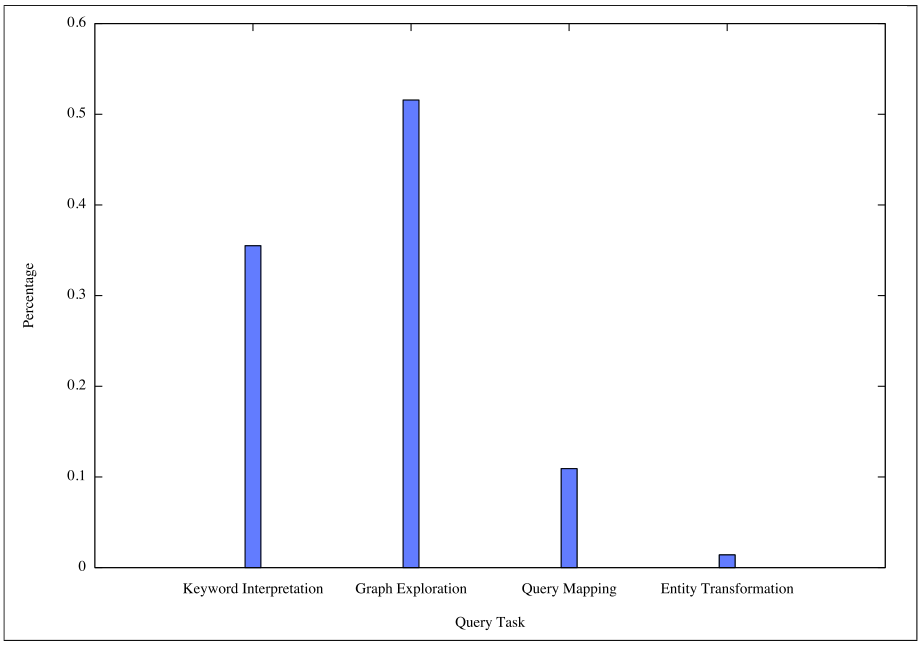

Figure 6.13: Percentage of query tasks to overall query processing for the Semantic Web Dog Food dataset.

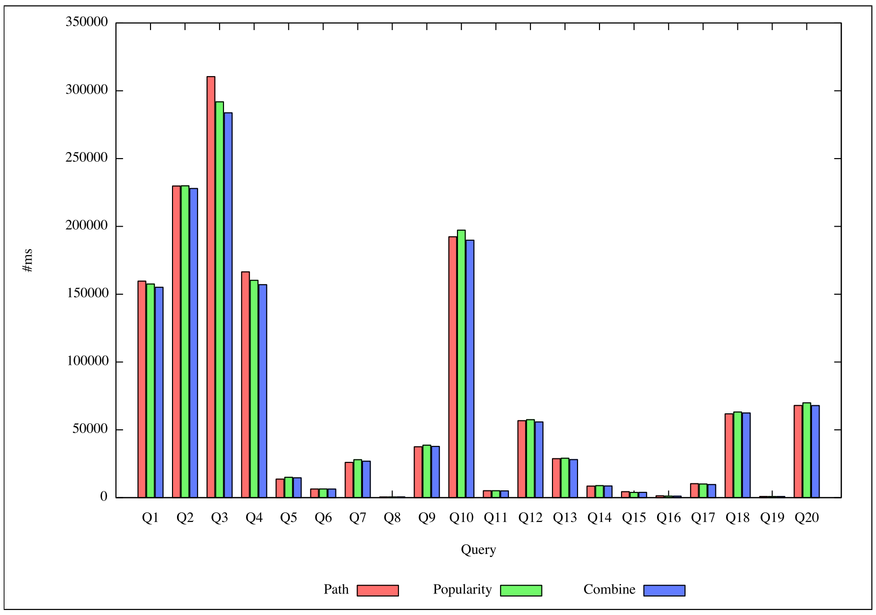

Figure 6.14: Query time performance for different scoring functions for the DBLP dataset.

time performance of the overall query processing. This is justified by the characteristics of the DBLP dataset, as mentioned in Subsection 6.1.3; indeed, the dataset contains tremendously many more triples than those of the History ontology and the Semantic Web Dog Food dataset, while the summary graph is extremely simple, so the task of graph exploration is trivial.

Last, Figure 6.16 shows the percentage contribution of each query task to the average query processing time. This percentage is the average contribution over all queries. Query mapping and entity transformation preserve their low contribution to the overall query processing time, but in contrast to previous results, the task of graph exploration is as easy as that of query mapping. In contrast, the contribution of the keyword interpretation's task has become much greater and takes almost all the time spent in query processing.

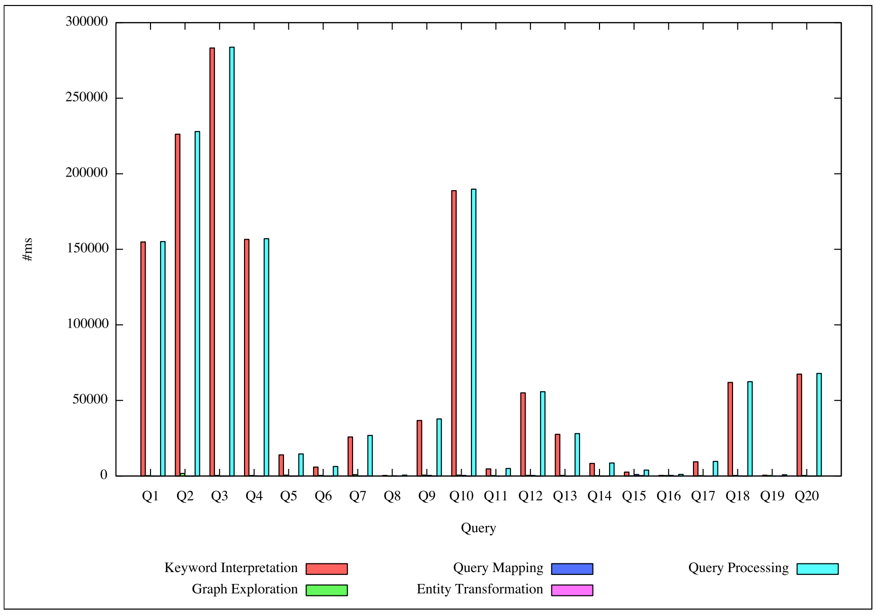

Figure 6.15: Performance of query tasks for different queries for the DBLP dataset.

