NUI Galway
OÉ Gaillimh

Searching Web Data: an Entity Retrieval Model

Renaud Delbru

Supervisor: **Dr. Giovanni Tummarello**
Internal Examiner: **Prof. Stefan Decker**
External Examiner: **Dr. Jérôme Euzenat**
External Examiner: **Dr. Fabrizio Silvestri**

Dissertation submitted in pursuance of the degree of Doctor of Philosophy

Digital Enterprise Research Institute Galway
National University of Ireland, Galway / Ollscoil na hÉireann, Gaillimh

December 14, 2011

Abstract

More and more (semi) structured information is becoming available on the Web in the form of documents embedding metadata (e.g., RDF, RDFa, Microformats and others). There are already hundreds of millions of such documents accessible and their number is growing rapidly. This calls for large scale systems providing effective means of searching and retrieving this semi-structured information with the ultimate goal of making it exploitable by humans and machines alike.

This dissertation examines the shift from the traditional web document model to a web data object (entity) model and studies the challenges and issues faced in implementing a scalable and high performance system for searching semi-structured data objects on a large heterogeneous and decentralised infrastructure. Towards this goal, we define an entity retrieval model, develop novel methodologies for supporting this model, and design a web-scale retrieval system around this model. In particular, this dissertation focuses on the following four main aspects of the system: reasoning, ranking, indexing and querying. We introduce a distributed reasoning framework which is tolerant against low data quality. We present a link analysis approach for computing the popularity score of data objects among decentralised data sources. We propose an indexing methodology for semi-structured data which offers a good compromise between query expressiveness, query processing and index maintenance compared to other approaches. Finally, we develop an index compression technique which increase both the update and query throughput of the system. The resulting system can index billions of data objects and provides keyword-based as well as more advanced search interfaces for retrieving the most relevant data objects.

This work has been part of the Sindice search engine project at the Digital Enterprise Research Institute (DERI), NUI Galway. The Sindice system currently maintains more than 100 million pages downloaded from the Web and is being used actively by many researchers within and outside of DERI. The reasoning, ranking, indexing and querying components of the Sindice search engine is a direct result of this dissertation research.

Declaration

I declare that this thesis is composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

Renaud Delbru

Contents

Part I.

Prelude

Chapter 1.

Introduction

On the Web, a growing number of data management scenarios have to deal with heterogeneous and inter-linked data sources. More and more structured and semi-structured data sources are becoming available. Hundreds of millions of documents embedding semi-structured data, e.g., HTML tables, Microformats, RDFa, are already published and publicly accessible. We observe a rapidly growing amount of online Semantic Web data repositories, e.g., the Linked Data community[1]. Also, we are expecting that millions of databases supporting simple web applications or more advanced Web 2.0 services, e.g., Wordpress or Facebook, will open their data to the Web. There is a growing demand for the exploitation of these data sources [AAB+09], and in particular there is the need for searching and retrieving information from these data sources.

With the current availability of data publishing standards and tools, publishing semi-structured data on the Web, which from here on we will simply call *Web Data*, is becoming a mass activity. Indeed, nowadays it is not limited to a few trained specialists any more. Instead, it is open to industries, governments and individuals. Some prominent fields in which examples of semi-structured data publishing efforts exist are:

e-government with the launch of `data.gov` by the US Federal Government, `http://data.gov.uk/` by the UK government or `data.vancouver.ca` by the City of Vancouver to publicly share open government data;

editorial world in which we can cite the effort of the New York Times or Reuters to publish rich metadata about their news articles;

[1]Linked Data: `http://linkeddata.org/`

e-commerce with for example BestBuy, which published product descriptions, store details and other information in a machine-readable format;

web content management systems such as Drupal or Wordpress that provides means to expose their structured information as Web Data.

social networking such as the Facebook's Open Graph Protocol which enables any web page to become a rich object in a social graph.

e-science such as Bio2RDF[2] which offers the integration of over 30 major biological databases.

Also, the emergence of Web 2.0 communities in which users join ad-hoc communities to create, collaborate and curate domain-specific information has created many "closed data silos" that are not reusable outside the communities. Some of these communities, e.g., Last.fm or LiveJournal, are already publishing open data, and it is expected that other communities will follow the lead. The Web is becoming an universal interactive medium for exchanging data, being unstructured, semi-structured or structured. The result is a sea of data objects ranging from traditional web documents to records in online databases and covering various domains such as social information or business data.

However, the usefulness of Web Data publishing is clearly dependent of the ease by which data can be discovered and consumed by others. Taking the e-commerce example, how can a user find products matching a certain description pattern over thousands of e-commerce data sources ? By entering simple keywords into a web search system, the results are likely to be irrelevant since the system will return pages mentioning the keywords and not the matching products themselves. Current search systems are inadequate for this task since they have been developed for a totally different model, i.e., a Web of Documents. The shift from *documents* to *data objects* poses new challenges for web search systems. One of the main challenge is to develop efficient and effective methods for searching and retrieving data objects from a world of distributed and heterogeneous data sources.

Throughout this thesis, we introduce a novel methodology for searching semi-structured information coming from distributed data sources. This methodology applied directly and has been motivated by the Web Data scenario. We introduce a logical view of the distributed data sources and propose a boolean model to search over these data sources. The logical view is built around two important concepts, the concept of *dataset* and

[2]Bio2RDF: `http://bio2rdf.org/`

entity. Based on this model, we propose appropriate methods to search data objects and improve the search results. All of the methods are designed especially for large scale scenarios. We finally show how to integrate and interconnect these novel techniques into a single system, a search engine for Web Data, which retains the same class of scalability than traditional web information retrieval systems.

1.1. On Web Search and Retrieval

The term *Information Retrieval* is defined by Baeza-Yates and Ribeiro-Neto [BYRN99] as the task of representing, storing, organising, and accessing *information items*. The goal of Information Retrieval systems is to help users accessing the information they need. Fundamentally, a user accesses information by formulating an *information request* that summarises his information need and can be processed by the Information Retrieval system. An information request is generally expressed by a *query* composed of a set of *keywords* (or index terms). Given a query, the task of an Information Retrieval system is to retrieve information items that might be useful, or *relevant*, in some sense to the user.

Information Retrieval systems are focusing on the retrieval of information as opposed to traditional Database systems which focusses on efficiently solve the user query. The task is to provide a *partial* list of information items where each item possibly holds information about a certain subject vaguely defined by the keyword query. On the contrary, the task of a Database system is to provide a *complete* list of data items that satisfies a clearly defined query. In fact, the main differences between them lie on the type of requests, vague versus clear, and on the type of result which is more of an informative nature for an Information Retrieval system and is more precise for a Database system.

1.1.1. Information Retrieval for Web Documents

Information Retrieval systems for the Web, i.e., web search engines, are mainly devoted to finding relevant web documents[3] in response to a user's query. The information items that such systems have to deal with are documents. We say that the retrieval system is following a *document-centric* model, since the whole system is organised around the concept of document. The representation of a document which is commonly adopted by

[3]We are deliberately excluding multimedia data which are out of scope of the present work.

web search systems is the *full-text* logical view [BYRN99] where a document is seen as a set or sequence of words.

The retrieval process works as follow. A web search engine provides to the user a full-text search interface where the user can compose a query using keywords and operators to combine keywords. The query is parsed and processed to compute a list of matching documents. The query processing is generally supported by specialised index structures such as inverted indexes for fast computation. Finally, the list of documents are ordered by relevance with respect to the query according to a certain scoring mechanism.

There are many reasons behind the success of web search engines. Users can use a more natural interface, i.e., keywords, to formulate their request as opposed to database style query with restricted syntax and vocabulary. This considerably lowers the technical barrier of use for end-users. Moreover, a web search engine compared to a database is more tolerant to vaguely formulated requests. The system always tries to return the most relevant results to a vaguely defined request. In addition, web users tolerate false-positive[4] and false-negative[5] errors in the search results as long as some correct results are returned.

With the scale and nature of the Web, Information Retrieval systems had to face new problems. Web search engines have to provide access to billions of documents. Indexing, query processing and ranking at this scale requires advanced techniques to create systems that provide good quality results with sub-second response time [BYCJ+07]. While Information Retrieval systems for Web Data have to face similar problems, they have to deal with an additional factor: the information is semi-structured instead of being unstructured. Moving from unstructured to semi-structured information raises new challenges but also provides new opportunities for developing more advanced search techniques.

1.1.2. Information Retrieval for Web Data

Data on the Web, or Web Data, comes from a variety of sources other than traditional databases. For example, Web Data includes structured information extracted from text, software logs and sensors, or metadata embedded in documents. Compared to web

[4] A false-positive error occurs when the system rejects a relevant document from the search results
[5] A false-negative error occurs when the system returns an irrelevant document in the search results

documents, which can be considered as *unstructured data*[6], each of these data sources is providing some kind of structure. However, since each of these data sources has its own defined schema, ranging from loosely to strictly defined, the data structure does not follow strict rules as in a database. We say that the data is *semi-structured* [Abi97].

Even in one data source, the schema might not be fixed and may change as the information grows. The information structure evolves over time, and new records can require new attributes. Let's take as an example the Web Data coming from a listing site such as eBay. As the number of items is very large and the item descriptions have a rich and evolving structure which cannot be captured, it is not possible to define an apriori schema. A solution to such a scenario is to adopt a semi-structured data definition, a schema that places loose constraints on the data and where values are raw and not strongly typed [Flo05].

The semi-structured information items that are found on the Web are very diverse. These items can describe a person, a product, a location, a document, etc. In fact, these items can describe any kind of *entities*. We consider these items, which from here on we will simply call *entity descriptions*, as the principal units of information to be searched and retrieved. We envision that the task of an Information Retrieval system for Web Data is to provide a list of entity descriptions, where each of them is holding information that the user perceives as being valuable with respect to some information request. For these reasons, we organise our retrieval model around the concept of *entity* as opposed to the concept of *document* found in traditional web search engines. We say that the retrieval system is following an *entity-centric* model. This model does not adhere to the full-text logical any more. Compared to a document which is commonly seen as a sequence of words, an entity description is a more complex data object which is seen as a set of attribute and value pairs and possibly a set of relations with other entities.

While most of the users are searching the Web with keywords, machines and experienced users which are generally more aware of the data structures need to express more complex query to interact with the system. The use of boolean connectives or quote phrases are just the beginning of the complex search queries we can expect from machines. Being able to express not only content, i.e., keywords, but also structure in the search query enables the formulation of more precise and targeted requests. For example, when querying e-commerce databases for a particular product, a user will naturally search on explicit specifications such as the manufacturer, the model, the price or even specific

[6]The term "unstructured data" refers to data which does not have a clear machine-understandable structure as opposed to "structured data" found in relational database.

properties to the product such as the capacity for an hard-drive or the focal length for a camera. Such complex search queries cannot be handled by a simple keyword-based interface that the current web search engines provide.

1.2. Challenges in Information Retrieval for Web Data

We have to face new challenges when building a Web Data search engine. Such a search engine has to deal with highly heterogeneous information sources, to offer advanced search interfaces and to provide to many simultaneous clients an access to billions of entities in millions of data sources with sub-second response times. New index structures as well as new index optimisations are needed to incorporate semi-structured information in the system while sustaining a fast query computation and an efficient system maintenance. With respect to the search quality, new techniques have to be developed to exploit the semi-structured nature of the data in order to improve the search quality. Relations, or links, between entities have a meaning, as opposed to hyperlinks between documents. Entities are inter-linked with different types of relations which might be more or less informative. Also, the graph topology on Web Data might have similar or different characteristics than the ones found on a Web of documents. Link analysis algorithms must be extended to take into account these characteristics in order to deliver better effectiveness. In addition, Web Data is generally based on formal description and enables a system to derive new logical consequences which can then be used for improving search results. However, logic-based techniques are computationally intensive and are sensitive to noisy data and unexpected use of data. It is still unclear if a system can apply logic-based techniques on a very large scale and with low data quality without having significant repercussions on its performance.

1.3. Scope of Research

The scope of this thesis is to study the problem of Information Retrieval in distributed semi-structured information sources. More precisely, our study is focused on the challenges in building a large scale Information Retrieval system for Web Data. By large scale, we mean a system that is deployed on a very large number of machines, that grows as the

data volume increases and that many clients can access simultaneously. Also, the system must use available capacity effectively as it is distributed across multiple machines.

The main problem is to cope with the shift from the traditional document-centric model to the entity-centric model, and to analyse its impacts on the search techniques and main components of a search system. There is a need to reassess the data model, which is currently document-oriented, as well as the boolean search model which does not provide foundations for querying semi-structured data. In addition to scale gracefully with a large amount of data and a large number of simultaneous clients, an Information Retrieval system for Web Data requires techniques that are robust against low data quality which is principally due to the heterogeneity of Web Data.

Our study focusses on two main aspects: the quality of search results and the retrieval efficiency of the search system. With respect to the quality of search results, we investigate the feasibility of logical approaches in an Information Retrieval system by taking a context-dependent view over the data. Also, we examine link analysis methods for measuring the importance of an entity by exploiting the peculiar properties of link between datasets and entities. With respect to the retrieval efficiency of the system, we are concerned with the evaluation of different indexing techniques for semi-structured data. We compare the indexing techniques based on their query expressiveness, the performance of the query processing and the cost of maintenance operations. Also, we investigate compression techniques for optimising the index, as they have a direct impact on the overall performance of the system.

1.4. What the Thesis is not about

As we are interested in searching semi-structured data, and more precisely in searching entities among heterogeneous and decentralised information sources, we assume in our model that the information sources provide way to identify and access the data about a particular entity through standard data exchange protocol. Moreover, for the purpose of the link analysis method, we assume that these data sources are inter-linked, i.e., that entities from one information source can have relationships with entities from other information sources. These relationships are explicitly defined by the data sources themselves by referring to entities from other data sources using a global identifiers, e.g., using URIs in the Web Data scenario.

The data acquisition part of a search engine is an important aspect. Crawling semi-structured data is different than crawling documents, it should require a particular attention. However, this task is out of scope of this thesis and instead we use currently state-of-the-art techniques in this domain in order to discover new information sources, and to gather and keep up to date the entity descriptions from various information sources.

With respect to the search quality, we concentrate on a link analysis approach to compute static ranks of semi-structured data items. The thesis does not investigate the problem of query-dependent ranking for semi-structured data. We implicitly consider our boolean search model as an Extended Boolean Model [SFW83], i.e., we consider term weights to rank search results in accordance with a query. In that context, we present a simple term weighting function and discuss how to integrate it in the query processing framework. However, we do not evaluate in term of effectiveness this query-dependent ranking approach and also we do not investigate other approaches.

Finally, while the user interaction with an Information Retrieval system is important to assess, we are not interested in this aspect as we mainly focus on the underneath mechanism and performance of the system. In addition, we believe that such a system must be principally available for automatic processing by machines or applications. We therefore consider the query-based interface as the primary method to interact with the system. In the rest of the thesis, we suppose that the system provides an interface with which a query can be sent. As an answer to the query, the system returns a list of results ordered by relevance to the query. We do not investigate if other more advanced search interfaces, e.g., faceted-based interface or aggregated search interface such as Sig.ma [TCC+10], increase the usability of the system.

1.5. Outline of the Thesis

The thesis is divided in four parts. The first part, *Foundations*, introduces the framework and the main concepts that are used throughout the document. The second part, *Methods*, proposes new techniques for searching entities and improving search results. The third part, *System*, describes an Information Retrieval system implementing our techniques. Finally, the fourth part concludes the thesis.

In the rest of the thesis, we will often use the term *Web Data* to denote the semi-structured data that is published on the Web, and the term *Web of Data* to denote the infrastructure that supports data publishing on the Web.

Part I: Foundations

In Chapter 2, we examine the problem of the management of decentralised heterogeneous information sources and give an overview of the current approaches for managing these data sources. We then define our model for Web Data, centred around the concepts of dataset and entity. We also discuss some requirements when searching semi-structured data and define our boolean search model.

In Chapter 3, we review existing works from domains such as reasoning with Web Data, ranking techniques based on link analysis, indexing and searching semi-structured data, inverted indexes and indexing optimisation using compression.

Part II: Methods

Chapter 4 proposes an application of *contextual reasoning* for Web Data. We start by discussing what are the benefits of such an approach for web data search engines. We show how to apply and extend a contextual reasoning formalism to our data model. We then discuss what are the technical requirements for a high performance and distributed implementation.

In Chapter 5, we present a method that extends PageRank to rank inter-linked datasets and entities. We discuss experimental results pertaining to the performance of our approach.

Chapter 6 discusses the possible index data structures for supporting our search model. We study different methods and analyse their advantages and disadvantages. We then present our approach and compare its performance with the other approaches using experimental results.

In Chapter 7, we introduce a new class of compression algorithms for inverted indexes. We show that we can achieve substantial performance improvements and that such a technique increases the update and query throughputs of a search engine for web data.

Part III: System

Chapter 8 describes the Sindice search engine. The system is built around the model of datasets and entities and integrates the techniques presented in this thesis. We give on overview of the system infrastructure before explaining how each part is implemented and interconnected together.

Finally, Chapter 9 recalls the main findings of the research and discusses the tasks that remain.

1.6. Contribution

The main contribution of this work is a novel methodology for searching semi-structured information coming from distributed and heterogeneous data sources on a large-scale scenario. The methodology is exploited to develop an information retrieval system for Web Data. Towards that goal, we investigate the following aspects: reasoning, ranking, indexing and querying and index compression. Specific contributions include:

- a formal model for distributed and heterogeneous semi-structured data sources as well as a boolean search model for supporting the entity retrieval task in Chapter 2.
- a contextual reasoning formalism for our model that enables distributed reasoning over a very large data collection and that copes with heterogeneous and noisy data in Chapter 4.
- a hierarchical link analysis methodology to compute the popularity of web data sources and entities that provides better quality results and that is efficient to compute in Chapter 5.
- a node-based indexing scheme based on inverted indexes that supports our boolean search model and that scales gracefully with a very large data collection on limited hardware resources in Chapter 6;
- a novel class of compression techniques for node-based inverted indexes that increases the update and query throughputs of the information retrieval system in Chapter 7.

Part II.

Foundations

Chapter 2.

An Entity Retrieval Model for Web Data*

In this chapter, we introduce an entity retrieval model for semi-structured information found in distributed and heterogeneous data sources. This model is used as a common framework for all the methods presented in this thesis. Before defining our model, we start by examining the problem of management of heterogeneous information sources in a decentralised setting. We review three major data models for semi-structured information as well as two infrastructures for managing heterogeneous information sources. Then, based on the Web Data scenario, we introduce a data model that encompasses the core concepts that are found in distributed and heterogeneous information sources. We first give an overview of the core abstract concepts before defining the formal data model. We then discuss some requirements for searching semi-structured information and introduce our boolean search model and query algebra.

2.1. Management of Decentralised and Heterogeneous Information Sources

In the quest for lowering the cost of data acquisition and integration, the Extensible Markup Language (XML) [BPSM+08] is a first widely deployed step towards this vision. XML provides a common syntax to share and query user-defined data across distributed systems. More recently, the *Semantic Web*[1] vision attracted much attention in a similar

*This chapter is partially based on [CDTT10]

[1]Semantic Web Activity Statement: `http://www.w3.org/2001/sw/Activity.html`

context. The Semantic Web is a set of technologies developed as part of the W3C for the publishing and automatic processing of semi-structured data on the Web. Also, the Database community has introduced the concept of *Dataspaces* [HFM06] as a new abstraction model for data management of heterogeneous information sources. These previous and ongoing efforts towards a better management of distributed and heterogeneous data sources motivate our effort to define a retrieval model that is compatible with the existing approaches. Before defining our entity retrieval model, we first review these efforts in order to understand their common characteristics.

2.1.1. Semi-Structured Data Models

The Web provides access to a large number of information sources of all kinds. Some of these information sources contain raw data where the structure exists but has to be extracted, for example HTML tables. Some of them contain data that has structure but where data is not as rigid and regular as data found in a database system. Data that is neither raw nor very strictly typed or that does not follow a fixed schema is called *semi-structured data* [Abi97]. Semi-structured data can also arise when integrating several structured data sources since multiple data sources will not adopt the same schema and the same conventions for their values. We review in the followings three major models for semi-structured data exchange over distributed information sources: the Object Exchange Model, the Extensible Markup Language and the Resource Description Framework.

Object Exchange Model - OEM

One of the seminal work in semi-structured data model is the Object Exchange Model model [PGMW95]. OEM served as the basic data model in many projects such as Tsimmis [CGMH$^+$94], Lore [QAU$^+$96] and Lorel [AQM$^+$97]. The Object Exchange Model is a model for exchanging semi-structured data between distributed object-oriented databases. In the Object Exchange Model, the data is a collection of objects, where each object can be either atomic or complex. An atomic object represents some base data types such as an integer, a string or an image. A complex object represents a set of (attribute, object) pairs. The Object Exchange Model is a graph-based data model, where the nodes are the objects and the edges are labelled with attribute names, and in which some leaf nodes have an associated atomic value.

Extensible Markup Language - XML

XML is a standard proposed by the W3C in 1998 to complement HTML for data exchange on the Web. The XML infrastructure developed by W3C includes a set of technologies and languages: abstract data models such as Infoset [CT04] and XML Schema [FW04]; and the declarative processing languages such as XPath or XQuery [WFM+07]. Nested, tagged elements are the building blocks of XML. Each tagged element can have attribute value pairs and sub-elements. The sub-elements may themselves be tagged elements, or may be just atomic data such as text. The XML standard puts no restrictions on tags, attribute names or nesting relationships for data objects, making it well suited for semi-structured data.

While XML is universally accepted and embraced by a variety of communities for representing and interchanging information, it does not solve the general problem of interoperability [Flo05]. In particular, the meaning of the tags and elements must be agreed a-priori. XML provides a syntax for presenting and organising semi-structured data, but lacks of a framework to formally describe the data encoded in the document.

Resource Description Framework - RDF

The Resource Description Framework (RDF) [KC04] is a generic data model that can be used for interoperable data exchange. In a nutshell, RDF makes easier the understanding of a resource description by a machine and makes possible its automatic processing. In RDF, a resource description is composed of statements about the resource. A statement in RDF is a triple consisting of a subject, a predicate, and an object and asserts that a subject has a property with some value. A set of RDF statements forms a directed labelled graph. Thus, RDF is a graph-based data model similarly to OEM. However, RDF propose some additional features such as the use of URIs as identifiers which helps to avoid ambiguities and enables the creation of a global data graph.

RDF also provides a foundation for more advanced assertional languages, for instance the vocabulary description language RDF Schema [BG04] or the web ontology language OWL [MvH04]. These assertional languages help to formally define and exchange a common set of terms which describes a domain. RDF Schema is a simple ontology definition language for modelling the semantics of data and offers a basis for logical reasoning. With RDF Schema, one can define RDF vocabularies, i.e., hierarchy of classes and properties, as well as the domains and ranges of properties. More expressive

definitions of the vocabulary terms can be written with the Web Ontology Language. OWL is a set of expressive languages based on description logic and can be used to bridge terminology differences and thus enhance interoperability. Ontologies can be seen as explicit integrated schemas that are reused across distributed and potentially numerous information sources. These assertional languages has been seen by many as the solution to overcome the problem of interoperability of XML.

2.1.2. Infrastructures for the Management of Decentralised and Heterogeneous Information Sources

Semantic Web

The Semantic Web [BLHL01] envisions an evolutionary stage of the World Wide Web in which software agents create, exchange, and process machine-readable information distributed throughout the Web. The Semantic Web is an extension of the current Web and acts as a layer of resource descriptions on top of it. Its aim is to combine web resources with machine-understandable data to enable people and computers to work in cooperation and to simplify the sharing and handling of large quantities of information.

The Semantic Web can be seen as a large knowledge-base formed by information sources that serve information as RDF through documents or SPARQL [PS08] endpoints. A fundamental feature of the Semantic Web is that the RDF data graph is decentralised: it has no single knowledge-base of statements but instead anyone can contribute by making new statements available in a public web space. Information sources on the Semantic Web might have something in common or complement each other. By using shared identifiers, i.e., Uniform Resource Identifiers (URIs) [CCD+01], and shared terms, information can be merged to provide useful services to both humans and software clients.

The Semantic Web infrastructure provides a technology stack, depicted in Figure 2.1, to support such a vision. The bottom layer is composed of the URI syntax which provides means to identify resources and XML which provides a standardised syntax for the data. On top of that, the Semantic Web infrastructure proposes standardised technologies, such as RDF as a data model, RDF Schema and OWL to formally describe the vocabularies of the data, and SPARQL for querying and retrieving data.

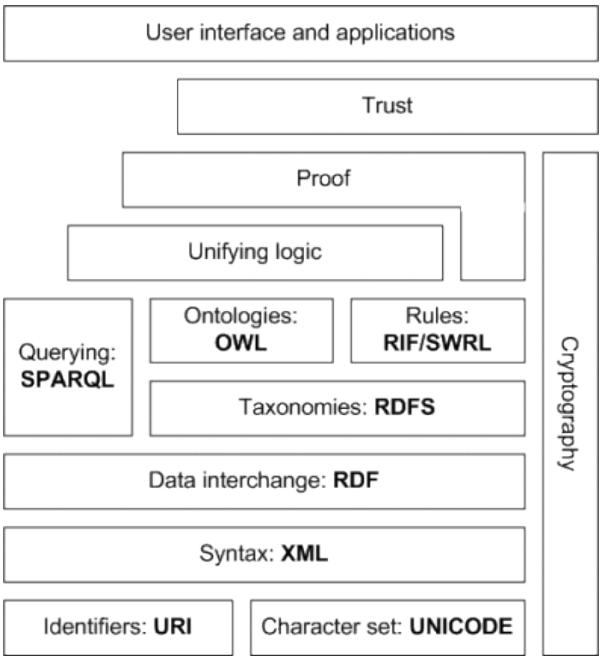


Figure 2.1.: The Semantic Web architecture in layers

Dataspaces

Dataspaces [HFM06] are a new abstraction model for data management in scenarios with a large number of diverse, loosely interrelated data sources, and propose the design and development of DataSpace Support Platforms (DSSPs). In a nutshell, a DSSP offers a suite of services that enables developers to focus on the specific challenges of their applications, rather than on the recurring challenges involved in dealing consistently with large amounts of disparate data. In particular, a DSSP helps to identify sources in a dataspace and inter-relate them by offering basic query mechanisms over them. For example, a DSSP can provide keyword search over all of its data sources, similar to that provided by existing desktop search systems.

Unlike data integration systems that offer uniform access to heterogeneous data sources, dataspaces do not assume that all the semantic relationships between sources are known and have been specified. The goal of dataspaces is to provide useful services whenever possible, and assist users in creating tighter semantic relationships when they see fit. Scenarios in which we want to manage dataspaces include personal data on ones desktop, collections of data sources in enterprises, government agencies, collaborative scientific projects, digital libraries, and large collections of data sources on the Web.

2.1.3. Common Characteristics in Decentralised and Heterogeneous Information Sources

In the next sections, we define an entity retrieval model for decentralised semi-structured information sources that is generic enough to cover the previously described data models and infrastructures. By defining a compatible retrieval model and using it as a framework through the whole thesis, we make the methodologies introduced in this thesis applicable on both the Semantic Web, the Dataspaces and possibly other scenarios.

A first step towards the definition of such a retrieval model is to understand the similarities across the existing data models and infrastructures, and to establish the most important characteristics of decentralised semi-structured information sources. These characteristics will shape in broad outline the retrieval model.

We can distinguish two common characteristics between the two infrastructures, i.e., the Semantic Web and the Dataspaces, are

- the existence of a multitude of inter-connected data sources; and
- the semi-structured nature of the data.

Also we can remark that the previous semi-structured data models can be abstracted by a graph-based data model. Among these models, we discern the following common features:

- the data graph is composed of nodes and edges which respectively represents entities and the relations between these entities;
- the existence of a mechanism for identifying a particular entity in the data graph;
- besides relationships with other entities, each entity is composed of a set of attribute and value pairs;

2.2. Abstract Model

The goal of this section is to define an abstract model for decentralised semi-structured information sources. In the following, we take the Web Data scenario as an example to abstract and define the core components of our model.

We define Web Data as part of the Hypertext Transfer Protocol (HTTP) [BLFN96] accessible Web that returns semi-structured information using standard interchange formats and practices. For example, The Linked Data community advocates the use of HTTP content negotiation and redirect[2] to provide either machine or human readable versions of the data. The standard data interchange formats include HTML pages which embed RDFa or Microformats as well as RDF models using different syntaxes such as RDF/XML [Bec04].

2.2.1. Practical Use Case

To support the Web Data scenario, we take as an example the case of a personal web dataset. A personal web dataset is a set of semi-structured information published on one personal web site. The web site is composed of multiple documents, either using HTML pages embedding RDFa or using plain RDF files. Each of these documents contains a set of statements describing one or more entities. By aggregating the semi-structured content of these documents altogether, we can recreate a dataset or RDF graph. Figure 2.2 depicts such a case. The content, or database, of a web site `http://renaud.delbru.fr/` is exposed through various accessible online documents, among them `http://renaud.delbru.fr/rdf/foaf` and `http://renaud.delbru.fr/-publications.html`. The former document provides information about the owner of the personal web site and about its social relationships. The second document provides information about the publications authored by the owner of the web site. Each of them is providing complementary pieces of information which when aggregated provide a more complete view about the owner of the web site.

2.2.2. Model Abstraction

In the previous scenario, it is possible to abstract the following core concepts in semi-structured data web publishing: *dataset*, *entity* and *view*:

A dataset is a collection of entity descriptions. One dataset is usually the content of a database which powers a web application exposing metadata, be this a dynamic web site with just partial semantic markups or a Semantic Web database which exposes its content such as the Linked Open Data datasets. Datasets however can

[2]URL redirection is a directive that allows to indicate that further action, e.g., fetch a second URL, needs to be taken by the user agent in order to fulfil the request.

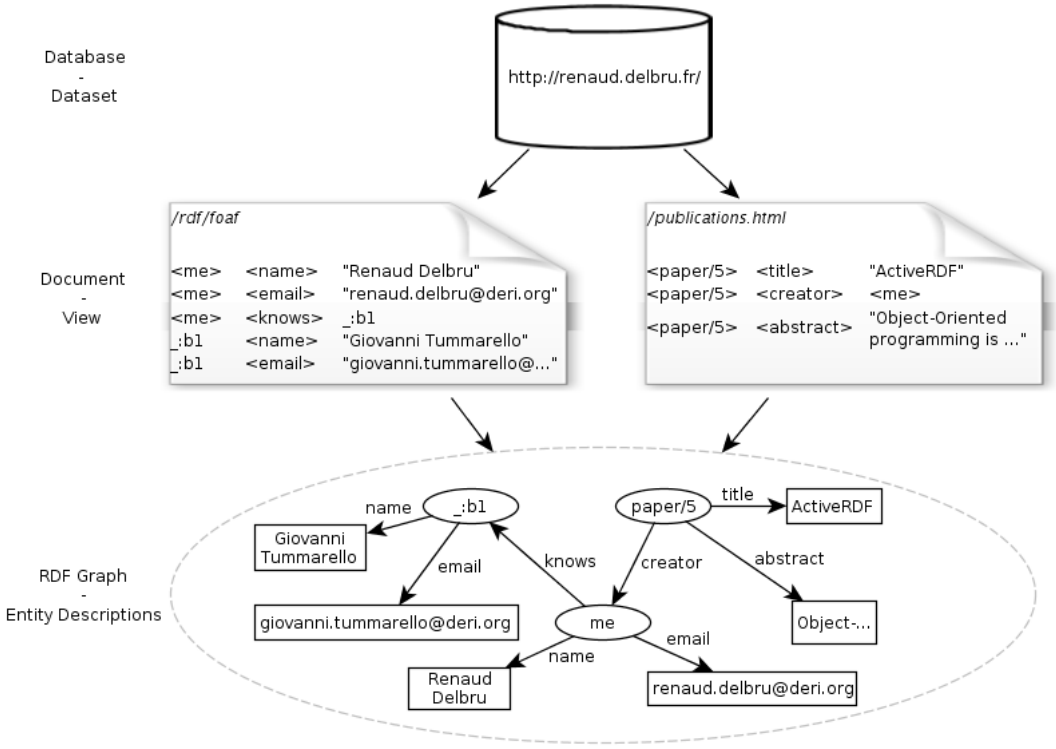


Figure 2.2.: A sample of a personal web dataset. The dataset `http://renaud.delbru.fr/`, i.e., a database, makes available partial *view* of its content in the form of documents containing semi-structured data. The aggregation of all the views enables the reconstruction of the dataset in the form of a data graph.

also come in the form of a single RDF document, e.g., an individual FOAF file posted on a person's homepage. A dataset is uniquely identified by a URI, e.g., `http://renaud.delbru.fr/` as depicted in Figure 2.2.

An entity description is a set of assertions about an entity and belongs to a dataset. Typical examples of entities include documents, people, events, products, etc. The assertions provide information regarding the entity such as attributes, for instance, the firstname and surname for a person, or relationships with other entities, for instance, the family members for a person. An entity description has a unique identifier, i.e., a URI, which can be used to retrieve the set of assertions from a dataset using standard data exchange protocols, e.g., HTTP.

A View represents a single HTTP accessible piece of information, retrieved by resolving a URI, that provides a full or partial view of the dataset. In case of Linked Open Data, a typical view is the RDF model returned when one dereferences the URI of a resource. The Linked Open Data views are 1 to 1 mapping from the URI to the complete entity description. This however is more the exception than the rule for other kind of web data publishing where most often only partial entity descriptions are provided in the views. For example in the case of microformats or RDFa, views are pages that talk about different aspects of the entity, e.g., a page listing social contacts for a person, a separate page listing personal data as well as a page containing all the posts by a user. As shown in Figure 2.2, the union of all the views provided within a context, e.g., a web site, might enable an agent, e.g., a crawler, to reconstruct the entire dataset. There is no guarantee on this since it is impossible to know if we are in possession of every pieces of information about an entity. The problem of retrieving the views and reconstructing a dataset is more related to data acquisition which is out of scope in this thesis. We assume in the rest of the thesis that all the views have been gathered and that the datasets have been reconstructed.

This three layer model, which is graphically represented in Figure 2.3, forms the base for the methodologies and the infrastructure described in this thesis.

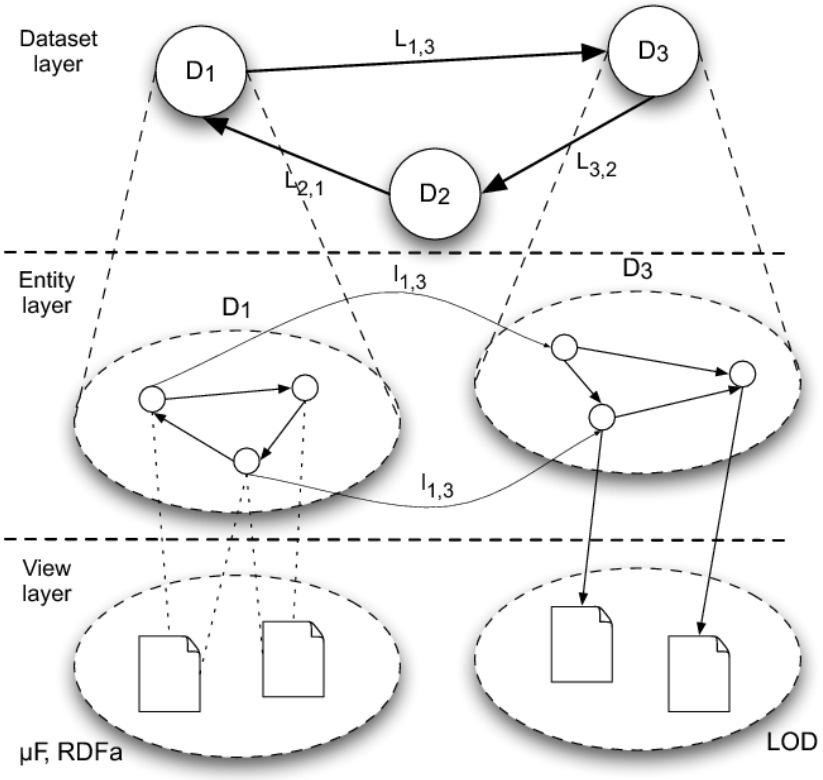


Figure 2.3.: The three-layer model of Web Data

2.3. Formal Model

For the purpose of this work, we need a generic graph model that supports the previous scenario and covers the three layer model discussed previously. First, we define a labelled directed graph model that covers the various type of Web Data sources, i.e., Microformats, RDFa, Semantic Web repositories, etc. This graph model represents datasets, entities and their relationships. With respect to this graph model, we define formally the following concepts: *dataset*, *entity*, *entity relationship* and *entity attribute*.

2.3.1. Data Graph

Let V be a set of nodes and A a set of labelled edges. The set of nodes V is composed of two non-overlapping sets: a set of entity nodes V^E and a set of literal nodes V^L. Let $\mathcal{L}$ be a set of labels composed of a set of node labels $\mathcal{L}^V$ and a set of edge labels $\mathcal{L}^A$.

The Web Data is defined as a graph G over $\mathcal{L}$, and is a tuple $G = \langle V, A, \lambda \rangle$ where $\lambda : V \rightarrow \mathcal{L}^V$ is a node labelling function. The set of labelled edges is defined as $A \subseteq \{(e, \alpha, v) | e \in V^E, \alpha \in \mathcal{L}^A, v \in V\}$. The components of an edge $a \in A$ will be denoted by $source(a)$, $label(a)$ and $target(a)$ respectively.

2.3.2. Dataset

A dataset is defined as a subgraph of the Web Data graph:
Definition 2.1 (Dataset)**.** A dataset D over a graph $G = \langle V, A, \lambda \rangle$ is a tuple $D = \langle V_D, A_D, \mathcal{L}_D^V, \lambda \rangle$ with $V_D \subseteq V$ and $A_D \subseteq A$.

We identify a subset $\mathcal{L}_D^V \subseteq \mathcal{L}_V$ of node labels to be *internal* to a dataset D, i.e., the set of entity identifiers and literals that originates from this dataset. For example, such a set might include the URIs and literals defined by the naming authority of the dataset [HKD09].
Definition 2.2 (Internal Node)**.** A node $v \in V_D$ is said to be internal to a dataset D if $\lambda(v) \in \mathcal{L}_D^V$.

Analogously, we identify as *external* a node with a label that does not belong to $\mathcal{L}_D^V$.
Definition 2.3 (External Node)**.** A node $v \in V_D$ is said to be external to a dataset D if $\lambda(v) \notin \mathcal{L}_D^V$.

We denote $V_D^{\underline{E}} \subseteq V_D^E$ as being the set of internal entity nodes for a dataset D and $V_D^{\overline{E}} \subseteq V_D^E$ as being the set of external entity nodes for a dataset D. We assume that a literal node is always *internal* to a dataset D. A literal is always defined with respect to an entity and within the context of a dataset. Therefore, if two literal nodes have identical labels, we consider them as two different nodes in the graph G. This assumption does not have consequences for the methods described in this thesis. In fact, this can be seen as a denormalisation of the data graph by increasing the number of nodes in order to simplify data processing.

Two datasets are not mutually exclusive and their entity nodes $V_D^E \subseteq V_D$ may overlap, i.e., $V_{D_1}^E \cap V_{D_2}^E \neq \emptyset$, since

- two datasets can contain identical *external* entity nodes, and
- the *internal* entity nodes in one dataset can be *external* entity nodes in another dataset.

In a dataset, we identify two types of edges: *intra-dataset* and *inter-dataset* edges. An *intra-dataset* edge is connecting two *internal* nodes, while an *inter-dataset* edge is connecting one *internal* node with one *external* node.

Definition 2.4 (Intra-Dataset Edge)**.** An edge $a \in A_D$ is said to be *intra-dataset* if $\lambda(source(a)) \in \mathcal{L}_D^V, \lambda(target(a)) \in \mathcal{L}_D^V$.

Definition 2.5 (Inter-Dataset Edge)**.** An edge $a \in A_D$ is said to be *inter-dataset* if $\lambda(source(a)) \in \mathcal{L}_D^V, \lambda(target(a)) \notin \mathcal{L}_D^V$ or if $\lambda(source(a)) \notin \mathcal{L}_D^V, \lambda(target(a)) \in \mathcal{L}_D^V$.

We group inter-dataset links into *linkset*. For example, in Figure 2.3 the inter-dataset links $l_{1,3}$ between D_1 and D_3 are aggregated to form the linkset $L_{1,3}$ on the dataset layer.

Definition 2.6 (Linkset)**.** Given two datasets D_i and D_j, we denote a linkset between D_i and D_j with $L_{\alpha,i,j} = \{a | label(a) = \alpha, source(a) \in D_i, target(a) \in D_j\}$ the set of edges having the same label α and connecting the dataset D_i to the dataset D_j.

2.3.3. Entity Attribute-Value Model

The data graph provides information about an entity including its relationships with other entities and its textual attributes (literals). Consequently, a subgraph describing an entity can be extracted from the data graph.