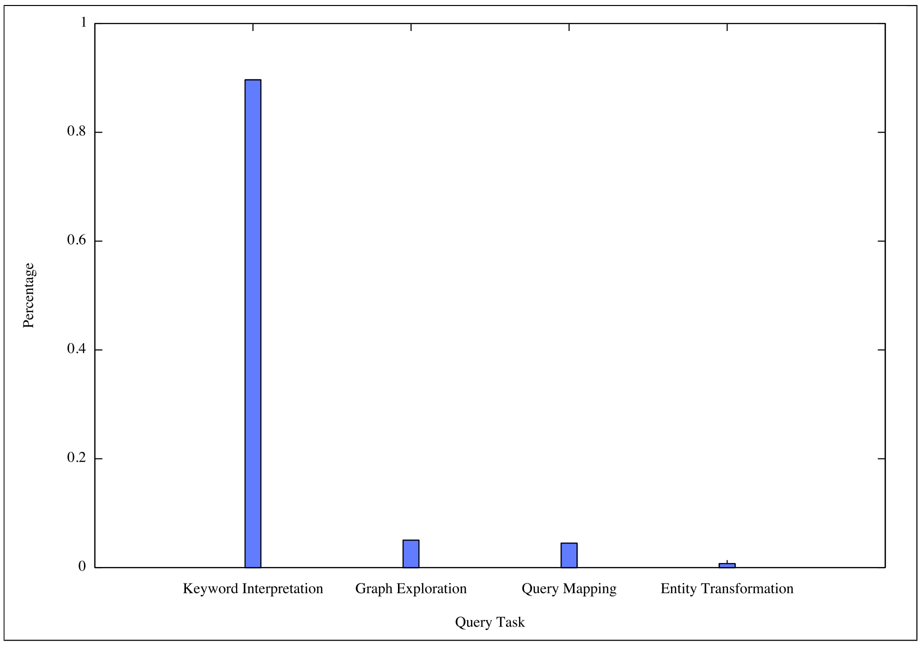

Figure 6.16: Percentage of query tasks to overall query processing for the DBLP dataset.

6.3 Measuring Effectiveness

The effectiveness of the keyword system implemented in the context of this dissertation is measured using both measures for unranked retrieval sets and ranked retrieval results. For the first case, the two most frequent and basic measures for information retrieval effectiveness are *precision* and *recall*. Apart from these two, the *F measure* is employed as well. For the second case, the chosen measure is the *normalized discounted cumulative gain* (NDCG) [MRS, CMS09]. In the following, definitions for these measures are given, as well as justifications for their use.

Precision (P) is the fraction of retrieved documents[11] that are relevant.

$$P = \frac{\#(\text{relevant items retrieved})}{\#(\text{retrieved items})}$$

Recall (R) is the fraction of relevant documents that are retrieved.

$$R = \frac{\#(\text{relevant items retrieved})}{\#(\text{relevant items})}$$

These two measures are very useful in combination, because one may be more important than the other in many circumstances. For example, typical web servers would prefer precision to recall on account of examining only the first page of results (high precision). In contrast, a PC user searching for files in the file system would prefer recall to precision, on account of being interested in finding all desired files. In general, someone would like to get some amount of recall while tolerating only a certain percentage of irrelevant results.

A single measure that trades off precision versus recall is the F measure, which is the weighted harmonic mean of precision and recall:

$$F = \frac{1}{\alpha\frac{1}{P} + (1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P + R} \quad \text{, where} \quad \beta^2 = \frac{1-\alpha}{\alpha} \quad \text{and} \quad \alpha \in [0,1]$$

The default *balanced F measure* equally weights precision and recall, which means making $\alpha = 1/2$ or $\beta = 1$. It is commonly written as F_1.

$$F_1 = \frac{2PR}{P+R}$$

To evaluate the effectiveness of the ranking mechanism of the keyword search system the $NDCG$ measure is employed. This measure is designed for situations of nonbinary notions of relevance (i.e., in the interval $[0,1]$), in contrast to the binary notion of relevance (relevant/nonrelevant) on which the precision, recall, and F measures are based. $NDCG$ is a popular measure for evaluating web search engines and related tasks. It makes the following assumptions:

[11]In this dissertation, the notion of a document is casted to that of an entity. In the following, these terms shall be used interchangeably.

1. Highly relevant documents are more useful than marginally relevant document.
2. The lower the ranked position of a relevant document, the less useful it is for the user, since it is less likely to be examined.

$NDCG$ is evaluated over some number k of top search results. The property of $NDCG$ is that accumulates the scores of the first k top search results penalizing each one according to its rank position. Most of the times, this penalty is expressed by the factor $\frac{1}{log_2(1+r)}$, where r is the rank position. If the relevance of a document i is expressed as rel_i, then the DCG measure at k is given by the following formula:

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(1+i)}$$

Another, equivalent expression to the above is the following:

$$DCG_k = rel_1 + \sum_{i=2}^{k} \frac{rel_i}{log_2(i)}$$

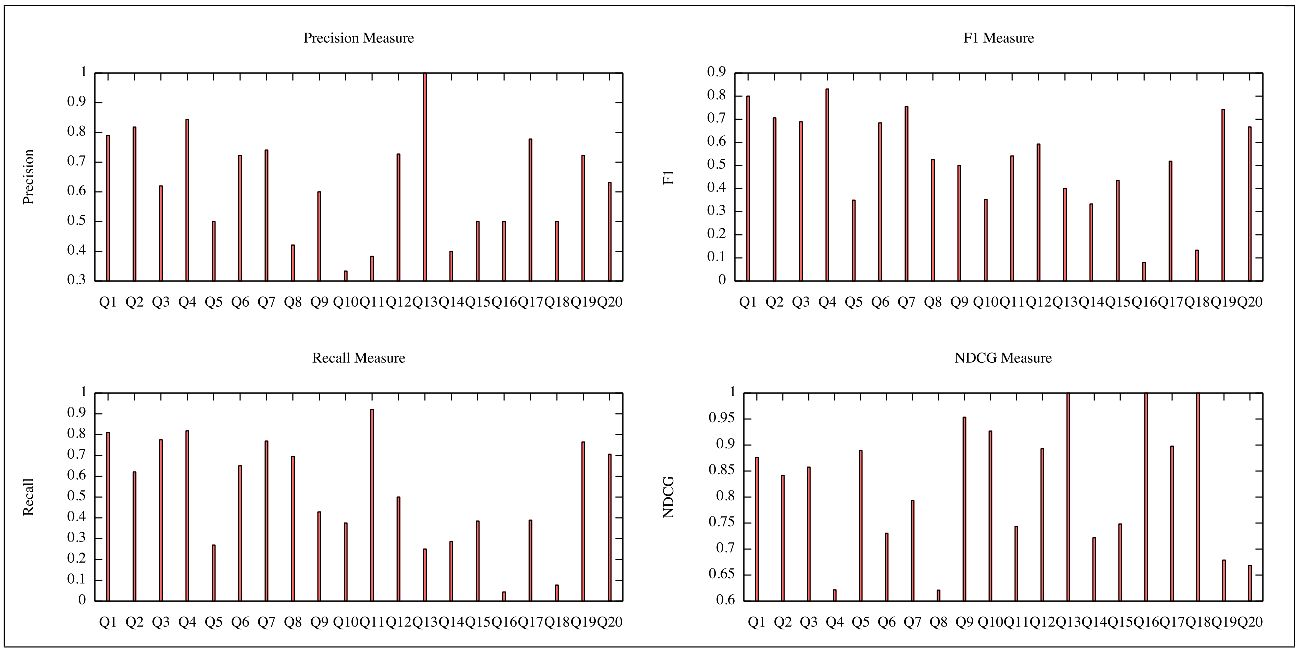

Figure 6.17: Effectiveness evaluation results for the History ontology.

Comparing a search engine's effectiveness from one query to the next cannot be consistently achieved using DCG alone due to the various sizes of the result lists. DCG can be normalized across queries by sorting documents of a result list by relevance, producing an ideal DCG at position k. This is named as $IDCG$. For a query, the normalized DCG or $NDCG$ at k is computed as:

$$NDCG_k = \frac{DCG_k}{IDCG_k}$$

Then, the $NDCG$ values for all queries can be averaged to obtain a measure of the average effectiveness of a search engine's ranking algorithm. Note that in a perfect ranking algorithm, the DCG will be the same as the $IDCG$, producing an $NDCG$ of 1.0. All $NDCG$ calculations are then relative values on the interval $[0, 1]$ and so are cross-query comparable.

The evaluation results concerning effectiveness are depicted in Figure 6.17.

6.4 Conclusions

This chapter discussed the methodology followed for evaluating the developed keyword querying system and commented on its outcome. In particular, the evaluation was conducted on three different datasets, which exhibit different quantitative and qualitative characteristics in terms of, namely, the History ontology — developed in the context of the Papyrus project —, the Semantic Web Dog Food dataset, and the DBLP dataset. The keyword querying system evaluated both in terms of efficiency and effectiveness.

Regarding efficiency, the dimensions over which the keyword querying system was evaluated are scalability and performance. In terms of scalability, load, service, data, geographic, and domain scalability were considered. In terms of performance, the construction time of keyword/graph indexes, the load time of graph index, and the query response time were examined. Overall, the keyword querying system is scalable in terms of load, and exhibits a modest performance. The evaluation outcome identified two query tasks that have to be improved: keyword interpretation and graph exploration. For the first, the employed LuceneSail index component has to be substituted with another one. For the latter, the graph exploration algorithm has to be improved so as to allow for subgraph sharing between different keyword interpretations avoiding the generation of duplicates. In general, the employed Sesame RDF store could be substituted with another one, which would allow for using a database backend for storing the RDF data. Currently, the storage backend is the Sesame's Native repository, which stores and indexes RDF data on top of the filesystem. While it supports a database backend in general, in particular, it does not provide inference support, which is crucial for building the graph index.

Regarding effectiveness, the employed metrics were precision/recall, the balanced F measure, and the normalized discounted cumulative gain. Judging by the outcome, the keyword querying system achieves a fair to good performance.

Chapter 7

Epilogue

This dissertation studied related work on keyword-based search on structured and semi-structured data in the last decade, and compared it with that developed in the field of IR. It discussed the state-of-the-art on evaluating methodologies for systems providing keyword-search functionality and concluded that it is far from good, without providing any solid and rigid methodology, as opposed to the respective evaluation process developed in the field of IR. Fortunately, this area has already started to attract the interest of the research community leading to initiatives, such as INEX, and other interesting and promising evaluation methodologies based on large knowledge bases, such as Wikipedia. Furthermore, it discussed the new challenges that the research community around keyword-based search should address. Towards understanding and fulfilling future requirements, a keyword-based system was designed, implemented, and extensively evaluated according to current standard methodologies. The dimensions of the evaluation process were efficiency and effectiveness.

The proposed system is based and improves the work in [TWRC09]. The system employs a keyword-based query language extended with temporal constructs in the form of all Allen's thirteen temporal relations: before, after, overlaps, etc. The system's data model is able to capture indefinite temporal information defining the time-limits over which a certain fact is valid. Indefinite information concerning a time point can be given as an interval in which the point must lie. In the case of indefinite information about time intervals, the starting and ending time points of an interval are defined in the same way.

Next, we discuss directions of our future work, and elaborate on ideas for improving our system.

7.1 Future Work

Our future work around keyword-based search on RDF databases focuses on the following aspects:

Keyword index. Implement or reuse another keyword index for RDF literals to improve the performance of the keyword interpretation task. Based on the evaluation results of the DBLP dataset, it is clear that the LuceneSail implementation of a keyword index does not perform well for datasets containing a large amount of RDF literals. As another shortcoming of the LuceneSail implementation is that it does not support the solution sequence modifier, $LIMIT$, supported in the specification of SPARQL. The $LIMIT$ modifier could be possibly employed to restrict the top-k' keyword interpretations for a specific keyword.

A possible solution on this could be the use of BigOWLIM repository[1]. BigOWLIM is a high-performance semantic repository, implemented in Java and packaged as a Storage and Inference Layer (SAIL) for the Sesame RDF database. Aside from other special features, it provides full-text search functionality integrating the Lucene index, and allows powerful hybrid queries to be expressed inside SPARQL queries. Because of the fact that BigOWLIM is packaged as SAIL, our implementation is fully-compatible and adaptable.

Graph Exploration. The graph exploration algorithm could be further improved not allowing multiple explorations of the same subgraphs for a specific query. Currently, multiple keyword interpretations might produce the same subgraphs. This occurs when two or more keyword interpretations are, for example, instances of the same classes. Omitting these keyword elements, the other part of the exploration, which is conducted on the schema graph of the RDF data, is totally the same.

Another direction for graph exploration would be to explore the graph, if possible, by subgraphs, producing subgraphs in decreasing score order. Currently, graph exploration is conducted based on the concept of a path, which leads to produced subgraphs in unordered scores. This makes top-k computation a difficult task.