The simplest form of description for an entity node e is a *star graph*, i.e., a subgraph of a dataset D with one central node e and one or more edges. The Figure 2.4 shows

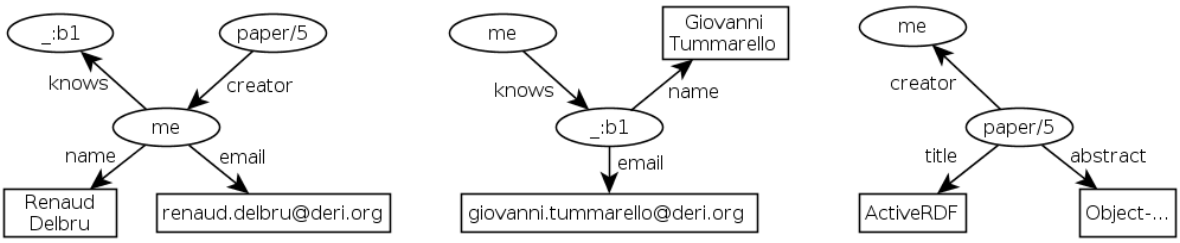


Figure 2.4.: A visual representation of the RDF graph from Figure 2.2 divided into three entities identified by the nodes *me*, *_:b1* and *paper/5*.

me

Attribute	Value
name	Renaud Delbru
email	renaud.delbru@deri.org
knows	_:b1
creator $^{-1}$	paper/5

paper/5

Attribute	Value
title	ActiveRDF
abstract	Object-oriented programming is ...
creator	me

_:b1

Attribute	Value
name	Giovanni Tummarello
email	giovanni.tummarello@...
knows $^{-1}$	me

Figure 2.5.: An example of EAV model derived from the three subgraphs from Figure 2.4.

how the RDF graph from Figure 2.2 can be split into three entities *me*, *_:b1* and *paper/5*. Each entity description forms a sub-graph containing the incoming and outgoing relations of the entity node. More complex graphs might be extracted, but their discussion is out of scope of this thesis.

An entity description is defined as a tuple $\langle e, A_e, V_e \rangle$ where $e \in V_D^E$ the entity node, $A_e \subseteq \{(e, \alpha, v) | \alpha \in \mathcal{L}_D^A, v \in V_e\}$ the set of labelled edges representing the attributes and $V_e \subseteq V_D$ the set of nodes representing values.

Conceptually, an entity is represented by an identifier, a set of attributes and a set of values for these attributes. Such a model is similar to the Entity Attribute Value model (EAV) [NMC+99] which is typically used for representing highly sparse data with heterogeneous attributes. Under this model, we can depict the dataset example in Figure 2.2 as a set of EAV tables in Figure 2.5.

In its most simple form, an attribute is composed by one edge connecting two nodes, the central entity node e with a value node v, being entity or literal. However, this is not always the case. More complex attributes such as multi-valued attributes are composed of more than one edge. We present and define in the following the possible type of attribute we aim to support.

Atomic Attribute: An atomic attribute is the simplest form of attribute. It represents an atomic characteristic of an entity, for example the name of a person or the title of a publication.
Definition 2.7 (Atomic Attribute)**.** Given an entity e, an atomic attribute is defined as being an edge $(e, \alpha, v)|v \in V_e, \alpha \in \mathcal{L}_A$

Single-Valued Attribute: A single-valued attribute is an attribute that holds exactly one value. For example, the birthdate is a single-valued property since a person has only one birthdate.
Definition 2.8 (Single-Valued Attribute)**.** Given an entity e, a single-valued attribute is an atomic attribute with exactly one value $(e, \alpha, v)|\exists! v\ target(\alpha) = v$.

Multi-Valued Attribute: An attribute is multi-valued when it has more than one value. For example, the email address of a person can be a multi-valued attribute since a person has possibly more than one email address.
Definition 2.9 (Multi-Valued Attribute)**.** Given an entity e, a multi-valued attribute is a set of edges having the same label $\alpha \in \mathcal{L}_A$ and with different value nodes.

This Entity Attribute-Value model is expressive enough to cover most of the entity descriptions that can be found on the Web. Based on this data model, we introduce a boolean search model which enables the retrieval of entities. We define formally this search model with a query algebra. The definition of a precise search model which covers the possible expected search use cases is primordial. Without this search model, it is difficult to compare the usability and precision of our retrieval system with others.

2.4. Search Model

The search model presented here adopts a semi-structural logical view as opposed to the full-text logical view of traditional web search engines. We do not represent an entity by a bag of words, but we instead consider its set of attribute-value pairs. The search model is therefore restricted to search entities using a boolean combination of attribute-value

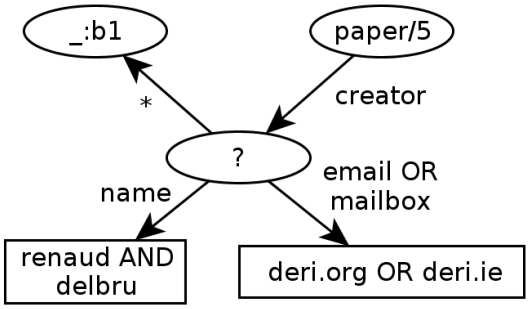


Figure 2.6.: A star-shaped query matching the description graph of the entity *me* from Figure 2.4. *?* stands for the bound variable and $\star$ for a wildcard.

pairs. We do not support relational queries such as the one found in relational database. We aim to support three types of queries:

full-text query: the typical keyword based queries, useful when the data structure is unknown;

structural query: complex queries specified in a star-shaped structure, useful when the data schema is known;

semi-structural query: a combination of the two where full-text search can be used on any part of the star-shaped query, useful when the data structure is partially known.

These queries provide a gradual increase of the complexity of the query semantics. This gradual increase enables to accommodate various kind of information requests, from vague using full-text queries to precise using structural queries. The type of request depends on the awareness of the data structure by the user and on the user expertise. We remind the reader that we expect the main users to be machines, which need complex query semantic in certain cases to generate and ask complex requests.

The main use case for which this search model is developed is entity search: given a description pattern of an entity, i.e., a star-shaped queries such as the one in Figure 2.6, locate the most suitable entities and datasets. This means that, in terms of granularity, the search needs to move from a "document" centric point of view (as per normal web search) to a "dataset-entity" centric point of view.

Contrary to the full-text logical view where words are assigned to a single bag of words, in the semi-structured view, the words are assigned to multiple distinct bag of words, one for each node and edge label. Consequently, it is possible to distinguish when words occur in a same value or different values and avoid false-positive answers.

Dataset	*Entity*	*Attribute*	*Value*
delbru.fr	me	name	Renaud Delbru
delbru.fr	me	knows	_:b1
delbru.fr	paper/5	creator	me

(a) Entity Attribute Value relation R_1

Dataset	*Entity*	*Attribute*	*Value*
delbru.fr	paper/5	title	ActiveRDF
delbru.fr	paper/5	creator	me
delbru.fr	me	name	Renaud Delbru

(b) Entity Attribute Value relation R_2

Table 2.1.: An example of two Entity Attribute Value relations, R_1 and R_2

2.4.1. Search Query Algebra

In this section, we introduce a formal model of our search query language. For clarity reasons, we adopt a similar model as the relational query algebra [Mai83], where the inputs and outputs of each operator are "relations". Every operators in the algebra accept one or two relation instances as arguments and return a relation instance as a result.

We first describe the relations that are used in the algebra before introducing the basic operators of the algebra.

Relations

For the purpose of the query algebra, we define an entity attribute value table as a relation $\langle dataset, entity, attribute, value \rangle$ where the fields

dataset holds the label of a dataset D;

entity holds the label of the entity node $e \in V_D^E$;

label holds the attribute label $\alpha \in \mathcal{L}_A$;

value holds the label of the value node.

Table 2.1a and Table 2.1b depict two relations that are derived from the EAV model of Figure 2.5. These two relations are used as inputs in the following examples.

In the next algebra formulas, we will use d to denote the fields *dataset*, e for *entity*, *at* for *attribute* and v for *value*.

Dataset	*Entity*	*Attribute*	*Value*
delbru.fr	me	name	Renaud Delbru
delbru.fr	paper/5	author	me

(a) $R_1 \cap R_2$

Dataset	*Entity*	*Attribute*	*Value*
delbru.fr	me	knows	_:b1

(b) $R_1 \setminus R_2$

Dataset	*Entity*	*Attribute*	*Value*
delbru.fr	me	name	Renaud Delbru
delbru.fr	me	knows	_:b1
delbru.fr	paper/5	title	ActiveRDF
delbru.fr	paper/5	creator	me

(c) $R_1 \cup R_2$

Table 2.2.: An example of the set operations applied on the relations R_1 and R_2

Set Operations

Since relations are considered as sets, boolean operations such as union, intersection and set-difference are applicable on relations.

Union The union of two relations $R \cup S$ returns a relation containing all the relation instances occurring in either R or S. For example, Table 2.2c shows the union of the two relations R_1 and R_2.

Intersection The intersection of two relations $R \cap S$ returns a relation containing all the relation instances occurring in both R and S. For example, Table 2.2a shows the intersection of the two relations R_1 and R_2.

Set-difference The set-difference of two relations $R \setminus S$ returns a relation containing all the relation instances occurring in R but not in S. For example, Table 2.2b shows the set-difference of the two relations R_1 and R_2.

Keyword Selection

The search unit is a keyword $k \in \mathcal{K}$ where $\mathcal{K}$ is the lexicon of the Web Data graph, i.e., the set of distinct words occurring in $\mathcal{L}$.

Dataset	*Entity*	*Attribute*	*Value*
delbru.fr	me	name	Renaud Delbru

(a) $\sigma_{v:renaud}(R_1)$ or $\sigma_{v:\text{“}renaud\ delbru\text{”}}(R_1)$

Dataset	*Entity*
delbru.fr	me

(b) $\pi_{d,e}(\sigma_{v:renaud}(R_1))$

Table 2.3.: An example showing the selection and projection operations

Let $W : \mathcal{L} \rightarrow \mathcal{K}$ a function that maps a label $l \in \mathcal{L}$ to a set of words $K \subset \mathcal{K}$. Given a keyword k, if $k \in W(l)$, it means that the word denoted by k appears at least one time in the label l. We say that k matches l. A keyword is therefore restricted to match the label of a node, being an entity, a value, or an attribute.

The *Keyword Selection* operator σ is a unary operator on relations. The selection operator allows to specify the relation instances to retain through a keyword selection condition.

Definition 2.10 (Keyword Selection)**.** Given a keyword selection condition c and a relation R, the keyword selection operator $\sigma_c(R)$ is defined as a set of relation instances $\{r|r \in R\}$ for which the condition c is true.

The most basic form of a keyword selection condition is to test if a given keyword k occurs in one of the field of a relation R. For example, one can test if the keyword k occurs in the dataset node label:

$$\sigma_{d:k}(R) : \{r|r \in R, k \in W(r.d)\}$$

or in the entity node label:

$$\sigma_{e:k}(R) : \{r|r \in R, k \in W(r.e)\}$$

or in the attribute label:

$$\sigma_{at:k}(R) : \{r|r \in R, k \in W(r.at)\}$$

or in the value node label:

$$\sigma_{v:k}(R) : \{r|r \in R, k \in W(r.v)\}$$

The selection operator has the following properties. Proves of these properties can be found in [Mai83].

idempotent: multiple applications of the same selection operator have no additional effect beyond the first one:

$$\sigma_{field:k}(\sigma_{field:k}(R)) = \sigma_{field:k}(R)$$

commutative: the order of the selections has no effect on the result:

$$\sigma_{field:k_1}(\sigma_{field:k_2}(R)) = \sigma_{field:k_2}(\sigma_{field:k_1}(R))$$

distributive: the selection is distributive over the setminus, intersection and union operators:

$$\sigma_{field:k}(R \gamma S) = \sigma_{field:k}(R)\ \gamma\ \sigma_{field:k}(S) \text{ where } \gamma = \cap, \cup \text{ or } \setminus$$

In general, the keyword selection condition is defined by a boolean combination of keywords using the logical operators $\wedge, \vee$ or $\neg$. Phrase selection (i.e., a sequence of keywords) is also possible using the syntax `field:"k1 k2"`. A keyword selection using a boolean combination of keywords is identical to a boolean combination of keyword selections. Also, two nested selections are equivalent to an intersection of tw selections.

$$\begin{aligned}
\sigma_{field:k_1}(R) \cap \sigma_{field:k_2}(R) &= \sigma_{field:k_1 \wedge field:k_2}(R) \\
\sigma_{field:k_1}(R) \cup \sigma_{field:k_2}(R) &= \sigma_{field:k_1 \vee field:k_2}(R) \\
\sigma_{field:k_1}(R) \setminus \sigma_{field:k_2}(R) &= \sigma_{field:k_1 \neg field:k_2}(R) \\
\sigma_{field:k_1}(\sigma_{field:k_2}(R)) &= \sigma_{field:k_1 \wedge field:k_2}(R)
\end{aligned}$$

Dataset and Entity Projection

The projection operator π allows to extract specific columns, such as *dataset* or *entity*, from a relation. For example, the expression:

$$\pi_{d,e}(R)$$

returns a relation with only two columns, dataset and entity, as shown in Figure 2.3b

The projection is idempotent, a series of projections is equivalent to the outermost projection, and is distributive over set union but not over intersection and set-difference [Mai83]:

$$\begin{aligned}
\pi_e(\sigma_{field:k_1}(R_1) \cap \sigma_{field:k_2}(R_2)) &\neq \pi_e(\sigma_{field:k_1}(R_1)) \cap \pi_e(\sigma_{field:k_2}(R_2)) \\
\pi_e(\sigma_{field:k_1}(R_1) \cup \sigma_{field:k_2}(R_2)) &= \pi_e(\sigma_{field:k_1}(R_1)) \cup \pi_e(\sigma_{field:k_2}(R_2)) \\
\pi_e(\sigma_{field:k_1}(R_1) \setminus \sigma_{field:k_2}(R_2)) &\neq \pi_e(\sigma_{field:k_1}(R_1)) \setminus \pi_e(\sigma_{field:k_2}(R_2))
\end{aligned}$$

2.4.2. Examples of Algebra Queries

Keyword Query

Q1: Find all entities matching keywords *deri* and *renaud*.

$$Q1 = \pi_e(\sigma_{v:deri}(R)) \cap \pi_e(\sigma_{v:renaud}(R)) \tag{Q1}$$

Value Query

Q2: Find all entities with a value matching keywords *renaud* and *delbru*.

$$Q2 = \pi_e(\sigma_{v:renaud}(\sigma_{v:delbru}(R))) = \pi_e(\sigma_{v:renaud \wedge v:delbru}(R)) \tag{Q2}$$

Attribute Query

Q3: Find all entities with a value matching keywords *john* and *smith* associated to an attribute matching the keyword *author*.

$$Q3 = \pi_e(\sigma_{at:author}(\sigma_{v:john \wedge v:smith}(R))) = \pi_e(\sigma_{at:author \wedge v:john \wedge v:smith}(R)) \tag{Q3}$$

Q4: Find all entities with two values, the first one matching keywords *john* and *smith* and the second one matching keywords *Peter* and *Brown*, associated to an attribute

matching the keyword *author*.

$$\begin{aligned} R_1 &= \pi_e(\sigma_{at:author}(\sigma_{v:john \wedge v:smith}(R))) \\ R_2 &= \pi_e(\sigma_{at:author}(\sigma_{v:peter \wedge v:brown}(R))) \\ Q4 &= R_1 \cap R_2 \end{aligned} \tag{Q4}$$

Star Query

Q5: Find all entities with a value matching keywords *john* and *smith* associated to an attribute matching the keyword *author*, and a value matching keywords *search* and *engine* associated to an attribute matching the keyword *title*.

$$\begin{aligned} R_1 &= \pi_e(\sigma_{at:author}(\sigma_{v:john \wedge v:smith}(R))) \\ R_2 &= \pi_e(\sigma_{at:title}(\sigma_{v:search \wedge v:engine}(R))) \\ Q5 &= R_1 \cap R_2 \end{aligned} \tag{Q5}$$

Q6: Find all entities with a value matching the keyword *acm* associated to an attribute matching the keyword *journal* or with a value matching the keyword *sigir* associated to an attribute matching the keyword *conference*, and with a value matching keywords *search* and *engine* associated to an attribute matching the keyword *title* or *abstract*.

$$\begin{aligned} R_1 &= \pi_e(\sigma_{at:publisher}(\sigma_{v:acm}(R))) \\ R_2 &= \pi_e(\sigma_{at:conference}(\sigma_{v:sigir}(R))) \\ R_3 &= \pi_e(\sigma_{at:title \vee at:abstract}(\sigma_{v:search \wedge v:engine}(R))) \\ Q6 &= (R_1 \cup R_2) \cap R_3 \end{aligned} \tag{Q6}$$

Q7: Find all entities with a value matching keywords *john* and *smith* associated to an attribute matching the keyword *author*, but not with a value matching keywords *peter* and *brown* associated to an attribute matching the keyword *author*.

$$
\begin{aligned}
R_1 &= \pi_e(\sigma_{at:author}(\sigma_{v:john \wedge v:smith}(R))) \\
R_2 &= \pi_e(\sigma_{at:author}(\sigma_{v:peter \wedge v:brown}(R))) \\
Q7 &= R_1 \setminus R_2
\end{aligned} \tag{Q7}
$$

Context Query

Q8: Find all entities from the dataset *biblio* with a value matching keywords *john* and *smith* associated to an attribute matching the keyword *author*, and a value matching keywords *search* and *engine* associated to an attribute matching the keyword *title*.

$$
\begin{aligned}
R_1 &= \pi_e(\sigma_{at:author}(\sigma_{v:john \wedge v:smith}(R))) \\
R_2 &= \pi_e(\sigma_{at:title}(\sigma_{v:search \wedge v:engine}(R))) \\
Q8 &= \pi_e(\sigma_{d:biblio}(R_1 \cap R_2))
\end{aligned} \tag{Q8}
$$

Q9: Find all datasets with two entities, the first one with a value matching keywords *john* and *smith* associated to an attribute matching the keyword *name*, and the second one with a value matching keywords *search* and *engine* associated to an attribute matching the keyword *title*.

$$
\begin{aligned}
R_1 &= \pi_d(\sigma_{at:name}(\sigma_{v:john \wedge v:smith}(R))) \\
R_2 &= \pi_d(\sigma_{at:title}(\sigma_{v:search \wedge v:engine}(R))) \\
Q9 &= \pi_d(R_1 \cap R_2)
\end{aligned} \tag{Q9}
$$

2.5. Conclusion

We presented in this chapter the data model and the boolean search model that form the framework of the information retrieval system described in this thesis. In the rest of the thesis, when we employ the term *dataset* or *entity*, we implicitly refer to the concepts and definitions presented in this chapter. In particular, the formalisation of the reasoning and ranking techniques introduced in Chapter 4 and Chapter 5 respectively are based on the data model. The indexing and querying techniques presented in Chapter 6 is based on the data and query model.

This formal data model enables to cover to a certain degree the Semantic Web and Dataspaces infrastructures. The Entity Attribute-Value model is generic enough to be compatible with various semi-structured data models such as OEM, XML or RDF. This generic framework enables the techniques presented in this thesis to be applicable on a wide variety of scenarios, and not only in the context of Web Data search. In conclusion, we provide a generic framework to simplify and unify access to a multitude of decentralised information sources.

Chapter 3.

Background

This chapter provides a review of distinct domains, each one relates to one of the methods introduced in this thesis. Section 3.1 reviews scalable reasoning techniques for web data, discusses their limitations and presents the foundations on which we developed the context-dependent reasoning technique presented in Chapter 4. Section 3.2 reviews the link analysis techniques for the Web and the Web of Data. We discuss the current limitations that are tackled by the technique introduced in Chapter 5. The remaining sections are related to Chapter 6 and Chapter 7. In Section 3.3, we examine the notion of Information Retrieval and review the current techniques for retrieval of semi-structured data. We gives an overview of the existing search query models for semi-structured data and discuss the challenges encountered in building them. Section 3.4 introduces the inverted index, the data structure at the basis of the indexing system presented in Chapter 6. Section 3.4.3 introduces the compression techniques for inverted indexes that are compared with our technique in Chapter 7.

3.1. Reasoning over Web Data

Reasoning over semi-structured entity description enables to make explicit what would otherwise be implicit knowledge: it adds value to the information and enables an entity-centric search engine to ultimately be more competitive in terms of precision and recall [MF03]. The drawback is that inference can be computationally expensive, and therefore drastically slow down the process of indexing large amounts of information. Therefore, large scale reasoning through parallelisation is one requirement of information retrieval systems for web data.

Moreover, we can not expect web data to always adhere to strict rules. Web data is highly heterogeneous and unexpected usage of data and data schema is common. For example, data can be erroneous or crafted for malicious purposes. As a consequence, there is a risk for a reasoner to infer undesirable logical assertions which is harmful for the Information Retrieval system. These assertions increase the noise in the data collection and decrease the precision of the system, loosing the benefits that reasoning should provide. In addition, such inferences add an unnecessary computational overhead which augments the demand of computational resources and limits the performance of the system. Therefore, a second requirement of inference engines for web data is the ability to cope with versatile data quality.

In this section, we first provide an overview of the current approaches for large scale reasoning over web data. Next, we review existing contextual reasoning approaches and other works related to our context-dependent reasoning technique. Finally, we introduce the concept of *Context* on the Semantic Web and its formal framework as it is defined by Guha [GMF04]. This framework is used in Chapter 4 in the development our context-dependent reasoning mechanism.

3.1.1. Distributed Reasoning

Recently, we have seen major interests towards large scale reasoning based on parallel distribution of inference. [WH09, UKOvH09] present a parallel RDFS reasoning. In [UKM+10], they extend their approach to cover the OWL Horst fragment [tH05]. SAOR [HHP09, HPPD10] aims to cover a larger subset of OWL2RL.

In these approaches, the separation of terminological data from assertional data, which are commonly known as the *T-Box* and *A-Box* respectively[1], is a key point for parallel inference. The separation of T-Box from A-Box allows to distribute reasoning among a cluster of machines more easily. Since the T-Box is relatively small, the T-Box is replicated to each cluster node. The A-Box is partitioned and distributed among the nodes. Each node calculates the closure of its A-Box partition.

With respect to computational optimisation, [UKOvH09, HHP09, HPPD10] have identified properties about the ordering of rules in order to avoid multiple passes over the data when possible. In addition, [UKM+10] proposes a fixpoint iteration to compute inference of rules that requires multiple A-Box joins. SAOR avoids rules which require

[1] slightly abusing Description logics terminology

A-Box joins and consequently is less complete than [UKM+10]. [WH09] does not need A-Box joins since it is restricted to RDFS where there are no dependencies between partitions.

With respect to IO load, [UKM+10, HHP09, HPPD10] which deal with richer logics than RDFS necessitate generally more than one pass over the data. This means that the data is read multiple times from disk. For very large data collection, this is time and resource consuming. With respect to inference maintenance, [UKM+10, HHP09, HPPD10] do not have the possibility of updating the inferences. If the data changes, the full closure has to be computed again. These approaches are therefore unsuitable for dynamic systems such as web search engines where data is updated continuously.

In our context-dependent reasoning technique, we also separate T-Box from A-Box, but compared to the previous approaches, we partition and distribute the A-Box on a per context fashion. The statements in each A-Box partition belong to a same context, e.g., a document and its related documents through implicit or explicit imports. Each A-Box partition is small enough to be fully loaded in memory. We can then perform conventional reasoning in memory and avoid to write and read intermediate results on disk. In addition, the notion of context enables efficient updates. If the data changes, then only the closure of the related contexts has to be recomputed.

Although [WH09, UKOvH09, UKM+10] have demonstrated to scale in the order of hundreds of millions or hundreds of billions triples, their experiments are focussed on scalability issues, disregarding the problem of data quality and its consequences. Only SAOR proposes a technique called *Authoritative Reasoning* to avoid undesirable inference results. Their system performs an analysis of the authority of web data sources, and imposes rule restrictions if non-authoritative T-Box statements are detected. Our context-dependent reasoning technique adopts a completely different approach against the problem of noisy data and undesirable inferences. Instead of imposing rule restrictions, we consider a context as a closed world where inferences are "quarantined" within the context.

3.1.2. Contextual Reasoning

The notion of context has been extensively studied since early 1990s, starting with a first formalisation by Guha [Guh92] followed by McCarthy [McC93] and by Giunchiglia [Giu93]. For more information about the domain, [SB04] presents a survey and comparison of

the main formalizations of context. There are also recent works that adapt context formalisations to the Semantic Web such as [BGvH+04, GMF04].

The previous references provide a strong theoretical background and framework for developing a context-dependent reasoning mechanism for the Semantic Web. Up to our knowledge, only [SBPR07] propose a concrete application of management of contextualized RDF knowledge bases. But no one has described a methodology for efficient large scale reasoning that deals with contextuality of web documents.

Assumption-based Truth Maintenance Systems [dK86] are somehow comparable to our approach since such a system keeps track of dependency between facts in multiple views (contexts) in order to maintain consistency in a knowledge base.

A side remark in [DTM08] suggests that the authors follow a very similar strategy to ours in determining the ontological closure of a document (see Section 4.2.1), but no details on efficient contextual confinement and reuse of inferences, which are the main contributions of Chapter 4, are discussed.

3.1.3. Contexts on the Semantic Web

The URI of a dataset or of an entity description provides a natural way to define a context for the data it contains. Naming an RDF graph has multiple benefits [CBHS05]. It helps tracking the provenance of each statement. In addition, since named graphs are treated as first class objects, this enables the description and manipulation of a set of statements just as for any other resource. In this section, the notion of context refers to the entity URI at which the entity description is retrievable. For our purposes, it is sufficient to understand, in this section, that by *context* we identify the set of statements of an entity description.

Guha [GMF04] proposed a context mechanism for the Semantic Web which provides a formal basis to specify the aggregation of contexts. Within his framework, a context denotes the scope of validity of a statement. This scope is defined by the symbol *ist* ("is true in context"), introduced by Guha in [Guh92]. The notation $ist(c, \varphi)$ states that a proposition φ is true in the context c.

The notions of context and lifting rules presented in the following are based on Guha's context mechanism [GMF04]. Its aim is to control the integration of data and ultimately avoid the aggregation of data that may result in undesirable inferred assertions.

Context

In Guha's formalism, a context is a first class resource, instance of the class *Context*. Each data source (or collection of data sources) is abstracted into a context. The contents of the data source(s) are said to be true in that context.

Aggregate Context

An *Aggregate Context* is a subclass of *Context*. Its content is composed by the content retrieved by resolving its URI, and by the contents lifted from other contexts. An aggregate context must contain the full specification of what it imports. In our case, each entity description is considered an *Aggregate Context*, since it always contains the specification of what it imports through explicit or implicit import declarations, as explained in Section 4.2.1.

Lifting Rules

Since contexts are first class objects, it becomes possible to define expressive formulae whose domains and ranges are contexts. An example is the so called *Lifting Rule* that enables to *lift* axioms from one context to another. In Guha's context mechanism, a lifting rule is a property type whose domain is an *Aggregate Context* and range a *Context*. A lifting rule can simply import all of the statements from a context to another, but can also be defined to select precisely which statements have to be imported.

Import Lifting Rules

Guha's context mechanism defines the *importsFrom* lifting rule. It is the simplest form of lifting and corresponds to the inclusion of one context into another.
Definition 3.1. Let c_2 be a context with a set of propositions $\varphi(x)$, and c_1 a context with an *owl:imports* declaration referencing c_2. Then the set of propositions $\varphi(x)$ is also true in the context c_1:

$$ist(c_2, \varphi(x)) \wedge ist(c_1, importsFrom(c_1, c_2)) \rightarrow ist(c_1, \varphi(x))$$

3.2. Link Analysis for Ranking Web Resources

3.2.1. PageRank

PageRank [PBMW99] is a ranking system that originates from Web search engines. The ranking system, based on a random walk algorithm, evaluates the probability of finding a random web surfer on any given page. The algorithm assumes an hyperlink from a page i to a page j as an evidence of the importance of page j. In addition, the amount of importance that i is conferring to j is determined by the importance of i itself and inversely proportional to the number of pages i points to. PageRank can easily be adapted to our data model from Section2.3. By regarding pages as entities and hyperlinks as links between entities, we can formulate PageRank as follows:

Let $L(i) = \{target(l) | \forall l \in A, source(l) = i\}$ be the set of entities linked by an entity $i \in V^E$ and $B(j) = \{source(l) | \forall l \in A, target(l) = j\}$ be the set of entities that points to $j \in V^E$. The PageRank $r(j)$ of an entity $j \in E$ is given by:

$$r^k(j) = \alpha \sum_{i \in B(j)} \frac{r^{k-1}(i)}{|L(i)|} + \frac{(1-\alpha)}{|V^E|} \,. \tag{3.1}$$

A fixed-point iteration approach is commonly used where the computation of a rank score $r^k(j)$ at a step k uses the rank scores of the step $k-1$. The operation is repeated until all scores r stabilise to within some threshold.

The PageRank formula is composed of two parts weighted by a damping factor α, usually set to 0.85. The first component provides the contribution $\sum_{i \in B_j} \frac{r^{k-1}(i)}{|L(i)|}$ of incoming links to entity j. The factor $\frac{1}{|L(i)|}$ defines a uniform distribution of the importance of entity i to all the entities i points to. This distribution can be modified in order to provide a distribution based on the weight of a link. The second component $\frac{1}{|V^E|}$ denotes the probability that the surfer will jump to j from any other random entity from the collection.

3.2.2. Weighted Link Analysis

When working with more heterogeneous links, standard approaches do not provide accurate results since links of different types can have various impact on the ranking computation. In [XG04, BYD04], the authors extend PageRank to consider different

types of relations between entities. PopRank [NZWM05], an object-level link analysis, proposes a machine learning approach to assign a "popularity propagation factor" to each type of relation. ObjectRank [BHP04] applies authority-based ranking to keyword search in databases. However, these works do not consider the features of links such as their specificity and cardinality to assign weights in an unsupervised fashion as we propose in Chapter 5. Furthermore, these approaches are too complex to apply on web-scale since they require multiple times the current processing power of current web search engines. A major task is to bring down this computational power requirement.

3.2.3. Hierarchical Link Analysis

Recent works [KHMG03, EAT04, WD04, XYZ+05, FLW+06, BLMP06] exploit the hierarchical data structure of distributed environments and of the Web. [KHMG03] suggests a hierarchical model of the Web based on the notions of domains, directories and pages, and shows the desirable computational properties of such an approach. In [XYZ+05], the authors show that hierarchical ranking algorithm outperforms qualitatively other well-known algorithms, including PageRank. However, such models have never been applied on semi-structured data sources with distinct semantics and none of them are considering weighted relations between supernodes as we propose in Chapter 5.

3.2.4. Semantic Web Link Analysis

SemRank [AMS05] proposes a method to rank semantic relations using information-theory techniques but is solely focus on ranking and retrieval of relationships. The Swoogle search engine [DPF+05b] is the first one to propose an adaptation of PageRank for Semantic Web resources, OntoRank. In ReconRank [HHD06], a link analysis is applied at query time for computing the popularity of resources and documents. The above algorithms only consider the individual web resources, disregarding the semantics and structure of the Web of Data. They are therefore costly on a web-scale and provide sub-optimal results.

3.3. Indexing and Searching Semi-Structured Data

On heterogeneous information sources, heterogeneity is a common feature that characterises the datasets and makes impractical exact queries due to the lack of knowledge of (1) the vocabularies used and, (2) how data is organized. Essentially, the user should be able to express his search needs by means of keywords combined with boolean operators and to include varying degrees of structure in his search queries, so that he can better specify his needs according to the partial knowledge of the data schema he may have.

Searching as it has been defined in Chapter 2 is a much simpler form of querying compared to database query languages where complex join queries can be expressed. There is a need to clarify what is feasible and usable [JCE+07] in terms of search over semi-structured datasets. These raise new challenges such as the definition of usable and effective models for searching semi-structured data, as well as efficient algorithms and data structures to support their applicability on a web-scale scenario.

While there is a large spectrum of techniques for indexing and searching semi-structured data allowing more or less query expressiveness, none of them have dealt with the issue of designing algorithms and data structures for large scale processing. Each technique has its advantages and inconvenients: some techniques are very efficient to answer complex queries, but are costly to scale, while others provide less flexibility in their query language but are more suitable for large-scale scenarios. There is a need to investigate these indexing schemes and to design an index that provides the best compromise between query expressiveness and web scalability.

In this section, we first give a retrospective of traditional retrieval systems and retrieval systems for structured documents, before presenting an overview of the current approaches for indexing and querying RDF data. We then review existing search models for semi-structured and structured data and the various indexing techniques to support them.

3.3.1. Traditional Retrieval Systems

Traditional Information Retrieval systems are based around the concepts of text and document. An Information Retrieval system typically indexes a set of documents where the document content is a distribution of terms over a vocabulary. Given a query

composed by a combination of terms, Information Retrieval engines are able to efficiently retrieve a list of documents ranked by their relevance with respect to the query.

Information Retrieval is a well studied field, and techniques such as inverted indexes have proved to scale to the size of the Web. Many optimisations [BYRN99, WMB99, BYCJ+07, MRS08] have been developed for efficiently storing and querying a large amount of information.

The problem with the traditional model is its inability to capture the structure of a document, since a document is seen as a "bag of words". In order to query the content as well as the structure of the document, a large number of techniques for structured text retrieval has been developed.

3.3.2. Retrieval of Structured Documents

In the past decades, many models for textual database have been developed to support queries integrating content (words, phrases, etc.) and structure (for example, the table of contents). Amongst them, we can cite the hybrid model, PAT expressions, overlapped lists and proximal nodes. We refer the reader to [BYN96] for a comparison of these models. By mixing content and structure, more expressive queries become possible such as relations between content and structural elements or between structural elements themselves. For example, it becomes possible to restrict the search of a word within a chapter or a section, or to retrieve all the citations from a chapter.

With the increasing number of XML documents published on the Web, new retrieval systems for the XML data model have been investigated. Certain approaches [GSBS03, CMKS03, LWC07] have investigated the use of keyword-based search for XML data. Given a keyword query, these systems retrieve document fragments and use a ranking mechanism to increase the search result quality. However, such systems are restricted to keyword search and do not support complex structural constraints. Therefore, other approaches [STW05, TSW05] have investigated the use of both keyword-based and structural constraints. In the meantime, various indexing techniques have been developed for optimising the processing of XPath query, the standardised query language for XML [WFM+07]. The three major techniques are the node index scheme [LM01, HWYY05, HHJ+06], the graph index scheme [GW97, CSF+01, WJW+05], and the sequence index scheme [WPFY03, RM04, MJCW04, FLMM06, Gri08]. Node index scheme relies on node labelling techniques [SCCS09] to encode the tree structure of an XML

document in a database or in an inverted index. Graph index schemes are based on secondary indexes that contain structural path summaries in order to avoid join operations during query processing. Sequence index schemes encode the structure into string sequences and use string matching techniques over these sequences to answer queries.

More recently, the Database community has investigated the problem of search capabilities over semi-structured data in Dataspaces [HFM06]. [DH07] proposes a sequence index scheme to support search over loosely structured datasets using conditions on attributes and values.

3.3.3. Retrieval of RDF Data

With the growth of RDF data published on the Semantic Web, Semantic Web applications demand scalable and efficient RDF data management systems. Different RDF storage strategies have been proposed based on RDBMS [BG03], using native index [HUHD07, BB07, WKB08, NW08] or lately using column-oriented DBMS using vertical partitioning over predicates [AMMH07].

RDF data management systems excel in storing large amount of RDF data and are able of answering complex conjunctive SPARQL queries involving large joins. Typical SPARQL queries are graph patterns combining SQL-like logical conditions over RDF statements with regular-expression patterns. The result of a query is a set of subgraphs matching precisely the graph pattern defined in the query. This search and retrieval paradigm is well adapted for retrieving data from large RDF datasets with a globally known schema such as a social network graph or a product catalog database. However, such approaches are of very limited value when it comes for searching over highly heterogeneous data collections, for example where data comes from many information sources, each one defining its own schema. This kind of environment is typical on the Semantic Web or in Dataspaces. The variance in the data structure and in the vocabularies makes it difficult to write precise SPARQL queries. In addition, none of the above approaches uses structural and content similarity between the query and the data graph for ordering results with respect to their estimated relevance with the query. This latter aspect is the rationale behind Web and enterprise search systems and is often considered as the only appropriate search paradigm.

Also, RDF data management systems are difficult to deploy on a very large scale. A native RDF quad index[2] is used to lookup any combination of s, p, o, c directly, which means the index is a data structure based on the notion of quad patterns. A naive implementation would need $2^4 + 1 = 17$ indices, one for each quad patterns and one inverted index for full-text search on literals. If prefix lookups are supported by the index, only 6 indices are necessary for implementing the 16 quad patterns [HUHD07, BB07]. Due to their need of large number of indexes for efficient query processing, they are (1) difficult to deploy on a large scale and (2) computationally expensive to update.

Other approaches [GMM03, DPF+05a, HHUD08, CGQ08] carried over keyword-based search to Semantic Web and RDF data in order to provide ranked retrieval using content-based relevance estimation. In addition, such systems are more easy to scale due to the simplicity of their indexing scheme. However, such systems are restricted to keyword search and do not support structural constraints. These systems do not consider the rich structure and semantic annotations provided by RDF data. A first step towards a more powerful search interface combining imprecise keyword search with precise structural constraints has been investigated by Semplore [WLP+09]. Semplore [WLP+09] has extended inverted index to encode RDF graph approximations and to support keyword-based tree-shaped queries over RDF graphs. However, the increase of query capabilities comes at a cost, and the scalability of the system becomes limited.

3.3.4. Search Models for Semi-Structured Data

In addition to standardised query languages such as SPARQL, XQuery or XPath, a large number of search models for semi-structured data can be found in the literature. Some of them focus on searching structured databases [ACD02, BHN+02, HP02, LYMC06], XML documents [CMKS03, GSBS03, LWC07] or graph-based data [AQM+96, KPC+05, LOF+08] using simple keyword search. Simple keyword-based search has the advantages of (1) being easy to use by users since it hides from the user any structural information of the underlying data collection, and (2) of being applicable on any scenarios. On the other hand, the keyword-based approach suffers from limited capability of expressing various degrees of structure when users have a partial knowledge about the data structure.

Other works [KSI+08, MMVP09, WLP+09, DH07, FBGV09] have extended simple keyword-based search with semantic queries capabilities. [DH07, FBGV09] propose a

[2]specifically a quad is a statement s, p, o with a fourth element c called "context" for naming the RDF graph, generally to keep the provenance of the RDF data.

partial solution to the lack of expressiveness of the keyword-based approach by allowing search using conditions on attributes and values. [KSI+08, MMVP09, WLP+09] present more powerful query language by adopting a graph-based model. However, it is unclear what is the model to adopt to maximise the usability, if a simpler method such as [DH07] is sufficient for searching over loosely structured data or if more complex models are required. Moreover, the increase of query expressiveness is tied with the processing complexity, and the graph-based models [KSI+08, MMVP09, WLP+09] are not applicable on a very large scale.

The search model introduced in Chapter 2 is similar to [DH07, FBGV09], i.e., it is defined around the concept of attribute-value pairs. However, our model is more expressive since it differentiates between single and multi-valued attributes and it considers the provenance of the information.

3.4. Inverted Indexes and Inverted Lists

In this section, we first introduce the inverted index which is the data structure at the core of the indexing technique presented in Chapter 6. We then describe a list of index compression techniques, along with their implementation. The compression algorithms will be compared with our compression technique in Chapter 7.

3.4.1. Inverted Index Structure

An inverted index is composed of (1) a lexicon, i.e., a dictionary of terms that allows fast term lookup; and (2) of an inverted index, containing one inverted list per term. An inverted list is a stream of integers, composed of a list of document identifiers that contain the term, a list of term frequencies in each document and a list of term positions in each document at which the term appears.

In a interleaved index [AM06], the inverted index is composed of a single inverted file where the inverted lists are stored contiguously. A pointer to the beginning of the inverted list in the inverted file is associated to each term in the lexicon. Therefore, the inverted list of a term can be accessed efficiently by performing a single term lookup on the lexicon. For performance and compression efficiency, it is best to store separately each data stream of an inverted list [AM06]. In a non-interleaved index organisation, the

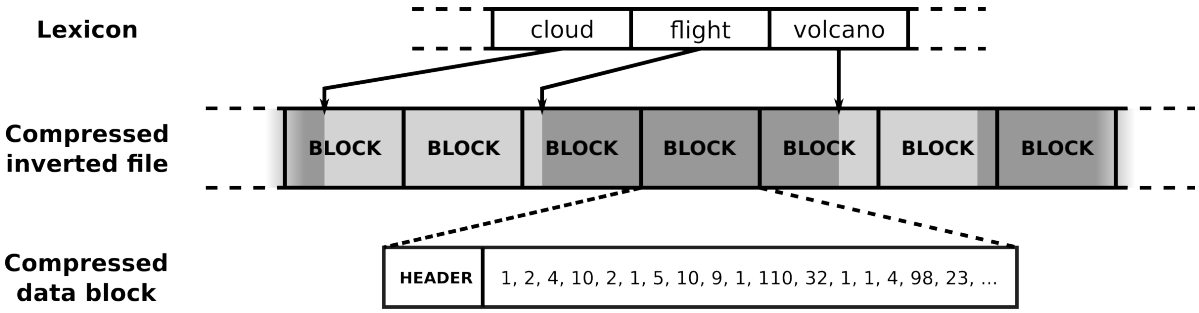


Figure 3.1.: Inverted index structure. Each lexicon entry (term) contains a pointer to the beginning of its inverted list in the compressed inverted file. An inverted file is divided into blocks of equal size, each block containing the same number of values. A compressed block is composed of a block header followed by a compressed list of integers.

inverted index is composed of three inverted files, one for each inverted lists, i.e., list of documents, term frequencies and term positions. Each inverted file stores contiguously one type of list, and three pointers are associated to each term in the lexicon, one pointer for each inverted file.