Schema-agnostic data model. The employed data model in our work is a schema-aware data model. For the RDF data of an organization, the RDF data exported from a DBMS, or for RDF data that adhere to a specific schema in general, a schema-aware data model is the proper one. Conversely, the most appropriate data model for future demands is the schema-agnostic; the next challenge is the integration and exploitation of semi-structured data and linked data, stored in different formats (RDF, XML, tables and records, etc.) and data sources (as ontologies, knowledge bases, databases, etc.).

Taking into account the evaluation of our work and the experience gained, we aim at developing a schema-agnostic or a hybrid data model to satisfy future demands.

Keyword-search algorithm modifications. In each current form, the algorithm doing keyword-search, for a given keyword query, is invoked k' times, that is, as many times as the number of keyword interpretations is. One alternative to this would be to place all interpretations in the summary graph, and then invoke the algorithm

[1] `http://www.ontotext.com/owlim/version-map.html`

just once. Then, a connecting element would be a graph element which is on a path of at least one keyword interpretation from each keyword of the query. The trade-off of this approach is that the graph exploration time and space may be increased dramatically, because of keeping all possible paths for each keyword interpretation. Certainly, experiments have to be conducted to decide what is the most efficient approach.

Complexity analysis. Apart from evaluating our system through experimentation, it is crucial to evaluate it theoretically also. The employed keyword search algorithm has to be analyzed in terms of time and space complexity.

Identification of keyword phrases in the query. Frequently enough, keyword phrases, i.e., keywords that should be found together as given in the query, are not enclosed in quotes. This results in the increase of keyword interpretations, and also in the ignorance of the underlying semantic linkage of such keywords. Consider for example the keyword query "artificial intelligence". The keyword interpretation task would derive all interpretations for the keyword "artificial", which might be an artificial lake, barrier, flower, language, respiration, intelligence, satellite, etc., and then, all interpretations for the keyword "intelligence", such as intelligence service, latest intelligence, intelligence officer, intelligence quotient, artificial intelligence, etc. These are the good case interpretations, because they comprise a keyword phrase with the two keywords being frequently used together. In addition, the keyword interpretation task would derive other interpretations in which keyword the "artificial" would be present. Certainly, in the context of a keyword phrase, such as the above, it is very unlikely that such interpretations would be of any value. In this respect, a statistical model is needed, such as the n-gram model employed in the field of IR, which recognizes such keyword phrases and group them as one. The identification of keyword phrases in the query certainly improves retrieval effectiveness.

Appendix A

Queries used in evaluation

A.1 History ontology

The queries posed to the History ontology dataset and their respective answers are shown in Table A.1 and A.2 respectively. Both sets have been compiled by expert historians. As far as the answers are concerned, they reflect the entities of the History ontology, which are relevant to each keyword query.

Table A.1: Queries for the History ontology dataset

Query ID	Keyword Query
QH1	cloning ethics
QH2	cloning “public opinion”
QH3	cloning controversy
QH4	cloning “research institutions”
QH5	“pharmaceutical industry” discovery
QH6	“stem cell” “public opinion"
QH7	“stem cell” ethics
QH8	“stem cell” controversy
QH9	“gene therapy” discovery
QH10	“genetically modified organisms” artifacts
QH11	biotechnology “academic disciplines”
QH12	“wind power” “climate change”
QH13	windmill “public opinion”
QH14	“climate change” “public opinion”
QH15	“climate change” “public policy”
QH16	“pollution” “public policy”
QH17	recycling “wind power”
QH18	“ecological disasters” “public opinion”
QH19	“renewable energy” industry

Continued in next page

Table A.1 – Queries for the History ontology dataset (cont'd)

Query ID	Keyword Query
QH20	biofuels controversy

Table A.2: Expected answers for the History ontology queries

Query ID	Expected Entities
QH1	Cloned organism
	Cloning
	Daniel Callahan
	Dolly the Sheep
	Grenada Genetics
	Hastings Center
	Human Cloning
	Hwang Woo-suk
	Ian Wilmut
	Institute of Society Ethics and Life Science
	Keith H. S. Campbell
	Organism cloning
	Panayiotis Zavos
	Peter Hoppe
QH2	Alta Genetics
	Clone Rights United Front
	Injaz
	Celera Corporation
	Cellular cloning
	Cloned organism
	Cloning
	Cloning of Dolly
	Dolly the Sheep
	First cat cloning
	First cloning of a frog
	First cloning of mice
	First cow cloning
	First gaur ox cloning
	First pig cloning
	First rabbit cloning
	Human Cloning
	Hwang Woo-suk
	Keith H. S. Campbell

Continued in next page

Table A.2 – Expected answers for the History ontology queries (cont'd)

Query ID	Expected Entities
	Organism cloning
	Pandora's box metaphor
	Playing God metaphor
	Professor Hwang
	Tetra, the monkey
QH3	Clone Rights United Front
	Cloned organism
	Cloning
	Cloning Controversy
	Cloning of Dolly
	Dolly the Sheep
	Grenada Genetics
	Human Cloning
	Hwang Woo-suk
	Ian Wilmut
	Injaz
	Keith H. S. Campbell
	Molecular cloning
	Organism cloning
	Panayiotis Zavos
	PPL Therapeutics
	Professor Hwang
	Robert Briggs
	Tetra, the monkey
	First cat cloning
	First cloning of a frog
	First cloning of mice
	First cow cloning
	First gaur ox cloning
	First pig cloning
	First rabbit cloning
	Human Cloning
	Padora's box metaphor
	Risk assessment
QH4	Celera Corporation
	Cloning
	Cloning of Dolly
	Dolly the Sheep
	First cat cloning

Continued in next page

Table A.2 – Expected answers for the History ontology queries (cont'd)

Query ID	**Expected Entities**
	First cloning of a frog First cloning of mice First cow cloning First gaur ox cloning First pig cloning First rabbit cloning Grenada Genetics Human Cloning Injaz J. Craig Venter Institute Organism Cloning PPL Therapeutics Tetra, the monkey
QH5	Biochemistry Biogen Idec, Inc. Biological engineering Biotechnology Biotechnology industry Cancer Dechema Disease Elmer Gaden Ernst Chain Human Disease J. Craig Venter Institute Mad cow disease Molecular biology Novartis International AG Pharmaceutical industry Race Roslin institute SARS Treatment
QH6	Hashmi family Whitaker family Hwang Woo-suk Stem cell Hans Adolf Eduard Driesch Discovery of human embryonic stem cell

Continued in next page

Table A.2 – Expected answers for the History ontology queries (cont'd)

Query ID	Expected Entities
	Pandora's box metaphor Playing God metaphor Chimera - monster metaphor
QH7	Discovery of human embryonic stem cell Hashmi family Whitaker family Embryologist Ian Wilmut Embryonic stem cell Hwang Woo-suk Institute of Society Ethics and Life Science Stem cell Hastings Center
QH8	Hashmi family Whitaker family Embryologist Ian Wilmut Embryonic stem cell Hwang Woo-suk Pandora's box metaphor Playing God metaphor Stem cell Stem cell controversy Risk assessment
QH9	Alzheimer's disease Blue gene metaphor Criminal gene metaphor Embryologist Ian Wilmut Engineering - machine metaphor Fatty gene metaphor Gene Gene shifters metaphor Gene therapy Genetic engineering George Beadle James A. Shapiro Ian Wilmut
QH10	Chimera - monster metaphor Designer metaphor Gene Genetic modification

Continued in next page

Table A.2 – Expected answers for the History ontology queries (cont'd)

Query ID	Expected Entities
	Genetically modified organism
	Genetics
QH11	Biochemistry
	Bioinformatics
	Biological engineering
	Biology
	Biology-based technology
	Biomedical engineer
	Biomedical technology
	Biophysics
	Biotechnology
	Molecular biology
QH12	Atmosphere of Earth
	Climate change
	Renewable energy
	Renewable energy source
	Wind farm
	Wind farm controversy
	Wind power
	Wind power industry
	Windmill
QH13	Renewable energy
	Turbine
	Wind farm
	Wind farm controversy
	Wind power
	Wind power industry
	Windmill
QH14	Amazon rainforest
	Arctic Shrinkage
	Climate change
	Environmental degradation
	Environmental movement
	Environmentalism
	Glacial Warming
	Global Warming
	Global warming controversy
	Green Party
	Greenhouse effect

Continued in next page

Table A.2 – Expected answers for the History ontology queries (cont'd)

Query ID	Expected Entities
QH15	Air pollution Amazon rainforest Arctic Shrinkage Batteries recycling Climate change Deforestation Environmental degradation Global Warming Greenhouse effect Milan Climate Change Conference
QH16	Air pollution Batteries recycling Biomedical waste Chemical pollution Electronics recycling Environmental pollution Environmentalism Ferrous metals recycling Glass recycling Industrial effluents Industrial waste Non-ferrous metals recycling Oil pollution Paper recycling Plastics recycling Pollution Radioactive pollution Recycling Textiles recycling Timber recycling Thermal pollution Water pollution
QH17	Batteries recycling Electronics recycling Environmentalism Ferrous metals recycling Glass recycling Non-ferrous metals recycling Paper recycling

Continued in next page

Table A.2 – Expected answers for the History ontology queries (cont'd)

Query ID	Expected Entities
	Plastics recycling Pollution Recycling Textiles recycling Timber recycling
QH18	Arctic Shrinkage Chemical pollution Chernobyl Nuclear Accident Ecological disaster Ecosystem Environmental degradation Environmental pollution Global Warming Greenhouse effect Industrial waste Radioactive pollution Oil pollution
QH19	Biofuel Biomass Electric industry Electric power industry Electric utility industry Geothemal power controversy in US, Hawaii Geothermal power Hydroelectric plant Photovoltaic panel Photovoltaic power station Power station Renewable energy Solar power Solar power station Water wheel Wave farm Wave power Wind farm Wind power industry
QH20	Bioalcohol Biodiesel Bioethers

Continued in next page

Table A.2 – Expected answers for the History ontology queries (cont'd)

Query ID	Expected Entities
	Biofuel
	Biofuel controversy
	Biogas
	Biomass
	First generation biofuel
	Second generation biofuel
	Third generation biofuel
	Fourth generation biofuel
	Green Party
	Environmental movement
	Environmentalism

A.2 Semantic Web Dog Food

The queries posed to the Semantic Web Dog Food dataset are shown in Table A.3. These queries have been extracted from the query logs of the Semantic Web Dog Food web site and reflect a part of the top queries posed to this site for the year of 2009. The author of this dissertation is thankful to Knud Möller, who provided the logs. Expected answers to these queries are not provided, because of lack in users posing such queries.

Table A.3: Queries for the Semantic Web Dog Food dataset

Query ID	Keyword Query
QH1	ontology engineering
QH2	table top
QH3	Olegas Vasilecas
QH4	semantic web service discovery
QH5	koubarakis idreos
QH6	design semantics
QH7	Test Collection Construction for QA system using Ontology Instance Triplet
QH8	semantic web algorithms
QH9	ontology data integration
QH10	semantic web and models
QH11	collective intelligence
QH12	ontology editor
QH13	interlinking Open Data on the Web
QH14	semantic wiki
QH15	semantic information retrieval

Continued in next page

Table A.3 – Queries for the Semantic Web Dog Food dataset (cont'd)

Query ID	Keyword Query
QH16	linked data + architecture
QH17	automatic name disambiguation
QH18	Positioning
QH19	NLP NLP OR social OR networks
QH20	crawling the semantic web

A.3 DBLP

The queries posed to the DBLP dataset are shown in Table A.4. The first fifteen queries have been extracted from commonly used queries in related work and 5 PhD users involved in the author's institution, while the other five have been compiled by the author himself. Expected answers to these queries are not provided, because of lack in users posing such queries.

Table A.4: Queries for the DBLP dataset

Query ID	Keyword Query
QH1	jagadish optimization
QH2	jeff dynamic optimal
QH3	abiteboul adaptive algorithm
QH4	hector jagadish performance improving
QH5	krishnamurthy parametric "query optimization"
QH6	naughton dewitt "query processing"
QH7	divesh jignesh jagadish timber querying xml
QH8	idreos koubarakis tryfonopoulos
QH9	semantic tagging aaai
QH10	multiple instance learning
QH11	gunopulos subspace
QH12	first-order progression
QH13	gunopulos papapetrou matching
QH14	zaniolo ICDE stream
QH15	mohan
QH16	halevy abiteboul
QH17	papadimitriou topological
QH18	k-server koutsoupias
QH19	ioannidis "parametric query optimization"
QH20	r-tree sellis

References

[AAH$^+$10] Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors. *The Semantic Web: Research and Applications, 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part II*, volume 6089 of *Lecture Notes in Computer Science*. Springer, 2010.