We now describe the implementation[3] of an inverted file on disk. An inverted file is partitioned into blocks, each block containing a fixed number of integers as shown in Figure 3.1. Blocks are the basic units for writing data to and fetching data from disk, but also the basic data unit that will be compressed and decompressed. A block starts with a block header. The block header is composed of the length of the block in bytes and additional metadata information that is specific to the compression technique used. Long inverted lists are often stored across multiple blocks, starting somewhere in one block and ending somewhere in another block, while multiple small lists are often stored into a single block. For example, 16 inverted lists of 64 integers can be stored in a block of 1024 integers[4]. We use blocks of 1024 integers in our experiments with respect to the CPU cache, since this was providing the best performance. The overall performance of all the compression techniques were decreasing with smaller block sizes.

[3]This implementation is the one found in the open-source project Apache Lucene.

[4]In that case, the compression techniques are independent of the size of the inverted lists. Therefore, compared to [ZLS08], we do not have to rely on secondary compression techniques for short inverted lists.

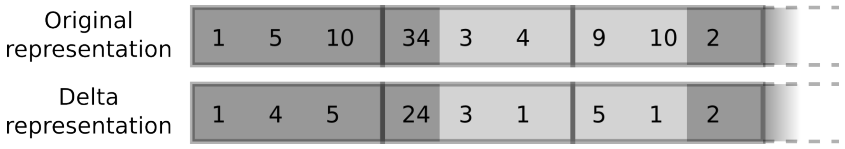


Figure 3.2.: Inverted file with blocks of size 3 using the delta encoding method. The upper inverted file contains the three original lists of integers (1 to 34, 3 to 10, and 2 to ...). The lower inverted file contains the three lists in delta code.

3.4.2. Delta Encoding of Inverted Lists

Inverted lists represent an important proportion of the total index size and are generally kept on disk. It is desirable to use compression techniques in order to reduce disk space usage and transfer overhead. On one hand, efficient decompression can provide faster transfer of data from disk to memory, and therefore faster processing, since the time of fetching and decompressing a data segment is less than fetching an uncompressed form [BC07]. On the other hand, efficient compression can provide faster indexing since the time of compressing and writing a data segment is less than the time to write an uncompressed form, as we will show in Chapter 7. To summarise, compression is useful for saving disk space, but also to maximise IO throughput and therefore to increase update and query throughput.

An inverted index is composed of inverted lists, each one being an ordered list of integers. The main idea of index compression is to encode the list of integers using as few bits as possible. Instead of storing the raw integer in a 32 bit machine word, the goal is to store each integer using the smallest number of bits possible.

One common practice [MRS08] is to rely on "delta encoding". The delta encoding technique stores the difference between the previous integer and the current one in the list, instead of storing the raw integer. This allows to encode an ordered list of integers using much smaller integers, which theoretically can be encoded in less bits. Delta encoding with block-based inverted files is depicted in Figure 3.2. Compression techniques are then used to encode these integers with the smallest number of bits possible.

3.4.3. Algorithms for Compressing Inverted Lists

We describe next the five compression algorithms selected for the experiments and discuss their implementation. With respect to the Binary Interpolative Coding [MS00], it has

been shown to provide a very good compression rate and it could have been a reference for comparing compression rates. However, it is very inefficient in decoding, and we found that Rice is competitive enough in term of compression rate to use it as a reference.

Rice Coding

In Rice, an integer n is encoded in two parts: a quotient $q = \lfloor \frac{n}{2^b} \rfloor$ and a remainder $r = n \bmod 2^b$. The quotient is stored in unary format using $q+1$ bits while the remainder is store in binary format using b bits. In our implementation, the parameter b is chosen per block such that 2^b is close to the average value of the block.

The main advantage of Rice is its very good compression ratio. However, it is in general the slowest method in term of compression and decompression. The main reason is that Rice needs to manipulate the unary word one bit at a time during both compression and decompression, which is very demanding in CPU cycles.

Variable Byte Coding (VByte)

Variable Byte compression encodes an integer with a variable number of bytes. VByte is byte-aligned, i.e., coding and decoding is done one byte at a time. Each byte consists of 7 bits to encode the partial binary representation of the integer, and one bit used as status flag to indicate if the following byte is part of the current number. For example, the number $298 = 1 \cdot 2^8 + 42 \cdot 2^0$ is encoded using two bytes **1**0000001 **0**0101010. The most significant bit **1** of the first byte indicates that the next byte is part of the current integer and must be decoded. The most significant bit **0** of the second byte indicates that there is no more bytes to decode for the current integer.

The advantages of VByte are: (1) it is simple to implement; and (2) its overall compression and decompression performance are good. Compared to bitwise techniques like Rice, VByte requires a single branching condition for each byte which is more cost-effective in term of CPU cycles. However, the branching condition leads to branch mispredictions which makes it slower than CPU optimised techniques such as the one presented next. Moreover, VByte has a poor compression ratio since it requires one full byte to encode one small integer (i.e., $\forall n < 2^7$).

Simple Coding Family

The idea behind the Simple coding is to pack as many integers as possible into one machine word (being 32 or 64 bits). We describe one Simple coding method (referred to as S-64 in our experiments) based on 64-bit machine words, recently introduced in [AM10]. In our experiments, we report only S-64 results since its overall performance was always superior to Simple9 [AM05]. In S-64, each word consists of 4 status bits and 60 data bits. The 4 status bits are used to encode one of the 16 possible configurations for the data bits. A description of the 16 configurations can be found in [AM10]. S-64 wastes generally less bits than Simple9 and therefore provides a better compression ratio.

While providing good compression ratio, decompression is done efficiently by reading one machine word at a time and by using a precomputed lookup table over the status bits in order to select the appropriate optimised routine (one routine per configuration) to decode the data bits using shift and mask operations only.

One disadvantage is that compression cannot be done efficiently. The typical implementation is to use a sliding window over the stream of integers and to find the best configuration, i.e., the one providing the best compression ratio, for the current window. This generally requires repetitive try and error iterations over the possible configurations at each new window. In addition, Simple coding has to perform one table lookup per machine word which costs more CPU cycles than the other techniques presented next.

Frame of Reference (FOR)

FOR determines the range of possible values in a block, called a *frame*, and maps each value into this range by storing just enough bits to distinguish between the values [GRS98]. In the case of the delta-encoded list of values, since the probability distribution generated by taking the delta tends to be naturally monotonically decreasing, one common practice [ZHNB06, AM10] is to choose as frame the range $[0, max]$ where max is the largest number in the group of delta values.[5]

Given a frame $[0, max]$, FOR needs $\lceil \log_2(max + 1) \rceil$ bits, called *bit frame* in the rest of the paper, to encode each integer in a block. The main disadvantage of FOR is that it is sensitive to outliers in the group of values. For example, if a block of 1024 integers

[5]This assumes that a group of values will always contain 0, which is not always the case. However, we found that taking the real range $[min, max]$ was only reducing the index size by 0.007% while increasing the complexity of the algorithm.

contains 1023 integers inferior to 16, and one value superior to 128, then the bit frame will be $\lceil \log_2(128+1) \rceil = 8$, wasting 4 bits for each other values.

However, compression and decompression is done very efficiently using highly-optimised routines [ZHNB06] which avoid branching conditions. Each routine is loop-unrolled to encode or decode m values using shift and mask operations only. Listing 3.1 and 3.2 show the routines to encode or decode 8 integers with a bit frame of 3. There is a compression and decompression routines for each bit frame.

Given a block of n integers, FOR determines a frame of reference for the block and encodes the block by small iterations of m integers using the same compression routine at each iteration. Usually, and for questions of performance, m is chosen to be a multiple of 8 so that the routines match byte boundaries. In our implementation, FOR relies on routines to encode and decode 32 values at a time.

The selection of the appropriate routine for a given bit frame is done using a pre-computed lookup table. The compression step performs one pass only over the block to determine the bit frame. Then, it selects the routine associated to the bit frame using the lookup table. Finally, the bit frame is stored using one byte in the block header and the compression routine is executed to encode the block. During decompression, FOR reads the bit frame, performs one table lookup to select the decompression routine and executes iteratively the routine over the compressed block.

```
compress3(int[] i, byte[] b)
  b[0] = (i[0] & 7)
       | ((i[1] & 7) << 3)
       | ((i[2] & 3) << 6);
  b[1] = ((i[2] >> 2) & 1)
       | ((i[3] & 7) << 1)
       | ((i[4] & 7) << 4)
       | ((i[5] & 1) << 7);
  b[2] = ((i[5] >> 1) & 3)
       | ((i[6] & 7) << 2)
       | ((i[7] & 7) << 5);
```

Listing 3.1: Loop unrolled compression routine that encodes 8 integers using 3 bits each

```
decompress3(byte[] b, int[] i)
  i[0] = (b[0] & 7);
  i[1] = (b[0] >> 3) & 7;
  i[2] = ((b[1] & 1) << 2)
       | (b[0] >> 6);
  i[3] = (b[1] >> 1) & 7;
  i[4] = (b[1] >> 4) & 7;
  i[5] = ((b[2] & 3) << 1)
       | (b[1] >> 7);
  i[6] = (b[2] >> 2) & 7;
  i[7] = (b[2] >> 5) & 7;
```

Listing 3.2: Loop unrolled decompression routine that decodes 8 integers represented by 3 bits each

Patched Frame Of Reference (PFOR)

PFOR [ZHNB06] is an extension of FOR that is less vulnerable to outliers in the value distribution. PFOR stores outliers as exceptions such that the frame of reference $[0, max]$

is greatly reduced. PFOR first determines the smallest max value such that the best compression ratio is achieved based on an estimated size of the frame and of the exceptions. Compressed blocks are divided in two: one section where the values are stored using FOR, a second section where the exceptions, i.e., all values superior to max, are encoded using 8, 16 or 32 bits. The unused slots of the exceptions in the first section are used to store the offset of the next exceptions in order to keep a linked list of exception offsets. In the case where the unused slot is not large enough to store the offset of the next exceptions, a *compulsive exception* [ZHNB06] is created.

For large blocks, the linked list approach for keeping track of the position of the exceptions is costly when exceptions are sparse since a large number of compulsory exceptions has to be created. [ZHNB06] proposes to use blocks of 128 integers to minimise the effect. [YDS09] proposes a non-compulsive approach where the exceptions are stored along with their offset in the second block section. We choose the latest approach since it has been shown to provide better performance [YDS09]. During our experimentations, we tried PFOR with frames of 1024, 128 and 32 values. With smaller frames, the compression rate was slightly better than the compression rate of AFOR-1 (which will be presented next). However, the query time performance was decreasing. We therefore decided to use PFOR with frames of 1024 values as it was providing a good reference for query time.

The decompression is performed efficiently in two phases. First, the list of values are decoded using the FOR routines. Then, the list of values is *patched* by: (1) decompressing the exceptions and their offsets and (2) replacing in the list the exception values. However, the compression phase cannot be efficiently implemented. The main reason is that PFOR requires a complex heuristic that require multiple passes over the values of a block in order to find the frame and the set of exceptions providing the highest compression.

Vector of Split Encoding

At the time of the writing, we came to knowledge of the existence of a novel work, the Vector of Split Encoding (VSE) [VS10], which is similar to the Adaptive Frame of Reference (AFOR) technique presented in Chapter 7. The two methods can be considered as an extension of FOR to make it less sensitive to outliers by adapting the encoding to the value distribution. To achieve this, the two methods are encoding a list of values by partitioning it into frames of variable lengths, and rely on algorithms to automatically find the list partitioning.

Apart from the minor differences in the implementation design, the main difference is that VSE is optimised for achieving a high compression rate and a fast decompression but disregards the efficiency of compression, while our method is optimised for achieving a high compression rate, a fast decompression but also a fast compression speed. VSE is using a Dynamic Programming method to find the optimal partitioning of a list. While this approach provides a higher compression rate than the one proposed in AFOR, the complexity of such a partitioning algorithm is $O(n \times k)$, with the term n being the number of values and the term k the size of the larger frame, which greatly impacts the compression performance. In AFOR, we use instead a local optimisation algorithm that is less effective but faster to compute.

Part III.

Methods

Chapter 4.

Context-Dependent Reasoning for Web Data*

4.1. Introduction

This chapter is concerned on how to efficiently reason over a very large collection of entity descriptions that have been published on the Web. As discussed in Section 3.1, reasoning over semi-structured data increases the precision and recall of an information retrieval system. However, due to the scale, the heterogeneity and the dynamic nature of web data, a reasoning engine for web information retrieval system must (1) scale out through parallelisation over a cluster of machines; (2) cope with unexpected data usage; and (3) maintain coherent inferences against data updates.

A common strategy for reasoning with web data is to put several entity descriptions together and to compute the deductive closure across all the entity descriptions. While this approach is computationally convenient, it turns out to be sometimes inappropriate since the integration of information from different data sources can lead to unexpected results. For example, if an ontology other than the FOAF vocabulary itself extends `foaf:knows` as a symmetric property, an inferencing agent should not consider this axiom outside the scope of the entity description that references this particular ontology. Not doing so would severely decrease the precision of the system by diluting the search results with many false positives.

Moreover, a system is unable to efficiently maintain coherent inferences when data changes with such a strategy. The typical approach is to recompute the complete

*This chapter is partially based on [DPTD08]

deductive closure whenever the data is modified. This is unsuitable for dynamic systems such as web search engines where data updates are expected to arrive continuously.

For these reasons, a fundamental requirement has been to confine T-Box assertions and reasoning tasks into contexts in order to track the provenance of inference results. By tracking the provenance of each single T-Box assertion, we are able to prevent one ontology to alter the semantics of other ontologies on a global scale. In addition, we are able to find the logical consequences that are altered by a change in the data and therefore to maintain coherent inferences.

In Section 4.2, we introduce the core concepts our approach. Section 4.3 describes a context-dependent TBox, called an ontology base. The conceptual model of the ontology base is detailed and our query and update strategies are discussed. Section 4.4 discusses the distributed implementation and Section 4.5 an empirical evaluation before concluding in Section 4.7.

4.2. Context-Dependent Reasoning of RDF Models

It would be practically impossible, and certainly incorrect, to apply reasoning over a single model composed of all the data found on the Web. Letting alone the computational complexity, the problem here is clearly the integration of information from different sources: data is published in a particular context, and a naive integration of information coming from different contexts can result in undesirable inferences.

We therefore ensure that the context is maximally preserved when reasoning with web data. We proceed in a way that we consider to be a form of *divide-and-conquer* approach. Instead of considering one single model during reasoning, we divide the model into smaller independent models, each one representing a particular "context". For example, instead of taking a single model composed of various datasets, we can divide it on a per-entity basis where each context represents an entity description. However, the notion of context is flexible and is not limited to entity description. We could instead define a dataset or a view as being a context.

Each context is considered as a closed world where inferencing results are confined, therefore limiting the scope of undesirable inferences. In addition, it becomes easier to deal with data updates. When an entity description changes, the reasoning engine has to recompute the inferences of the associated context only.

To reason on contexts, we assume that the ontologies that these contexts refer to are either included explicitly with `owl:imports` declarations or implicitly by using property and class URIs that link directly to the data describing the ontology itself. This later case should be the standard if the W3C best practices [MBS08] for publishing ontologies and the Linked Data principles [BL06] are followed by data publishers. As ontologies might refer to other ontologies, the import process then needs to be recursively iterated as explained in Section 4.2.1.

A naive approach would be to execute such a recursive fetching for each entity description and to create an aggregate context (see Section 3.1.3 for a definition) composed by the original description plus the imported ontologies. At this point the deductive closure of the aggregate context can be computed as explained in Section 4.2.2.

Such a naive procedure is however obviously inefficient since a lot of processing time will be used to recalculate the T-Box deductions which could be instead reused for possibly large numbers of other entity descriptions. Therefore, we propose a mechanism to store and reuse such deductions in Section 4.3.

4.2.1. Import Closure of RDF Models

On the Semantic Web, ontologies are published in order to be easily reused by third parties. OWL provides the `owl:imports` primitive to indicate the inclusion of a target ontology inside an RDF model. Conceptually, importing an ontology brings the content of that ontology into the RDF model.

The `owl:imports` primitive is transitive. That is, an import declaration states that, when reasoning with an ontology O, one should consider not only the axioms of O, but the entire *import closure* of O. The *import closure* of an ontology O is the smallest set containing the axioms of O and of all the axioms of the ontologies that O imports. For example, if ontology O_A imports O_B, and O_B imports O_C, then O_A imports both O_B and O_C.

Implicit Import Declaration

The declaration of `owl:imports` primitives is a not a common practice on the Semantic Web. Most published RDF models do not contain explicit `owl:imports` declarations.

For example, among the 110 million of documents in Sindice, only 650 thousands are declaring at least one `owl:imports`.

Instead, RDF models generally refer to classes and properties of existing ontologies by their URIs. For example, most FOAF profile documents do not explicitly import the FOAF ontology, but instead just refer to classes and properties of the FOAF vocabulary. Following the W3C best practices [MBS08] and Linked Data principles [BL06], the URIs of the classes and properties defined in an ontology should be resolvable and should provide the machine-processable content of the vocabulary.

In the presence of dereferenceable class or property URIs, we perform what we call an *implicit import*. By dereferencing the URI, we attempt to retrieve a *view* containing the description of the ontological entity and to include its content inside the source RDF model.

Also the implicit import is considered transitive. For example, if a RDF model refers to an entity E_A from an ontology O_A, and if O_A refers to an entity E_B in an ontology O_B, then the model imports two ontologies O_A and O_B.

Import Lifting Rules

Guha's context mechanism defines the *importsFrom* lifting rule (see Section 3.1.3 for a definition) which corresponds to the inclusion of one context into another. The *owl:imports* primitive and the implicit import declaration are easily mapped to the *importsFrom* rule.

A particular case is when import relations are cyclic. Importing an ontology into itself is considered a null action, so if ontology O_A imports O_B and O_B imports O_A, then the two ontologies are considered to be equivalent [BvHH+04]. Based on this definition, we extend Guha's definition to allow cycles in a graph of *importsFrom*. We introduce a new symbol eq, and the notation $eq(c_1, c_2)$ states that c_1 is equivalent to c_2, i.e., that the set of propositions true in c_1 is identical to the set of propositions true in c_2.

Definition 4.1 (Cyclic Import Rule)**.** Let c_1 and c_2 be two contexts. If c_1 contains the proposition $importsFrom(c_1, c_2)$ and c_2 the proposition $importsFrom(c_2, c_1)$, then the two contexts are considered equivalent:

$$ist(c_2, importsFrom(c_2, c_1)) \wedge ist(c_1, importsFrom(c_1, c_2)) \rightarrow eq(c_1, c_2)$$

4.2.2. Deductive Closure of RDF Models

In context-dependent reasoning, the deductive closure of an entity description is the set of assertions that is entailed in the aggregate context composed of the entity description and its ontology import closure. Before defining formally the deductive closure of an aggregate context, we discuss the incomplete reasoning aspect of our approach.

Incomplete Reasoning with Web Data

When reasoning with Web data, we can not expect to deal with a level of expressiveness of OWL-DL [dBG+07], but would need to consider OWL Full. Since under such circumstances, we can not strive for a complete reasoning anyway, we therefore content ourselves with a finite entailment regime based on a subset of RDFS and OWL, namely the ter Horst fragment [tH05]. Such a deductive closure provides useful RDF(S) and OWL inferences such as class hierarchy, equalities or property characteristics (inverse functional properties or annotation properties), and is sufficient with respect to increasing the precision and recall of the search engine.

A rule-based inference engine is currently used to compute full materialisation of the entailed statements with respect to this finite entailment regime. In fact, a finite deduction is a requirement in such a setting, which in terms of OWL Full can only be possible if the entailment regime is incomplete (as widely known the RDF container vocabulary alone is infinite already [Hay04]). This is deliberate and we consider a higher level of completeness hardly achievable on the Web of Data: in a setting where the target ontology to be imported may not be accessible at the time of the processing, for instance, we just ignore the imports primitives and are thus anyway incomplete from the start.

Also the context-dependent reasoning approach misses inferences from rules that requires multiple A-Box joins across contexts, for example with a transitivity rule across entity descriptions. However, completeness is not our aim. Instead our goal is to take the best out of the data we have and in a very efficient way.

Deductive Closure of Aggregate Context

We now explain how the deductive closure of an aggregate context is performed. Given two contexts c_1 and c_2, for example an entity description and an ontology, their axioms are lifted (see Section 3.1.3 for a definition) into an aggregate context labelled $c_1 \wedge c_2$.

The deductive closure of the aggregate context is then computed using the rule-based inference engine.

It is to be noticed that the deductive closure of an aggregate context can lead to inferred statements that are not true in any of the source contexts alone. For example, if a context c_1 contains an instance x of the class *Person*, and a context c_2 contains a proposition stating that *Person* is a subclass of *Human*, then the entailed conclusion that x is a human is only true in the aggregate context $c_1 \wedge c_2$:

$$\begin{aligned} ist(c_1, Person(x)) \quad &\wedge \quad ist(c_2, subClass(Person, Human)) \\ &\rightarrow \quad ist(c_1 \wedge c_2, Human(x)) \end{aligned}$$

The set of inferred statements that are not true in any of the source contexts alone are called *aggregate entailment*:

Definition 4.2 (Aggregate Entailment)**.** Let c_1 and c_2 be two contexts with respectively two propositions φ_1 and φ_2, $ist(c_1, \varphi_1)$ and $ist(c_2, \varphi_2)$, and $\varphi_1 \wedge \varphi_2 \models \varphi_3$, such that $\varphi_2 \not\models \varphi_3$, $\varphi_1 \not\models \varphi_3$, then we call φ_3 a newly entailed proposition in the aggregate context $c_1 \wedge c_2$. We call the set of all newly defined propositions an aggregate entailment and denote it as Δ_{c_1,c_2}:

$$\begin{aligned} \Delta_{c_1,c_2} \quad = \quad &\{ist(c_1, \varphi_1) \wedge ist(c_2, \varphi_2) \models ist(c_1 \wedge c_2, \varphi_3) \\ &\text{and } \neg(ist(c_1, \varphi_3) \vee ist(c_2, \varphi_3))\} \end{aligned}$$

The aggregate entailment property enables the reasoning engine to confine inference results to specific contexts and therefore protects other contexts from unexpected data usage. Unexpected data usage in one context will not alter the intended semantics of other contexts, if and only if no direct or indirect import relation exists between them.

Note that - by considering (in our case (Horn) rule-based) RDFS/OWL inferences only - aggregate contexts enjoy the following monotonicity property[1]: if the aggregate context $c_1 \subseteq c_2$ then $ist(c_2, \phi)$ implies $ist(c_1, \phi)$, or respectively, for overlapping contexts, if $ist(c_1 \cap c_2, \phi)$ implies both $ist(c_1, \phi)$ and $ist(c_2, \phi)$. This property is exploited in our ontology base, which is described next, to avoid storing duplicate inferred statements.

[1]We remark here that under the addition of possibly non-monotonic rules to the Semantic Web architecture, this context monotonicity only holds under certain circumstances [PFH06].

4.3. Context-Dependent Ontology Base

A problem when reasoning over a large number of entity descriptions independently is that the process of computing the ontology import closure and its deductive closure has to be repeated for each entity description. This is inefficient since the computation of the import closure and the deductive closure is resource demanding and can in fact be reused for other entity descriptions. The import closure necessitates to execute multiple web requests that are network resource demanding and time consuming, while the computation of the deductive closure is CPU bound. In addition, we observe in Section 4.5 that the computation of the T-Box closure is more CPU intensive than the computation of the A-Box closure. This observation suggests to focus on the optimisation of the T-Box closure computation. Thanks to the smaller scale of the T-Box with respect to the A-Box, we can store the computed ontology import closure as well as the deductive closure in a *ontology base* in order to reuse them in later computation.

The ontology base, which can be seen as a persistent context-dependent T-Box, is in charge of storing any ontology discovered on the web along with their import relations. The ontology base also stores the inference results that has been performed in order to reuse them later. The ontology base serves the inference engine by providing the appropriate and pre-computed T-Box when reasoning over an entity description.

In the next, we first introduce the basic concepts used in the formalisation of the ontology base. We then discuss an optimised strategy to update the ontology base. We finally describe how to query the ontology base.

4.3.1. Ontology Base Concepts

The ontology base uses the notion of named graphs [CBHS05] for modelling the ontologies and their import relations. The ontology base relies on a rules-based inference engine to compute the deductive closure of either one ontology or of a combination of them.

Ontology Entity

An *Ontology Entity* is a property, instance of `rdf:Property`, or a class, instance of `rdfs:Class`. The ontology entity must be identified by an URI (we exclude entities that

are identified by a blank node) and the URI must be resolvable and must point to a document containing the ontological entity description.

Ontology Context

An *Ontology Context* is a named graph composed by the ontology statements that have been retrieved after dereferencing the entity URIs of the ontology. The content of this graph consists of the union of the descriptions of all properties and classes associated to a same ontology namespace. According to best practices [MBS08], properties and classes defined in an ontology should have the same URI namespace.

Usually, this means that the data associated with the ontology context simply reflects the content of the ontology document as found on the Web. There are cases however, e.g., in the case of the OpenCyc ontology[2], where each property and class has its own view. The ontology context is therefore composed by the union of all these views.

Ontology Network

An ontology context can have import relationships with other ontology contexts. A directed link l_{AB} between two contexts, O_A and O_B, stands for O_A imports O_B. A link l_{AB} is mapped to an *importsFrom* lifting rule and serves as a pointer to a frame of knowledge that is necessary for completing the semantics of O_A.

Definition 4.3 (Ontology Network)**.** An ontology network is a directed graph $\langle \mathcal{O}, \mathcal{L} \rangle$ with $\mathcal{O}$ a set of vertices (ontology contexts) and $\mathcal{L}$ a set of directed edges (import relations).

The import closure of an ontology can be seen as an activation of a subset of the ontology network.

Definition 4.4 (Import Closure)**.** The import closure of an ontology context O_A is a subgraph $\langle \mathcal{O}', \mathcal{L}' \rangle$ such as:

$$\mathcal{O}' \subseteq \mathcal{O}, \mathcal{L}' \subseteq \mathcal{L} \mid \forall O \in \mathcal{O}', \exists\, path(O_A, O)$$

where a path is a sequence of n vertices $O_A, O_B, \ldots, O_n$ such that from each of its vertices there is an edge to the next vertex.

[2]OpenCyc: `http://sw.opencyc.org/`

For the purpose of this work, we consider the import closure to also contain the deductive closure of the union of all the ontologies. For example, given two ontology contexts O_A and O_B, with O_A importing O_B, the import closure of O_A will consist of the context aggregating O_A, O_B and the deductive closure of the union of the two ontologies (i.e., the entailment of O_A and O_B as well as the *aggregate entailment* Δ_{O_A,O_B}).

4.3.2. Ontology Base Update

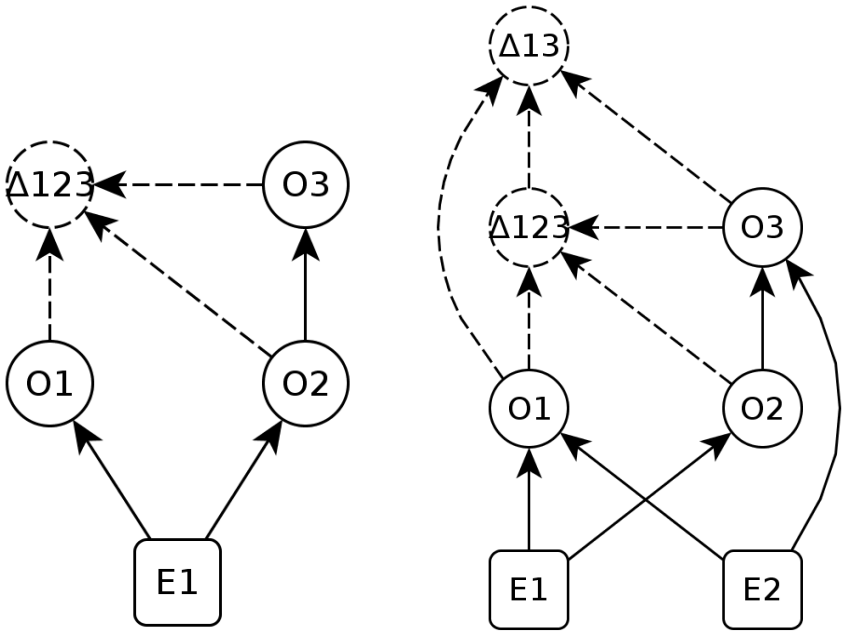
When a new ontology context is added to the ontology network, its import closure is materialised. Then, the deductive closure is computed by first lifting the axioms of the import closure and by computing the newly aggregate entailment as explained in Definition 4.2. Finally the new statements are added to the ontology network so that they are never duplicated.

This is better explained by an example. In Figure 4.1a, an entity E_1 imports explicitly two ontologies O_1 and O_2. The import closure of E_1 is calculated and this is found to include also O_3 since O_3 is imported by O_2. At this point, the following step are performed:

1. The deductive closure of O_1 and O_3 is computed separately and stored in their respective nodes.
2. The deductive closure of O_2 is computed by first lifting the axioms of O_3. The entailed propositions Δ_{O_2,O_3} are stored in O_2.
3. Finally, the deductive closure of O_1, O_2 and O_3 is calculated. The entailed statements Δ_{O_1,O_2,O_3} resulting from the reasoning over O_1, O_2 and O_3 together but that are not found already in any of the source contexts are stored in a virtual context Δ_{123}.

At this point a new entity E_2 comes which only imports O_1 and O_3 as shown in Figure 4.1b. The update strategy will:

1. calculate the deductive closure of O_1 and O_3 and store the new assertions Δ_{O_1,O_3} in a new virtual context Δ_{13};
2. subtract these triples from the content of the previous context Δ_{123}. Δ_{123} is connected to Δ_{13} by an import relation.



(a) The ontology network after processing an entity E_1.

(b) The ontology network after processing a second entity E_2. The virtual context Δ_{123} has been split in 2 parts and now imports Δ_{13}.

A last optimisation is based on Definition 4.1. Whenever a cycle is detected into the ontology network, the ontology contexts present in the cycle are aggregated into one unique context.

4.3.3. Ontology Base Querying

The ontology base is used by the A-Box reasoner to retrieve the appropriate T-Box for each entity description being processed. The A-Box reasoner first performs an analysis of the entity description and extracts all the references to the ontological terms being used. It then queries the ontology base with the set of ontological terms.

The ontology base maps each ontological terms to their ontology and computes the import closure for each of the ontology. Finally, it merges the different import closures, including the aggregate entailments Δ, into one model and returns this model to the A-Box reasoner to be used as T-Box.

For example in Figure 4.1b, the entity E_1 imports the two ontologies O_1 and O_2. This lifts the contexts $\{O_1, O_2, O_3, \Delta_{13}, \Delta_{123}\}$ into the entity E_1 before computing the A-Box deductive closure.

Sometimes, the computed T-Box can be large for two reasons: (1) when the import closure is large, i.e., a large number of ontologies are merged together; or (2) when large ontologies are present in the import closure. As a consequence, the amount of information transferred through the network between the A-Box reasoner and the ontology base increases. In addition, the memory requirement of the A-Box reasoner also increases. However, we found that the A-Box reasoner does not need the complete T-Box, i.e., the complete set of ontologies, but that only a subset of the T-Box could be used instead without having any impact on the A-Box inference result. This T-Box subset is composed of the union of the descriptions of all the ontological entities used in the original entity description. To extract the description of one ontological entity, the ontology base relies on a technique similar to the Symmetric Concise Bounded Description [Sti05] where not only blank nodes are followed, but also the symmetric property `owl:inverseOf`. [BvHH+04] states that "an axiom of the form `P1 owl:inverseOf P2` asserts that for every pair (x,y) in the property extension of P1, there is a pair (y,x) in the property extension of P2, and vice versa". By not including the description of the inverse property, the A-Box reasoner might miss some logical assertions about the property.

4.4. Implementation

The ontology base is implemented using a RDF database to store the ontology statements in their context. A secondary index is used to store the import relations between the contexts. A caching mechanism is used on top of the ontology base to cache frequent requests. The caching mechanism is especially useful when processing multiple entities from a single dataset. Since entities from a same dataset are likely to be described with the same ontologies, the requests to the ontology base are identical and the cache hit rate increases.

The reasoning engine which is used by the ontology base is especially designed and optimised to compute entailments in memory. Each term (i.e., URIs, Blank Nodes and Literals) in a statement is mapped to a unique identifier (integer). Statements are indexed using in-memory data structures, similar to triple tables, in order to lookup any kind of statement patterns. Rules are then checked against the index in an iterative

manner, with one rule being applied at a given iteration. The result of the rule is then added to the index before proceeding to the next iteration. Iterations continue until a fixpoint is reached. For rules that requires joins between multiple statements, since we are working with a small amount of data and a small number of elements, we rely on an efficient merge-join algorithm where both relations are sorted on the join attribute using bit arrays. The bit arrays are then intersected using bitwise operations.

The A-Box reasoning is distributed on a cluster of machines using the following strategy. In our approach, the A-Box is divided on a per-context basis (in our settings, on a per-entity basis). Each context provides a chunk of data which is distributed to different computing nodes. A computing node acts as A-Box reasoner and has its own ontology base. The A-Box rule engine is based on the same rule engine used by the ontology base.

Since each chunk of data is relatively small, the deductive closure of the A-Box can be entirely performed in memory without relying on disk accesses. With respect to other distributed approaches that perform reasoning on the global model, we avoid to read and write multiple times the data directly from the disk, and therefore we obtain better performance. Finally, it is to be noticed that such a distributed model scales linearly with the number of available nodes in the cluster.

4.5. Performance Evaluation

In this section, we evaluate the performance of our approach on three datasets:

Geonames: is a geographical database and contains 13.8 million of entities[3].

DBPedia: is a semi-structured version of Wikipedia and contains 17.7 million of entities[4].

Sindice: is a representative subset of the Web of Data containing, at the time of the writing, more than 110 million of documents. It is composed of Semantic Web online repositories and pages with Microformats or RDFa markups crawled on a regular basis for more than three years.

We first start by comparing the size of the T-Box and A-Box data for each dataset. We then benchmark the performance of the context-dependent reasoning approach with and without an ontology base. We finally estimate the performance in a distributed setting.

[3]Geonames: `http://www.geonames.org/`
[4]DBpedia: `http://dbpedia.org/`

		T-Box Size		A-Box Size	
Dataset	Ontology	Explicit	Implicit	Explicit	Implicit
Geonames	15	1820	4005	6	14
DBPedia	8	657	1556	7	16
Sindice	14	2085	4601	69	170

Table 4.1.: Statistics about T-Box and A-Box of 100.000 random entity descriptions.

4.5.1. Quantifying T-Box and A-Box

We perform a random sample of 100.000 entities for each dataset. For each entity, we apply our context-dependent reasoning approach and record various characteristics such as:

Ontology: the number of ontologies in the ontology closure;
T-Box - Explicit: the number of axioms in the original ontology closure;
T-Box - Implicit: the number of entailed statements from the ontology closure;
A-Box - Explicit: the number of facts in the original entity description;
A-Box - Implicit: the number of inferred facts from the entity description and the ontology closure.

Table 4.1 reports the arithmetic mean of the 100.000 measurements for each dataset.

We can notice that in general the ontology import closure is relatively large with in average 14 and 15 ontologies per entity on the Sindice and Geonames dataset respectively. The DBpedia dataset has in average less ontologies. This can be explained by the fact that this dataset contains many small entity descriptions which solely consist of a few links of type *dbpedia:wikilink* with other entities. However, we report that it also contains some fairly complex ones having an import closure of more than 30 ontologies.

These ontologies account for a large number of axioms, up to 2085 for Sindice dataset. The deductive closure provides a larger number of additional axioms. On all the datasets, the ratio of inferred statements is around 2.2, i.e., 2.2 implicit axioms in average are inferred for 1 explicit axiom.

The A-Box is relatively small compared to the T-Box. The ratio between T-Box and A-Box statements varies greatly across datasets, with 300:1 for Geonames, 94:1 for

DBPedia and 30:1 for Sindice. In average, the ratio of inferred statements on the A-Box varies between 2 and 3, which is very close to the inference ratio of the T-Box.

These statistics show that, when reasoning over an entity, there is more data to be processed in the T-Box than there is in the A-Box, suggesting that most of the computational complexity will reside on the T-Box level. Therefore, these observations strengthen the needs of an appropriate caching system such as the ontology base to reuse T-Box computation whenever it is possible.

4.5.2. Optimisation with Ontology Base

In the next experiment, we show the performance improvements provided by the ontology base. We compare the time to reason over an entity with and without ontology base. We then show the gradual performance improvement as the ontology base is being updated.

Experimental Settings

The hardware system we use in our experiments is a 2 x Opteron 250 @ 2.4 GHz (2 cores, 1024 KB of cache size each) with 4GB memory and a local SATA disk. The operating system is a 64-bit Linux 2.6.31-20-server. The version of the Java Virtual Machine (JVM) used during our benchmarks is 1.6.0_20. The reasoning engine and the ontology base is written in Java. The A-Box reasoner runs on a single thread.

Experimental Design

In the first experiment, we reason sequentially over 100.000 randomly selected entities with and without the ontology base activated. When the ontology base is deactivated, we measure the time to compute the deductive closure of the T-Box and A-Box for each entity. When the ontology base is activated, we measure the time to request the T-Box from the ontology base and to perform the deductive closure of the A-Box. Beforehand, we load all the ontology closures, i.e., their import closures and their deductive closures, in the ontology base.

In the second experiment, we start with an empty ontology base and record the evolution of the computational time while the ontology base is being created,i.e., while the import and deductive closures are being computed and stored.

In the two experiments, we ignore the time spent by the ontology base to resolve URIs and fetch the associated documents from the Web. This is deliberate since such a time is dependent of the network resources available and of the availability of the documents on the Web at the time of the fetching.

Experimental Results

Figure 4.1 reports the total time to reason over the 100.000 entities for each dataset. The total time is composed of the time spent computing the A-Box and T-Box closure. The bar *D*, *G* and *S* denotes the time to reason without an ontology base on DBpedia, Geonames and Sindice dataset respectively. The bar *D-OB*, *G-OB* and *S-OB* denotes the reasoning time with an ontology base.

A first observation is that the time increases with the complexity of the dataset and with the number of statements inferred. DBpedia is the fastest to process following by Geonames and then Sindice. We can also notice that the time to compute the closure of the A-Box on Sindice is much higher than on the other datasets, which correlates with the larger number of A-Box statement inferred found in Table 4.1.

The most interesting result is the improvement provided by the ontology base. On the three datasets, we can observe that the computation of the T-Box closure is the most time consuming operation. However, it becomes negligible when the ontology base is activated.

Figure 4.2 depicts the performance improvement of the reasoning engine while the ontology base is being updated, i.e., when the import and deductive closure are being computed for each new ontology. We analyse the measurements by computing the cumulative average of the reasoning time over the past five entities. We can see that at the beginning, while the ontology base is still empty, the reasoning time is very high with more than 3 seconds. However, the processing time rapidly decreases under the second and starts to reach its steady state of performance after 10000 entities.

4.5.3. Distributed Performance

Since the distributed model scales linearly with the number of nodes available in the cluster, we show in Figure 4.3 an estimation of the number of entities per second that can be processed on a given number of nodes. This estimation is based on the previous results

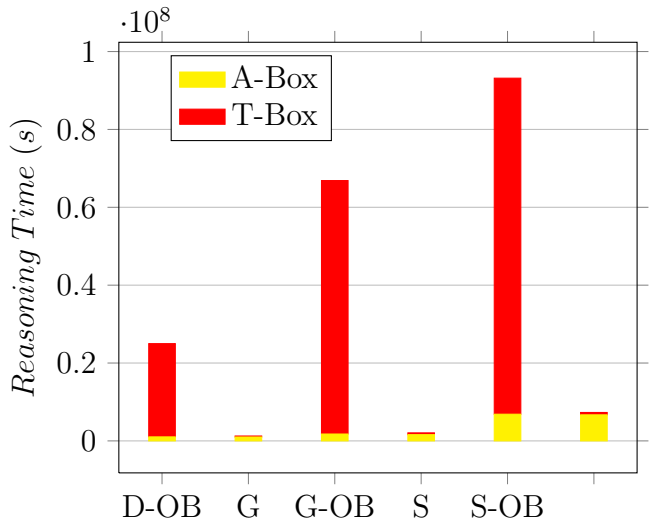


Figure 4.1.: The total time in second to reason over 100.000 randomly selected entities. Each dataset is represented by two bars, the first one reporting the reasoning time without the ontology base and the second one the time with the ontology base. Each bar is divided in two parts in order to differentiate the time spent to compute the A-Box and T-Box closures.