[ABK$^+$07] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary G. Ives. Dbpedia: A nucleus for a web of open data. In Aberer et al. [ACN$^+$07], pages 722–735.

[ACD02] Sanjay Agrawal, Surajit Chaudhuri, and Gautam Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE* [DBL02], pages 5–16.

[ACD$^+$03] B. Aditya, Soumen Chakrabarti, Rushi Desai, Arvind Hulgeri, Hrishikesh Karambelkar, Rupesh Nasre, Parag, and S. Sudarshan. User interaction in the banks system. In Umeshwar Dayal, Krithi Ramamritham, and T. M. Vijayaraman, editors, *ICDE*, pages 786–788. IEEE Computer Society, 2003.

[ACN$^+$07] Karl Aberer, Key-Sun Choi, Natasha Fridman Noy, Dean Allemang, Kyung-Il Lee, Lyndon J. B. Nixon, Jennifer Golbeck, Peter Mika, Diana Maynard, Riichiro Mizoguchi, Guus Schreiber, and Philippe Cudré-Mauroux, editors. *The Semantic Web, 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007*, volume 4825 of *Lecture Notes in Computer Science*. Springer, 2007.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[All81] James F. Allen. An interval-based representation of temporal knowledge. In Patrick J. Hayes, editor, *IJCAI*, pages 221–226. William Kaufmann, 1981.

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[BCT07] K. Bollacker, R. Cook, and P. Tufts. Freebase: A shared database of structured general human knowledge. In *Proceedings of the national conference on Artificial Intelligence*, volume 22, page 1962. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[BEP$^+$08] Kurt D. Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In Wang [Wan08], pages 1247–1250.

[BG00] D. Brickley and R.V. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Technical report, W3C Recommendation, 2000.

[BHN$^+$02] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE* [DBL02], pages 431–440.

[BHP04] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: Authority-based keyword search in databases. In Mario A. Nascimento, M. Tamer Özsu, Donald Kossmann, Renée J. Miller, José A. Blakeley, and K. Bernhard Schiefer, editors, *VLDB*, pages 564–575. Morgan Kaufmann, 2004.

[BKTV10] S. Bykau, N. Kiyavitskaya, C. Tsinaraki, and Y. Velegrakis. Bridging the Gap between Heterogeneous and Semantically Diverse Content of Different Disciplines. *FlexDBIST*, August 2010.

[Bud91] R. Bud. Biotechnology in the twentieth century. *Social studies of science*, 21(3):415–457, 1991.

[BW99] C. Buckley and J. Walz. Smart in trec 8. In *Proc. of TREC*, 1999.

[BW07] Holger Bast and Ingmar Weber. The completesearch engine: Interactive, efficient, and towards ir& db integration. In *CIDR*, pages 88–95. www.crdrdb.org, 2007.

[CMS09] B. Croft, D. Metzler, and T. Strohman. Search engines: Information retrieval in practice. 2009.

[Coh98] William W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In Laura M. Haas and Ashutosh Tiwary, editors, *SIGMOD Conference*, pages 201–212. ACM Press, 1998.

[Coh00] William W. Cohen. Whirl: A word-based information representation language. *Artif. Intell.*, 118(1-2):163–196, 2000.

[COZ07] Chee Yong Chan, Beng Chin Ooi, and Aoying Zhou, editors. *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*. ACM, 2007.

[CRW05] Surajit Chaudhuri, Raghu Ramakrishnan, and Gerhard Weikum. Integrating db and ir technologies: What is the sound of one hand clapping? In *CIDR*, pages 1–12, 2005.

[DBL02] *Proceedings of the 18th International Conference on Data Engineering, 26 February - 1 March 2002, San Jose, CA*. IEEE Computer Society, 2002.

[DBL09] *Proceedings of the 25th International Conference on Data Engineering, ICDE 2009, March 29 2009 - April 2 2009, Shanghai, China*. IEEE, 2009.

[DCG$^+$08] Pedro DeRose, Xiaoyong Chai, Byron J. Gao, Warren Shen, AnHai Doan, Philip Bohannon, and Xiaojin Zhu. Building community wikipedias: A machine-human partnership approach. In *ICDE*, pages 646–655. IEEE, 2008.

[DKP$^+$09] Nilesh N. Dalvi, Ravi Kumar, Bo Pang, Raghu Ramakrishnan, Andrew Tomkins, Philip Bohannon, Sathiya Keerthi, and Srujana Merugu. A web of concepts. In Jan Paredaens and Jianwen Su, editors, *PODS*, pages 1–12. ACM, 2009.

[ERS$^+$09] Shady Elbassuoni, Maya Ramanath, Ralf Schenkel, Marcin Sydow, and Gerhard Weikum. Language-model-based ranking for queries on rdf-graphs. In David Wai-Lok Cheung, Il-Yeol Song, Wesley W. Chu, Xiaohua Hu, and Jimmy J. Lin, editors, *CIKM*, pages 977–986. ACM, 2009.

[FCI10] V. Fernandez, K. Chandramouli, and E. Izquierdo. Exploiting Complementary Resources for Cross-Discipline Multimedia Indexing and Retrieval. *User Centric Media*, pages 109–116, 2010.

[FGM05] Jenny Rose Finkel, Trond Grenager, and Christopher D. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL*. The Association for Computer Linguistics, 2005.

[FLS$^+$08] Miriam Fernández, Vanessa Lopez, Marta Sabou, Victoria S. Uren, David Vallet, Enrico Motta, and Pablo Castells. Semantic search meets the web. In *ICSC*, pages 253–260. IEEE Computer Society, 2008.

[FLS$^+$09] M. Fernandez, V. Lopez, M. Sabou, V. Uren, D. Vallet, E. Motta, and P. Castells. Using TREC for cross-comparison between classic IR and ontology-based search models at a Web scale. In *Workshop: Semantic search workshop at 18th International World Wide Web Conference*. Citeseer, 2009.

[FR97] Norbert Fuhr and Thomas Rölleke. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, 1997.

[Fuh95] Norbert Fuhr. Probabilistic datalog - a logic for powerful retrieval methods. In Edward A. Fox, Peter Ingwersen, and Raya Fidel, editors, *SIGIR*, pages 282–290. ACM Press, 1995.

[GHV05] Claudio Gutierrez, Carlos Hurtado, and Ro Vaisman. Temporal RDF. In *European Conference on the Semantic Web (ECSW 05)*, pages 93–107, 2005.

[GHV07] Claudio Gutierrez, Carlos Hurtado, and Alejandro Vaisman. Introducing Time into RDF. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):207–218, 2007.

[GK02] Norbert Gövert and Gabriella Kazai. Overview of the initiative for the evaluation of xml retrieval (inex) 2002. In Norbert Fuhr, Norbert Gövert, Gabriella Kazai, and Mounia Lalmas, editors, *INEX Workshop*, pages 1–17, 2002.

[GKS08] Konstantin Golenberg, Benny Kimelfeld, and Yehoshua Sagiv. Keyword proximity search in complex data graphs. In Wang [Wan08], pages 927–940.

[Gra91] Gösta Grahne. *The Problem of Incomplete Information in Relational Databases*, volume 554 of *Lecture Notes in Computer Science*. Springer, 1991.

[HBN$^+$01] Arvind Hulgeri, Gaurav Bhalotia, Charuta Nakhe, Soumen Chakrabarti, and S. Sudarshan. Keeyword search in databases. *IEEE Data Eng. Bull.*, 24(3):22–32, 2001.

[HGP03] Vagelis Hristidis, Luis Gravano, and Yannis Papakonstantinou. Efficient ir-style keyword search over relational databases. In *VLDB*, pages 850–861, 2003.

[HKPS06] Vagelis Hristidis, Nick Koudas, Yannis Papakonstantinou, and Divesh Srivastava. Keyword proximity search in xml trees. *IEEE Trans. on Knowl. and Data Eng.*, 18:525–539, April 2006.

[HLC08a] Yu Huang, Ziyang Liu, and Yi Chen. extract: a snippet generation system for xml search. *Proc. VLDB Endow.*, 1:1392–1395, August 2008.

[HLC08b] Yu Huang, Ziyang Liu, and Yi Chen. Query biased snippet generation in xml search. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, pages 315–326, New York, NY, USA, 2008. ACM.

[HP02] Vagelis Hristidis and Yannis Papakonstantinou. Discover: Keyword search in relational databases. In *VLDB*, pages 670–681. Morgan Kaufmann, 2002.

[HV06] Carlos A. Hurtado and Alejandro A. Vaisman. Reasoning with Temporal Constraints in RDF. In *Principles and Practice of Semantic Web Reasoning*, pages 164–178. Springer, 2006.

[HWY07] Hao He, Haixun Wang, Jun Yang 0001, and Philip S. Yu. Blinks: ranked keyword searches on graphs. In Chan et al. [COZ07], pages 305–316.

[JCE$^+$07] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian, Yunyao Li, Arnab Nandi, and Cong Yu. Making database systems usable. In Chan et al. [COZ07], pages 13–24.

[Kat] Kiyavitskaya N. Tympas A. Katifori, A. Papyrus deliverable d3.1: Ontologies for news and historical content.

[Kiy] N. Kiyavitskaya. Documentation on papyrus ontologies. Technical report, University of Trento, Italy.

[KJ02] Jaana Kekäläinen and Kalervo Järvelin. Using graded relevance assessments in IR evaluation. *Journal of the American Society for Information Science and Technology*, 53(13):1120–1129, 2002.

[Kou94] Manolis Koubarakis. Database models for infinite and indefinite temporal information. *Inf. Syst.*, 19(2):141–173, 1994.

[KPC$^+$05] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In Klemens Böhm, Christian S. Jensen, Laura M. Haas, Martin L. Kersten, Per-Åke Larson, and Beng Chin Ooi, editors, *VLDB*, pages 505–516. ACM, 2005.

[KSI10] Georgia Koutrika, Alkis Simitsis, and Yannis E. Ioannidis. Explaining structured queries in natural language. In Feifei Li, Mirella M. Moro, Shahram Ghandeharizadeh, Jayant R. Haritsa, Gerhard Weikum, Michael J. Carey, Fabio Casati, Edward Y. Chang, Ioana Manolescu, Sharad Mehrotra, Umeshwar Dayal, and Vassilis J. Tsotras, editors, *ICDE*, pages 333–344. IEEE, 2010.

[KST09a] A. Katifori, E. Savaidou, and A. Tympas. Tradeoffs in seeking to automate historical research in digitized media archives: Historians of media meeting media informaticians. 2009.

[KST09b] Akrivi Katifori, Eirini Savaidou, and Aristotle Tympas. Making history courses relevant and attractive to engineering and science majors by bringing archival research within their reach: The papyrus initiative. In *INTED Conference*, Valencia, Spain, March 2009.

[KZGM09] Georgia Koutrika, Zahra Mohammadi Zadeh, and Hector Garcia-Molina. Data clouds: summarizing keyword search results over structured data. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '09, pages 391–402, New York, NY, USA, 2009. ACM.

[LLZ07] Yi Luo, Xuemin Lin, Wei Wang 0011, and Xiaofang Zhou. Spark: top-k keyword query in relational databases. In Chan et al. [COZ07], pages 115–126.

[LOF$^+$08] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In Wang [Wan08], pages 903–914.

[LS99] O. Lassila and R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Technical report, W3C Recommendation, 1999.

[LT10] Günter Ladwig and Thanh Tran. Combining query translation with query answering for efficient keyword search. In Aroyo et al. [AAH$^+$10], pages 288–303.

[LUM06] Yuangui Lei, Victoria S. Uren, and Enrico Motta. Semsearch: A search engine for the semantic web. In Steffen Staab and Vojtech Svátek, editors, *EKAW*, volume 4248 of *Lecture Notes in Computer Science*, pages 238–245. Springer, 2006.

[LYMC06] Fang Liu, Clement T. Yu, Weiyi Meng, and Abdur Chowdhury. Effective keyword search in relational databases. In Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis, editors, *SIGMOD Conference*, pages 563–574. ACM, 2006.

[McC96] A.K. McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering, 1996.

[MRS] C.D. Manning, P. Raghavan, and H. Sch
"utze. An introduction to information retrieval.

[MSG$^+$08] E. Minack, L. Sauermann, G. Grimnes, C. Fluit, and J. Broekstra. The Sesame LuceneSail: RDF Queries with Full-text Search. Technical report, NEPOMUK Technical Report 2008-1, Feb. 2008 ps Q PC Q SEL Q, 2008.

[MYP07] Alexander Markowetz, Yin Yang, and Dimitris Papadias. Keyword search on relational data streams. In Chan et al. [COZ07], pages 605–616.

[NMS+07] Zaiqing Nie, Yunxiao Ma, Shuming Shi, Ji-Rong Wen, and Wei-Ying Ma. Web object retrieval. In Carey L. Williamson, Mary Ellen Zurko, Peter F. Patel-Schneider, and Prashant J. Shenoy, editors, *WWW*, pages 81–90. ACM, 2007.