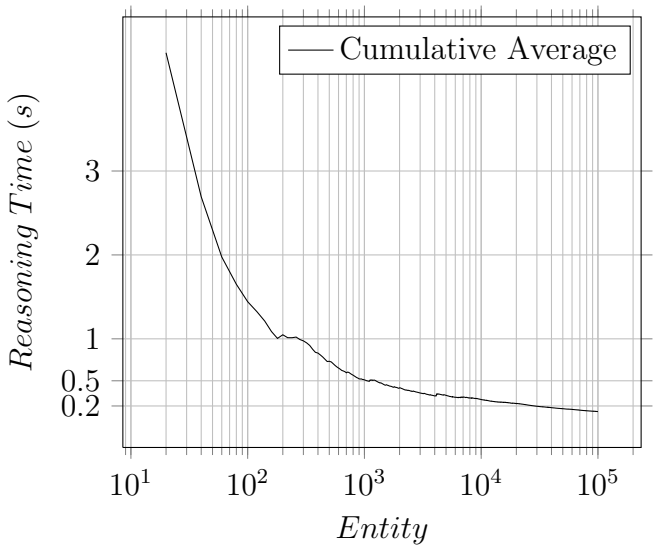


Figure 4.2.: The cumulative average of the reasoning time per entity in second over a sequence of 100.000 randomly selected entities from the Sindice dataset. The ontology base is activated but empty at the beginning.

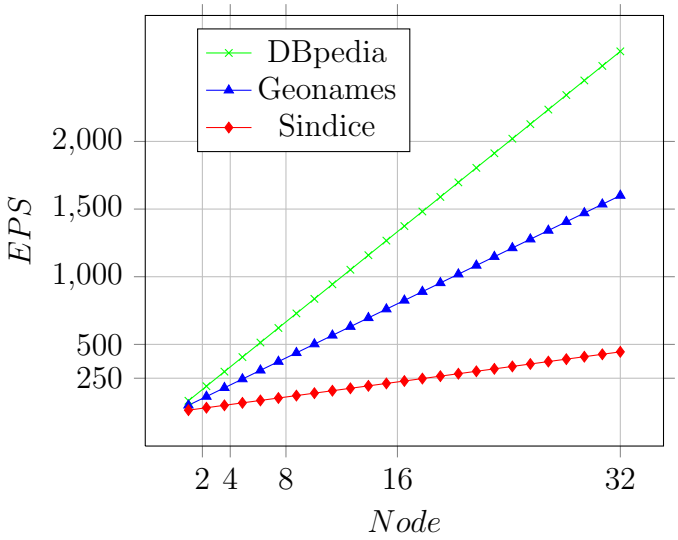


Figure 4.3.: An estimation of the number of entities per second (EPS) that can be processed on a given number of nodes. The estimation is based on the previous results where 83.3 (DBpedia), 50 (Geonames) and 13.9 (Sindice) entities per second in average is processed by a single node.

where in average 83.3, 50 and 13.9 entities per second where processed on DBpedia, Geonames and Sindice respectively. A system with 32 nodes is able to ingest more than 2000 entities per second on DBPedia, more than 1500 on Geonames and more than 400 on heterogeneous data collection such as Sindice. It is to be noticed that these results come from a prototypical implementation, still to be subject to many possible technical optimisations.

4.6. Discussion and Future Works

Sometimes the context-dependent reasoner is unable to complete the ontology import closure within a reasonable amount of time. This could happen for a variety of reasons including slow internet services or large distributed ontologies such as OpenCyc[5]. We adopted a twofold approach to tackle this problem. Firstly we added a configurable depth-limited blacklisting mechanism that filters certain ontologies after reaching a certain depth in the import closure. As a consequence, in the presence of large import closure, we only build a partial T-Box. This is however sufficient to produce a almost complete A-Box results. Secondly we added a time constraint for completing the reasoning. If the

[5]OpenCyc: `http://sw.opencyc.org/`

reasoner is unable to finish its computation within the time constraint, we simply stop the computation and use the partially computed result.

Also, the technique presented in this chapter is mainly focussed on the T-Box level. The import relations between ontologies provide a good support for lifting rules. On A-Box level, it is not clear which relations should be consider as lifting rules. We will investigate equality relations such as the `owl:sameAs` relations in the future.

4.7. Conclusions

In this chapter, we discussed a context dependent methodology for large scale web data reasoning. We first define the problem conceptually and then illustrate how to create an ontology repository which can provide the materialization of implicit statements while keeping track of their provenance. Our implementation currently support a subset of RDFS and OWL. We find this level of inference to be in line with Sindice's target objective to support the RDF community of practice, e.g., the Linked Data community, which usually relies only on RDFS and OWL features covered by the OWL ter Horst fragment [tH05].

Reasoning on Web Data enables the Sindice search engine to be more effective in term of precision and recall for the entity retrieval task. As an example, it would not be possible to answer a query on FOAF social networking files asking for entities which have label "giovanni" unless reasoning is performed to infer `rdfs:label` from the property `foaf:name`.

The context mechanism allows Sindice to avoid the deduction of undesirable assertions in RDF models, a common risk when working with the Web Data. This context mechanism does not restrict the freedom of expression of data publishers. Data publishers are still allowed to reuse and extend ontologies in any manner, but the consequences of their modifications will be confined in their own context, and will not alter the intended semantics of the other RDF models published on the Web.

Chapter 5.

Link Analysis over Web Data*

5.1. Introduction

As seen in Chapter 2, the Web of Data is composed of inter-linked datasets. The content of every dataset can be transposed into a graph model, representing entities and their relationships. As the Web of Data graph is very large, containing billions of nodes and edges, developing scalable link analysis algorithms the entity popularity scores at this scale is becoming an important requirement.

Current link analysis algorithms [DPF+05b, HHD06] for the Web of Data consider exclusively the graph of entities. In addition to their high computational complexity, they suffer from not taking into account the semantics of datasets and produce sub-optimal results. For example, given a dataset about employees, one would like to find the most skilled employee and not the most popular one. In certain cases, it is preferable to adapt the algorithms to the nature of the dataset. In this chapter, we introduce a two-layer model for the Web of Data and provide justifications for this two-layer model. Based on this model, we propose a novel ranking algorithm called DING (for Dataset rankING). DING performs ranking in three steps:

- dataset ranks are computed by performing link analysis on the graph of inter-connected dataset;
- for each dataset, entity ranks are computed by performing link analysis on the local entity collection;

*This chapter is partially based on [DTC+10b]

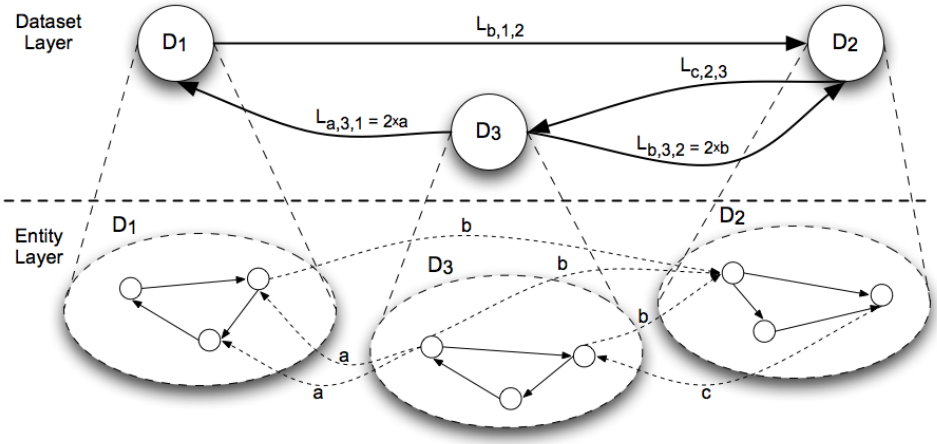


Figure 5.1.: The two-layer model of the Web of Data. Dashed edges on the entity layer represent inter-dataset links that are aggregated into linksets on the dataset layer.

- the popularity of the dataset is propagated to its entities and combined with their local ranks to estimate a global entity rank.

5.2. A Two-Layer Model for Ranking Web Data

In this section, we introduce a two layer model for the Web of Data pictured in Figure 5.1. This two-layer model is based on the formal model introduced in Chapter 2. The top layer, or dataset layer, is composed of a collection of *datasets* (see Definition 2.1) connected by *linksets* (see Definition 2.6). The lower layer, or entity layer, is composed of independent graphs of entities. These graphs of entities are composed of the *internal nodes* (see Definition 2.2) and *intra-dataset edges* (see Definition 2.4) of their respective datasets.

5.2.1. Quantifying the Two-Layer on the Web of Data

In this section, we provide evidence of the two-layer model and its desirable computational properties by quantifying the locality of the links and the dataset size distribution. We perform the following experiments. We first take the datasets described below and count how many of the links are intra-dataset and how many are inter-dataset. Then, we analyse the dataset size distribution on a subset of the Web of Data.

Dataset	Intra	Inter
DBpedia	88M (93.2%)	6.4M (6.8%)
Citeseer	12.9M (77.7%)	3.7M (22.3%)
Geonames	59M (98.3%)	1M (1.7%)
Sindice	287M (78.8%)	77M (21.2%)

Table 5.1.: Ratio between intra-dataset and inter-dataset links.

DBpedia is a semi-structured version of Wikipedia and contains 17.7 million of entities[1].

Citeseer is a semi-structured version of Citeseer from the RKBExplorer initiative and contains 2.48 million of entities[2] .

Geonames is a geographical database and contains 13.8 million of entities[3].

Sindice contains 60 million of entities[4] among 50.000 datasets (including the previous). It is a representative subset of the Web of Data. It is composed of Semantic Web online repositories and pages with microformats or RDFa markups crawled on a regular basis for more than two years.

Table 5.1 shows that 78.8% of the links are intra-dataset. Such connectivity statistics are not far from the previous results of [KHMG03] where 83.9% of the links from a large web pages dataset are intra-domain links. On individual datasets, inter-dataset links in DBpedia represent only 6.8% of its total links. For Citeseer, the number of inter-dataset links is higher than other datasets but can be explained by the fact that this dataset is using an external ontology to describe its data, hence most of its inter-dataset links point to only one external dataset (the dataset ontology). Geonames is representative of a "dataset sink", a dataset loosely linked with other datasets. These numbers confirm a high degree of locality on the Web of Data, and suggest the two-layer model proposed in this paper.

Figure 5.2 depicts the distribution of the size of all datasets found in the Sindice data collection. The distribution nearly follows a powerlaw and corresponds to previous research on the size of web sites [FLW+06]. We observe that the majority of the datasets

[1]DBpedia: `http://dbpedia.org/`
[2]Citeseer: `http://citeseer.rkbexplorer.com/`
[3]Geonames: `http://www.geonames.org/`
[4]At the time of the experiment, the Sindice data collection was composed of only 60 million of entities.

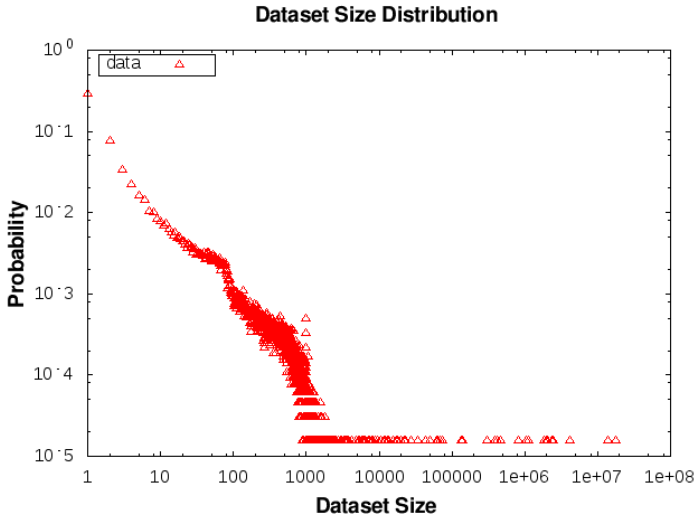


Figure 5.2.: Probability distribution of the size of datasets.

contain less than 1000 nodes which indicates that local rank computation within these graphs can be performed in a efficient manner in memory.

5.2.2. The Dataset Graph

The top layer, or dataset graph, can be seen as an approximation of the data graph G (as defined in Section 2.3.1). Instead of considering entities and links between these entities, we are using higher-level information such as *datasets* and *linksets*.

The resulting graph (50.000 nodes, 1.2M of linksets) is orders of magnitude smaller than the original graph G (60M nodes, 364M of links). As a consequence, it can be easily kept in memory (in the case of Sindice) and the popularity of the datasets can be computed on demand.

5.2.3. The Entity Graph

The lower layer, or entity graph, is composed of disjoint graphs D (see Definition 2.1) each of them being a collection of internal nodes and intra-dataset edges. The direct consequence is that the computation of the local entity ranks can be computed in an independent manner on a per dataset basis and can be easily parallelised.

Since computations are performed independently, the complexity that would dominate is that of the largest dataset, e.g., DBpedia in our case. However, the majority of the graphs has a small number of nodes as shown in Figure 5.2. This means that the majority of the graphs can be kept in memory and rank computation can be performed without the performance penalty of IO accesses generally encountered when processing very large graphs.

5.3. The DING Model

In this section, we start by introducing an unsupervised method for weighting links and linksets. Next, the DING algorithm is described. We first reformulate the original PageRank algorithm (see Section 3.2.1) for computing dataset ranks. We present two local entity ranking algorithms as well as a list of dataset-dependent algorithms that is known to outperform standard algorithms for certain type of dataset. We finally explain how to combine dataset ranking with local entity ranking in order to estimate a global entity ranking.

5.3.1. Unsupervised Link Weighting

In Figure 5.1 the probability of the user going from D_3 to D_1 is likely to be different from the probability of going to D_2 since the label and number of links associated to $L_{a,3,1}$ are not the same as the ones associated to $L_{b,3,2}$. The goal is to define a linkset weighting function $w_{\sigma,i,j}$.

Weights can be assigned based on the cardinality (the number of links) of a linkset and on the specificity of its label. Our approach, called "Link Frequency - Inverse Dataset Frequency" (LF-IDF), is derived from TF-IDF to measure the relevance of a linkset. The *link frequency* measures the importance of the linkset between two datasets, and is defined as follow:

$$LF(L_{\sigma,i,j}) = \frac{|L_{\sigma,i,j}|}{\sum_{L\tau,i,k} |L_{\tau,i,k}|} \tag{5.1}$$

where $|L_{\sigma,i,j}|$ is the cardinality of the considered linkset between datasets D_i and D_j, and the denominator is the cardinality of all the other linksets having D_i as source.

The *inverse dataset frequency* measures the general importance of a link given its label, and is defined as follow:

$$IDF(\sigma) = \log \frac{N}{1 + freq(\sigma))} \tag{5.2}$$

where N denotes the total number of datasets and $freq(\sigma)$ is the number of occurrences of the label σ in the collection of datasets.

We define the linkset weighting function w as being the product between the *link frequency* and the *inverse dataset frequency*:

$$w_{\sigma,i,j} = LF(L_{\sigma,i,j}) \times IDF(\sigma) \tag{5.3}$$

The *link frequency* and *inverse dataset frequency* can be computed dynamically given accumulated statistical information in a database. The LF-IDF scheme assigns a higher degree of importance to link with a high frequency between two datasets and a low dataset frequency. Former results [TUD+09] have shown that link weighting improves dataset ranking.

5.3.2. DING Algorithm

The DING algorithm is an extension of PageRank (see Equation 3.1) for the two-layer graph model presented in Section 5.2. Instead of visiting web pages, the random surfer browses datasets. The random walk model is as follows:

1. At the beginning of each browsing session, a user randomly selects a dataset.
2. Then, the user may choose one of the following actions:
   a) Selecting randomly an entity in the current dataset.
   b) Jumping to another datasets that is linked by the current dataset.
   c) Ending the browsing.

According to the hierarchical random walk model, we can apply a two-stage computation. In the first stage, we calculate the importance of the top level dataset nodes as explained next. The second stage calculates the importance of entities within a dataset as explained in Section 5.3.4.

5.3.3. Computing DatasetRank

Since the dataset surfing behaviour is the same as in PageRank, we can obtain the importance of dataset nodes by adapting Equation 3.1 to the weighted dataset graph. The DatasetRank formula is defined as:

$$r^k(D_j) = \alpha \sum_{L\sigma,i,j} r^{k-1}(D_i) w_{\sigma,i,j} + (1-\alpha) \frac{|V_{D_j}^E|}{\sum_{D \subset \mathcal{G}} |V_D^E|} \tag{5.4}$$

The rank score $r(D_j)$ of a dataset is composed of a first part corresponding to the rank contribution from the datasets linking to D_j and of a second part corresponding to the probability of a random jump to D_j from any dataset in the collection. The probability of selecting a dataset during a random jump is proportional to its size, i.e., the number of internal entity nodes or $|V_{D_j}^E|$, normalised by the total size of the data graph G, i.e., $\sum_{D \subset \mathcal{G}} |V_D^E|$. The distribution factor $w_{\sigma,i,j}$ is defined by Equation 5.3. The two parts are combined using the damping factor $\alpha = 0.85$, since we observed that this value provides also good results in our experimental evaluation.

5.3.4. Computing Local Entity Rank

A method used in layered ranking algorithms is to assign to the page the importance of the supernode [EAT04]. In our case this would correspond to assign the DatasetRank score of a dataset to all its entities. In large datasets, such as DBpedia, this approach does not hold since a query is likely to return many entities from a same dataset. This unnecessarily pushes part of the ranking problem at query-time. Instead we need to assign a score combining both the importance of a dataset and the importance of an entity within the dataset.

Next, we present two generic algorithms, the weighted EntityRank and the weighted LinkCount, that compute entity ranks on any type of graphs. However, we argue that entity ranking is strongly dependent of the semantic of the dataset. A list of existing dataset-dependent algorithms is discussed afterwards.

Weighted EntityRank

The Weighted EntityRank method computes the importance of an entity node within a dataset. It adapts the PageRank formulae from Equation 3.1 and is applied on graphs composed of the internal entities and the intra-links of a dataset. We use the LF-IDF weighting scheme from Equation 5.3 as distribution factor. On the entity layer, the *link frequency* from Equation 5.1 is therefore always equal to one divided by the total number of links of a node.

Reusing the same weighting scheme over the two layers allows to maintain consistency of the surfer's preferences in terms of links to follow. However one could also consider computing the link weights using for example a localized version of the *inverse dataset frequency* from Equation 5.2, where $freq(\sigma)$ would be the frequency of the label within the dataset. Like PageRank, the robustness against spam of the EntityRank method makes it a good choice for datasets build on non-controlled user inputs.

Weighted LinkCount

The Weighted LinkCount is a stripped off alternative to EntityRank when the dataset can be assumed mostly deduplicated and spam-free. This is often true for very well curated datasets like DBpedia. The rank $r(j)$ of an entity node j is given by

$$r(j) = \sum_{l_{\sigma,i,j}} w(l_{\sigma,i,j})$$

where $w(l_{\sigma,i,j})$ is the LF-IDF weight of the link from i to j. LinkCount is more efficient to compute than EntityRank since it only needs one "iteration" over the data collection.

Dataset-Dependent Entity Ranking

Datasets on the Web of Data may have their own semantic and may come with a variety of graph structures. For example, we can mention generic graphs coming from user inputs, hierarchical graphs, bipartite graphs, etc. A complete taxonomy of the different graph structures among existing datasets is beyond the scope of this thesis, but several examples are presented in Table 5.2.

Graph Structure	Dataset	Algorithm
Generic, Controlled	DBpedia	LinkCount
Generic, Open	Social Communities	EntityRank
Hierarchical	Geonames, Taxonomies	DHC
Bipartite	DBLP	CiteRank

Table 5.2.: List of various graph structures with targeted algorithms

While EntityRank and LinkCount represent good generic solutions for local entity ranking, as shown in Section 5.5.2, an approach which takes into account the peculiar properties of each dataset will give better results. Considering that in literature there are already a notable amount of ranking algorithms that are domain-dependent or specific to certain graph structure, such as [SG09, WXYM06] for citation networks or Dissipative Heat Conductance [XYZ+05] for strongly hierarchical datasets like taxonomies or geo-databases, DING has been designed to exploit better alternatives to LinkCount and EntityRank. One can also define its own algorithm using dataset-dependent ranking criteria. For example, we could rank products of a e-commerce database based on their customer ratings and reviews.

5.3.5. Combining DatasetRank and Entity Rank

A straightforward approach for combining dataset and local entity ranks is to adopt a purely probabilistic point of view by interpreting the dataset rank $r(D)$ as the probability of selecting the dataset and the local entity rank $r(e)$ as the probability of selecting an entity within this dataset. Hence we would have the global score $r_g(e)$ defined as

$$r_g(e) = P(e \cap D) = r(e) * r(D)$$

However, this approach favours smaller datasets. In the probabilistic model all ranks in a dataset sum to 1. Therefore, the local entity ranks is much higher in small datasets than in larger ones. As a consequence any small dataset receiving even a single link is likely to have its top entity scores way above many of the top ones from larger datasets. The solution is to normalize the local ranks to a same *average* based on the dataset size. In our experiments we use the following formula for combining the rank of an entity e

with the rank of a dataset D:

$$r_g(e) = r(D) * r(e) * \frac{|V_D^E|}{\sum_{D' \subset G} |V_{D'}^E|}$$

5.4. Scalability of the DING approach

A precise evaluation of the scalability of our approach is not the goal of this paper. Moreover, [KHMG03] has shown that hierarchical ranking algorithms provide speedup in computation compared to standard approaches. However, we report here some performance results from the DING method when applied on a real use-case scenario, i.e., the Sindice search engine.

Given the small size of the dataset graph as shown in Section 5.2.2, the graph can be fully held in memory and rank computation can be performed on demand. A single iteration of DatasetRank computation takes 200ms to process 50k datasets on commodity hardware (Intel Xeon E5410 Quad Cores), a good quality rank can hence be obtained in a matter of seconds. If we define a measure of convergence of the algorithm at an iteration $k+1$ as in Equation 5.5, the algorithm converges to a 0.1% threshold in 32 iterations, which represents 5 seconds.

$$\rho(k+1) = \max_{D_i \in \mathcal{D}} \frac{|r^{k+1}(D_i) - r^k(D_i)|}{r^k(D_i)} \tag{5.5}$$

Since the size of the majority of the datasets is in order of thousands of nodes as shown in Section 5.2.3, their entity graph can also be held entirely in memory making the computation of entity ranks more effective. Moreover, since the computation of entity ranks in one dataset is independent of the entity rank from another datasets, the computation can be easily distributed over a cluster of machines.

On the other hand, the computation of entity ranks in large datasets can become a heavy operation considering that the largest dataset (e.g., DBpedia) is containing over tenths of millions entities and links. For such a dataset, we fall back on standard methods to parallelise the computation such as the *Map-Reduce* programming model [DG08]. Computing[5] the local entity ranks of the DBpedia dataset with a 0.1% precision took 55

[5]including Input/Output disk operations as well as data preprocessing

iterations of 1 minute each on a Map-Reduce cluster composed of three Intel Xeon quad cores. In the case of LinkCount, the computation would require only one iteration.

In addition, separating dataset ranks and local entity ranks minimizes the amount of computation required when updating the data collection. For example, a new dataset D_i which has links to several other datasets has to be indexed by Sindice. With standard non-hierarchical ranking models, the ranks of all entities would have to be recomputed. In contrast, with the DING model the only set of ranks to be computed are (1) the ranks of entities in the dataset D_i; and (2) the dataset ranks which can be recomputed in a matter of seconds. This decreases the cost of the update from being proportional to the size of the Web of Data to being proportional to the size of the dataset D_i and of the dataset graph.

5.5. Experiments and Results

We introduced a novel ranking model, shown that it can adapt to dataset semantics and gave evidences about its desirable computational properties. But it is not yet clear if the DING model provides worst, similar or better performance than standard approaches. In order to assess the performance of DING, we conduct three experiments. The baseline algorithm that we use for comparison is a global version of EntityRank (GER). This algorithm is similar to the one described in Section 5.3.4 with the only difference that it operates on the full data graph G. We use the datasets presented in Section 5.2.1 for the first two experiments.

The first experiment investigates the impact of the link locality in the data graph by comparing the performance of the two generic local algorithms, the local EntityRank (LER) and local LinkCount (LLC), with GER. The experiment is done without user intervention by measuring the correlation of the ranking produced by the different algorithms. The second experiment evaluates the effectiveness of the local algorithms and of the DING approach with a user study in order to judge if they provides worst, similar or better performance than the baseline approach. The third experiment is a comparison of the effectiveness of the algorithms in term of precision and recall.

Algorithm	DBpedia	Citeseer	Geonames
LLC	0.79	0.86	0.73
LER	0.88	0.97	0.78

Table 5.3.: Spearman's correlation between LLC and LER with GER. There is a strong correlation between the local entity ranks and the global ones, indicating a high degree of link locality in the data graph.

5.5.1. Accuracy of Local Entity Rank

We compare the performance of LER and LLC with GER on individual datasets. We measure the Spearman's correlation [Mel07] of the two local algorithms with GER using the full entity rank list of three datasets: DBpedia, Citesser and Geonames. For GER, the corresponding lists of entity ranks in extracted as follows. We first apply GER on the Sindice data graph to compute a global entity rank for the Sindice dataset. Then, we extract the global entity ranks for each of the three datasets.

The Spearman's correlation coefficient tests the strength of the relationship between two variables, i.e., ranks produced by LER or LLC and GER. The values varies between 1 (a perfect positive correlation) and -1 (a perfect negative correlation). A value of 0 means no particular correlation.

The results are presented in Table 5.3. While LLC performs slightly worse than LER, the Spearman's correlation indicates a strong correlation of the two algorithms with GER. These results confirm that, on individual datasets, GER can be well approximated using computational methods of lower complexity such as LER or LLC due to the high degree of link locality in the data graph.

5.5.2. User Study

Usually, in Information Retrieval, an evaluation of the system assesses its retrieval effectiveness, expressed in terms of recall and precision. Data collections such as the one provided by TREC[6] or INEX[7] are employed to judge the relevance of the results produced by the system. In particular, The TREC and INEX Entity Tracks are corpus

[6]Text Retrieval Conference: `http://trec.nist.gov/`
[7]INitiative for the Evaluation of XML Retrieval: `http://www.inex.otago.ac.nz/`

especially created for the evaluation of entity-related searches. However, such evaluation corpus are designed for keyword search over a single dataset. This is not suitable for our use cases since our goal is to measure the effectiveness of ranking when queries of various complexity are used among inter-linked datasets. Therefore, in order to evaluate qualitatively the DING methodology, we decided to perform an user study where users provide relevance judgements for each algorithm.

Design

The user study is divided into two experiments: (1) the first one (Exp-A) assesses the performance of local entity ranking on the DBpedia dataset; (2) the second one (Exp-B) assesses the performance of the DING approach on the full Sindice's page-repository. Each experiment includes 10 queries, varying from simple keyword queries to more complex structured queries (SPARQL). Each participant receives a questionnaire, as the ones found in Appendix A, containing a description of the query in human language, and three lists of top-10 results. Each result is described by the human-readable label and the URI of the entity. The first list corresponds to the ranking results of GER. For Exp-A, the second and third lists correspond to the ranking results of LER and LLC while for Exp-B the ranking results are from DatasetRank combined with LER and LLC (DR-LER and DR-LLC resp.). The second and third lists are named randomly "Ranking A" or "Ranking B" so no information about the ranking algorithms and no correlation between Ranking A and B on two questionnaires can be inferred. The three lists are ordered using both the query-dependent evidence, i.e., the relevance score of the entity description with respect to the query as explained in Section 6.4.4, and the query-dependent evidence, i.e., the static scores produced by the link analysis algorithms. Compared to the next experiment, the evidences are combined linearly.

Participants

Exp-A evaluation is performed on 31 participants, and Exp-B evaluation on 58 participants. The participants consist of researchers, doctoral and master students and technicians. All of the participants are familiar with search engines, but a few of them familiar with entity search engines.

Task

The task is to rate "Ranking A" in relation to the standard one using categorical variable, then to rate "Ranking B" in relation to the standard one. The participants have to choose between 5 categories: Better (B), Slightly Better (SB), Similar (S), Slightly Worse (SW), Worse (W).

Measure

We use the Pearson's chi-square to perform the test of "goodness of fit" between O, the observed frequency distribution (the participant's judgements) of the previous categories, and E, an expected theoretical uniform distribution (equiprobable) of these categories, in order to establish whether or not the observed distribution differs from the theoretical distribution. Our null hypothesis is that the observed frequency distribution is uniform. We then interpret the contribution to chi-square of each category.

Exp-A Results

Tables 5.4a and 5.4b report the results of the chi-square test for LER and LLC respectively. For the tests to be significant at the 1% level, with 4 degrees of freedom, the value for chi-square has to be at least 13.3. Since the chi-square test yields 49.48 for LER and 14 for LLC, we can reject the null hypothesis for the two tests. It bears out that a large proportion of the population (+71% of contribution to chi-square) considers LER similar to the GER. For LLC, a majority of the population (+53% of contribution to chi-square) considers it similar to GER, and this is reinforced by the fact that a minority (−31% of contribution to chi-square) considers it worse.

To conclude, at 1% significance level, LER and LLC provides similar results than GER. However, there is a more significant proportion of the population that considers LER more similar to GER.

Exp-B Results

Tables 5.5a and 5.5b report the results of the chi-square test for DR-LER and DR-LLC respectively. For the tests to be significant at the 1% level, with 4 degrees of freedom,

Rate	O_i	E_i	$\frac{(O_i-E_i)^2}{E_i}$	$\%\chi^2$
Better	0	6.2	6.2	−13%
Slightly Better	7	6.2	0.1	+0%
Similar	21	6.2	35.33	+71%
Slightly Worse	3	6.2	1.65	−3%
Worse	0	6.2	6.2	−13%
Totals	31	31	$\chi^2 = 49.48$	

(a) Chi-square calculation for LER

Rate	O_i	E_i	$\frac{(O_i-E_i)^2}{E_i}$	$\%\chi^2$
Better	3	6.2	1.65	−12%
Slightly Better	8	6.2	0.52	+4%
Similar	13	6.2	7.46	+53%
Slightly Worse	6	6.2	0.01	−0%
Worse	1	6.2	4.36	−31%
Totals	31	31	$\chi^2 = 14$	

(b) Chi-square calculation for LLC

Table 5.4.: Chi-square test for Exp-A. The column $\%\chi^2$ gives, for each modality, its contribution to χ^2 (in relative value).

Rate	O_i	E_i	$\frac{(O_i-E_i)^2}{E_i}$	$\%\chi^2$
Better	12	11.6	0.01	+0%
Slightly Better	12	11.6	0.01	+0%
Similar	22	11.6	9.32	+57%
Slightly Worse	9	11.6	0.58	−4%
Worse	3	11.6	6.38	−39%
Totals	58	58	$\chi^2 = 16.31$	

(a) Chi-square calculation for DR-LER

Rate	O_i	E_i	$\frac{(O_i-E_i)^2}{E_i}$	$\%\chi^2$
Better	7	11.6	1.82	−9%
Slightly Better	24	11.6	13.26	+65%
Similar	13	11.6	0.17	+1%
Slightly Worse	10	11.6	0.22	−1%
Worse	4	11.6	4.98	−24%
Totals	58	58	$\chi^2 = 20.45$	

(b) Chi-square calculation for DR-LLC

Table 5.5.: Chi-square test for Exp-B. The column $\%\chi^2$ gives, for each modality, its contribution to χ^2 (in relative value).

the value for chi-square has to be at least 13.3. Since the chi-square test yields 16.31 for DR-LER and 20.45 for DR-LLC, we can reject the null hypothesis for the two tests. It bears out that a good proportion of the population (+57% of contribution to chi-square) considers DR-LER similar to GER, strengthen by the fact that a minority (−39% of contribution to chi-square) considers it worse. For DR-LLC, a large proportion of the population (+65% of contribution to chi-square) considers it slightly better than GER, and this is comforted by the fact that a minority (−24% of contribution to chi-square) considers it worse.

To conclude, at 1% significance level, the two algorithms give a profile of preference quite different. It appears that DR-LLC provides a better effectiveness. Indeed, a large proportion of the population find that its results are slightly better than GER, and this is reinforced by a few number of people finding it worse.

5.5.3. Entity Search Track at SemSearch 2010

In order to verify the previous findings, a third experiment is performed by participating to an evaluation for semantic entity search organised by the Semantic Search workshop[8]. The Semantic Search evaluation focus on matching and ranking entities in the semantic data search scenario. GER, DR-LER and DR-LLC are compared in the setting of this evaluation.

Query

The entity search track provides a set of 92 queries that are focused on the task of entity retrieval using keyword-based search. These queries represent a sample extracted from the Yahoo! Web and Microsoft Live search query log. Each query is a plain list of keywords which refer to one particular entity. In other words, the queries ask for one particular entity (as opposed to a set of entity).

Corpus

The dataset is based on the Billion Triple Challenge dataset[9], which represents a sample of Web data crawled from publicly available sources. The dataset is published using the N-Quad syntax [10], a syntax similar to N-Triples but with a fourth element specifying the URI of the RDF document containing the triples. Therefore, the same triple might be contained in several different documents.

Relevance Judgement

The relevance judgement is based on the top-10 results of each query. Results are evaluated via the three point scale (0) Not Relevant, (1) Relevant and (3) Perfect Match. A perfect match is a description of a resource that matches the entity to be retrieved by the query. A relevant result is a resource description that is related to the entity, i.e., the entity is contained in the description of that resource. Otherwise, a resource description is not relevant. The assessment of the results is based on the recall, precision, f-measure

[8]Semantic Search 2010 Workshop: `http://km.aifb.kit.edu/ws/semsearch10/`
[9]BTC Dataset: `http://challenge.semanticweb.org/`
[10]N-Quads: `http://sw.deri.org/2008/07/n-quads/`

and the mean average precision. The relevance judgement and assessment of the results is done by the program committee of the workshop.

Entity Description Extraction

A pre-processing step of the data is necessary before indexing in order to extract the entity descriptions, i.e., star graphs as shown in Figure 2.4, from the dataset. First, the set of n-quads from the dataset is ordered by context and subject. Then, we perform a scan on the sorted n-quads and extract all the n-quads having the same context and subject. The extracted subset of n-quads forms an entity description. Compared to Figure 2.4, we extract only the outgoing relations of an entity node. Furthermore, we filter all entity descriptions that are composed of only one or two triples in order to reduce the index size and the noise.

Entity Description Preprocessing

Each entity description is then indexed by SIREn, the Information Retrieval engine presented in Chapter 6. SIREn, before indexing, performs pre-processing tasks for normalising the data. Literals as well as URIs are tokenised using various rules. Tokenising URIs is useful since it allows to perform keyword search on URI's parts. For example one could just use the local name, e.g. `person`, to match `foaf:person` ignoring the namespace. Then each token is lower-cased. We also filter out stopwords and words with only one character. No stemming is performed.

Query-Independent Ranking

The dataset and entity ranks are computed using the Sindice data repository instead of the original BTC dataset. While the Sindice data repository and the BTC dataset are largely overlapping, the data coming from Sindice are kept up to date and better interlinked. The Sindice data collection is therefore a more representative subset of the current Web of Data, and provides more precise dataset and entity ranks.

Combining Rank Evidence

We combine the query-dependent and query-independent evidence in order to compute the final entity score. We adopt one of the method presented in [CRZT05].

In our experiment, we retrieve the full ordered list of entities matching for each query. We integrate the static score S_s with the query score S_q in order to obtain the final entity score S_f. We finally rerank the entity list based on the final score.

The query score is computed by SIREn as explained in Section 6.4.4. The query score is normalized using its logarithm $\log(S_q)$ and the static score using the sigmoid function [CRZT05] with parameters $w = 1.8$, $k = 1$ and $a = 0.6$. The final formula is $S_f = \log(S_q) + w * \frac{S_s^a}{k^a + S_s^a}$.

Results

The assessments of the results for each ranking technique, i.e., GER, DR-LER and DR-LLC, are reported in Figure 5.3. The results indicates that DR-LER and DR-LLC provide slightly better results than GER. These results strengthen the findings of the user study from Section 5.5.2. However, compared to the results of the user study, DR-LLC does not seem to provide better results than DR-LER. This may be due to the queries which are of a different type. In this experiment, the queries are restricted to keywords and targeted to find a specific entity. In the user study, the queries varies from simple keywords to more advanced and structured forms, and some of the queries are targeted to find a set of entities such as "Give me a list of projects dealing with the Semantic Web".

5.6. Conclusion and Future Work

We presented DING, a novel two-layer ranking model for the Web of Data. DING is specifically designed to address the Web Data scenario, i.e, computing the popularity score of entities on a web-scale graph. We explained its desirable computational properties compared to alternative approaches and displayed experimental evidence of improved ranking quality. Furthermore, DING introduces the idea of using dataset-specific ranking algorithms for improving ranking. Further works need to be done in the area of automation of graph structure recognition. This would allow better match of specific ranking

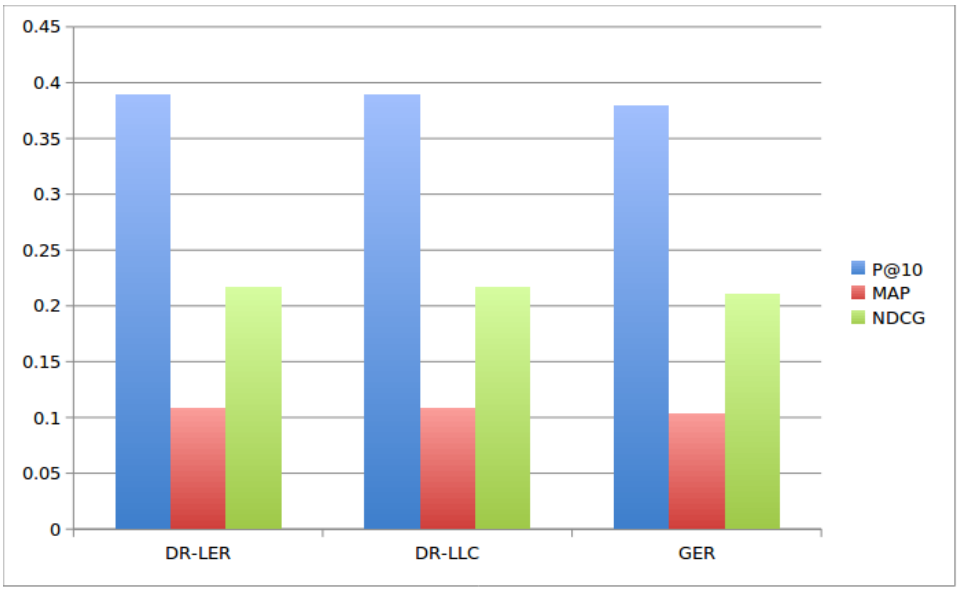


Figure 5.3.: Assessments of the ranking methods at the Entity Search Track of the Semantic Search 2010 workshop.

algorithms to a particular graph structure or semantic, and improve the quality of the ranking on a heterogeneous Web.

Chapter 6.

Semi-Structured Indexing for Web Data*

6.1. Introduction

In this chapter, we present the Semantic Information Retrieval Engine, SIREn, a system based on Information Retrieval (IR) techniques and conceived to index the entire "Web of Data" and search "entities". The requirements have therefore been:

1. Support for the multiple formats which are used on the Web of Data;
2. Support for entity centric search;
3. Support for context (provenance) of information: entity descriptions are given in the context of a website or dataset;
4. Support for semi-structural full text search, top-k query, incremental index maintenance and scalability via shard over clusters of commodity machines.

With respect to point 1 and 2, we have developed SIREn to support the Entity Attribute-Value model and the boolean search model from Section 2.3.3 and Section 2.4 respectively, since it covers RDF, Microformats and likely other forms of semi-structured data that can be found of the Web. Finally, we will see in Section 6.4 that the dataset-entity centric indexing enables SIREn to leverage well known Information Retrieval techniques to address the point 3 and 4.

We advocate the use of a *node indexing scheme* for indexing semi-structured data, a technique coming from the XML Information Retrieval world. We analyse and compare

*This chapter is partially based on [DTC+10a]

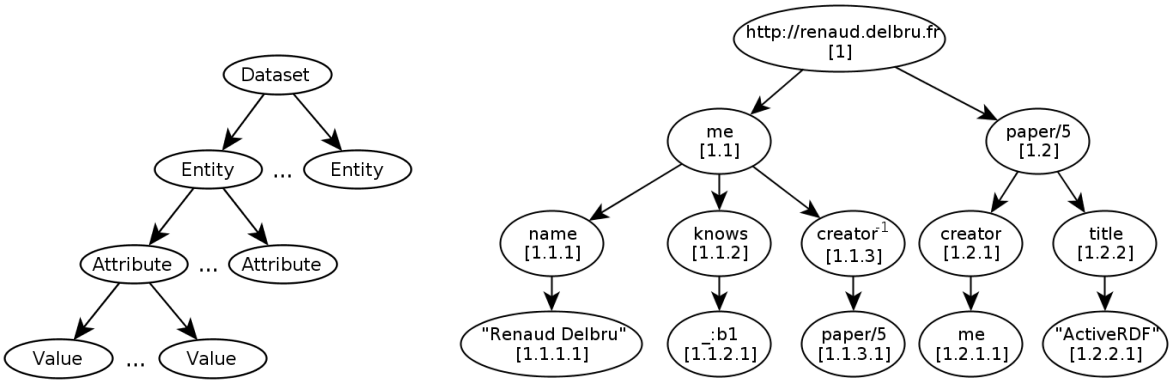
the theoretical performances and other criteria of SIREn against three other indexing techniques for entity retrieval. We show that the node indexing scheme offers a good compromise between query expressiveness, query processing time and update complexity and scales well with a very large data collection. The resulting system inherits from many characteristics of Information Retrieval systems such as web like scalability, incremental updates and top-k queries among others.

The chapter is organized as follows. We present the node-labelled data model in Section 6.2 and the associated query model in Section 6.3. We describe in Section 6.4 how to extend inverted lists as well as update and query processing algorithms to support the node labelled data model. An analysis of the differences and theoretical performances between SIREn and other entity retrieval systems is given in Section 6.5. In Section 6.6, experimental results are shown using large real world data collections and against other well known RDF management systems.

6.2. Node-Labelled Tree Model for RDF

SIREn adopts a node-labelled tree model to capture the relation between datasets, entities, attributes and values. The tree model is pictured in Figure 6.1a. Taking RDF data model as example, the tree has four different kind of nodes: context (dataset), subject (entity), predicate (attribute) and object (value). The tree encodes a set of RDF quads (which can be seen as a Dataset-Entity Attribute Value table from Section 2.3.3), where each branch of the tree represents one quad (or one row of the Dataset-Entity Attribute Value table). Each node can refer to one or more terms. In the case of RDF, a term is not necessarily a word from a literal, but can be an URI or a local blank node identifier. The incoming relations, i.e., RDF quads where the entity identifier appears at the object position, are symbolised by a attribute node with a $^{-1}$ tag in Figure 6.1b. This model is not limited to quad relations, and could in theory be used to encode longer paths such as 2-hop relations but this is beyond the scope of this thesis.

A node-labelled tree model enables to encode and efficiently establish relationships between the nodes of a tree. The two main types of relations are *Parent-Child* and *Ancestor-Descendant* which are also core operations in XML query languages such as XPath. To support these relations, the requirement is to assign unique identifiers, called *node labels*, that encode the relationships between the nodes. Several node labelling



(a) Conceptual representation of the node-labelled tree model

(b) Node-labelled tree model using Dewey's encoding of the example dataset from Section 2.2.1

Figure 6.1.: The node-labelled tree model

schemes have been developed [SCCS09] but in the rest of the thesis we use a simple prefix scheme, the *Dewey Order* encoding [BVT+02].

In Dewey Order encoding, each node is assigned a vector that represents the path from the tree's root to the node and each component of the path represents the local order of an ancestor node. Using this labelling scheme, structural relationships between elements can be determined efficiently. An element u is an ancestor of an element v if label(u) is a prefix of label(v). Figure 6.1b depicts a data tree where nodes have been labelled using Dewey's encoding. Given the label $\langle 1.1.1.1 \rangle$ for the term `Renaud`, we can find that its parent is the attribute `name`, labelled with $\langle 1.1.1 \rangle$.

The node labelled tree is embedded into an inverted index. The inverted index stores for each term occurrence its node label, i.e., the path of elements from the root node (dataset) to the node (attribute or value) that contains the term. A detailed description of how this tree model is encoded into an inverted index is given in Section 6.4.

6.3. Query Model

In this section, we present a set of query operators over the content and the structure of the node-labelled tree which covers the boolean search model presented in Section 2.4. We will present the query operators of SIREn and whenever possible compare them with the search query algebra from Section 2.4.1.

6.3.1. Content operators

The content operators are the only ones that access the content of a node and are orthogonal to the structure operators. The atomic search element is a keyword. Multiple keywords can be combined with traditional keyword search operations. Such operations include boolean operators (intersection, union, difference), proximity operators (phrase, near, before, after, etc.), fuzzy or wildcard operators, etc.

These operators give the ability to express complex keyword queries. A keyword query is used to retrieve a particular set of nodes. Interestingly, it is possible to apply these operators not only on literals, but also on URIs, if URIs are tokenized and normalized. For example one could just use an RDF local name, e.g., `name`, to match `foaf:name` ignoring the namespace.

With respect to the search query algebra of Section 2.4.1, the content operators are mapped to the *keyword selection* operators. The query algebra only defines the boolean operators, but it is easy to see how to extend the algebra for including proximity or other operators. Also, the content operators allow to restrict keyword search to a particular type of nodes, being either dataset, entity, attribute or value. However, in the following we assume that this restriction is implicit and thus is not shown in the following examples.

6.3.2. Structure operators

The structure operators are accessing the structure of the data tree. The atomic search element is a node. Multiple nodes can be combined using tree and set operators. The tree operators, i.e. the Ancestor-Descendant and Parent-Chile operators, allow to query node relationships and to retrieve the paths matching a given pattern. The combination of paths are possible using set operators, enabling the computation of star-shaped queries such as the one pictured in Figure 2.6.

Ancestor-Descendant - *A//D*

A node A is the ancestor of a node D if it exists a path between A and D. The operator checks the node labels and retains only relations where the label of A is a prefix of the label of D. For example, in Figure 6.1b, the operation `renaud.delbru.fr // paper/5` will retain the relations `[1] // [1.1.3.1]` and `[1] // [1.2]`. With respect to the

query algebra from Section 2.4.1, we interpret an Ancestor-Descendant operator as a keyword selection applied on a second operation. For example, the query Q8 can be interpreted as an Ancestor-Descendant operator where $\sigma_{d:biblio}$ is the ancestor and $R_1 \cap R_2$ is the descendant.

Parent-Child - *P/C*

A node P is the parent of a node C if P is an ancestor of C and C is exactly one level above P. The operator checks the node labels and retains only relations where the label of P is the longest prefix matching the label of C. For example, in Figure 6.1b, the operation `made / paper-9` will retain the relation `[1.1.2] / [1.1.2.1]`. With respect to the query algebra, we also interpret a Parent-Child operator as a keyword selection applied on a second operation. Query Q3 can be interpreted as an Parent-Child operator where $\sigma_{at:author}$ is the parent and $\sigma_{v:john \wedge v:smith}$ is the child.

Set manipulation operators

These operators allow to manipulate nodes (dataset, entity, attribute and value) as sets, implementing union ($\cup$), difference ($\setminus$) and intersection ($\cap$). These operators are mapped one to one to the set operators found in the query algebra. For example, Query Q5 can be interpreted as an intersection $R_1 \cap R_2$ between two Parent-Child operators, $\sigma_{at:author}(\sigma_{v:john \wedge v:smith})$ and $\sigma_{at:title}(\sigma_{v:search \wedge v:engine})$.

Projection

The *dataset* and *entity* projection defined in Section 2.4.1 are simply performed by applying a filter over the node labels in order to keep the dataset and entity identifiers and filter out unnecessary identifiers such as the attribute or value identifier.

6.3.3. SPARQL Interpretation

In this section we discuss the extension by which, given the above discussed operators, it is possible to support a subset of the standard SPARQL query language. Such an interpretation is useful to understand the parallel between SPARQL and SIREn queries and to compare the performances of RDF databases against our approach in Section 6.6.5.

SPOC	POCS	OCSP	CPSO	CSPO	OSPC
(?,*,*,?)	(?,p,*,?)	(?,*,o,?)	(?,*,*,c)	(s,*,*,c)	(s,*,o,?)
	p	*o*	*c*	*c/s*	*s//o*
(s,*,*,?)	(?,p,o,?)	(?,*,o,c)	(?,p,*,c)	(s,p,*,c)	
s	*p/o*	*c//o*	*c//p*	*c/s/p*	
(s,p,*,?)	(?,p,o,c)	(s,*,o,c)			
s/p	*c//p/o*	*c/s//o*			
(s,p,o,?)					
s/p/o					

Table 6.1.: Quad patterns covered by outgoing relations and their interpretation with the SIREn operators. The *?* stands for the elements that are retrieved and the *** stands for a wildcard element.

By indexing outgoing relations alone, we can show to cover the quad access patterns listed in Table 6.1. A quad lookup is performed using the A//D or P/C operators. The covered quad patterns are a subset of the quad patterns covered by conventional RDF data management systems [HD05]. In fact, they give the ability to retrieve information about variables that are restricted to be at the subject or context position. By indexing also incoming relations, it becomes possible to retrieve information about variables at the object position.

6.4. Implementing the Model

We describe in this section how the tree model and query model is implemented on top of an inverted index. We first describe how we extend the inverted index data structure before explaining the incremental index updating and query processing algorithms. We also discuss how query results are ranked during query processing.

6.4.1. Inverted Index Structure

The node labels, or Dewey's vectors, that are associated with each term are stored within an inverted index. Compared to the inverted index described in Section 3.4.1, the difference is situated in the structure of the inverted lists. Originally, an inverted list is

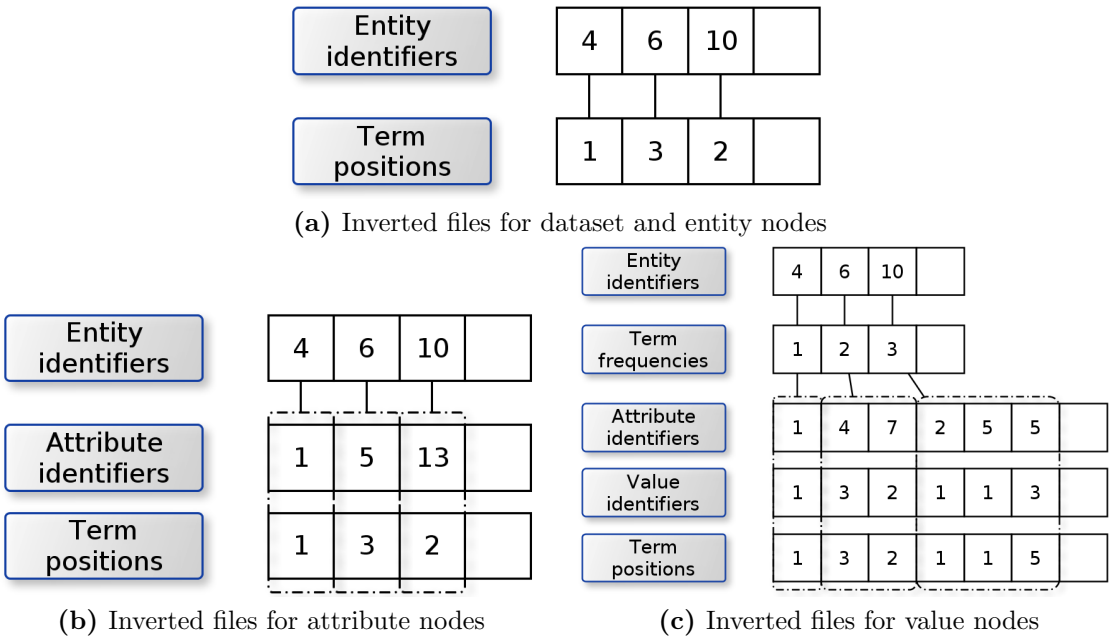
composed of a list of document identifiers, a list of term frequencies and a list of term positions. In our implementation, an inverted list is composed of five different streams of integers: a list of entity identifiers, a list of term frequencies, a list of attribute identifiers, a list of value identifiers and a list of term positions. The term frequency corresponds to the number of time the term has been mentioned in the entity description. The term position corresponds to the relative position of the term within the node. Each data stream is stored in a separated inverted file as explained in Section 3.4.1.

However, it is unnecessary to associate each term to five streams of integers. For example, a term appearing in an attribute node does not have value identifiers and a term in an entity node does not have attribute and value identifiers. Also, we assume that terms from entity, attribute and dataset nodes always appear a single time in the node. In this case, the term frequency is always equal to one, and it becomes unnecessary to store it. Such a choice has an impact on the way entities are ranked, since the term frequency is usually used for computing the impact of a term with respect to an entity description. However, this impact is minimal since the probability to have more than one occurrence of a same term in the entity, attribute and dataset nodes is very low. By carefully selecting what information is stored for each term, the index size is reduced and the overall performance is improved since less data has to be written during indexing and read during query processing. The strategy is to associate each term to a set of different inverted files depending on which node the term appears. Figure 6.2 depicts the inverted files associated to each type of nodes. We associate the following inverted files for each type of nodes:

dataset and entity: Terms from a dataset or entity node are associated to a list of entity identifiers and to their relative position within the node as shown in Figure 6.2a. The position of a term within a node is necessary to support phrase and proximity queries.

attribute: Terms from an attribute node are associated to a list of entity identifiers, a list of attribute identifiers and to their relative position within the node as shown in Figure 6.2b.

value: Terms from a value node are associated to a list of entity identifiers, a list of term frequencies, a list of attribute identifiers, a list of value identifiers and to their relative positions within the node as shown in Figure 6.2c. Here, the frequency of a term is not only necessary for computing the score impact of a term within a node, but it is also useful to associate more than one attribute identifier, value identifier and position to the current term occurrence. We consider that a term can



(a) Inverted files for dataset and entity nodes

(b) Inverted files for attribute nodes

(c) Inverted files for value nodes

Figure 6.2.: Diagram showing the set of inverted lists and their inter-connection for each type of terms.

appear more than once in the entity description or in a same node. Contrary to the other nodes where integers of each list have a one to one association, we can see in Figure 6.2c that each entity identifier is associated to a variable number of attribute identifiers, value identifiers and positions. The records are ordered first by attribute identifier then by value identifier and finally by position.

We can notice that we are not storing the dataset identifier of the Dewey's vector. Instead, we are encoding the relation between dataset terms and entity identifiers. This can be considered as a simplification of the data model from Section 6.2. This approach only enables a partial support of context queries since context queries such as Q9 can not be answered efficiently. However, this reduces the update complexity and therefore enables more efficient incremental updates. Such a choice is discussed more in details in Section 6.7.1.

A more compact representation for node-based index structure is to represent node labels as delta values, a technique first introduced in [SdDTZ97]. The key idea of the delta compression is to store the difference between consecutive values instead of the values themselves as explained in Section 3.4.2. If two consecutive term occurrences have overlapping paths, most of the delta values of the Dewey's vector will be zero. In addition, in our model, the delta values are much smaller than those obtained when indexing

text documents. This is due (1) to the usually more verbose and repetitive nature of structured data, e.g., the same URI used multiple times, and (2) to the locality of the attribute identifiers, the value identifiers and the term positions. The delta values are then compressed with the variable-length byte encoding, which is explained in Section 3.4.3. More advanced compression techniques could be implemented instead and this aspect is explored in Chapter 7.

6.4.2. Incremental Update of the Inverted Lists

The proposed model supports incremental updates of entities as it is performed for documents in traditional inverted indexes [ZM06]. Adding an entity corresponds to adding a set of statements to the inverted index. The statements are first transformed into a node-labelled tree data model as in Figure 6.1b. Then, for each term of the tree, the associated inverted lists are accessed and updated in consequence given the Dewey's vectors and positions of the term.

For example, to add a new occurrence of a term t from a value node, the following operations are performed:

1. the inverted files of t is accessed by performing a lookup in the lexicon;
2. a new entry is appended to the list of entity identifiers, term frequencies, attribute identifiers, value identifiers and positions.

If the same term t appears in a entity or dataset node, then the operations are similar than the previous ones with the difference that only the list of entity identifiers and term positions is accessed and updated.

The complexity of insertion of one term occurrence is $O(log(n) + k)$, where the term $log(n)$ denotes the cost of looking up a term in a lexicon of n terms and the term k denotes the cost of appending an integer to k inverted lists. The lexicon lookup is the predominant cost during the insertion of one term occurrence. However, updates are usually performed by batches of multiple entities. In this case, the update complexity becomes linear with the number of term occurrences since a lexicon lookup is only performed once per term.

Compared to common RDF databases, we do not support the deletion on a statement granularity, but we support the deletion of a dataset or entity. When an entity is removed, its identifier is inserted into a *deletion table*. When a dataset is deleted, the associated

entity identifiers are inserted into the deletion table. During query processing, each entity identifiers is checked against the deletion table in $O(1)$ to ensure that it has not been deleted. The deletion table is integrated back to the inverted index only when a certain amount of deletion is sufficient to amortize the cost of a such maintenance operation.

6.4.3. Query Processing

The evaluation of a query works in a bottom-up fashion. First, matching on the content (terms) of a node is performed, then node information is used during list intersection for filtering the result candidates that do not belong to a same node or branch of the tree.

The intersection of two inverted lists is the most common operation during query processing. For example, a content operator such as the boolean intersection and phrase proximity operators relies on a list intersection. The structure operators such as the Ancestor-Descendant, Parent-Child and boolean intersection operators also rely on list intersection. The methodology is identical for all of them and is described by the following merge algorithm:

1. The inverted list of each term is retrieved.
2. We position the pointers to the first element of each inverted list.
3. We then walk through the inverted lists simultaneously. At each step, we perform the following operations:
   a) We first compare their node information.
   b) If the comparison is positive,
      i. we add the entity identifier to the result list;
      ii. we advance the list pointers to their next position.
   c) If the comparison is negative, we move forward the list pointer with the smallest identifier to its next position.

However, the comparison between node information that is performed at step 3.a is slightly different depending on the query operator employed. In the case of a boolean intersection between two words, the algorithm compares first their entity identifiers, then their attribute identifiers and finally compare their value identifiers. In the case of a proximity operator, the position information is additionally compared. Concerning Ancestor-Descendant and Parent-Child operators, the comparison is restricted to the elements of the ancestor node to mimic the node label prefix matching as explained in Section 6.2. For example, if a word from a dataset node and a word from a value node

are intersected to check a Ancestor-Descendant relation, then only the entity identifier is compared.

Given the query `made / paper-9`, the query evaluation is performed as follows. In the following examples, we display the node information as a Dewey's vector and omit the position information for simplicity.

1. Postings List Fetching
   a) Retrieve the inverted list for the term "name": `[1.0],[1.2]`
   b) Retrieve the inverted list for the term "paper-9": `[1.2.1]`
2. Inverted List Merging
   a) position the pointers to the first element of the two lists.
   b) compare the entity identifiers. The two entity identifiers are equal to `1`.
   c) compare the attribute identifiers. The first pointer has an attribute identifier `<0>` inferior to the second pointer `<2>`. We move the first pointer to the next occurrence.
   d) compare the attribute identifiers. This time, the two attribute identifiers are equal to `2`. We have a match and add the entity identifier in the result list.

The worst-case complexity of a query evaluation is in time linear to the total number of term occurrences in the inverted list [MRS08]. In the average case, the complexity of an intersection is reduced to sub-linear time with the use of self-indexing [MZ96] over the entity identifiers in order to skip and avoid unnecessary record comparisons.

Each query operator delivers output in sorted order. Multiple operators can be nested without losing the sorted order of the output, therefore enjoying the concept of interesting orderings [Gra93] enabling the use of effective merge-joins without intermediate result sorting.

6.4.4. Top-K Query Processing

Top-k query processing is a crucial requirement in an environment such as the Web that involve a massive amount of data. End-users are more interested in the most relevant query answers from the potentially huge answer space. However, current query processors of triple stores do not handle query ranking efficiently. In contrast, SIREn gives a relevance measure for each search result using an adaptation of the well-known tf-idf approach.

In this section, our goal is not to present a new ranking scheme but instead to show how to apply existing schemes such as TF-IDF [Spa72] or BM25F [RZT04] to compute top-k results at query time based on the keywords and the structure of the matching sub-graphs. Also, it is to be noticed that the ranking scheme described here is the one used during the ranking experiments in Section 5.5.2 and Section 5.5.3.

The scoring function evaluates individuals by computing the score of the matching terms with the *term frequency - inverse entity frequency*, or TF-IEF. The *inverse entity frequency* measures the general importance of the term across the entity collection, and is defined as follow:

$$tf - ief(t, e) = \frac{n_{t,e}}{\sum_k n_{k,e}} \times \log \frac{|V^E|}{1 + n_{t,e}} \tag{6.1}$$

where $n_{t,e}$ is the number of occurrences of the term t in the entity e and $|V^E|$ denotes the total number of entities in the collection. Since URIs are treated as terms, we can apply the weighting scheme on URIs (or parts of them if URIs are normalized). For example, the TF-IEF scheme will give a low weight to an attribute or a value that occurs in many entities as, for example, for ubiquitous predicates like `rdf:type` or generic classes like `foaf:Agent`.

The score of an entity e is simply the sum of the score of the matching values weighted by the score of their relative attributes. Moreover, we apply the link analysis algorithm from Chapter 5 for assigning static weights $\omega(e)$ to entities. Given a boolean query q, a value v, an attribute a and an entity e, the score of e relative to q is:

$$\begin{aligned}
score(q, v) &= \sum_{t \in q, t \in v} tf.ief(t, e) \\
score(q, a) &= \sum_{v \in a} score(q, v) \times \sum_{t \in q, t \in a} tf.ief(t, e) \\
score(q, e) &= \sum_{a \in e} score(q, a) \times spread(q, e)
\end{aligned}$$

with

$$spread(q, e) = \frac{(|q| - |v|) + 1}{|q|}$$

where $|q|$ stands for the number of distinct terms in the query q and $|v|$ the number of values matching at least one keyword.

The spread normalisation factor favours queries that match terms within a single value. More the query terms are spread over multiple values, smaller the normalisation factor will be. For example, if a query is composed of 3 terms and all the terms occur in a same value, then the normalisation factor will be equal to $\frac{(3-1)+1}{3} = 1$. On the contrary, if each term occurs in a different value, the normalisation factor will be equal to $\frac{(3-3)+1}{3} = 1/3$.

In addition, we add a weight factor to words from a entity and dataset nodes. We consider keywords matching parts of the entity or dataset URI as a strong indication of the relevance of the entity. For example, the keyword *renaud* has a higher score impact when it matches the entity URI *http://renaud.delbru.fr/rdf/foaf#me* than when it matches the value *Renaud Delbru*.

During query processing the posting lists of all query terms are combined through a multi-way merge process. In such a concurrent traversal we compute the score of one entity at a time, similarly to the document-at-a-time scoring in text database. During this process, a fixed size candidate list of entities in order of increasing score is maintained in a priority queue and is then returned as the top-k result list.

6.5. Comparison among Entity Retrieval Systems

In this section, we evaluate four entity retrieval systems: SIREn based on a node-labelled index, field-based indexes [DH07], RDF databases [NW08] based on quad tables and Semplore [WLP+09]. These techniques are representative of the current approaches for entity retrieval.

Field-based indexing schemes are generally used in standard document retrieval systems such as Apache Lucene[1] to support basic semi-structured information like document's fields, e.g., the title. A field index is a type of sequence index scheme (see Section 3.3.2) that constructs lexicon terms by concatenating the field or attribute name, e.g., the predicate URI, with the terms from the content of this field. For example, in the graph depicted in Figure 2.4, the index terms for the entity "giovanni" and its predicate *name* will be represented as *name:giovanni* and *name:tummarello*. In fact, the field index encodes the relation between an attribute and a value directly within the lexicon.

[1] Apache Lucene: `http://lucene.apache.org/`

Criteria	Node Index	Field Index	Quad Table	Semplore
Dictionary Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n))$	$O(log(n))$
Quad Lookup	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n))$
Join in Quad Lookup	Yes	No	No	No
Star Query Evaluation	Sub-Linear	Sub-Linear	$O(n)$	$O(n*log(n))$
Update Cost	$O(log(n))$	$O(log(n*m))$	$O(log(n)+log(k))$	$O(log(n)+log(l))$
Multiple Indices	No	No	Yes	Yes
Query Expressiveness	Star	Star	Graph	Tree
Full-Text	Yes	Yes (on literals)	No	Yes (on literals)
Multi-Valued Support	Yes	No	Yes	No
Context	Partial	Partial	Yes	Partial
Precision (false positive)	No	Yes	No	Yes

Table 6.2.: Summary of comparison among the four entity retrieval systems

Semplore is an Information Retrieval engine for querying Semantic Web data which supports hybrid queries, i.e., a subset of SPARQL mixed with full text search. Semplore is also built on inverted lists and relies on three inverted indexes: (1) an ontology index that stores the ontology graph (concepts and properties), (2) an individual path index that contains information for evaluating path queries, and (3) an individual content index that contains the content of the textual properties.

In the following, we assume that term dictionaries as well as quad tables are implemented with a b+-tree. The comparison is performed according to the following criteria: *Processing Complexity*, *Update Complexity*, *Query Expressiveness* and *Precision*. *Processing Complexity* evaluates the theoretical complexity for processing a query (lookups, joins, etc.). *Update Complexity* evaluates the theoretical complexity of maintenance operations. *Query Expressiveness* indicates the type of queries supported. *Precision* evaluates if the system returns any false answers in the query result set.

6.5.1. Processing Complexity

Since the field-based index encodes the relation between an attribute and a value term in the dictionary, its dictionary may quickly become large when dealing with heterogeneous data. A dictionary lookup has a complexity of $O(log(n*m))$ where n is the number of terms and m the number of attributes. This overhead can have a significant impact on

the query processing time. In contrast, the other systems has a term dictionary of size n and thus a dictionary lookup complexity of $O(log(n))$.

To lookup a quad or triple pattern, the complexity of the node and field index is equal to the complexity of looking up a few terms in the dictionary. In contrast, RDF databases have to perform an additional lookup on the quad table. The complexity is $O(log(n) + log(k))$ with $log(n)$ the complexity to lookup a term in the dictionary and $log(k)$ the complexity to lookup a quad in a quad table, with k being the number of quads in the database. In general, it is expected to have considerably more quads than terms, with k generally much larger than n. Therefore, the quad table lookup has a substantial impact on the query processing time for very large data collection.

For quad patterns containing two or more terms, for example (?c,?s,p,o), the node index has to perform a merge-join between the posting lists of the two terms in order to check their relationships. However, this kind of join can be performed on average in sub-linear time. On the contrary, the other indexes do not have to perform such a join, since the field index encodes the relationship between predicate and object in the dictionary, the quad table in the b+-tree and Semplore in the inverted list for each term occurrence (but only for URI terms and not literal terms). Furthermore, in Semplore, access patterns where the predicate is not specified trigger a full index scan which is highly inefficient.

For evaluating a star-shaped query (joining multiples quad patterns), each index has to perform a join between the results of all the patterns. Such a join is linear with the number of results in the case of the quad table, and sub-linear for the node and field index with the use of the self-indexing method [MZ96]. In contrast, Semplore has often to resort to possibly expensive external sort before merge-join operations.

6.5.2. Update Complexity

In a b+-tree system the cost of insertion of one quad represents the cost of searching the related leaf node, i.e., $O(log(n) + log(k))$, the cost of adding a leaf node if there is no available leaf node and the cost of rebalancing the tree. These costs become problematic with large indices and requires advanced optimizations [Gra06] that in return cause a degradation in query performance. In contrast, the cost of insertion for a node and field index is equal to the cost of a dictionary lookup as discussed in Section 6.4.2, which is $O(log(n))$ and $O(log(n * m))$ for the node index and the field index

respectively. Furthermore, quad tables are specific to access patterns. Hence multiple b+-tree indexes, one for each access pattern, have to be maintain which limits effective caching. Concerning the size of the indexes, all of them are linear with the data.

Concerning Semplore, the original system could not perform updates or deletions of triples without a full re-indexing. The authors have recently [WLP+09] proposed an extension for incremental maintenance operations based on the *landmark* [LWP+03] technique but the update complexity remains sustained. The update cost is $O(log(n) + log(l))$ with l the number of landmarks in the inverted list. The fact that Semplore uses multiple indexes and landmarks considerably increase the update complexity. For example, index size and creation time reported in [WLP+09] are higher than for the state-of-the-art RDF database RDF-3X [NW08].

6.5.3. Query Expressiveness

In term of query expressiveness, RDF databases have been designed to answer complex graph-shaped queries which are a superset of the queries supported by the other systems. On the other hand, the other systems are especially designed to support natively full-text search which is not the case for quad table indexes. Compared to field index and Semplore, the node index provides more flexibility since it enables to search keywords on every parts of a quad. In addition, node indexes support set operations on every nodes, giving the ability to express queries on both URI and literal multi-valued properties. For example, a field-based index and Semplore cannot process the query Q2 without potentially returning false-positive results.

While Semplore supports relational, tree shaped, queries, it does not index relations between a resource and a literal. Hence, it is not possible to restrict full-text search of a literal using a predicate, e.g., queries such as `(?s, <foaf:name>, "renaud")`.

The node indexing scheme, field-based indexing scheme and Semplore only support context queries partially. The three systems are using an identical technique that consists of encoding the relation between context terms with the entity identifiers as explained in Section 6.4. This approach makes difficult the processing of the context query Q9.

6.5.4. Precision

The field indexing scheme encodes the relation between an attribute and and a value term in the index dictionary, but loses an important structural information: the distinction between multiple values. As a consequence, if an attribute is multi-valued, the field index may return false-positive results. Semplore suffers from a similar problem: it aggregates all the values of an entity, disregarding the attribute, into a single bag of words. On the contrary, the node index and the quad table are able to distinguish distinct values and do not produce wrong answers.

6.6. Experiments

In this section, we compare the performance of SIREn against RDF databases based on some of the above criteria. We assess the space requirement, the index creation time and the query processing performance. The aim is to show the benefits of using a system like SIREn for web entity retrieval compared to common approaches based on RDF databases. While RDF databases are very efficient to answer complex queries, we show that for the simpler task of entity retrieval, carefully designed systems can provide substantial benefits in term of scalability while sustaining fast query time.

At the time the experiments were performed, the prototype system was not including the recent advances in index compression that are described in Chapter 7. Instead, the prototype was relying on the variable-length byte encoding technique (see Section 3.4.3). The gain of performance using a more advanced compression technique such as the Adaptive Frame of Reference presented in Chapter 7 can theoretically be reflected in these experiment results. In particular, we can expect a decrease of index size by a factor of 3 and a decrease of 30% of query processing time.

6.6.1. Experimental Settings

The experimental setup is as follows. SIREn is implemented on top of Apache Lucene 2.4[2]. The first RDF database is OpenRDF Sesame 2.0[3] with its native backend, an open-source system which is commonly used as baseline for comparing quad store performances, e.g.,

[2]Apache Lucene: `http://lucene.apache.org/`
[3]OpenRDF Sesame: `http://www.openrdf.org/`

in [NW08]. The second system is the state-of-the-art triple store RDF-3X [NW08]. All the RDF databases are based on quad or triple tables and backed by b+-tree indexes. We report that it is impossible for us to compare Semplore because at the time of the writing it is not being made available for this purpose. We also do not compare field-based index due to their query expressiveness limitations. In a previous publication [DTC+09], we reported experimental results showing the decrease of performance of field-based index compared to SIREn when the number of fields increases.

The machine that serves the experiments is equipped with 8Gb ram, 2 quad core Intel processors running at 2.23 Ghz, 7200 RPM SATA disks, linux 2.6.24-19, Java Version 1.6.0.06 and GCC 4.2.4. All the following benchmarks are performed with cold-cache by using the `/proc/sys/vm/drop_caches` interface to flush the kernel cache and by reloading the application after each query to bypass the application cache. Running the benchmarks under cold-cache enables more precise measurements which take into account the disk IO costs of the dictionary lookups and query evaluation.

6.6.2. Data Collection

For the experiments, we use two datasets. The first one, called "Real-World" has been obtained by random sampling the content of the Sindice search engine. The real world dataset consists of 10M triples (approximately 2Gb in size), and contains a balanced representation of triples coming from the Web of Data, e.g., RDF and Microformats data published online. The second dataset is the MIT Barton dataset that consists of 50M triples (approximatively 6Gb in size). Each dataset has been split on a per entity basis as shown in Figure 2.4 with outgoing relations only. During data preprocessing, URIs are not tokenised.

6.6.3. Index Size

The first experiment compares the index size of the three systems. The index size comprises the size of the lexicon and the size of the indices. Sesame is configured to create a single quad table (p,o,c,s). RDF-3X creates all the possible triple tables plus additional tables for query optimizations. SIREn creates a single inverted index.

The final index size for each system is reported in Table 6.3a. With respect to SIREn, Sesame exhibits at least a two-fold increase in index size on the real-world dataset and

Dataset	SIREn	Sesame	RDF-3X
Barton	789	3400	3076
Real-World	296	799	1138

(a) Index size in MB per dataset and system

Dataset	SIREn10	SIREn100	Sesame	RDF-3X
Barton	3	1.5	266	11
Real-World	1	0.5	47	3.6

(b) Indexing time in minutes per dataset and system

Table 6.3.: Report on index size and indexing time

a four-fold increase on Barton. RDF-3X exhibits a four-fold increase in index size on the two datasets. With respect to the original dataset size, we observe that SIREn exhibits a index size ratio of 13-15%, whereas for Sesame and RDF-3X the ratio is approximately 50%. While the index size is linear with the size of the data collection for all the systems as discussed in Section 6.5, we can observe that the duplication of indices in RDF databases is causing a significant increase in index size compared to SIREn.

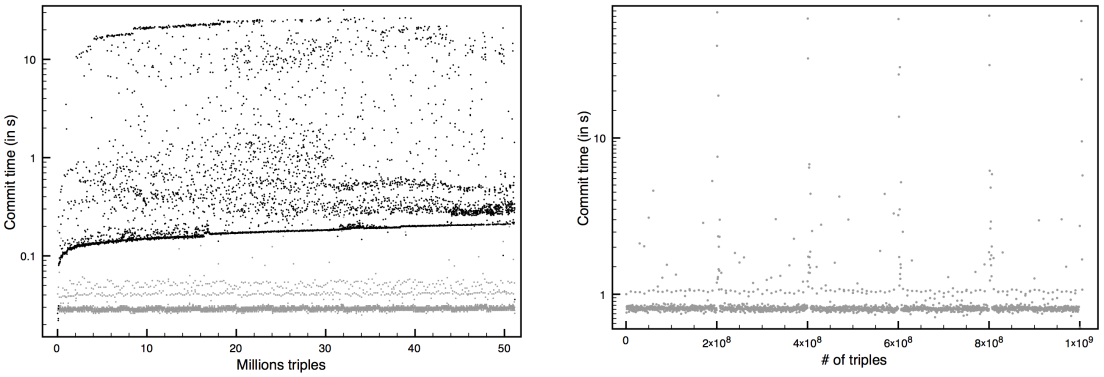
6.6.4. Insertion Time

In Table 6.3b, we report the index creation time for the two datasets. For SIREn we report two cases: SIREn10 is the time to construct the index by batch of 10000 triples while SIREn100 is the time by batch of 100000 triples. Concerning RDF-3X, it is important to notice that (1) it does not support context, therefore it indexes triples and not quads, and that (2) it does not support incremental indexing; RDF-3X needs the full dataset beforehand in order to construct the indexes in a single batch process, as opposed to Sesame which supports incremental updates. We can see from the results in Table 6.3b that SIREn is 50 to 100 times faster than Sesame and 3 to 6 times faster than RDF-3X.

In the next test, we plot the performance of SIREn and Sesame in an incremental update benchmark. The Figure 6.3a shows the commit times for an incremental 10.000 triples batch on the two systems[4]. The graph is reported in logarithmic scale. While the absolute time is significant, the important result is the constant time exhibited by SIREn for incremental updates, as compared to Sesame performance which progressively decreases as the number of quads increases as explained in Section 6.5.

In Figure 6.3b, the commit time of SIREn is plotted for a synthetic dataset constructed by replicating Barton 20 times to reach 1 billion triples. The total index creation time is

[4]We omit the commit times for the Real-World dataset since the results were similar to the Barton dataset



(a) Plots showing the commit time every 10.000 triples during the index creation on Barton

(b) Plots showing the commit time every 500.000 triples during the index creation on the billion triples dataset

Figure 6.3.: Dark dots are Sesame commit time records while gray dots are SIREn commit time records

31 minutes. We can notice that SIREn keeps a constant update time during the entire indexing. Outliers are due to periodic merges of the index segments. These results show that SIREn scales well with a large number of triples and provides significant improvement in terms of incremental update compared to RDF databases.

6.6.5. Query Time Execution

For the query time execution benchmark, we created multiple sets of queries with increasing complexity. The first set of queries (A*) consist of simple term lookups (URIs or literals). The second set of queries (B*) contains triple pattern lookups. The other sets consist of a combination of triple patterns using different set operators (intersection: C*, union: D*, exclusion: E). The queries are provided in Appendix B. For each query we average 50 query execution times without considering the final mapping between the result ids and their string values. The query time reported in Table 6.4a is the CPU time, i.e., user time and system time, used by the current thread. The query times are reported in Table 6.4a for the Barton dataset and in Table 6.4b for the Real-World dataset.

Since RDF-3X does not support native full-text search, we were unable to test queries involving this aspect. With respect to query E only SIREn was able to execute it since RDF-3X does not support the `bound` operator that is necessary to implement *exclusion* of triple patterns. With respect to Sesame, we decided not to include it in this test as

System	A1	A2	B1	C1	C2	D1	D2	E
RDF-3X	16.12	0.12	1.38	1.16	0.38	0.23	0.14	X
SIREn	2.79	0.02	1.33	1.71	0.95	0.36	0.03	0.96

(a) Barton dataset

System	A1	A2	B1	B2	C1	C2	D1	E
RDF-3X	0.29	0.12	0.17	0.18	0.21	0.13	0.28	X
SIREn	0.23	0.03	0.04	0.05	0.09	0.08	0.16	0.53

(b) Real-World dataset

Table 6.4.: Querying time in seconds

during the experimentation phase we found that the results that we have obtained were consistently outperformed by RDF-3X.

The first observation is that on the Real-World dataset, SIREn performs significantly better, approximately 2 to 4 times, in 6 queries out of 7 while performing similarly in one, A1, a query which produces a very large amount of results. In these queries and due to skewness of real-world data, SIREn are able to take advantage of its sub-linear merge-join by skipping unnecessary record comparisons.

On the Barton dataset, we notice however that for 3 queries out of 7 SIREn performs approximately 5 to 6 faster than RDF-3X, while resulting slower but comparable in 3 out of 7. In a particular query, C2, SIREn under-performs approximately 3 times. The explanation is that this query uses multiple triple access patterns that requires SIREn to perform more term lookups and merge-joins compared to RDF-3X. However, despite this overhead, the next experiment shows that SIREn scales well with a very large amount of data.

6.6.6. Scalability

We evaluate SIREn scalability by indexing a dataset composed of 1 billion entities described with approximately 10 billion triples (one Terabyte of data). The dataset is derived from the billion triple challenge dataset[5]. To avoid hitting the limit of 2 billion

[5]Semantic Web Challenge: `http://challenge.semanticweb.org/`

	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Time	0.75	1.3	1.4	0.5	1.5	1.6	4
Hits	7552	9344	3.5M	57K	448	8.2M	20.7M

Table 6.5.: Querying time in seconds and number of hits for the 10 billion triples benchmark

entities due to the current implementation, we remove entities with only one or two triples and duplicate the remaining triples to reach 10 billion. With respect to RDF databases, we were unable to load that amount of data on a single computer. We therefore excluded them from this experiment.

Since the dataset is different from the one in previous experiments, we use a different albeit comparable set of queries which is also provided in Appendix B. The query time reported in Table 6.5 is the CPU time, i.e., user time and system time, used by the current thread.

Q1 to Q4 are predicate-object lookups using terms that are more or less frequent. In Q1 and Q2, we request for an infrequent predicate-object. The first query, while giving a result set of similar size, performs approximately two times better than Q2. The difference is that Q1 uses an infrequent predicate while Q2 a very frequent one, which in the latter case causes an overhead during the merge-join. However, Q3 and Q4 use also frequent terms and, despite of the large increase of hits, the performance is similar or even better than Q2, which underlines that the complexity is dependent of the length of the term inverted list, i.e., the predicate inverted list in the case of Q2.

Q5 performs a union between two infrequent predicate-object using a frequent property term. Again, we can observe the overhead caused by the merge-join between a frequent predicate and an object term. While both Q6 and Q7 contain frequent predicates and return a large number of results, we can observe that the system scales linearly between Q6 and Q7 with the number of hits. The overhead of the merge-join becomes less significant.

This scalability experiment shows that SIREn, even if there is a slight overhead when frequent predicates are used in the query, scales well with a large number of triples and provides in all the cases reasonable query times, which makes it suitable for the web entity retrieval scenario.

6.7. Discussion

6.7.1. Handling Context

As we explained in Section 6.4, the current implementation supports partially context queries. Context queries such as Q8 that restrict entity matching to a certain context are possible. However, it is not possible to support efficiently context queries such as Q9 that retrieve all contexts involving two or more entity queries. The reason is that it is not possible at query time to know the context identifier associated to each entity, thus making impossible the intersection of the two entity relations based on an equality condition over their context identifier.

In order to support such an intersection, we would have to encode the relation between the dataset node with all the other nodes by storing a list of dataset identifiers in every inverted lists. However, this means that the update complexity will increase since it is necessary to keep a global order in the dataset identifier lists and a local order in the entity identifier lists with respect to each dataset. With such requirements, it is difficult to implement an efficient incremental update procedure.

6.7.2. Query Processing Overhead

We have explained in Section 6.6.5 and shown in Section 6.6.6 that the node indexing scheme suffers from an overhead during query processing. The overhead occurs during the intersection of the inverted lists when processing attribute queries (see Section 2.4.2). The overhead becomes more noticeable when intersection is performed between two very common terms, e.g., an ubiquitous predicate such as `rdf:type` and a frequent class such as `foaf:Person`. We show in Chapter 7 that this overhead can be greatly reduced with an appropriate compressing method.

6.8. Conclusion and Future Work

We presented SIREn, a node-based indexing scheme for semi-structured data. SIREn is designed for indexing very large datasets and handling the requirements of indexing and querying Web Data: constant time incremental updates and very efficient entity lookup using semi-structural queries with full text search capabilities. With respect to

DBMS and IR systems, SIREn positions itself somewhere in the middle as it allows semi-structural queries while retaining many desirable IR features: single inverted index, effective caching, top-k queries and efficient index distribution over shards.