[PAAG+] J.R. Perez-Aguera, J. Arroyo, J. Greenberg, J. Perez-Iglesias, and V. Fresno. Using BM25F for Semantic Search.

[PAAG+10] José R. Pérez-Agüera, Javier Arroyo, Jane Greenberg, Joaquín Pérez-Iglesias, and Víctor Fresno. Inex+dbpedia: a corpus for semantic search evaluation. In Michael Rappa, Paul Jones, Juliana Freire, and Soumen Chakrabarti, editors, *WWW*, pages 1161–1162. ACM, 2010.

[PPT10] G. Paci, G. Pedrazzi, and R. Turra. Wikipedia-based approach for linking ontology concepts to their realisations in text. *LREC. To appear*, 2010.

[PY08] Ken Q. Pu and Xiaohui Yu. Keyword query cleaning. *Proc. VLDB Endow.*, 1:909–920, August 2008.

[QYC09] Lu Qin, Jeffrey Xu Yu, and Lijun Chang. Keyword search in databases: the power of rdbms. In Ugur Çetintemel, Stanley B. Zdonik, Donald Kossmann, and Nesime Tatbul, editors, *SIGMOD Conference*, pages 681–694. ACM, 2009.

[QYCT09] Lu Qin, Jeffrey Xu Yu, Lijun Chang, and Yufei Tao. Scalable keyword search on large data streams. In *ICDE* [DBL09], pages 1199–1202.

[RVMB09] Flavio Rizzolo, Yannis Velegrakis, John Mylopoulos, and Siarhei Bykau. Modeling concept evolution: A historical perspective. In Alberto H. F. Laender, Silvana Castano, Umeshwar Dayal, Fabio Casati, and José Palazzo Moreira de Oliveira, editors, *ER*, volume 5829 of *Lecture Notes in Computer Science*, pages 331–345. Springer, 2009.

[RW94] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *In Proceedings of SIGIR'94*, pages 232–241. Springer-Verlag, 1994.

[RWJ+96] S.E. Robertson, S. Walker, S. Jones, M.M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. pages 109–126, 1996.

[Sal81] G. Salton. The smart environment for retrieval system evaluation–advantages and problem areas. *Information retrieval experiment*, pages 316–329, 1981.

[Sch98] Thomas Schutz. Retrieval of complex objects, considering sgml documents as example (in german). Master's thesis, University of Dortmund, Computer Science Department, 1998.

[SKAI08] Alkis Simitsis, Georgia Koutrika, Yannis Alexandrakis, and Yannis E. Ioannidis. Synthesizing structured text from logical database subsets. In Alfons Kemper, Patrick Valduriez, Noureddine Mouaddib, Jens Teubner, Mokrane Bouzeghoub, Volker Markl, Laurent Amsaleg, and Ioana Manolescu, editors, *EDBT*, volume 261 of *ACM International Conference Proceeding Series*, pages 428–439. ACM, 2008.

[SKI08] Alkis Simitsis, Georgia Koutrika, and Yannis E. Ioannidis. Précis: from unstructured keywords as queries to structured databases as answers. *VLDB J.*, 17(1):117–149, 2008.

[SKW08] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: A large ontology from wikipedia and wordnet. *J. Web Sem.*, 6(3):203–217, 2008.

[TCRS07] Thanh Tran, Philipp Cimiano, Sebastian Rudolph, and Rudi Studer. Ontology-based interpretation of keywords for semantic search. In Aberer et al. [ACN$^+$07], pages 523–536.

[TLR] T. Tran, G. Ladwig, and S. Rudolph. iStore: Efficient RDF Data Management Using Structure Indexes for General Graph Structured Data.

[TMH10] Thanh Tran, Tobias Mathäß, and Peter Haase. Usability of keyword-driven schema-agnostic search. In Aroyo et al. [AAH$^+$10], pages 349–364.

[Tro08] Raphaël Troncy. Bringing the iptc news architecture into the semantic web. In Amit P. Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy W. Finin, and Krishnaprasad Thirunarayan, editors, *International Semantic Web Conference*, volume 5318 of *Lecture Notes in Computer Science*, pages 483–498. Springer, 2008.

[TVKM10] C. Tsinaraki, Y. Velegrakis, N. Kiyavitskaya, and J. Mylopoulos. A Context-based Model for the Interpretation of Polysemous Terms. *ODBASE*, 2010.

[TWRC09] Thanh Tran, Haofen Wang, Sebastian Rudolph, and Philipp Cimiano. Top-k exploration of query candidates for efficient keyword search on graph-shaped (rdf) data. In *ICDE* [DBL09], pages 405–416.

[Ull88] Jeffrey D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.

[Voo01] Ellen M. Voorhees. The philosophy of information retrieval evaluation. In Carol Peters, Martin Braschler, Julio Gonzalo, and Michael Kluck, editors, *CLEF*, volume 2406 of *Lecture Notes in Computer Science*, pages 355–370. Springer, 2001.

[Wan08] Jason Tsong-Li Wang, editor. *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*. ACM, 2008.

[Web10] William Webber. Evaluating the effectiveness of keyword search. *IEEE Data Eng. Bull.*, 33(1):54–59, 2010.

[Wei07] Gerhard Weikum. Db&ir: both sides now. In Chan et al. [COZ07], pages 25–30.

[WKRS09] Gerhard Weikum, Gjergji Kasneci, Maya Ramanath, and Fabian M. Suchanek. Database and information-retrieval methods for knowledge discovery. *Commun. ACM*, 52(4):56–64, 2009.

[WLP$^+$09] Haofen Wang, Qiaoling Liu, Thomas Penin, Linyun Fu, Lei Zhang, Thanh Tran, Yong Yu, and Yue Pan. Semplore: A scalable ir approach to search the web of data. *J. Web Sem.*, 7(3):177–188, 2009.

[WMZ08] William Webber, Alistair Moffat, and Justin Zobel. Statistical power in retrieval experimentation. In *Proceeding of the 17th ACM conference on Information and knowledge management*, CIKM '08, pages 571–580, New York, NY, USA, 2008. ACM.

[WPZ$^+$06] Shan Wang, Zhaohui Peng, Jun Zhang, Lu Qin, Sheng Wang, Jeffrey Xu Yu, and Bolin Ding. Nuits: a novel user interface for efficient keyword search over databases. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 1143–1146. VLDB Endowment, 2006.