SIREn is in production at the core of the Sindice semantic search engine. At the time of the writing, SIREn serves over 110 million harvested web pages containing RDF or Microformats and answers several tens of thousands queries per day on a single machine.

Future works will concentrate on increasing the query expressiveness of the system. We have to investigate the feasibility of path-based queries or associative queries which will enable to query relations between entities. However, supporting such queries while keeping a system in the same class of scalability is still an open problem. Also, the current scoring function for top-k query processing is quite simplistic. Future work is necessary to extend the scoring function in order to take into account the structure of the data.

Chapter 7.

Adaptive Frame Of Reference for Compressing Inverted Lists

7.1. Introduction

The performance of Information Retrieval systems is a key issue in large web search engines. The use of appropriate compression techniques is partially accountable for the current performance achievement of web search engines. Compression is not only useful for saving disk space, but it also maximises IO throughput [BC07] and therefore increases query throughput. In this paper, we introduce a new class of compression techniques for inverted indexes that provides fast query response time, good compression ratio but also fast indexing time.

In the past years, compression techniques have focussed on CPU optimised compression algorithms [GRS98, AM05, ZHNB06, AM10]. It has been shown in [AM05, ZHNB06, AM10] that the decompression performance depends on the complexity of the execution flow of the algorithm. Algorithms that require branching conditions tend to be slower than algorithms optimised to avoid branching conditions. In fact, simplicity over complexity in compression algorithms is a key for achieving high performance. The challenge is however to obtain a high compression rate while keeping the execution flow simple.

Previous works [AM05, BV05, ZHNB06, ZLS08, YDS09, AM10] have focussed solely on two factors, the compression ratio and the decompression performance, disregarding the compression performance. While decompression performance is essential for query throughput, compression performance is crucial for update throughput. We therefore propose to study compression techniques with an additional third factor, the compression

performance. We show that compression performance is also dependent on an optimised execution flow.

In the remainder of this chapter, we introduce a new class of compression algorithms, the Adaptive Frame of Reference. We compare our approach against a number of state-of-the-art compression techniques for inverted indexes. We perform a detailed experimental evaluation based on three factors: indexing time, compression ratio and query processing time. With respect to the node-based index presented in Chapter 6, we show that significant performance improvements can be achieved on the indexing time and compression ratio while maintaining one of the fastest query processing time.

7.2. Adaptive Frame of Reference

The Adaptive Frame Of Reference (AFOR) attempts to retain the best of FOR, i.e., a very efficient compression and decompression using highly-optimised routines, while providing a better tolerance against outliers and therefore achieving a higher compression ratio. Compared to PFOR, AFOR does not rely on the encoding of exceptions in the presence of outliers. Instead, AFOR partitions a block into multiple frames of variable length, the partition and the length of the frames being chosen appropriately in order to adapt the encoding to the value distribution.

To elaborate, AFOR works as follow. Given a block B of n integers, AFOR partitions it into m distinct frames and encodes each frame using highly-optimised routines. Each frame is independent from each other, i.e., each one has its own *bit frame*, and each one encodes a variable number of values. This is depicted in Figure 7.1 by *AFOR-2*. Along with each frame, AFOR encodes the associated bit frame with respect to a given encoder, e.g., a binary encoder. In fact, AFOR encodes (resp., decodes) a block of values by:

1. encoding (resp., decoding) the bit frame;
2. selecting the compression (resp., decompression) routine associated to the bit frame;
3. encoding (resp., decoding) the frame using the selected routine.

Finding the right partitioning, i.e., the optimal configuration of frames and frame lengths per block, is essential for achieving high compression ratio [VS10]. If a frame is too large, the encoding becomes more sensitive to outliers and wastes bits by using an inappropriate bit frame for all the other integers. On the contrary, if the frames are too

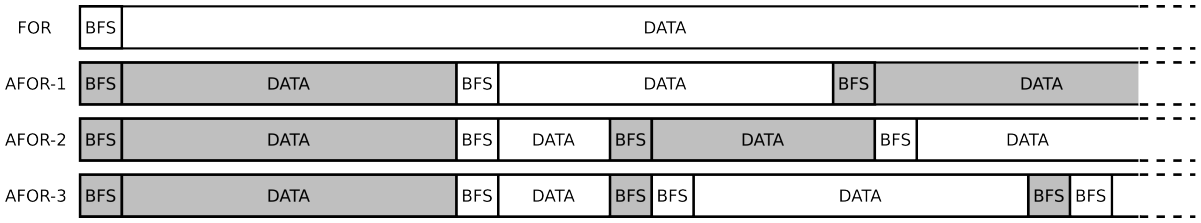


Figure 7.1.: Block compression comparison between FOR and AFOR. We alternate colours to differentiate frames. AFOR-1 denotes a first implementation of AFOR using a fixed frame length. AFOR-2 denotes a second implementation of AFOR using variable frame lengths. AFOR-3 denotes a third implementation using variable frame lengths and the frame stripping technique. *BFS* denotes the byte storing the bit frame selector associated to the next frame, and enables the decoder to select the appropriate routine to decode the following frame.

small, the encoding wastes too much space due to the overhead of storing a larger number of bit frames. The appropriate strategy is to rely on large frames in the presence of a dense sequence of values, and on small frames in the presence of sparse sequence of values. Also, alternating between large and small frames is not only important for achieving high compression ratio but also for achieving high performance. If frames are too small, the system has to perform more table lookups to select the appropriate routine associated to each frame, and as a consequence the compression and decompression performance decrease. Therefore, it is best to rely on large frames instead of multiple smaller frames when it is possible. To find a block partitioning, our solution uses a local optimisation algorithm which is explained next.

7.2.1. Partitioning Blocks into Variable Frames

Finding the optimal configuration of frames and frame lengths for a block of values is a combinatorial problem. For example, with three different frame lengths (32, 16 and 8) and a block of size 1024, there are 1.18×10^{30} possible combinations. While such a combinatorial problem can be solved via Dynamic Programming algorithms [VS10], the complexity of such algorithms is still $O(n \times k)$, with the term n being the number of integers and the term k the size of the larger frame, and therefore greatly impacts the compression performance. We remind the reader that we are interested not only by fast decompression speed and high compression ratio, but also by fast compression speed. Therefore, in our experiments, we do not rely on the optimal configuration. Instead, we

use a local optimisation algorithm that provides a satisfactory compression rate and that is efficient to compute.

AFOR computes the block partitioning by using a sliding window over a block and determines the optimal configuration of frames and frame lengths for the current window. Given a window of size w and a list of possible frame lengths, we compute beforehand the possible configurations. For example, for a window size of 32 and three different frame lengths, 32, 16 and 8, there are six configurations: [32], $[16, 16]$, $[16, 8, 8]$, $[8, 16, 8]$, $[8, 8, 16]$, $[8, 8, 8, 8]$. The size of the window as well as the number of possible frame lengths are generally chosen to be small in order to reduce the number of possible configurations. Then, we first compute the bit frames of the smallest frames by doing one pass over the values of the window as shown in the Algorithm 1 (lines 1-5). On the previous example, this means that we compute the bit frames for the configuration $[8, 8, 8, 8]$. The `bitFrames` array stores the bit frame for each of frame of this configuration. Given these bit frames, we are able to compute the bit frames of all the other frames. The second step, lines 6-12 in Algorithm 1, iterates over the possible configurations and estimates the size of each configuration in order to find the optimal one for the current window. Given the previously computed `bitFrames` array, the `EstimateSize` function computes the cost of encoding the window for a given configuration, accounting also the overhead of storing the bit frames. For example, for the configuration $[8, 8, 8, 8]$ with four frames of size 8 each, and with four associated bit frames, b_1 to b_4, the size of the encoding is computed as follow: $(4 \times 8) + 8 \times \sum_{i=1...4} b_i$, where $8 \times \sum_{i=1...4} b_i$ is the size (in bits) of the four encoded frames and 4×8 is the overhead (in bits) to store the four bit frames.

This simple algorithm is efficient to compute, in particular if the window size is small and if there is a few number of possible frame lengths. However, it is easy to see that such a method does not provide the optimal configuration for a complete block. There is a trade-off between optimal partitioning and complexity of the algorithm. One can possibly use a more complex method for achieving a higher compression if the compression speed is not critical. However, this is not the case for a web search engine where high update throughput is crucial. We decided to use this method since in our experiments we found that a small window size of 32 values and three frame lengths, 32, 16 and 8, were providing satisfactory results in term of compression speed and compression ratio. More details about our implementations of AFOR are given next. As a final reminder, we point out that we are interested by fast compression speed and by presenting the general benefits of AFOR. We therefore do not compare our partitioning scheme with *optimal* partitioning in this chapter, and defer this task for a future work.

Algorithm 1: The algorithm that finds the best configuration of frames and frame lengths for a window W of size w.

input : A window W of size w
input : The smallest frame length l
output : The best configuration for the window

```
1  for i ← 0 to w/l do
2  |   for j ← i × l to (i + 1) × l do
3  |   |   bitFrames[i] ← max(bitFrames[i], ⌈log2(W[j] + 1)⌉);
4  |   end
5  end
6  bestSize ← MaxSize;
7  foreach configuration c of the possible configurations do
8  |   if EstimateSize(c, bitFrames) < bestSize then
9  |   |   bestSize ← EstimateSize(c);
10 |   |   bestConf ← c;
11 |   end
12 end
```

7.2.2. Frame Stripping

In an inverted list, it is common to encounter a long sequence of 1 to encode. For example, this occurs with terms that appear frequently in many entities. With RDF data, such a very common term might be a predicate URI or a ubiquitous class URI. As a consequence, the list of entity identifiers is composed of many consecutive identifiers, which is encoded as a list of 1 using the delta representation. Also, we noticed that when indexing batch of entities coming from a same dataset, the schema used across the entity descriptions are very similar. In that case, all the schema terms become associated with long runs of consecutive entity identifiers in the inverted index. A long run of 1 is also common in:

- the list of term frequencies for terms that appear frequently a single time in the entity description, e.g., class URIs;
- the list of value identifiers for terms that appear frequently in single-valued attributes;
- the list of term positions for nodes holding a single term, e.g., URIs.

However, in presence of such long runs of 1, AFOR still needs to encode each value using 1 bit. For example, a frame of 32 values will encode a sequence of 1 using 32 bits. The goal of the *Frame Stripping* method is to avoid the encoding of such frames.

Our solution is to *strip* the content of a frame if and only if the frame is exclusively composed of 1. We encode such a case using a special bit frame. Compared to AFOR-2, the compressed block can contain frames which is then encoded by a single bit frame as depicted in Figure 7.1 by *AFOR-3*.

7.2.3. Implementation

We present three different implementations of the AFOR encoder class. We can obtain many variations of AFOR by using various sets of frame lengths and different parameters for the partitioning algorithm. We tried many of them during our experimentation and report here only the ones that are promising and interesting to compare.

AFOR-1

The first implementation of AFOR, referred to as AFOR-1 and depicted in Figure 7.1, is using a single frame length of 32 values. To clarify, this approach is identical to FOR applied on small blocks of 32 integers. This first implementation shows the benefits of using short frames instead of long frames of 1024 values as in our original FOR implementation. In addition, AFOR-1 is used to compare and judge the benefits provided by AFOR-2, the second implementation using variable frame lengths. Considering that, with a fixed frame length, a block is always partitioned in the same manner, AFOR-1 does not rely on the partitioning algorithm presented previously.

AFOR-2

The second implementation, referred to as AFOR-2 and depicted in Figure 7.1, relies on three frame lengths: 32, 16 and 8. We found that these three frame lengths give the best balance between performance and compression ratio. Additional frame lengths were rarely selected and the performance was decreasing due to the larger number of partitioning configurations to compute. Reducing the number of possible frame lengths was providing slightly better performance but slightly worse compression ratio. There is a trade-off between performance and compression effectiveness when choosing the right set of frame lengths. Our implementation relies on the partitioning algorithm presented earlier, using a window's size of 32 values and six partitioning configurations $[32], [16, 16], [16, 8, 8], [8, 16, 8], [8, 8, 16], [8, 8, 8, 8]$.

AFOR-3

The third implementation, referred to as AFOR-3 and depicted in Figure 7.1, is identical to AFOR-2 but employs the frame stripping technique. AFOR-3 implementation relies on the same partitioning algorithm than AFOR-2. However, we perform an additional step to find and strip frames composed of a sequence of 1 in the partitions.

Compression and decompression routines

For question of efficiency, our implementations rely on highly-optimised routines such as the ones presented in Listing 3.1 and 3.2, where each routine is loop-unrolled to encode or decode a fixed number of values using shift and mask operations only. In our implementation, there is one routine per bit frame and per frame length. For example, for a bit frame of 3, we need a different set of instructions depending on the size of the frame. For a frame length of 8 values, the routine encodes 8 values using 3 bits each as shown in Listing 3.1, while for a frame length of 32, the routine encodes 32 values using 3 bits each. One could use the routines for a frame length of 8 in a loop to encode and decode a frame of 32, instead of using especially made routines for 32 values, but at the cost of 4 loop conditions and 4 function calls.

Since AFOR-1 uses a single frame length, it only needs 32 routines for compression and 32 routines for decompression, i.e., one routine per bit frame (1 to 32). With respect to AFOR-2, since it relies on three different frame lengths, it needs 96 routines for compression and 96 routines for decompression. With respect to AFOR-3, one additional routine for handling a sequence of 1 is added per frame length. The associated compression routine is empty and does nothing since the content of the frame is not encoded. Therefore the cost is reduced to a single function call. The decompression routine consists of returning a list of 1. Such routines are very fast to execute since there are no shift or mask operations.

Bit frame encoding

We remind that the bit frame is encoded along with the frame, so that, at decompression time, the decoder can read the bit frame and select the appropriate routine to decode the frame. In the case of AFOR-1, the bit frame varies between 1 to 32. However, for AFOR-2, there are 96 cases to be encoded, where cases 1 to 32 refer to the bit frames for

a frame length of 8, cases 33 to 63 refer to the bit frames for a frame length of 16, and cases 64 to 96 refer to the bit frames for a frame length of 32. In AFOR-3, we encode one additional case per frame length with respect to the frame stripping method. Therefore, there is a total of 99 cases to be encoded. The cases 97 to 99 refer to a sequence of 1 for a frame length of 8, 16 and 32 respectively.

In our implementation, the bit frame is encoded using one byte. While this approach wastes some bits each time a bit frame is stored, more precisely 3 bits for AFOR-1 and 1 bits for AFOR-2 and AFOR-3, the choice is again for a question of efficiency. Since bit frames and frames are interleaved in the block, storing the bit frame using one full byte enables the frame to be aligned with the start and end of a byte boundary. Another possible implementation to avoid wasting bits is to pack all the bit frames at the end of the block. We tried this approach and report that it provides slightly better compression ratio, but slightly worse performance. Since the interleaved approach was providing better performance, we decided to use this one in our experiment.

Routine selection

A precomputed lookup table is used by the encoder and decoder to quickly select the appropriate routine given a bit frame. Compared to AFOR-1, AFOR-2 (and AFOR-3) has to perform more table lookups for selecting routines since AFOR-2 is likely to rely on small frames of 8 or 16 values when the value distribution is sparse. While these lookups cost additional CPU cycles, we will see next that the overhead is minimal.

7.3. Experiments

This section describes the benchmark experiments which aim to compare the techniques introduced by AFOR with the various compression methods described in Section 3.4.3. The first experiment measures the indexing performance based on two aspects: (1) the indexing time; and (2) the index size. The second experiment compares the query execution performance.

7.3.1. Experimental Settings

The hardware system we use in our experiments is a 2 x Opteron 250 @ 2.4 GHz (2 cores, 1024 KB of cache size each) with 4GB memory and a local 7200 RPM SATA disk. The operating system is a 64-bit Linux 2.6.31-20-server. The version of the Java Virtual Machine (JVM) used during our benchmarks is 1.6.0_20. The compression algorithms and the benchmark platform are written in Java and based on the open-source project Apache Lucene[1].

7.3.2. Experimental Design

Each measurement where made by (1) flushing the OS cache; (2) initialising a new JVM and (3) warming the JVM by executing a certain number of times the benchmark. The JVM warmup is necessary in order to be sure that the OS and the JVM have reached a steady state of performance, e.g., that the critical portion of code is JIT compiled by the JVM. The implementation of our benchmark platform is based on the technical advices from [Boy08], where more details about the technical aspects can be found.

7.3.3. Data Collection

We use three real web datasets for our comparison:

Geonames: a geographical database and contains 13.8 million of entities[2]. The size is 1.8GB compressed.

DBPedia: a semi-structured version of Wikipedia and contains 17.7 million of entities[3]. The size is 1.5GB compressed.

Sindice: a sample of the data collection currently indexed by Sindice. We randomly sampled half of the semi-structured documents. For each document, we extracted the entities as pictured in Figure 2.4. We filtered out all the entity descriptions containing less than two facts. There is a total of 130.540.675 entities. The size is 6.9GB compressed.

[1]Apache Lucene: `http://lucene.apache.org/`
[2]Geonames: `http://www.geonames.org/`
[3]DBpedia: `http://dbpedia.org/`

7.3.4. Indexing Performance

The performance of indexing is compared based on the index size (compression ratio), commit time (compression speed) and optimise time (compression and decompression speed). The indexing is performed by adding incrementally 10000 documents at a time and finally by optimising the index. For each batch of documents, the commit operation creates a small inverted index (called an *index segment*). The optimisation merges all the index segments into a single segment. We believe this to be a common operation for incremental inverted index.

We report the results of the indexing experiments in Table 7.1. The table comprises two columns with respect to the indexing time: the total commit time (*Total*) to add all the documents and the optimisation time (*Opt*). The time collected is the CPU time used by the current thread and comprises the user time and the system time. The index size in Table 7.1 is studied based on the size of the individual inverted file (entity, frequency, attribute, value and position) and on the total index size (by summing the size of the five inverted files). We also provide bar plots to visualise better the differences between the techniques.

Commit Time

Figure 7.2 shows the total time spent by each method. As might be expected, Rice is the slowest method due to its execution flow complexity. It is followed by FOR and PFOR. We can notice the inefficiency of PFOR in term of compression speed. On a large dataset (Sindice), VByte is the best-performing method while AFOR-1, AFOR-2, AFOR-3 and S-64 provide a similar commit time. On DBpedia, AFOR-1 and AFOR-2 are the best performing methods. On Geonames, AFOR-1, AFOR-2 and AFOR-3 are the best performing methods, while S-64 and VByte perform similarly. On smaller datasets (DBpedia and Geonames), VByte, FOR and PFOR perform similarly.

Optimisation Time

Figure 7.3 shows the optimise time for each methods. The time to perform the optimisation step is quite different due to the nature of the operation. The optimisation operation has to read and decompress all the index segments and compress them back into a

	Time (s)		Sizes (Gb)					
Method	Total	Opt	Ent	Frq	Att	Val	Pos	Total
AFOR-1	536	139	0.246	0.043	0.141	0.065	0.180	0.816
AFOR-2	540	142	0.229	0.039	0.132	0.059	0.167	0.758
AFOR-3	562	158	0.229	0.031	0.131	0.054	0.159	0.736
FOR	594	158	0.315	0.061	0.170	0.117	0.216	1.049
PFOR	583	167	0.317	0.044	0.155	0.070	0.205	0.946
Rice	612	239	0.240	0.029	0.115	0.057	0.152	0.708
S-64	558	162	0.249	0.041	0.133	0.062	0.171	0.791
VByte	577	164	0.264	0.162	0.222	0.222	0.245	1.335

(a) DBpedia

	Time (s)		Sizes (Gb)					
Method	Total	Opt	Ent	Frq	Att	Val	Pos	Total
AFOR-1	729	114	0.129	0.023	0.058	0.025	0.025	0.318
AFOR-2	732	107	0.123	0.023	0.057	0.024	0.024	0.307
AFOR-3	724	103	0.114	0.006	0.056	0.016	0.008	0.256
FOR	748	102	0.150	0.021	0.065	0.025	0.023	0.349
PFOR	741	134	0.154	0.019	0.057	0.022	0.023	0.332
Rice	787	183	0.133	0.019	0.063	0.029	0.021	0.327
S-64	740	123	0.147	0.021	0.058	0.023	0.023	0.329
VByte	737	141	0.216	0.142	0.143	0.143	0.143	0.929

(b) Geonames

	Time (s)		Sizes (Gb)					
Method	Total	Opt	Ent	Frq	Att	Val	Pos	Total
AFOR-1	13734	1816	2.578	0.395	0.942	0.665	1.014	6.537
AFOR-2	13975	1900	2.361	0.380	0.908	0.619	0.906	6.082
AFOR-3	13847	1656	2.297	0.176	0.876	0.530	0.722	5.475
FOR	14978	1749	3.506	0.506	1.121	0.916	1.440	8.611
PFOR	14839	2396	3.221	0.374	1.153	0.795	1.227	7.924
Rice	15571	3281	2.721	0.314	0.958	0.714	0.941	6.605
S-64	14107	2163	2.581	0.370	0.917	0.621	0.908	6.313
VByte	13223	3018	3.287	2.106	2.411	2.430	2.488	15.132

(c) Sindice

Table 7.1.: Total indexing time, optimise time and index size.

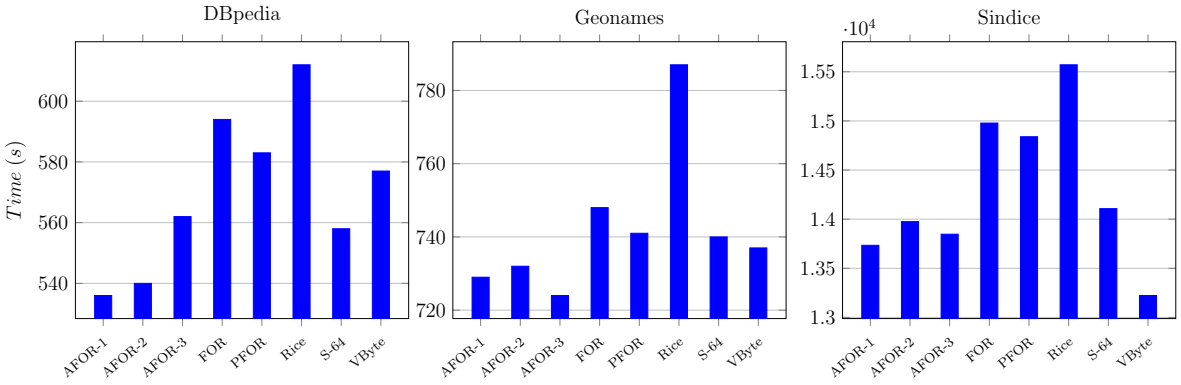


Figure 7.2.: The total time spent to commit batches of 10000 document.

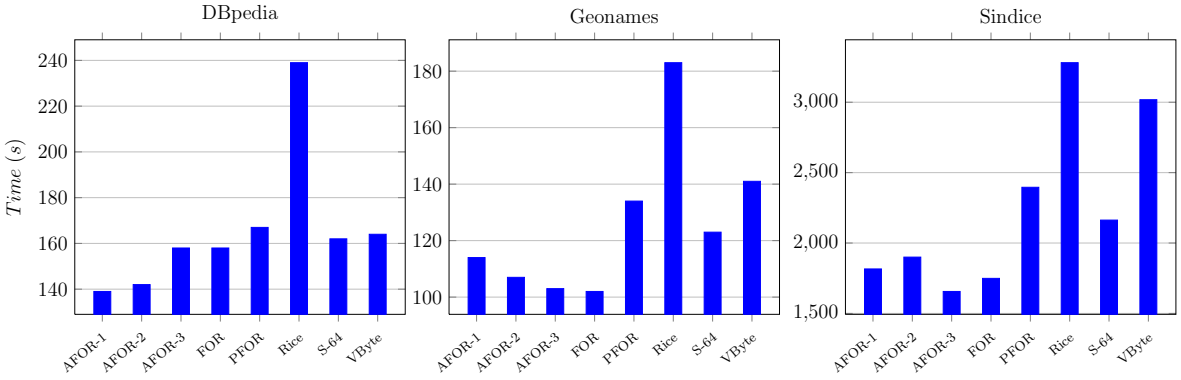


Figure 7.3.: The total time spent to optimise the complete index.

single segment. Therefore, decompression performance is also an important factor, and algorithms having good decompression speed becomes more competitive. For example, while FOR and PFOR was performing similarly than Rice in term of indexing time on the Sindice dataset, it is ahead of Rice in term of optimisation time. Rice is penalised by its low decompression speed. Similarly, S-64 provides close or even better optimisation performance than VByte due to its faster decompression. On smaller datasets (DBpedia and Geonames), VByte is performing well due to its good compression and decompression speed. However, we can notice on a large dataset (Sindice) the compression ratio and the decompression speed of VByte incur a large overhead. The best-performing methods are AFOR-1, AFOR-2 and AFOR-3, with AFOR-3 performing better on large datasets. The AFOR techniques take the advantage due their optimised compression and decompression routines and their good compression rate. AFOR-3 is even twice as fast as Rice on the Sindice dataset.

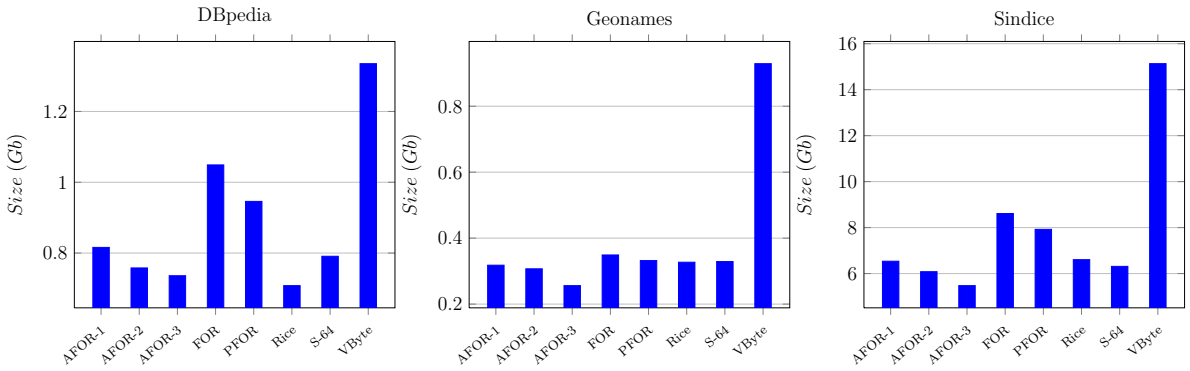


Figure 7.4.: The index size achieved by each compression technique.

Compression Ratio

Figure 7.4 shows the total index size achieved by each method. We can clearly see the inefficiency of the VByte approach. While VByte performs generally better than FOR on traditional document-centric inverted indexes, this is not true for inverted indexes based on a node indexing scheme. VByte is not adapted to such an index due to the properties of the delta-encoded lists of values. Apart from the entity file, the values are generally very small and the outliers are rare. In that case, VByte is penalized by its inability of encoding a small integer in less than a byte. On the contrary, FOR is able to encode many small integers in one byte. Also, while PFOR is less sensitive to outliers than FOR, the gain of compression rate provided by PFOR is minimal since outliers are more rare than in traditional inverted indexes. In contrast, AFOR and S-64 are able to better adapt the encoding to the value distribution and therefore provide a better compression rate. AFOR is even able to provide better compression ratio than Rice on the Geonames and Sindice dataset. Compared to AFOR-2, we can observe in Table 7.1 that AFOR-3 provides better compression rate on the frequency, value and position files, and slightly better on the entity file. This result corroborates the existence of long runs of 1 in these files, as explained in Section 7.2.2.

We will now discuss in more details the compression rate achievement of each technique per dataset.

DBPedia: Rice method provides the best compression on the DBpedia dataset, with the exception of the entity inverted file where AFOR-2 and AFOR-3 provide better compression. The second and third best compression is achieved by AFOR-2 and AFOR-3

respectively. We can see the benefits provided by the variable frame length and frame stripping techniques. Each one provides a substantial decrease of index size. S-64 provides the fourth best compression. It is interesting to notice that on all the files, except the entity inverted file, FOR provides better compression than VByte. This is explained by the fact that these files contain fairly small values which can be encoded in less than one byte. Also, we can see that PFOR is less sensitive to outliers than FOR and provides better compression with a gain of 100MB on the total index size. However, the exception management of PFOR shows some limitations: there is some size overhead on the entity file, and PFOR is not able to achieve the compression ratio of techniques such as AFOR. VByte provides the worst compression. VByte is inadequate against sequences of small values since it has to use one byte even for very small values.

Geonames: While the size of the Geonames dataset is slightly bigger than for the DBpedia dataset, the index size is smaller. This is explained by the fact that the data structure of Geonames is fairly repetitive across the collection of entities, and consists mainly of a set of relations and a few short literals. Therefore, a higher compression ratio is achieved by every methods.

Rice does not provide anymore the best compression ratio. The best compression ratio is achieved by any AFOR techniques. Again, we can observe that both the variable frame length and frame stripping techniques provide a substantial decrease of index size. Compared to Rice, AFOR-3 provides an index size reduction of 22%. S-64 provides similar compression than Rice. PFOR achieves a slightly better compression rate than FOR. VByte has again the worst compression ratio. With VByte, the total index size increases by four orders of magnitude compared to AFOR-3 and by three orders of magnitude compared to the other methods.

Sindice: The Sindice dataset is a more representative subset of the entity descriptions that can be found on the web. The descriptions generally contain a larger number of attributes than in the two previous datasets. It also contains more multi-valued attributes and textual properties.

Again, Rice does not provide the best compression rate. The best compression ratio is achieved by AFOR-2 and AFOR-3. Compared to AFOR, the index size decreases by 7% with the variable frame length technique, and by an additional 10% with the frame stripping technique. Even S-64 and AFOR-1 provides slightly better compression

rates than Rice, and achieves the third and fourth best compression respectively. In fact, compared to these methods, Rice shows some overhead on the entity, attribute and value inverted files. The exception management of PFOR shows only limited benefits compared to FOR, and even some overhead on the attribute inverted file. VByte delivers again the worst compression ratio with a two-fold or three-fold increase compared to other methods.

Conclusion

This experiment shows that the compression speed is also an important factor to take into consideration when designing a compression algorithm for an inverted index. Without a good compression speed, the update throughput of the index is limited. Also the experiment shows that the optimisation operation is dependent of the decompression performance, and its execution time can double without a good compression and decompression speed. With respect to index optimisation, the compression ratio must also be taken into consideration. While VByte provides in general correct commit times, we can observe on a large dataset (Sindice) that its performance during optimisation is limited by its poor compression ratio. Overall, the method providing the best balance between indexing time, optimise time and compression ratio is AFOR. AFOR-1 provides fast compression speed and better compression ratio than FOR and PFOR. AFOR-2 provides a notable additional gain in compression ratio and optimise time but undergoes a slight increase of indexing time. AFOR-3 provides another additional gain in compression ratio while providing better compression speed than AFOR-2.

7.3.5. Query Processing Performance

We now compare the decompression performance in real settings, where inverted indexes are answering queries of various complexities. We focus on two main classes of queries, the value and attribute queries, which are the core elements of a star-shaped query. Among these two classes, we identify types of keyword queries which represent the common queries received by a web search engine: conjunction, disjunction and phrase.

Query Generation

The queries are generated based on the selectivity of the words composing them. The word selectivity determines how many entities match a given keyword. The words are grouped into three selectivity ranges: high, medium and low. We differentiate also two groups of words based on their position in the data graph: attribute and value. We follow the technique described in [EDR05] to obtained the ranges of each word group. We first order the words by their descending frequency, and then take the first k words whose cumulative frequency is 90% of all word occurrences as high range. The medium range accounts for the next 5%, and the low range is composed of all the remaining words. For the phrase queries, we follow a similar technique. We first extract all the 2-gram and 3-gram[4] from the data collection. We then compute their frequency and sort them by descending frequency. We finally create the three ranges as explained above.

Value Queries: Value queries are divided into three types of keyword queries: conjunction, disjunction and phrase queries. These queries are restricted to match within one single value. Therefore, the processing of conjunction and disjunction queries relies on the entity, frequency, attribute and value inverted files. Phrase queries rely on one additional inverted file, the position inverted file.

Conjunction and disjunction queries are generated by taking random keywords from the high range group of words. 2-AND and 2-OR (resp. 4-AND and 4-OR) denotes conjunction and disjunction queries with 2 random keywords (resp. 4 random keywords). Similarly, a phrase query is generated by taking random n-grams from the high range group. 2-Phrase (resp. 3-Phrase) denotes phrase queries with 2-gram (resp. 3-gram). Benchmarks involving queries with words from low and medium ranges are not reported here for questions of space, but the performance results are comparable with the ones presented here.

Attribute Queries: An attribute query is generated by associating one attribute keyword with one value query. An attribute keyword is randomly chosen from the high range groups of attribute words. The associated value query is obtained as explained previously. An attribute query is intersecting the result of a value query with an attribute keyword.

[4] A n-gram is n words that appear contiguously

Query Benchmark Design

For each type of query, we generate a set of 200 random queries which is reused for all the compression methods and perform 100 measurements. Each measurement is made by performing n times the query execution of the 200 random queries, with n chosen so that the runtime is long enough to minimise the time precision error of the OS and machine (which can be 1 to 10 milliseconds) to a maximum of 1%. All measurements are made using *warm cache*, i.e., the part of the index read during query processing is fully loaded in memory. The measurement time is the CPU time, i.e., user time and system time, used by the current thread to process the 200 random queries.

Query execution time is sensitive to external events which can affect the final execution time recorded. For instance, background system maintenance or interruptions as well as cache misses or system exceptions can occur and perturb the measurements. All these events are unpredictable and must be treated as noise. Therefore, we need to quantify the accuracy of our measurements. As recommended in [Lil00], we report the arithmetic mean and the standard deviation of the 100 measurements. To assess differences between the algorithms, confidence intervals with 95% degree of confidence have been used. The design of the value and attribute query benchmarks includes three factors:

- *Algorithm* having height levels: AFOR-1, AFOR-2, AFOR-3, FOR, PFOR, Rice, S-64, and VByte;
- *Query* having six levels: 2-AND, 2-OR, 4-AND, 4-OR, 2-Phrase, and 3-Phrase; and
- *Dataset* having three levels: DBpedia, Geonames and Sindice.

Each condition of the design, e.g., AFOR-1 / 2-AND / WIKIPEDIA, contains 100 separate measurements.

Query Benchmark Results

We report the results of the query benchmarks in Table C.1 and Table C.2 for the value and attribute queries respectively. Based on these results, we derive multiple graphical charts to better visualise the differences between each algorithm. These charts are then used to compare and discuss the performances of each algorithm.

Figure 7.5 and Figure 7.6 report the query processing time for the value and attribute queries respectively. Figure 7.5a and Figure 7.6a depict the average processing time on

the boolean queries (2-AND, 2-OR, 4-AND, 4-OR) for the value and attribute queries respectively. Figure 7.5b and Figure 7.6b depict the average processing time on the phrase queries (2-Phrase, 3-Phrase) for the value and attribute queries respectively. Figure 7.5c and Figure 7.6c depict the average processing time on the boolean and phrase queries for the value and attribute queries respectively. The query processing time are obtained by summing up the average processing time of each query from Table C.1 and Table C.1. For example, the processing time of AFOR-1 on the DBpedia dataset in Figure 7.5b is obtained by summing up the processing times of the queries 2-Phrase (43.2 ms) and 3-Phrase (32.6 ms) reported in Table C.1.

Value Query:

In Figure 7.5c, and in particular on the Sindice dataset (large dataset), we can distinguish three classes of algorithms: the techniques based on FOR, a group composed of S-64 and VByte, and finally Rice. The FOR group achieves relatively similar results, with AFOR-2 slightly behind the others.

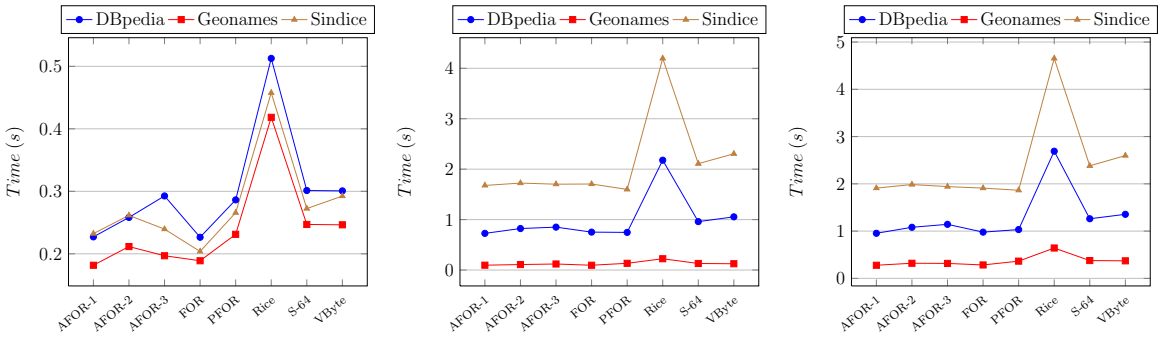
Rice has the worst performance for every query and dataset, followed by VByte. However, Rice performs in many cases twice as slow as VByte. In Figure 7.5a, S-64 provides similar performance to VByte on boolean queries but we can see in Figure 7.5b that it is faster than VByte on phrase queries. However, S-64 stays behind FOR, PFOR and AFOR in all the cases.

FOR, PFOR and AFOR have relatively similar performances on all the boolean queries and all the datasets. PFOR seems to provide generally slightly better performance on the phrase queries but seems to be slower on boolean queries.

Attribute Query:

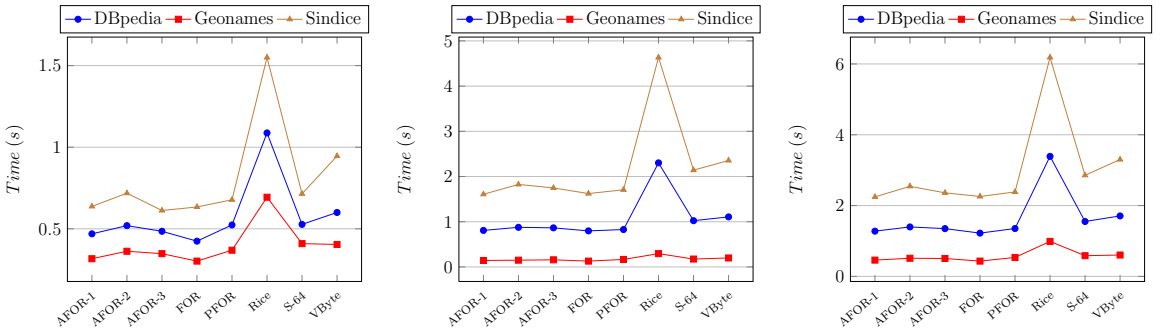
In Figure 7.5c, and in particular on the Sindice dataset (large dataset), we can again distinguish the same three classes of algorithms. However, the performance gap between S-64 and VByte becomes larger.

Rice has again the worst performance for every query and dataset. Compared to the performance on value queries, we can see in Figure 7.6a that S-64 provides similar performance to PFOR and AFOR-2 on boolean queries. FOR and AFOR-3 seem to be the best performing methods on boolean queries. With respect to the phrase queries in Figure 7.6b, S-64 has better performance than VByte. However, PFOR does not achieve any more the best performance on phrase queries. Instead, it seems that AFOR-2 and FOR achieve a slightly better processing time.



(a) Boolean Query Time **(b)** Phrase Query Time **(c)** Overall Query Time

Figure 7.5.: The average processing time for the value queries that is achieved by each compression technique. The time has been obtained by summing up the average query processing times of each type of query. The overall query processing time has been obtained by summing up the boolean and phrase query processing times.



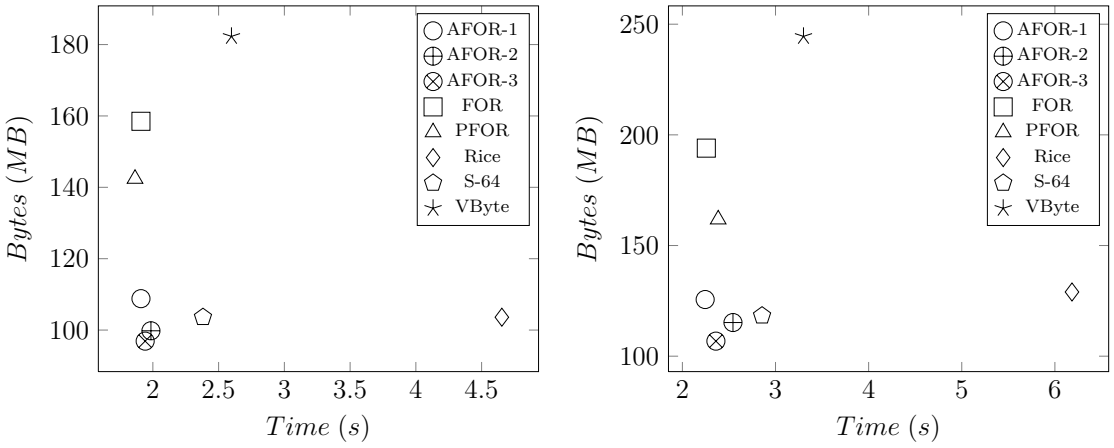
(a) Boolean Query Time **(b)** Phrase Query Time **(c)** Overall Query Time

Figure 7.6.: The average processing time for the attribute queries that is achieved by each compression technique. The time has been obtained by summing up the average query times of each type of query. The overall query time has been obtained by summing up the boolean and phrase query times.

FOR, PFOR and AFOR have again relatively similar performances on all the queries and all the datasets. AFOR-2 appears to be slower to some degree, while the gap between AFOR-3 and PFOR becomes less perceptible.

7.3.6. Performance Trade-Off

We report in Figure 7.7 the trade-off between query processing time and compression ratio among all the techniques on the Sindice dataset, the larger and more representative



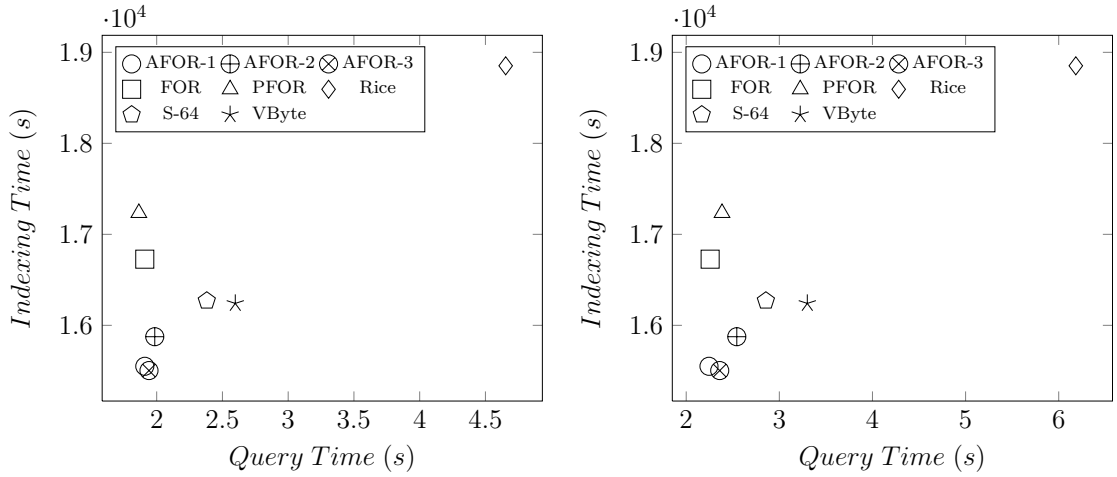
(a) Value Query **(b)** Attribute Query

Figure 7.7.: A graphical comparison showing the trade-off between querying time and compression ratio of the compression techniques on the Sindice dataset. The compression ratio is represented by the number of bytes read during the query processing.

dataset. The query time has been obtained by summing up the average query times of each query as in Figure 7.5 and Figure 7.6. The compression ratio is based on the number of bytes read during query processing which are reported in Table C.1 and Table C.2.

We can distinctively see that the AFOR techniques are close to Rice in term of compression ratio, while being relatively close to FOR and PFOR in term of query processing time. Compared to AFOR-1, AFOR-2 achieves a better compression rate in exchange of a slightly slower processing time. However, AFOR-3 accomplishes a better compression rate with a very close processing time to AFOR-1.

We report in Figure 7.8 the trade-off between query processing time and indexing time among all the techniques on the Sindice dataset. The indexing time has been obtained by summing up the commit and optimise time from Table 7.1. We can distinctively see that the AFOR techniques achieve the best trade-off between indexing and querying time. AFOR-3 produce very similar indexing and querying times to AFOR-1, while providing a much better compression rate. It is interesting to notice that PFOR provides a slightly better querying time than FOR but at the price of a much slower compression. Also, S-64 and VByte provide a relatively close performance trade-off. To conclude, AFOR-3 seems to offer the best compromise between querying time, indexing time, and compression rate.



(a) Value Query **(b)** Attribute Query

Figure 7.8.: A graphical comparison of the compression techniques showing the trade-off between querying time and indexing time on the Sindice dataset.

7.4. Discussion and Future Work

In general, even if FOR has more data to read and decompress, it still provides one of the best query execution time. The reason is that our experiments are performed using warm cache. We therefore ignore the cost of disk IO accesses and measure exclusively the decompression efficiency of the methods. With a cold cache, i.e., when IO disk accesses have to be performed, we expect a drop of performance for algorithms with a low compression ratio such as FOR and PFOR compared to AFOR-2 and AFOR-3. Future work will investigate this aspect.

Compression and decompression performance do not only depend on the compression ratio, but also on the execution flow of the algorithm and on the number of cycles needed to compress or decompress an integer. Therefore, CPU-optimised algorithms which provides at the same time a good compression ratio are more likely to increase the update and query throughputs of web search engines. In that context, AFOR seems to be a good candidate since it is well balanced in all aspects: it provides very good indexing and querying performance and one of the best compression ratio.

The Simple encoding family is somehow similar to AFOR. At each iteration, S-64 encodes or decodes a variable number of integers using CPU optimised routines. AFOR is however not tied to the size of a machine word, is simpler to implement and provides better compression ratio, compression speed and decompression speed.

Another interesting property of AFOR which is not discussed in this paper is its ability to skip quickly over chunks of data without having to decode them. This is not possible with techniques such as Rice, VByte or PFOR. AFOR has to decode the bit frame to know the length of the following frame, and is therefore able to deduce the position of the next bit frame. Such a characteristic could be leveraged to simplify the self-indexing of inverted files [MZ96]. In addition, one of the challenge with the self-indexing technique is how to place synchronisation points, called skips, over the list of integers. We would have to study the impact of aligning the synchronisation points with the partitioning produced by AFOR-2 and AFOR-3.

7.5. Conclusion

We presented AFOR, a novel class of compression techniques for inverted lists. AFOR is specifically designed to increase update and query throughput of web search engines. We have described three different implementations of the AFOR encoder class. We have compared AFOR to alternative approaches, and have shown experimental evidences that AFOR provides a well balanced trade-off between three factors: indexing time, querying time and compression ratio. In particular, AFOR-3 achieves similar query processing times than FOR and PFOR, a better compression rate than Rice and the best indexing times. In addition, AFOR is simple to implement and could become a new compression method of reference.

The results of the experiment lead to interesting conclusions. In particular and with respect to the node-based indexing scheme, we have shown (1) that VByte is inadequate due to its incapacity of efficiently encoding small values; and (2) that PFOR provides only limited benefits compared to FOR. On the contrary, techniques such as S-64 and AFOR which are able to adapt their encoding based on the value distribution yield better results. With respect to the earlier experiment performed in Section 6.6, the implementation of the node-based inverted index we were using was based on the VByte encoding technique. Therefore, the performance reported is not representative of what it could have been with an appropriate compression algorithm such as AFOR. In particular, we have seen that AFOR-3, compared to VByte, can provide a three-fold decrease of index size, divide by two the optimise time, and reduce by 30% the query processing time.

Part IV.

System

Chapter 8.

The Sindice Search Engine: Putting the Pieces Together*

This chapter describes how the different techniques presented in this thesis are integrated in the Sindice search engine infrastructure. We start by giving an overview of the Sindice infrastructure before describing in more details the main components of the architecture and how they are interlinked.

8.1. Sindice Overview

Developed by DERI's Data Intensive Infrastructures group[1], the Sindice project aims at building a scalable infrastructure to locate and integrate data from a multitude of web data sources. By using the Sindice API, for example, it is possible to use keywords or more complex queries to search for people, places, events and connections among them based on the semi-structured content of the documents found on the web.

Technically, Sindice is designed to provide these services fulfilling three main non functional requirements: scalability, run time performance and ability to cope with the many changes in standards and usage practices on the Web of Data.

The data collected using methods similar to those of conventional web search engines is then preprocessed. In this phase, the raw content harvested, e.g., the semantically annotated HTML, is analysed by a set of parsers which we are currently publishing

*This chapter is partially based on [CDTT10, CSD+08]
[1]Data Intensive Infrastructures Unit: `http://di2.deri.ie`

as open source under the name *any23*[2]. These parsers are used to extract RDF data from various formats. Different RDF serialization formats (RDF/XML, Notation 3, Turtle, N-Triples, RDFa) are supported, as well as several microformats (hCard, hEvent, hListing, hResume and others).

At this point, the content and semantics of the document, i.e., the complete or partial view over an entity, regardless of the original format, is represented in RDF. Information then reaches the context-dependent reasoning engine, discussed in Section 4, which is a part of the pre-processing step where inference is performed for each single entity to be indexed.

Indexing and searching is performed by SIREn, the Information Retrieval system presented in Chapter 6. SIREn enables the answering of both textual and semi-structural queries and forms the core of the Sindice public APIs. To be able to provide high quality top-k results, DING, the dataset-centric ranking strategy presented in Section 5, is employed. The static entity rank is combined with the query-dependent entity score as explained in Section 5.5.3. These elements and their interrelations are depicted in a high level view in Figure 8.1.

Finally, these services are leveraged as an API to create advanced applications which make use of Web of Data. This is the case of Sig.ma [TCC+10], an end-user targeted application that enables visual "entity-based search and information mashups".

8.2. Data Acquisition

There are many ways to make information available on the Web of Data. Even restricting to the RDF world, for example, an online database might be published as one single RDF dump. Alternatively, the LOD paradigm is based on using resolvable URIs to offer access to individual descriptions of resources. Other datasets might offer access to its data via a SPARQL endpoint or might expose web documents that embed RDFa.

If several of these options are offered simultaneously for the same database, the choice of access method can have significant effects on the amount of networking and computing resources consumed on the client and the server side. Such choices can have serious implications. Considering that results are generated from semantic queries, which tend to be resource intensive, it might be sensible to limit the query rate to one document

[2]any23: `http://code.google.com/p/any23/`

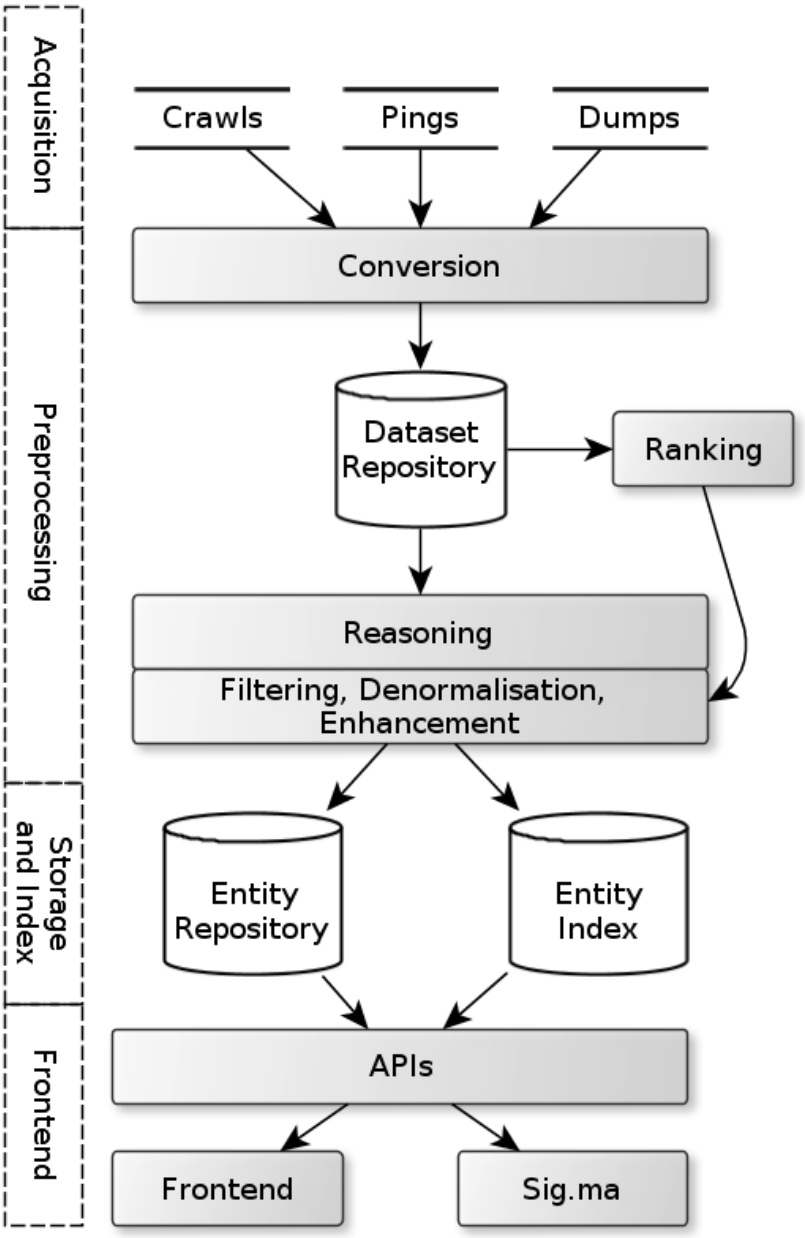


Figure 8.1.: Overview of the Sindice infrastructure stack.

per second. With this limit, however, crawling the 13.8 million entities of the Geonames dataset would take more than five months.

In this section we describe a methodology which Sindice supports fully that allows publishers to provide enough information about their publishing model and thus enables a smart selection of data access methods by clients and crawlers alike. The methodology, called Semantic Sitemaps [CSD+08], is based on extending the existing Sitemap Protocol by introducing several new XML tags for announcing the presence of RDF data and to deal with specific RDF publishing needs. Semantic Sitemaps are used extensively to index most of the LOD datasets. They represent probably the most peculiar method of data acquisition targeted at Semantic Web data, while other methods such as direct notifications of URLs to index, which we call Pings, and general crawling are more comparable to those used in conventional web search engine and will not be discussed.

8.2.1. Semantic Sitemap

The Semantic Sitemap extension [CDT07] has the concept of dataset at its core: datasets are well defined resources which can have one or more access methods. It is well defined what properties apply to a certain access method and what properties apply to a given dataset. Therefore properties that apply to that dataset will be directly related to all the data that can be obtained, independently from the access method. A publisher can host multiple datasets on the same site and can describe them independently using different sections of the Semantic Sitemap. While there is nothing that prevents information overlap or contradictions between different datasets, it is expected that this is not the case.

The Semantic Sitemap extension enables the description of a dataset via the tag `<sc:dataset>`, to be used at the same level as `<url>` tags in a regular sitemap. Access options for the datasets are given by additional tags such as `<sc:dataDump>`, `<sc:sparqlEndpoint>` and `<sc:linkedDataPrefix>`. RDF dataset dumps can be provided in formats such as RDF/XML, N-Triples and N-Quads. Optionally, dump files may be compressed in GZIP, ZIP, or BZIP2 format. If a sitemap contains several dataset definitions, they are treated independently. Listing 8.1 shows a sitemap file applying the extension.

The dataset is labelled as the *Example Corp. Product Catalog* and identified by `http://example.com/catalog.rdf#catalog`. Hence it is reasonable to expect further

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9"
        xmlns:sc="http://sw.deri.org/2007/07/sitemapextension">
  <sc:dataset>
    <sc:datasetLabel>
      Example Corp. Product Catalog
    </sc:datasetLabel>
    <sc:datasetURI>
       http://example.com/catalog.rdf#catalog
    </sc:datasetURI>
    <sc:linkedDataPrefix sc:slicing="subject-object">
       http://example.com/products/
    </sc:linkedDataPrefix>
    <sc:sparqlEndpoint sc:slicing="subject-object">
       http://example.com/sparql
    </sc:sparqlEndpoint>
    <sc:dataDump>
       http://example.com/data/catalogdump.rdf.gz
    </sc:dataDump>
    <sc:dataDump>
       http://example.org/data/catalog_archive.rdf.gz
    </sc:dataDump>
    <changefreq>weekly</changefreq>
  </sc:dataset>
</urlset>
```

Listing 8.1: Example of a Semantic Sitemap file

RDF annotations about the dataset `http://example.com/catalog.rdf`. The resources described in the dataset all have identifiers starting with `http://example.com/products/`, and their descriptions are served as Linked Data. A dump of the entire dataset is available, split into two files and the publisher states that dataset updates can be expected weekly.

8.3. Data Preprocessing and Denormalisation

8.3.1. Data Conversion

In a first step, the newly acquired data is converted into the N-Triples syntax. Dumps coming from semantic sitemaps are fetched, parsed and converted into N-Triples. Similarly, web documents coming from pings or crawls are parsed, embedded metadata are extracted and transformed into N-Triples.

The conversion step is distributed over a cluster of machines using Hadoop[3], an open-source implementation of the MapReduce programming model [DG08]. The Map task consists of converting the data into N-Triples, while the Reduce task consists of sorting and grouping the N-Triples statements by context, subject and view, the context

[3]Apache Hadoop: `http://hadoop.apache.org/`

being the dataset identifier, the subject being the entity identifier, and the view being the original provenance, i.e., URL of the document, of the statement. The view is necessary for keeping the dataset repository up to date, as explained in the next section.

We found the N-Triples syntax more adapted than RDF/XML or N3 for internal data management. N-Triples is more efficient to parse than any other syntaxes. In addition, since one statement is serialised by line, it enables stream-based processing, i.e., one line at a time, and more easy manipulation of the data. While the N-Triples syntax is more verbose and therefore more demanding in term of space, this is limited by using compressed data streams.

8.3.2. Dataset Repository

After being converted into N-Triples, the data is stored in the *dataset repository*. The goal of the dataset repository is to maintain a up to date vision of all the datasets and entities that have been currently acquired. The dataset repository is based on a distributed column-oriented datastores, HBase[4], an open-source implementation of BigTable [CDG+06].

In HBase, a table consists of a large number of sorted rows indexed by a row key and columns indexed by a column key (each row can have multiple different columns). The content is stored in a HBase cell. A cell is defined by a combination of a row and a column key. The primary key of a HBase table is the row key. HBase supports two basic operations on the row key: key lookup and key range scan.

The table is structured as follow. Each row of the table stores the serialised N-Triples statements of a particular view over an entity. A unique row key is generated by concatenating the dataset, entity and view identifiers, the view identifier being the URL of the document. A cell is hosting the statements of a view for a particular entity, and another cell a timestamp associated to the view. This table schema enables to keep unique rows for each dataset, entity and view which simplifies the table maintenance against updates. In addition, since the row keys are ordered first by datasets, then by entities and finally by views, it is possible to efficiently retrieve the description of an entity, by performing a range scan over all the views of a particular entity.

[4]Apache HBase: `http://hbase.apache.org/`

To keep data separated based on the view is necessary for update reasons. If the data acquisition provides a new version of a view, the change needs to be reflected in the dataset repository. A new version of a view can include data about new entities or can remove data about an entity. Therefore, in order to keep the data in a consistent state, all the related rows to the older view must be deleted and then new rows must be inserted. The update mechanism works in two steps. The first step overwrites rows with the same key while the second step is done asynchronously as explained next.

A deletion table is used to keep track of the deleted views on a per dataset basis. Each dataset is associated with a list of deprecated views. During a scan over the rows of an entity, the view identifier and its timestamp is checked against the deletion table. If the view has been deleted, the row is removed during the scan. The deletion table is integrated back to the table only when a certain amount of deletion is sufficient to amortize the cost of the maintenance operation, since it requires a full scan of the dataset in order to remove all deprecated views.

8.3.3. Entity Processing

Each entity description that has been modified in the dataset repository is exported by performing a range scan over the entity. The new entity description is then sent to a second pipeline for more advanced processing.

The first operation that is performed is reasoning. Reasoning is performed on each entity description using the context-dependent reasoning mechanism explained in Chapter 4. As discussed in Section 3.1, reasoning over entity description enables to make explicit what would otherwise be implicit data, and therefore increases the precision and recall of an information retrieval system. After reasoning, various other operations are performed such as data filtering, data denormalisation and data enhancement.

Data filtering consists of removing certain pieces of information that are considered as noise or useless for a search purpose. There are two kinds of filtering, one on a statement level, the second one on entity level. Performing a filtering on a statement level consists of removing certain statement from the entity description. The filtering is based on rules that are stored in a "black-list" of statement patterns. For example, among those rules, we can cite the "reflexive statement" rule that filters out all statements of the kind (x,p,x) or "generic statement" rules that filter out all generic statements such as (x,rdf:type,rdfs:Resource). Filtering on entity level consists of removing entities that do

not meet certain criteria. For example, the entity descriptions with less than 2 triples or coming from a particular domain are filtered out. The filtering of statements and entities increases the overall performance of the system without having an impact on the quality of the search results.

The data denormalisation consists of transforming the entity description, including the inferred data, into the node-labelled tree model presented in Section 6.2. The node-labelled tree is sent to the *entity index* for indexing. Prior to indexing, the data is enhanced with additional information, such as a pre-computed entity rank or the ontology import closure. A copy of the entity description, after reasoning, filtering and enhancement, but prior to denormalisation, is stored in the *entity repository*.

The pipeline is distributed over a cluster of machines using Hadoop with a Map task only. The distribution scheme is to partition on a per entity basis, where each Map task applies the complete pipeline of operations over one entity description.

8.3.4. Ranking

Ranking of datasets and entities are performed offline due to the cost of the operation. An offline job over the dataset repository is executed periodically to extract statistics about datasets, entities and their interconnections. These statistics are stored in a dedicated database, which are then used to compute a rank for each dataset using Equation 5.4 and for each entity using the Weighted LinkCount method from Section 5.3.4. In order to minimise the cost of the operation, the statistics are updated based on the changes in the dataset repository since the last job. The statistic extraction is only performed over the datasets that have been modified.

8.4. Distributed Entity Storage and Indexing

8.4.1. Entity Repository

The *entity repository* is a scalable storage system, based on HBase, for managing large collections of entity descriptions. The repository needs to perform two basic functions. First, it must provide an interface for the pipeline to store the entity descriptions. Second, it must provide an efficient access API to retrieve these entity descriptions.

The entity repository is storing the original entity description, the inferred data and additional metadata extracted during the entity processing, such as the ontology import closure, the entity rank, or the entity label. It is used by higher level applications, such as the search frontend or Sig.ma, to retrieve the content and information of an entity after having located them by using the entity index.

8.4.2. Entity Index

The *entity index*, which is based on the node indexing scheme from Chapter 6, provides an interface to search and locate entities based on semi-structural queries. In addition to index the entity descriptions, it also stores, for each entity, small pieces of information that are used at retrieval times, such as the entity rank or a small textual description of the entity. The entity rank is combined during query processing with the query-dependent score as explained in Section 5.5.3. The small description of an entity is used by the search result interface to generate a search snippet. The entity index also provides additional generic search operations to filter entities using the metadata information. For example, users can restrict search results to a set of entities using a particular ontology.

When large volumes of data are involved or when high query volumes must be supported, one machine may be not enough to support the load. To handle such a load, a combination of data distribution and replication is required. Distribution refers to the method where the entity collection and its index are split across multiple machines and where answers to the query as a whole must be synthesized from the various distributed sub-collection. The distribution scheme is to partition on a per entity basis, similarly to the document-based partitioning found in traditional information retrieval systems. Replication (or mirroring) involves making enough identical copies of the system so that the required query load can be handled.

8.5. Conclusions

Searching web data over distributed information sources is the basis for many applications today. Search engines for web data are increasingly needed to find appropriate information for later integration from a vast number of data sources. However, compared to traditional web search engines, search engines for web data must face new challenges due to the semi-structured nature of the information.

In this chapter, we have described one possible architecture of a search engine for web data. This architecture is built around the model described in Chapter 2. The core mechanisms of the system, i.e., reasoning, ranking and indexing, require a substantial design effort due to the web scale requirement. We have explained how to integrate all of these mechanisms in a distributed architecture that scales well with a large amount of data. We believe that such an architecture has the same class of scalability than traditional web information retrieval systems.

The architecture has been implemented and is currently at the core of the Sindice search engine. The Sindice search engine is continuously acquiring new data since the past three years, and, at the time of the writing, serves more than 110 million of documents. The set of APIs provided by Sindice simplifies the access to web data and makes the development of web data applications easier.

Part V.

Conclusions

Chapter 9.

Conclusions

In the last decade, we have seen the rise of standards for semi-structured machine processable information. But, it is only in the recent years that we have observed an increase of adoption of these standards and in parallel an exponential increase of data published on the Web. However, while machine-processable semi-structured data is becoming widespread, a problem remains and starts to be more and more critical: how can an application find data that is relevant for a particular task ? We believe that the answer is a framework that supports users and applications to locate and interrelate relevant pieces of information among distributed heterogeneous data sources. We consider the basis of such a framework as an information retrieval system that can cope with any kind of semi-structured data sources and that provides functionalities such as semi-structural search over all the data sources.

In this thesis, we have proposed a set of methodologies for designing such an information retrieval system. By using the Web Data scenario as an example, we have shown that a combination of careful algorithmic design enables such a system to cope with many distributed and heterogeneous information sources.

9.1. Summary of the Thesis

Throughout this thesis, we advocate the use of a data model designed around the concepts of datasets and entities. We have introduced a formal data model and a boolean query model suitable for the entity retrieval task. The formal model covers not only the Web Data scenario, but in fact all scenarios dealing with heterogeneous and distributed semi-structured information sources. We have presented multiple novel algorithms that forms

the core of the retrieval system. These algorithms have been designed by using the data and query model at a common basis and with an important focus on high-performance and scalability. More precisely, we have tackled three specific issues related to the design of an Information Retrieval system for heterogeneous and distributed semi-structured information sources:

Reasoning through contextualisation of the information. Reasoning on Web Data enables a search engine to be more effective in term of precision and recall for the entity retrieval task. The context mechanism allows the system to avoid the deduction of undesirable assertions, a common risk when working with Web Data. In addition, the contextualisation of the information provides a natural framework for efficient distributed computing.

Ranking by analysing the inter-connection of datasets and entities. A new approach, called DING, is specifically designed to address the Web Data scenario, i.e, computing the popularity score of entities on a web-scale graph. We explained its desirable computational properties compared to alternative approaches and displayed experimental evidence of improved ranking quality. Furthermore, DING introduces the idea of using dataset-specific ranking algorithms for improving entity ranking.

Indexing and Querying by developing an appropriate data structure for inverted indexes as well as a novel compression mechanism. We have investigated various indexing schemes for indexing and querying semi-structured data and shown that a node-based indexing scheme provides the best compromise between query expressiveness, query processing and index maintenance. The new approach, called SIREn, scales well with a very large amount of data while providing efficient entity lookup using semi-structural queries with full text search capabilities. Also, a new class of compression algorithm, called AFOR, for compressing node-based inverted indexes has been developed to increase both the update and query throughput of the system.

Finally, we have presented a system architecture that integrates the presented techniques. This architecture is currently at the core of the Sindice search engine, a system providing access to hundreds of millions of pieces of semi-structured information currently available on the Web. Together, these contributions demonstrate the feasibility of a novel kind of Information Retrieval system that exploits the semi-structured nature of the information while retaining the scalability of traditional web search engines.

9.2. Directions for Future Research

The Information Retrieval domain for Web Data is covering a wider number of problems than the ones addressed in this thesis. In addition, the results of our research leaves place for further developments. In the following, we provide a list of possible future directions of research related to the problem of semi-structured information retrieval:

Reasoning While import relations between ontologies provide a good support for lifting rules on a T-Box level, it is not clear which relations should be consider as lifting rules on the A-Box level. Lifting rules on A-Box will enable reasoning across entity descriptions, which could be beneficial for entity search in case of equality reasoning. Also, it is necessary to study deeper the impact of the logic complexity on the search quality.

Ranking We believe that it would be beneficial to automate graph structure recognition in order to select appropriate ranking algorithms for certain type of datasets. This will improve the effectiveness of link analysis on a heterogeneous web. Also, we believe that it is important to tackle the problem of query-dependent ranking for semi-structured data. XML query ranking methodologies are a good starting point, but need to be extended in order to take into account the specificity and cardinality of the attributes as well as other data structure specificities.

Semi-Structured Data Indexing The indexing model currently focus on the efficient processing of star-shaped queries. However, search queries can span over multiple entity descriptions that are somehow related. Therefore, the relational aspect of the entities is essential during query processing for improving search results. It is necessary to investigate indexing techniques that support the querying of relations between entities without sacrificing the scalability of the system. Towards this direction, we are currently developing an extension of the current indexing model that enables to index and query association between entities.

Inverted Index Optimisation The novel class of compression techniques introduced in our research open the door for a multitude of improvements. It will be beneficial to study more how to approximate optimal frame configurations. Also, the Adaptive Frame Of Reference enables better random accesses in a compressed list. We are currently extending self-indexing techniques of inverted lists in order to take advantages of such a property.

Result Consolidation Entity consolidation at query-time could be achieved in order to improve the usability of the system either by humans or machines. The search results is currently composed of disparate entities from various data sources. However, it often happens that multiple entity descriptions returned for a query are in fact describing the same entities. The aggregation of multiple descriptions into a single one might improve the comprehension of the search results by the user or might simplify the processing of data by a machine.

Machine-Machine Interaction The study of the interaction between a human and an Information Retrieval system has helped to understand how users search for information and has lead to better retrieval models and to innovative search techniques such as automatic query reformulation, result summarisation or faceted search. However, we believe that the main users of a retrieval system for Web Data will be machines or applications and not humans. Therefore, techniques such as result summarisation or faceted search which are designed for human becomes of limited use for a machine. The interaction between a machine and the Information Retrieval system might require a more in-depth study in order to understand how machines search for information. This could lead to more appropriate retrieval models and techniques that might help a machine to automate certain tasks, e.g., refining its search query, and as a result that might improve the quality of the retrieval task result.

Part VI.

Appendices

Appendix A.

Questionnaires of the User Study

This appendix provides a sample of the 20 questionnaires given to the participants of the user study for evaluating link analysis algorithms in Section 5.5.2.

Participant Questionnaire on DING (Dataset Ranking)

Data Intensive Infrastructure Unit (DI2)

June 18, 2009

The aim of this experiment is to rate various link analysis algorithms for ranking resources on the Web of Data.

1. Tell us how you rate *Ranking A* in relation to the standard one (leftmost list) by ticking just one value per scale in the table below:
   - ☐ Better
   - ☐ Slightly Better
   - ☐ Neutral
   - ☐ Slightly Worse
   - ☐ Worse
2. Comments about your choices (optional)

3. Tell us how you rate *Ranking B* in relation to the standard one (leftmost list) by ticking just one value per scale in the table below:
   - ☐ Better
   - ☐ Slightly Better
   - ☐ Neutral
   - ☐ Slightly Worse
   - ☐ Worse
4. Comments about your choices (optional)

Best american science fiction movies

Standard ranking	**Ranking A**	**Ranking B**
Wall-E http://dbpedia.org/resource/WALL-E	*Wall-E* http://dbpedia.org/resource/WALL-E	*Wall-E* http://dbpedia.org/resource/WALL-E
Species: The Awakening http://dbpedia.org/resource/Species_-_...	*Species: The Awakening* http://dbpedia.org/resource/Species_-_...	*The Fly* http://dbpedia.org/resource/The_Fly_(1...
Attack of the 50 Foot Woman http://dbpedia.org/resource/Attack_of_...	*Attack of the 50 Foot Woman* http://dbpedia.org/resource/Attack_of_...	*Stargate* http://dbpedia.org/resource/Stargate_(f...
Critters 4 http://dbpedia.org/resource/Critters_4	*The Thing* http://dbpedia.org/resource/The_Thing_...	*Spy Kids* http://dbpedia.org/resource/Spy_Kids
Real Genius http://dbpedia.org/resource/Real_Genius	*Critters 4* http://dbpedia.org/resource/Critters_4	*Robocop 2* http://dbpedia.org/resource/RoboCop_2
Critters 3 http://dbpedia.org/resource/Critters_3	*Real Genius* http://dbpedia.org/resource/Real_Genius	*Deep Impact* http://dbpedia.org/resource/Deep_Impact...
The Fly http://dbpedia.org/resource/The_Fly_(1...	*Outbreak* http://dbpedia.org/resource/Outbreak_(f...	*2001: A Space Odyssey* http://dbpedia.org/resource/2001:_A_Sp...
The Thing http://dbpedia.org/resource/The_Thing_...	*Critters 3* http://dbpedia.org/resource/Critters_3	*The Thing* http://dbpedia.org/resource/The_Thing_...
The Last Mimzy http://dbpedia.org/resource/The_Last_M...	*The Fly* http://dbpedia.org/resource/The_Fly_(1...	*Hollow Man* http://dbpedia.org/resource/Hollow_Man
Robocop 2 http://dbpedia.org/resource/RoboCop_2	*The Last Mimzy* http://dbpedia.org/resource/The_Last_M...	*Jumper* http://dbpedia.org/resource/Jumper_(film)

Figure A.1.: Questionnaire about the "Best american science fiction movies".

Participant Questionnaire on DING (Dataset Ranking)

Data Intensive Infrastructure Unit (DI2)

September 8, 2010

The aim of this experiment is to rate various link analysis algorithms for ranking resources on the Web of Data.

1. Tell us how you rate *Ranking A* in relation to the standard one (leftmost list) by ticking just one value per scale in the table below:
   - ☐ Better
   - ☐ Slightly Better
   - ☐ Neutral
   - ☐ Slightly Worse
   - ☐ Worse
2. Comments about your choices (optional)

3. Tell us how you rate *Ranking B* in relation to the standard one (leftmost list) by ticking just one value per scale in the table below:
   - ☐ Better
   - ☐ Slightly Better
   - ☐ Neutral
   - ☐ Slightly Worse
   - ☐ Worse
4. Comments about your choices (optional)

List of projects dealing with the Semantic Web

Standard ranking	**Ranking A**	**Ranking B**
RDF Book Mashup http://www4.wiwiss.fu-berlin.de/is-group...	*RDF Book Mashup* http://www4.wiwiss.fu-berlin.de/is-group...	*RDF Book Mashup* http://www4.wiwiss.fu-berlin.de/is-group...
SWSE http://sw.deri.org/2007/02/swsepaper/doa...	*The Cwm Project* http://www.w3.org/2000/10/swap/data#Cwm	*The Cwm Project* http://www.w3.org/2000/10/swap/data#Cwm
D2R Server - Publishing Relational Datab... http://www4.wiwiss.fu-berlin.de/is-group...	*D2RQ - Treating Non-RDF Databases as Vir...* http://www4.wiwiss.fu-berlin.de/is-group...	*Semantic Web Client Library* http://www4.wiwiss.fu-berlin.de/is-group...
Semantic Web Radar's DOAP document seria... http://rdfohloh.wikier.org/project/semra...	*RDF Description of D2RQ - Treating Non-R...* http://www4.wiwiss.fu-berlin.de/is-group...	*Disco - Hyperdata Browser* http://www4.wiwiss.fu-berlin.de/is-group...
D2RQ - Treating Non-RDF Databases as Vir... http://www4.wiwiss.fu-berlin.de/is-group...	*Semantic Web Radar's DOAP document seria...* http://rdfohloh.wikier.org/project/semra...	*D2RQ - Treating Non-RDF Databases as Vir...* http://www4.wiwiss.fu-berlin.de/is-group...
RDF Description of D2RQ - Treating Non-R... http://www4.wiwiss.fu-berlin.de/is-group...	*SWSE* http://sw.deri.org/2007/02/swsepaper/doa...	*RDF Description of D2RQ - Treating Non-R...* http://www4.wiwiss.fu-berlin.de/is-group...
SWAML http://swaml.berlios.de/doap.rdf	*SWAML* http://swaml.berlios.de/doap.rdf	*SWSE* http://sw.deri.org/2007/02/swsepaper/doa...

Figure A.2.: Questionnaire about a "List of projects dealing with the Semantic Web'.

Participant Questionnaire on DING (Dataset Ranking)

Data Intensive Infrastructure Unit (DI2)

September 8, 2010

The aim of this experiment is to rate various link analysis algorithms for ranking resources on the Web of Data.

1. Tell us how you rate *Ranking A* in relation to the standard one (leftmost list) by ticking just one value per scale in the table below:
   - ☐ Better
   - ☐ Slightly Better
   - ☐ Neutral
   - ☐ Slightly Worse
   - ☐ Worse
2. Comments about your choices (optional)

3. Tell us how you rate *Ranking B* in relation to the standard one (leftmost list) by ticking just one value per scale in the table below:
   - ☐ Better
   - ☐ Slightly Better
   - ☐ Neutral
   - ☐ Slightly Worse
   - ☐ Worse
4. Comments about your choices (optional)

List of resources matching keywords "Galway", "Ireland"

Standard ranking	**Ranking A**	**Ranking B**
Galway http://dbpedia.org/resource/Galway	*County Galway* http://dbpedia.org/resource/County_Galway	*County Galway* http://dbpedia.org/resource/County_Galway
County Galway http://dbpedia.org/resource/County_Galway	*Galway* http://dbpedia.org/resource/Galway	*Galway* http://dbpedia.org/resource/Galway
Galway GAA http://dbpedia.org/resource/Galway_GAA	*Galway GAA* http://dbpedia.org/resource/Galway_GAA	*Galway GAA* http://dbpedia.org/resource/Galway_GAA
7th century in Ireland http://dbpedia.org/resource/7th_century...	*UPC Ireland, UPC Communications Ireland ...* http://dbpedia.org/resource/UPC_Ireland	*UPC Ireland, UPC Communications Ireland ...* http://dbpedia.org/resource/UPC_Ireland
UPC Ireland, UPC Communications Ireland ... http://dbpedia.org/resource/UPC_Ireland	*Galway United, Galway United F.C.* http://dbpedia.org/resource/Galway_Unit...	*7th century in Ireland* http://dbpedia.org/resource/7th_century...
National University of Ireland, Galway, ... http://dbpedia.org/resource/National_Un...	*National University of Ireland, Galway* http://dbpedia.org/resource/National_Un...	*National University of Ireland, Galway* http://dbpedia.org/resource/National_Un...
Galway United, Galway United F.C. http://dbpedia.org/resource/Galway_Unit...	*Connacht* http://dbpedia.org/resource/Connacht	*Galway United, Galway United F.C.* http://dbpedia.org/resource/Galway_Unit...
County Galway, Ireland http://www.geonames.org/2964179/county-g...	*Galway West* http://dbpedia.org/resource/Galway_West...	*Irish Language* http://dbpedia.org/resource/Irish_langu...
Irish Language http://dbpedia.org/resource/Irish_langu...	*Irish Language* http://dbpedia.org/resource/Irish_langu...	*Connacht* http://dbpedia.org/resource/Connacht
Viscount Galway http://dbpedia.org/resource/Viscount_Ga...	*All-Ireland Minor Hurling Championship* http://dbpedia.org/resource/All-Ireland:.	*County Galway, Ireland* http://www.geonames.org/2964179/county-g...

Figure A.3.: Questionnaire about resources matching the keywords "Galway" and "Ireland".

Appendix B.

Query Sets of the Index Evaluation

B.1. Real-World's Query Set

```
# A1:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o .
}}

# A2:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s ?p <http://dbpedia.org/resource/Greenwich_Mean_Time> .
}}

# B1:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://dbpedia.org/property/awards> <http://dbpedia.org/resource/Nobel_Prize_in_physics> .
}}

# B2:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://dbpedia.org/property/settlementType> "Town" .
}}

# C1:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://www.w3.org/2000/01/rdf-schema#isDefinedBy> <http://semanticweb.org/wiki/Special:ExportRDF/
       Property:Name> .
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#DatatypeProperty> .
}}

# C2:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s ?p <http://dbpedia.org/resource/Category:American_documentary_films> .
  ?s ?q "Edith Massey" .
}}

# D1:
```

```
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
  ?s <http://xmlns.com/foaf/0.1/interest> <http://www.livejournal.com/interests.bml?int=music> .
  { ?s <http://xmlns.com/foaf/0.1/interest> <http://www.livejournal.com/interests.bml?int=tango> . }
  UNION
  { ?s <http://xmlns.com/foaf/0.1/interest> <http://www.livejournal.com/interests.bml?int=rock> . }
}}

# E:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
  OPTIONAL { ?s <http://xmlns.com/foaf/0.1/interest> ?x
             FILTER(?x = <http://www.livejournal.com/interests.bml?int=gothic+music>) } .
  FILTER(!bound(?x))
}}
```

Listing B.1: The query set for the Real-World dataset that has been used in the query benchmark of Section 6.6.5

B.2. Barton's Query Set

```
# A1:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o .
}}

# A2:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s ?p "Florence" .
}}

# B1:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://simile.mit.edu/2006/01/ontologies/mods3#
       Text> .
}}

# C1:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://simile.mit.edu/2006/01/ontologies/mods3#
       Text> .
  ?s ?p <http://simile.mit.edu/2006/01/language/iso639-2b/eng> .
}}

# C2:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://simile.mit.edu/2006/01/ontologies/mods3#
       Text> .
  ?s ?p <http://simile.mit.edu/2006/01/language/iso639-2b/eng> .
  ?s <http://simile.mit.edu/2006/01/ontologies/mods3#language> <http://simile.mit.edu/2006/01/language/
       iso639-2b/ger> .
```

```
}}

# D1:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  { ?s <http://simile.mit.edu/2006/01/ontologies/mods3#subject> <http://simile.mit.edu/2006/01/topic/
       Physics> }
  UNION
  { ?s <http://simile.mit.edu/2006/01/ontologies/mods3#subject> <http://simile.mit.edu/2006/01/topic/
       Biology> }
}}

# D2:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  { ?s ?p <http://simile.mit.edu/2006/01/topic/Physics> }
  UNION { ?s ?p <http://simile.mit.edu/2006/01/topic/Biology> }
}}

# E:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://simile.mit.edu/2006/01/ontologies/mods3#subject> <http://simile.mit.edu/2006/01/topic/
       History> .
  OPTIONAL { ?s <http://simile.mit.edu/2006/01/ontologies/mods3#issuance> ?x
             FILTER(?x = "Monographic") } .
  FILTER(!bound(?x))
}}
```

Listing B.2: The query set for the Barton dataset that has been used in the query benchmark of Section 6.6.5

B.3. 10 Billion's Query Set

```
# Q1:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://dbpedia.org/property/awards> <http://dbpedia.org/resource/Nobel_Prize_in_physics> .
}}

# Q2:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://www.w3.org/2004/02/skos/core#subject> <http://dbpedia.org/resource/Category:
       American_documentary_films> .
}}

# Q3:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2002/07/owl#Class> .
}}

# Q4:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://dbpedia.org/property/settlementType> "Town" .
}}
```

```
# Q5:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  { ?s <http://xmlns.com/foaf/0.1/name> "Samuel" . }
  UNION
  { ?s <http://xmlns.com/foaf/0.1/name> "Elisabeth" . }
}}

# Q6:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
}}

# Q7:
SELECT DISTINCT ?s ?g WHERE { GRAPH ?g {
  ?s <http://xmlns.com/foaf/0.1/knows> ?o .
}}
```

Listing B.3: The query set for the 10 Billion dataset that has been used in the query benchmark of Section 6.6.6

Appendix C.

Results of the Compression Benchmark

This appendix provides tables containing the results of the benchmarks that have been performed for comparing the query performance of the node-based index with various compression algorithms. These results have been used for generating the charts from Section 7.3.5.

	2 - AND			2 - OR			4 - AND			4 - OR			2 - Phrase			3 - Phrase		
Method	μ	σ	MB	μ	σ	MB	μ	σ	MB	μ	σ	MB	μ	σ	MB	μ	σ	MB
DBpedia																		
AFOR-1	32.6	1.3	1.8	42.9	1.2	2.0	63.2	2.4	3.6	88.4	2.5	4.0	218.4	13.7	14.2	508.3	4.4	36.6
AFOR-2	37.7	1.2	1.7	47.8	1.7	1.9	74.1	3.0	3.3	98.6	2.5	3.7	253.2	12.9	13.2	569.3	4.6	33.8
AFOR-3	44.2	1.2	1.7	52.3	11.2	1.8	86.0	3.7	3.3	110.1	5.0	3.6	256.7	13.9	13.1	593.9	32.2	33.6
FOR	31.5	1.4	2.3	46.8	10.6	2.5	61.9	2.9	4.5	86.3	2.5	5.1	220.5	13.2	18.9	531.1	35.3	48.5
PFOR	44.2	17.1	2.3	52.2	1.3	2.5	83.1	2.7	4.5	106.8	2.6	5.0	225.1	2.7	16.5	521.1	4.5	41.9
Rice	75.4	1.6	1.7	98.8	8.9	1.8	148.0	3.1	3.3	190.5	3.7	3.7	604.8	4.9	12.1	1573.0	6.4	29.9
S-64	42.4	3.0	1.9	57.8	1.6	2.0	83.3	2.4	3.6	117.8	2.4	4.0	291.0	15.3	13.7	668.8	6.0	35.0
VByte	45.8	17.8	2.7	57.2	1.5	2.9	81.0	2.9	5.2	116.7	2.3	5.9	330.5	13.0	21.9	723.9	5.8	57.8
Geonames																		
AFOR-1	29.3	1.5	1.4	30.7	9.2	1.4	62.7	8.8	2.9	59.0	2.8	2.9	35.3	1.8	1.7	60.6	2.0	3.1
AFOR-2	36.6	4.3	1.4	33.7	1.6	1.4	69.3	8.5	2.9	72.0	8.6	2.9	40.4	2.4	1.6	68.1	2.5	2.8
AFOR-3	32.6	1.7	1.3	32.8	1.4	1.3	65.5	3.2	2.7	66.0	2.7	2.7	40.1	1.8	1.5	79.5	2.5	2.7
FOR	30.4	8.3	1.5	31.7	1.5	1.5	63.4	4.6	3.0	63.4	2.9	3.0	35.8	12.4	2.2	58.9	4.5	4.1
PFOR	37.8	2.1	1.5	38.1	1.9	1.5	78.2	9.8	3.0	77.1	4.7	3.0	45.7	14.1	2.2	87.6	10.5	4.0
Rice	69.0	2.1	1.5	69.4	2.9	1.5	141.0	6.5	3.0	139.0	3.9	3.0	89.4	11.9	1.8	134.3	2.9	3.2
S-64	41.0	2.3	1.8	42.5	8.0	1.8	82.8	3.8	3.6	80.7	2.8	3.6	53.9	2.2	1.8	75.7	2.8	3.1
VByte	40.2	1.4	2.9	39.8	1.3	2.9	85.7	8.0	5.8	80.7	2.1	5.8	46.7	1.3	3.2	77.8	1.9	5.7
Sindice																		
AFOR-1	31.4	1.3	1.8	40.6	1.1	1.9	76.7	2.0	3.5	83.6	19.9	3.7	300.3	2.9	19.1	1377.0	5.7	78.8
AFOR-2	36.8	1.4	1.6	51.9	14.6	1.7	73.5	2.3	3.2	99.3	13.3	3.4	329.9	3.2	17.5	1394.0	5.9	72.4
AFOR-3	36.3	1.3	1.6	45.6	1.2	1.7	72.0	3.0	3.1	85.6	2.3	3.2	325.0	4.1	16.9	1377.0	6.1	70.4
FOR	35.9	17.9	2.3	37.3	1.1	2.4	60.1	2.3	4.5	70.5	2.0	4.7	323.6	30.3	28.3	1382.0	7.8	116.3
PFOR	40.5	1.7	2.3	49.8	2.1	2.4	81.4	10.0	4.5	94.1	2.4	4.7	316.6	3.1	25.5	1282.0	6.4	103.0
Rice	68.5	2.0	1.8	82.4	1.5	1.9	151.0	3.7	3.6	155.8	2.9	3.7	848.3	14.9	18.6	3348.0	6.7	74.0
S-64	40.9	1.4	1.8	52.5	1.9	1.9	81.1	2.7	3.6	97.9	2.3	3.8	408.6	17.1	18.0	1700.0	12.2	74.5
VByte	40.3	1.1	2.8	61.1	1.7	3.0	79.5	2.2	5.5	111.5	14.6	5.8	462.3	31.7	31.5	1843.0	6.7	133.7

Table C.1.: Query time execution for the value queries per query type, algorithm and dataset. We report for each query type the arithmetic mean (μ in millisecond), the standard deviation (σ in millisecond) and the total amount of data read during query processing (MB in megabyte).

	2 - AND			2 - OR			4 - AND			4 - OR			2 - Phrase			3 - Phrase		
Method	μ	σ	MB	μ	σ	MB	μ	σ	MB	μ	σ	MB	μ	σ	MB	μ	σ	MB
DBpedia																		
AFOR-1	47.1	1.6	2.4	134.6	2.2	7.3	87.4	16.5	4.1	200.1	3.6	10.7	244.0	2.8	15.3	564.1	31.2	37.6
AFOR-2	64.0	2.1	2.2	132.5	2.7	6.8	103.0	15.3	3.8	220.0	10.3	9.9	282.3	17.0	14.2	594.4	15.2	34.7
AFOR-3	54.5	2.2	2.1	136.0	2.1	5.9	104.1	8.5	3.7	190.3	3.4	8.7	264.0	3.2	13.9	600.4	4.3	34.4
FOR	54.4	18.6	3.0	116.2	2.5	9.2	77.2	3.0	5.2	176.6	2.9	13.4	239.3	3.7	20.4	558.3	37.3	49.8
PFOR	61.3	4.7	3.1	146.1	2.4	8.7	117.1	4.2	5.3	199.3	3.9	12.7	249.3	3.5	18.0	578.2	32.6	43.3
Rice	107.0	2.4	2.3	312.2	3.2	6.8	192.8	3.2	3.9	475.5	12.2	9.8	677.0	5.2	13.2	1625.0	7.6	30.9
S-64	64.0	12.1	2.4	144.5	4.9	6.9	103.7	3.9	4.1	215.0	4.6	10.1	316.9	3.7	14.7	706.3	5.4	35.9
VByte	59.0	1.9	3.8	165.6	2.3	14.8	110.8	16.6	6.3	264.8	21.0	20.8	339.9	2.9	24.1	767.3	37.1	59.8
Geonames																		
AFOR-1	42.9	2.1	1.7	84.0	2.7	2.4	71.9	2.5	3.2	117.9	3.5	4.4	64.2	2.0	2.1	78.8	2.5	3.4
AFOR-2	55.6	18.7	1.7	91.2	1.9	2.3	85.9	19.1	3.1	129.8	3.6	4.3	59.8	2.7	2.0	90.1	3.3	3.2
AFOR-3	50.5	19.6	1.5	70.2	2.0	1.9	89.8	23.4	2.9	137.2	23.8	3.5	69.9	11.6	1.7	87.5	3.5	2.9
FOR	41.6	2.6	1.9	80.5	2.2	2.6	68.8	2.9	3.4	111.3	3.7	4.6	51.9	2.6	2.7	77.4	3.9	4.7
PFOR	56.3	3.2	2.0	81.0	2.9	2.7	94.1	2.8	3.5	137.5	4.6	4.7	67.4	3.1	2.8	98.2	3.5	4.5
Rice	97.5	2.7	1.9	158.1	5.7	2.5	165.2	4.2	3.4	272.8	2.8	4.6	120.8	2.6	2.2	173.5	3.2	3.6
S-64	60.6	21.1	2.1	83.7	3.0	2.6	96.0	3.8	3.8	168.9	4.0	4.9	75.4	17.7	2.2	99.3	3.4	3.5
VByte	57.9	17.6	3.9	91.9	2.5	6.8	95.2	3.4	6.8	159.2	3.5	12.5	82.7	2.2	4.5	116.1	24.5	6.9
Sindice																		
AFOR-1	55.5	1.9	2.1	192.9	2.4	8.2	77.8	2.4	3.9	311.1	3.1	14.4	310.3	4.0	19.0	1297.0	6.2	78.0
AFOR-2	53.3	1.6	1.9	229.2	3.0	7.5	105.1	11.3	3.5	330.7	32.3	13.2	341.0	4.0	17.4	1484.0	5.6	71.7
AFOR-3	52.1	1.9	1.8	180.2	2.5	5.5	88.7	2.5	3.3	291.2	3.2	10.0	334.8	2.9	16.6	1413.0	5.9	69.6
FOR	46.4	8.0	3.0	197.0	3.2	15.3	76.2	2.6	5.3	314.3	3.3	25.4	319.1	4.2	29.1	1304.0	7.1	115.9
PFOR	67.4	14.6	2.8	193.9	2.9	9.2	100.5	3.2	5.0	316.1	3.2	17.0	358.7	3.1	25.5	1348.0	7.4	102.3
Rice	100.4	2.3	2.4	481.3	3.8	10.9	170.5	3.4	4.1	797.8	30.6	18.5	825.6	5.3	19.2	3808.0	7.6	73.9
S-64	58.8	2.4	2.1	213.2	13.8	7.5	99.9	2.3	3.9	342.7	4.0	13.3	416.9	16.0	17.8	1724.0	7.8	73.7
VByte	58.8	12.8	3.8	311.3	3.0	25.9	97.8	2.2	6.5	478.1	53.0	42.5	438.4	4.1	32.7	1916.0	6.3	133.1

Table C.2.: Query time execution for the attribute queries per query type, algorithm and dataset. We report for each query type the arithmetic mean (μ in millisecond), the standard deviation (σ in millisecond) and the total amount of data read during query processing (MB in megabyte).

Colophon

This thesis was made in LaTeX 2_ε using the "hepthesis" class[1]. The graphical charts have been generated using PGF/TikZ[2] and PGFPlots[3]. Diagrams have been drawn using Yed Graph Editor[4] and Inkscape[5]. The diagram from Figure 2.1 is coming from Wikipedia[6] and is in the public domain.

[1]Hepthesis: `http://www.ctan.org/tex-archive/help/Catalogue/entries/hepthesis.html`
[2]PGF/TikZ: `http://www.ctan.org/tex-archive/help/Catalogue/entries/pgf.html`
[3]PGFPlots: `http://www.ctan.org/tex-archive/help/Catalogue/entries/pgfplots.html`
[4]Yed Graph Editor: `http://www.yworks.com/en/products_yed_about.html`
[5]Inkscape: `http://inkscape.org/`
[6]`http://en.wikipedia.org/wiki/File:Semantic-web-stack.png`

Bibliography

[AAB+09] Rakesh Agrawal, Anastasia Ailamaki, Philip A. Bernstein, Eric A. Brewer, Michael J. Carey, Surajit Chaudhuri, Anhai Doan, Daniela Florescu, Michael J. Franklin, Hector Garcia-Molina, Johannes Gehrke, Le Gruenwald, Laura M. Haas, Alon Y. Halevy, Joseph M. Hellerstein, Yannis E. Ioannidis, Hank F. Korth, Donald Kossmann, Samuel Madden, Roger Magoulas, Beng Chin Ooi, Tim O'Reilly, Raghu Ramakrishnan, Sunita Sarawagi, Michael Stonebraker, Alexander S. Szalay, and Gerhard Weikum. The Claremont report on database research. *Communications of the ACM*, 52:9, 2009.

[Abi97] Serge Abiteboul. Querying Semi-Structured Data. In *Proceedings of the 6th International Conference on Database Theory*, pages 1–18, 1997.

[ACD02] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: a system for keyword-based search over relational databases. In *Proceedings of the 18th International Conference on Data Engineering*, pages 5–16, San Jose, California, 2002. IEEE Comput. Soc.

[AM05] Vo Ngoc Anh and Alistair Moffat. Inverted Index Compression Using Word-Aligned Binary Codes. *Information Retrieval*, 8(1):151–166, 2005.

[AM06] Vo Ngoc Anh and Alistair Moffat. Structured Index Organizations for High-Throughput Text Querying. In *Proceedings of the 13th International Conference of String Processing and Information Retrieval*, pages 304–315. Springer, 2006.

[AM10] Vo Ngoc Anh and Alistair Moffat. Index compression using 64-bit words. *Software: Practice and Experience*, 40(2):131–147, 2010.

[AMMH07] Daniel J. Abadi, Adam Marcus, Samuel R. Madden, and Kate Hollenbach. Scalable semantic web data management using vertical partitioning. In

VLDB '07: Proceedings of the 33rd international conference on Very large data bases, pages 411–422. VLDB Endowment, 2007.

[AMS05] Kemafor Anyanwu, Angela Maduko, and Amit Sheth. Semrank: ranking complex relationship search results on the semantic web. In *Proceedings of the 14th international conference on World Wide Web*, pages 117–127. ACM, 2005.

[AQM+96] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1:68–88, 1996.

[AQM+97] Serge Abiteboul, Dallan Quass, Jason McHugh, Jennifer Widom, and Janet L. Wiener. The Lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[BB07] Liu Baolin and Hu Bo. HPRD: A High Performance RDF Database. In *Proceedings of Network and Parallel Computing, IFIP International Conference*, volume 4672 of *Lecture Notes in Computer Science*, pages 364–374. Springer, September 2007.

[BC07] Stefan Büttcher and Charles L. A. Clarke. Index compression is good, especially for random access. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management - CIKM '07*, pages 761–770, New York, New York, USA, 2007. ACM Press.

[Bec04] Dave Beckett. RDF/XML Syntax Specification (Revised). W3C recommendation, W3C, February 2004.

[BG03] Dave Beckett and Jan Grant. Semantic Web Scalability and Storage: Mapping Semantic Web Data with RDBMSes. SWAD-Europe deliverable, W3C, January 2003.

[BG04] Dan Brickley and Ramanathan V. Guha. RDF vocabulary description language 1.0: RDF schema. W3C recommendation, W3C, February 2004.

[BGvH+04] Paolo Bouquet, Fausto Giunchiglia, Frank van Harmelen, Luciano Serafini, and Heiner Stuckenschmidt. Contextualizing ontologies. *Journal of Web Semantics*, 1(4):325–343, 2004.

[BHN+02] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan.

Keyword searching and browsing in databases using BANKS. In *Proceedings of the 18th International Conference on Data Engineering*, pages 431–440. IEEE Comput. Soc, 2002.

[BHP04] Andrey Balmin, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: authority-based keyword search in databases. In *Proceedings of the Thirtieth international conference on Very large data bases*, pages 564–575. VLDB Endowment, 2004.

[BL06] T. Berners-Lee. Linked data. W3C Design Issues, July 2006.

[BLFN96] Tim Berners-Lee, Roy Thomas Fielding, and Henrik Frystyk Nielsen. Hypertext Transfer Protocol – HTTP/1.0. RFC 1945, W3C, May 1996.

[BLHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[BLMP06] A. Z. Broder, R. Lempel, F. Maghoul, and J. Pedersen. Efficient pagerank approximation via graph aggregation. *Information Retrieval*, 9:123–138, 2006.

[Boy08] Brent Boyer. Robust Java benchmarking, 2008.

[BPSM+08] Tim Bray, Jean Paoli, C. Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible Markup Language (XML) 1.0 (Fifth Edition). W3C recommendation, W3C, November 2008.

[BV05] Paolo Boldi and Sebastiano Vigna. Compressed Perfect Embedded Skip Lists for Quick Inverted-Index Lookups. In Mariano Consens and Gonzalo Navarro, editors, *Proceedings of the 12th International Conference on String Processing and Information Retrieval*, volume 3772 of *Lecture Notes in Computer Science*, pages 25–28, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BvHH+04] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web Ontology Language Reference. W3C Recommendation, W3C, February 2004.

[BVT+02] Kevin Beyer, Stratis D. Viglas, Igor Tatarinov, Jayavel Shanmugasundaram, Eugene Shekita, and Chun Zhang. Storing and querying ordered xml using a relational database system. In *SIGMOD '02: Proceedings of the 2002*

ACM SIGMOD international conference on Management of Data, pages 204–215, New York, NY, USA, 2002. ACM.

[BYCJ+07] Ricardo Baeza-Yates, Carlos Castillo, Flavio Junqueira, Vassilis Plachouras, and Fabrizio Silvestri. Challenges on Distributed Web Retrieval. In *Proceedings of the 23rd International Conference on Data Engineering*, pages 6–20. IEEE, 2007.

[BYD04] Ricardo Baeza-Yates and Emilio Davis. Web page ranking using link attributes. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*, pages 328–329. ACM, 2004.

[BYN96] Ricardo Baeza-Yates and Gonzalo Navarro. Integrating contents and structure in text retrieval. *SIGMOD Rec.*, 25(1):67–79, 1996.

[BYRN99] Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., 1999.

[CBHS05] J. Carroll, C. Bizer, P. Hayes, and P. Stickler. Named graphs, provenance and trust. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 613–622, New York, NY, USA, 2005. ACM.

[CCD+01] Anthony B. Coates, Dan Connolly, Diana Dack, Leslie L. Daigle, Ray Denenberg, Martin J. Dürst, Paul Grosso, Sandro Hawke, Renato Iannella, Graham Klyne, Larry Masinter, Michael Mealling, Mark Needleman, and Norman Walsh. URIs, URLs, and URNs: Clarifications and Recommendations 1.0. Technical report, W3C, September 2001.

[CDG+06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A Distributed Storage System for Structured Data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, page 15. USENIX Association, 2006.

[CDT07] Richard Cyganiak, Renaud Delbru, and Giovanni Tummarello. Semantic Web Crawling: A Sitemap Extension. Technical report, DERI, NUI Galway, 2007.

[CDTT10] Michele Catasta, Renaud Delbru, Nickolai Toupikov, and Giovanni Tummarello. *Managing Terabytes of Web Semantics Data*, page 93. Springer, 2010.

[CGMH+94] Sudarshan S. Chawathe, Hector Garcia-Molina, Joachim Hammer, Kelly Ireland, Yannis Papakonstantinou, Jeffrey D. Ullman, and Jennifer Widom. The TSIMMIS Project: Integration of Heterogeneous Information Sources. In *Proceedings of 10th Anniversary Meeting of the Information Processing Society of Japan*, pages 7–18, 1994.

[CGQ08] Gong Cheng, Weiyi Ge, and Yuzhong Qu. Falcons: searching and browsing entities on the semantic web. In *Proceeding of the 17th international conference on World Wide Web*, pages 1101–1102. ACM, 2008.

[CMKS03] Sara Cohen, Jonathan Mamou, Yaron Kanza, and Yehoshua Sagiv. XSEarch: a semantic search engine for XML. In *Proceedings of the 29th international conference on Very large data bases - VLDB '2003*, pages 45–56. VLDB Endowment, 2003.

[CRZT05] Nick Craswell, Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Relevance weighting for query independent evidence. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 416–423, New York, NY, USA, 2005. ACM.

[CSD+08] Richard Cyganiak, Holger Stenzhorn, Renaud Delbru, Stefan Decker, and Giovanni Tummarello. Semantic sitemaps: efficient and flexible access to datasets on the semantic web. In *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, pages 690–704, 2008.

[CSF+01] Brian F Cooper, Neal Sample, Michael J Franklin, Gísli R Hjaltason, and Moshe Shadmon. A Fast Index for Semistructured Data. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 341–350, 2001.

[CT04] John Cowan and Richard Tobin. XML Information Set (Second Edition). W3C recommendation, W3C, February 2004.

[dBG+07] M. d'Aquin, C. Baldassarre, L. Gridinoc, S. Angeletou, M. Sabou, and E. Motta. Characterizing Knowledge on the Semantic Web with Watson. In *EON*, pages 1–10, 2007.

[DG08] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing

on large clusters. *Communications of the ACM*, 51(1):6, 2008.

[DH07] Xin Dong and Alon Halevy. Indexing dataspaces. *Proceedings of the 2007 ACM SIGMOD international conference on Management of data - SIGMOD '07*, page 43, 2007.

[dK86] Johan de Kleer. An Assumption-Based TMS. *Artif. Intell.*, 28(2):127–162, 1986.

[DPF+05a] Li Ding, Rong Pan, Tim Finin, Anupam Joshi, Yun Peng, and Pranam Kolari. Finding and Ranking Knowledge on the Semantic Web. In *Proceedings of the 4th International Semantic Web Conference*, pages 156–170, 2005.

[DPF+05b] Li Ding, Rong Pan, Timothy W. Finin, Anupam Joshi, Yun Peng, and Pranam Kolari. Finding and ranking knowledge on the semantic web. In *Proceedings of the International Semantic Web Conference*, pages 156–170, 2005.

[DPTD08] Renaud Delbru, Axel Polleres, Giovanni Tummarello, and Stefan Decker. Context Dependent Reasoning for Semantic Documents in Sindice. In *Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2008)*, 2008.

[DTC+09] Renaud Delbru, Nickolai Toupikov, Michele Catasta, Robert Fuller, and Giovanni Tummarello. SIREn: Efficient Search on Semi- Structured Documents. In *Lucene in Action, Second Edition*, chapter 11. Manning Publications Co., 2009.

[DTC+10a] Renaud Delbru, Nickolai Toupikov, Michele Catasta, Giovanni Tummarello, and Stefan Decker. A Node Indexing Scheme for Web Entity Retrieval. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*, pages 225–239. Springer, 2010.

[DTC+10b] Renaud Delbru, Nickolai Toupikov, Michele Catasta, Giovanni Tummarello, and Stefan Decker. Hierarchical Link Analysis for Ranking Web Data. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC 2010)*, pages 240–256. Springer, 2010.

[DTM08] Li Ding, Jiao Tao, and Deborah L. McGuinness. An initial investigation on evaluating semantic web instance data. In *WWW '08: Proceeding of the 17th international conference on World Wide Web*, pages 1179–1180, New

York, NY, USA, 2008. ACM.

[EAT04] Nadav Eiron, Kevin S. McCurley John A., and Tomlin. Ranking the web frontier. In *Proceedings of the 13th international conference on World Wide Web*, pages 309–318, New York, NY, USA, 2004. ACM.

[EDR05] Vuk Ercegovac, David J. DeWitt, and Raghu Ramakrishnan. The TEXTURE benchmark: measuring performance of text queries on a relational DBMS. In *Proceedings of the 31st international conference on Very large data bases*, pages 313–324. VLDB Endowment, 2005.

[FBGV09] George H. L. Fletcher, Jan Van Den Bussche, Dirk Van Gucht, and Stijn Vansummeren. Towards a theory of search queries. In *Proceedings of the 12th International Conference on Database Theory*, pages 201—-211, 2009.

[FLMM06] P. Ferragina, F. Luccio, G. Manzini, and S. Muthukrishnan. Compressing and searching XML data via two zips. In *Proceedings of the 15th international conference on World Wide Web - WWW '06*, pages 751–760, New York, New York, USA, 2006. ACM Press.

[Flo05] Daniela Florescu. Managing semi-structured data. *Queue*, 3:18–24, October 2005.

[FLW+06] Guang Feng, Tie-Yan Liu, Ying Wang, Ying Bao, Zhiming Ma, Xu-Dong Zhang, and Wei-Ying Ma. Aggregaterank: bringing order to web sites. In *Proceedings of the 29th annual international ACM SIGIR conference*, page 75. ACM Press, 2006.

[FW04] David C. Fallside and Priscilla Walmsley. XML Schema Part 0: Primer Second Edition. W3C recommendation, W3C, October 2004.

[Giu93] Fausto Giunchiglia. Contextual reasoning. *Epistemologia, special issue on I Linguaggi e le Macchine*, 345:345–364, 1993.

[GMF04] R. V. Guha, R. McCool, and R. Fikes. Contexts for the Semantic Web. In *International Semantic Web Conference*, pages 32–46, 2004.

[GMM03] R. Guha, Rob McCool, and Eric Miller. Semantic search. In *Proceedings of the 12th international conference on World Wide Web*, pages 700–709, 2003.

[Gra93] Goetz Graefe. Query evaluation techniques for large databases. *ACM*

Computing Surveys, 25(2):73, 1993.

[Gra06] Goetz Graefe. B-tree indexes for high update rates. *ACM SIGMOD Record*, 35(1):39, 2006.

[Gri08] Nils Grimsmo. Faster Path Indexes for Search in XML Data. In *Proceedings of the nineteenth conference on Australasian database*, pages 127–135, 2008.

[GRS98] Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. Compressing relations and indexes. In *Proceedings of the 14th International Conference on Data Engineering*, pages 370–379, Washington, DC, USA, 1998. IEEE Computer Society.

[GSBS03] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. XRANK: ranked keyword search over XML documents. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data - SIGMOD '03*, pages 16–27, New York, New York, USA, 2003. ACM Press.

[Guh92] R. V. Guha. *Contexts: a formalization and some applications*. PhD thesis, Stanford, CA, USA, 1992.

[GW97] Roy Goldman and Jennifer Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 436–445, 1997.

[Hay04] Patrick Hayes. RDF Semantics. W3C Recommendation, W3C, February 2004.

[HD05] A. Harth and S. Decker. Optimized Index Structures for Querying RDF from the Web. In *Proceedings of the Third Latin American Web Congress (LA-WEB'2005)*, pages 71–80. IEEE, 2005.

[HFM06] Alon Halevy, Michael Franklin, and David Maier. Principles of dataspace systems. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, Chicago, IL, USA, 2006.

[HHD06] Aidan Hogan, Andreas Harth, and Stefan Decker. Reconrank: A scalable ranking method for semantic web data with context. In *Proceedings of Second International Workshop on Scalable Semantic Web Knowledge Base*

Systems, Athens, GA, USA., 11 2006.

[HHJ+06] Wang Haixun, He Hao, Yang Jun, P.S. Yu, and J.X. Yu. Dual Labeling: Answering Graph Reachability Queries in Constant Time. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, pages 75–75. IEEE, 2006.

[HHP09] Aidan Hogan, Andreas Harth, and Axel Polleres. Scalable Authoritative OWL Reasoning for the Web. *International Journal on Semantic Web and Information Systems*, 5(2):49–90, 2009.

[HHUD08] Andreas Harth, Aidan Hogan, Jürgen Umbrich, and Stefan Decker. SWSE: Objects before documents! In *Proceedings of the Billion Triple Semantic Web Challenge, 7th International Semantic Web Conference*, 2008.

[HKD09] Andreas Harth, Sheila Kinsella, and Stefan Decker. Using Naming Authority to Rank Data and Ontologies for Web Search . In *The Semantic Web - ISWC 2009*, volume 5823 of *Lecture Notes in Computer Science*, pages 277 – 292, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[HP02] Vagelis Hristidis and Yannis Papakonstantinou. Discover: keyword search in relational databases. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 670–681, 2002.

[HPPD10] Aidan Hogan, Jeff Z. Pan, Axel Polleres, and Stefan Decker. SAOR: Template Rule Optimisations for Distributed Reasoning over 1 Billion Linked Data Triples. In *Proceedings of the 9th International Semantic Web Conference*. Springer, 2010.

[HUHD07] Andreas Harth, Jürgen Umbrich, Aidan Hogan, and Stefan Decker. YARS2: A Federated Repository for Querying Graph Structured Data from the Web. In *Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference*, volume 4825 of *Lecture Notes in Computer Science*, pages 211–224. Springer Verlag, November 2007.

[HWYY05] Hao He, Haixun Wang, Jun Yang, and Philip S. Yu. Compact reachability labeling for graph-structured data. In *Proceedings of the 14th ACM international conference on Information and knowledge management - CIKM '05*, pages 594–601, New York, New York, USA, 2005. ACM Press.

[JCE+07] H. V. Jagadish, Adriane Chapman, Aaron Elkiss, Magesh Jayapandian,

Yunyao Li, Arnab Nandi, and Cong Yu. Making database systems usable. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 13—-24, Beijing, China, 2007.

[KC04] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C recommendation, W3C, February 2004.

[KHMG03] Sepandar Kamvar, Taher Haveliwala, Christopher Manning, and Gene Golub. Exploiting the block structure of the web for computing pagerank. Technical Report 2003-17, Stanford InfoLab, 2003.

[KPC+05] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S. Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. Bidirectional expansion for keyword search on graph databases. In *Proceedings of the 31st international conference on Very large data bases*, pages 505–516, Trondheim, Norway, 2005.

[KSI+08] Gjergji Kasneci, Fabian M. Suchanek, Georgiana Ifrim, Shady Elbassuoni, Maya Ramanath, and Gerhard Weikum. NAGA: harvesting, searching and ranking knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, pages 1285–1288, Vancouver, Canada, 2008.

[Lil00] David J. Lilja. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2000.

[LM01] Quanzhong Li and Bongki Moon. Indexing and Querying XML Data for Regular Path Expressions. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 361–370, 2001.

[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 *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, pages 903–914, Vancouver, Canada, 2008.

[LWC07] Ziyang Liu, Jeffrey Walker, and Yi Chen. XSeek: a semantic XML search engine using keywords. In *Proceedings of the 33rd international conference on Very large data bases*, pages 1330–1333, 2007.

[LWP+03] Lipyeow Lim, Min Wang, Sriram Padmanabhan, Jeffrey Scott Vitter, and Ramesh Agarwal. Dynamic maintenance of web indexes using landmarks. In *Proceedings of the 12th international conference on World Wide Web*, page 102, 2003.

[LYMC06] Fang Liu, Clement Yu, Weiyi Meng, and Abdur Chowdhury. Effective keyword search in relational databases. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data - SIGMOD '06*, page 563, Chicago, IL, USA, 2006. ACM Press.

[Mai83] David Maier. *Theory of Relational Databases*. Computer Science Press, 1983.

[MBS08] Alistair Miles, Thomas Baker, and Ralph Swick. Best Practice Recipes for Publishing RDF Vocabularies. W3C working group note, W3C, 2008.

[McC93] John McCarthy. Notes On Formalizing Context. In *Proceedings of IJCAI-93*, pages 555–560, 1993.

[Mel07] Massimo Melucci. On rank correlation in information retrieval evaluation. *SIGIR Forum*, 41(1):18–33, 2007.

[MF03] J. Mayfield and T. Finin. Information retrieval on the Semantic Web: Integrating inference and retrieval. In *Proceedings of the SIGIR Workshop on the Semantic Web*, August 2003.

[MJCW04] Xiaofeng Meng, Yu Jiang, Yan Chen, and Haixun Wang. XSeq: an indexing infrastructure for tree pattern queries. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data - SIGMOD '04*, pages 941–942, New York, New York, USA, 2004. ACM Press.

[MMVP09] Federica Mandreoli, Riccardo Martoglia, Giorgio Villani, and Wilma Penzo. Flexible query answering on graph-modeled data. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*, pages 216—-227, Saint Petersburg, Russia, 2009. ACM.

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[MS00] Alistair Moffat and Lang Stuiver. Binary Interpolative Coding for Effective Index Compression. *Information Retrieval*, 3(1):25–47, 2000.

[MvH04] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. W3C recommendation, W3C, January 2004.

[MZ96] Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. *ACM Trans. Inf. Syst.*, 14(4):349–379, 1996.

[NMC+99] P M Nadkarni, L. Marenco, R. Chen, E. Skoufos, G. Shepherd, and P. Miller. Organization of heterogeneous scientific data using the EAV/CR representation. *Journal of the American Medical Informatics Association : JAMIA*, 6(6):478–493, 1999.

[NW08] Thomas Neumann and Gerhard Weikum. RDF-3X - a RISC-style Engine for RDF. *Proceedings of the VLDB Endowment*, 1(1):647–659, 2008.

[NZWM05] Zaiqing Nie, Yuanzhi Zhang, Ji-Rong Wen, and Wei-Ying Ma. Object-level ranking: bringing order to Web objects. In *Proceedings of the 14th international conference on World Wide Web*, page 567. ACM, 2005.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999.

[PFH06] A. Polleres, C. Feier, and A. Harth. Rules with contextually scoped negation. In *3rd European Semantic Web Conference (ESWC2006)*, volume 4011 of *LNCS*, Budva, Montenegro, June 2006. Springer.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 251–260. IEEE Comput. Soc. Press, 1995.

[PS08] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C recommendation, W3C, January 2008.

[QAU+96] Dallan Quass, Serge Abiteboul, Jeff Ullman, Janet Wiener, Jennifer Widom, Roy Goldman, Kevin Haas, Qingshan Luo, Jason McHugh, Svetlozar Nestorov, Anand Rajaraman, and Hugo Rivero. LORE: a Lightweight Object REpository for semistructured data. In *Proceedings of the 1996*

ACM SIGMOD international conference on Management of data - SIGMOD '96, page 549, New York, New York, USA, 1996. ACM Press.

[RM04] Praveen Roa and Bongki Moon. PRIX: indexing and querying XML using prufer sequences. In *Proceedings of the 20th International Conference on Data Engineering*, pages 288–299. IEEE Computer Society, 2004.

[RZT04] Stephen Robertson, Hugo Zaragoza, and Michael Taylor. Simple BM25 extension to multiple weighted fields. In *Proceedings of the Thirteenth ACM conference on Information and knowledge management - CIKM '04*, pages 42–49, New York, New York, USA, 2004. ACM Press.

[SB04] Luciano Serafini and Paolo Bouquet. Comparing formal theories of context in AI. *Artificial Intelligence*, 155(1-2):41, 2004.

[SBPR07] Heiko Stoermer, Paolo Bouquet, Ignazio Palmisano, and Domenico Redavid. A Context-Based Architecture for RDF Knowledge Bases: Approach, Implementation and Preliminary Results. In *RR*, pages 209–218, 2007.

[SCCS09] Haw Su-Cheng and Lee Chien-Sing. Node Labeling Schemes in XML Query Optimization: A Survey and Trends. *IETE Technical Review*, 26(2):88, 2009.

[SdDTZ97] Ron Sacks-davis, Tuong Dao, James A. Thom, and Justin Zobel. Indexing documents for queries on structure, content and attributes. In *Proceedings of International Symposium on Digital Media Information Base*, pages 236–245. World Scientific, November 1997.

[SFW83] Gerard Salton, Edward A. Fox, and Harry Wu. Extended Boolean information retrieval. *Communications of the ACM*, 26(11):1022–1036, 1983.

[SG09] Hassan Sayyadi and Lise Getoor. Futurerank: Ranking scientific articles by predicting their future pagerank. In *SDM*, pages 533–544, 2009.

[Spa72] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[Sti05] Patrick Stickler. CBD - Concise Bounded Description. W3C Member Submission, W3C, June 2005.

[STW05] Ralf Schenkel, Anja Theobald, and Gerhard Weikum. Semantic Similarity Search on Semistructured Data with the XXL Search Engine. *Information*

Retrieval, 8(4):521–545, 2005.

[TCC+10] Giovanni Tummarello, Richard Cyganiak, Michele Catasta, Scyzmon Danielczyk, Renaud Delbru, and Stefan Decker. Sig.ma: Live views on the Web of Data. *Journal of Web Semantics*, 2010.

[tH05] H. J. ter Horst. Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary. *Journal of Web Semantics*, 3(2-3):79–115, 2005.

[TSW05] Martin Theobald, Ralf Schenkel, and Gerhard Weikum. An Efficient and Versatile Query Engine for TopX Search. In *Proceedings of the 31st international conference on Very Large Data Bases*, pages 625–636, 2005.

[TUD+09] Nickolai Toupikov, Jürgen Umbrich, Renaud Delbru, Michael Hausenblas, and Giovanni Tummarello. DING! Dataset Ranking using Formal Descriptions. In *Linked Data on the Web Workshop (LDOW09), 18th International World Wide Web Conference (WWW09)*, 2009.

[UKM+10] Jacopo Urbani, Spyros Kotoulas, Jason Maassen, Frank van Harmelen, and Henri Bal. OWL Reasoning with WebPIE: Calculating the Closure of 100 Billion Triples. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *Proceedings of the 7th Extended Semantic Web Conference, ESWC2010*, volume 6088 of *Lecture Notes in Computer Science*, pages 213–227, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[UKOvH09] Jacopo Urbani, Spyros Kotoulas, Eyal Oren, and Frank van Harmelen. Scalable distributed reasoning using mapreduce. In *International Semantic Web Conference*, pages 634–649, 2009.

[VS10] Rossano Venturini and Fabrizio Silvestri. Vsencoding: Efficient coding and fast decoding of integer lists via dynamic programming. In *Proceedings of the nineteenth ACM conference on Conference on information and knowledge management- CIKM'10*, 2010.

[WD04] Yuan Wang and David J. DeWitt. Computing pagerank in a distributed internet search system. In *Proceedings of the Thirtieth international conference on Very large data bases*, pages 420–431, Toronto, Canada, 2004. VLDB Endowment.

[WFM+07] Norman Walsh, Mary Fernández, Ashok Malhotra, Marton Nagy, and Jonathan Marsh. XQuery 1.0 and XPath 2.0 data model (XDM). W3C recommendation, W3C, January 2007.

[WH09] Jesse Weaver and James A. Hendler. Parallel materialization of the finite rdfs closure for hundreds of millions of triples. In *International Semantic Web Conference (ISWC2009)*, pages 682–697, 2009.

[WJW+05] Wei Wang, Haifeng Jiang, Hongzhi Wang, Xuemin Lin, Hongjun Lu, and Jianzhong Li. Efficient processing of XML path queries using the disk-based F&B Index. In *Proceedings of the 31st international conference on Very Large Data Bases*, pages 145–156, 2005.

[WKB08] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore - sextuple indexing for semantic web data management. *Proceedings of the VLDB Endowment*, 1(1):1008–1019, 2008.

[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. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):177–188, 2009.

[WMB99] Ian H. Witten, Alistair Moffat, and Timothy C. Bell. *Managing gigabytes (2nd ed.): compressing and indexing documents and images*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[WPFY03] Haixun Wang, Sanghyun Park, Wei Fan, and Philip S. Yu. ViST: a dynamic index method for querying XML data by tree structures. In *Proceedings of the 2003 ACM SIGMOD international conference on on Management of data*, pages 110–121, New York, New York, USA, 2003. ACM Press.

[WXYM06] Dylan Walker, Huafeng Xie, Koon-Kiu Yan, and Sergei Maslov. Ranking scientific publications using a simple model of network traffic. *CoRR*, 2006.

[XG04] Wenpu Xing and Ali Ghorbani. Weighted pagerank algorithm. In *CNSR '04: Proceedings of the Second Annual Conference on Communication Networks and Services Research*, volume 0, pages 305–314, Washington, DC, USA, 2004. IEEE Computer Society.

[XYZ+05] Gui-Rong Xue, Qiang Yang, Hua-Jun Zeng, Yong Yu, and Zheng Chen. Exploiting the hierarchical structure for link analysis. In *Proceedings of*

the 28th annual international ACM SIGIR conference, pages 186–193, New York, NY, USA, 2005. ACM.

[YDS09] Hao Yan, Shuai Ding, and Torsten Suel. Inverted index compression and query processing with optimized document ordering. In *Proceedings of the 18th international conference on World Wide Web - WWW '09*, pages 401–410, New York, New York, USA, 2009. ACM Press.

[ZHNB06] M. Zukowski, S. Heman, Niels Nes, and Peter Boncz. Super-Scalar RAM-CPU Cache Compression. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, pages 59–59, Washington, DC, USA, 2006. IEEE Computer Society.

[ZLS08] Jiangong Zhang, Xiaohui Long, and Torsten Suel. Performance of compressed inverted list caching in search engines. In *Proceeding of the 17th international conference on World Wide Web - WWW '08*, pages 387–396, New York, New York, USA, 2008. ACM Press.

[ZM06] Justin Zobel and Alistair Moffat. Inverted files for text search engines. *ACM Computer Surveys*, 38(2):6, 2006.

List of figures

List of tables

Listings