2014

Semantic Web for Everyone: Exploring Semantic Web Knowledge Bases via Contextual Tag Clouds and Linguistic Interpretations

Xingjian Zhang
Lehigh University

Semantic Web for Everyone: Exploring Semantic Web Knowledge Bases via Contextual Tag Clouds and Linguistic Interpretations

by

Xingjian Zhang

Presented to the Graduate and Research Committee
of Lehigh University
in Candidacy for the Degree of
Doctor of Philosophy

in

Computer Science

Lehigh University

September, 2014

Approved and recommended for acceptance as a dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Accepted Date

Committee Members:

Jeff Heflin, Committee Chair and Dissertation Advisor

Associate Professor, Lehigh University

Donald Hillman, Committee Member

Professor, Lehigh University

Brian Davison, Committee Member

Associate Professor, Lehigh University

Wei-Min Huang, Committee Member

Professor, Lehigh University

Acknowledgment

Over the past seven years I have received support and encouragement from a great number of individuals. I first want to thank my advisor, Professor Jeff Heflin for his guidance and advice. I have learned a lot from him through the brainstorm of ideas, discussion on experiments and revisions on submitted papers. I would like to thank my dissertation committee Professors Donald Hillman, Brian Davison and Wei-min Huang for providing their valuable feedback on my work. I want also to thank Professor Xiaolei Huang for providing the opportunity to participate the "Multi-modal Cervix Classification System" project.

I would like to thank many members of SWAT Lab: Zhengxiang Pan, Abir Qasem, Ameet Chitnis, Yingjie Li, Yang Yu, Dezhao Song, Sambhawa Priya and Yi Luo, for their help in the projects and during my research, and I always enjoyed discussion with them to broaden my knowledge about other interesting research topics in Semantic Web. I would also like to thank other students in the department: Liangjie Hong, Na Dai, and Dawei Yin, who helped me with the questions related to their research interest. I also want to thank all the participants that helped with my research in the user study or providing their judgment for ground truth.

Also I want to thank my friends at Lehigh: Pu Zhao, Xuebin Wu, Dan Li, Xiaochen Zhang, and Xiaoxiao Ma, for all the help and fun you offer to my life at Lehigh. Finally I would thank my parents, for their encouragement and love, through every stage of my personal and academic life.

Contents

List of Tables

List of Figures

Abstract

The amount of Semantic Web data is huge and still keeps growing rapidly today. However most users are still not able to use a Semantic Web Knowledge Base (KB) effectively as desired to due to the lack of various background knowledge. Furthermore, the data is usually heterogeneous, incomplete, and even contains errors, which further impairs understanding the dataset. How to quickly familiarize users with the ontology and data in a KB is an important research challenge to the Semantic Web community.

The core part of our proposed resolution to the problem is the contextual tag cloud system: a novel application that helps users explore a large scale RDF(Resource Description Framework) dataset. The tags in our system are ontological terms (classes and properties), and a user can construct a context with a set of tags that defines a subset of instances. Then in the contextual tag cloud, the font size of each tag depends on the number of instances that are associated with that tag and all tags in the context. Each contextual tag cloud serves as a summary of the distribution of relevant data, and by changing the context, the user can quickly gain an understanding of patterns in the data. Furthermore, the user can choose to include different RDFS entailment regimes in the calculations of tag sizes, thereby understanding the impact of semantics on the data. To resolve the key challenge of scalability, we combine a scalable preprocessing approach with a specially-constructed inverted index and co-occurrence matrix, use three approaches to prune unnecessary counts for faster online computations, and design a paging and streaming interface. Via experimentation, we show how much our design choices benefit the

responsiveness of our system. We conducted a preliminary user study on this system, and find novice participants felt the system provided a good means to investigate the data and were able to complete assigned tasks more easily than using a baseline interface.

We then extend the definition of tags to more general categories, particularly including property values, chaining property values, or functions on these values. With a totally different scenario and more general tags, we find the system can be used to discover interesting value space patterns. To adapt the different dataset, we modify the infrastructure with new indexing data structure, and propose two strategies for online queries, which will be chosen based on different requests, in order to maintain responsiveness of the system.

In addition, we consider other approaches to help users locate classes by natural language inputs. Using an external lexicon, Word Sense Disambiguation (WSD) on the label words of classes is one way to understand these classes. We propose our novel WSD approach with our probability model, derive the problem formula into small computable pieces, and propose ways to estimate the values of these pieces. For the other approach, instead of relying on external sources, we investigate how to retrieve query-relevant classes by using the annotations of instances associated with classes in the knowledge base. We propose a general framework of this approach, which consists of two phases: the keyword query is first used to locate relevant instances; then we induce the classes given this list of weighted matched instances.

Following the description of the accomplished work, I propose some important future work for extending the current system, and finally conclude the dissertation.

Chapter 1

Introduction

As the Semantic Web has evolved over the last decade, the amount of interlinked structured data has grown tremendously. Billions of statements are published using Semantic Web languages, by people from all over the world. The knowledge covers many domains such as media, geographic data, publications, life science, and so on. In addition to academic researchers and organizations, more and more companies are publishing their data in Semantic Web format, such as NY Times, BBC, Nature Publishing Group, etc. While some of the statements are very domain specific knowledge (such as life science) created by domain experts, there are also many statements that bridge various vocabularies among different domains.

While such a huge amount of miscellaneous information potentially enables many different powerful applications, we also notice that there are some obstacles preventing people from taking full advantage of a Semantic Web knowledge base (KB). One of the

challenges is how to present a KB, particularly a multi-source or cross-domain KB, which is usually huge in terms of its terminology or the amount of data, to casual users and familiarize them with it so that they can quickly start building interesting queries and get useful answers. The terminology is formally defined as an ontology which represents knowledge as a hierarchy of concepts within a domain, using a shared vocabulary to denote the classes (types of things), properties (relations and attributes) and interrelationships of those concepts. The data contains facts associated with the terminological vocabulary within the KB. The users' unfamiliarity with the KB (which we refer to as their knowledge gap) arises due to various aspects with regard to both the ontology and data.

1. **Syntactic Correctness**: "What classes are available?" Knowing the ontological terms (classes and properties) is usually the first task. If the user does not know what terms are available in the KB, there is no way for the user to build a formal query.

2. **Semantic Correctness**: "Does this class refer to the concept I expect?" Knowing the terms also includes correctly understanding them, not only knowing their names. The meaning of a term can be found in the formal definition of ontological interrelations, or be implied by the actual usage in the data (which occasionally might be different from the ontological definition). Understanding the semantics of the terms is essential to construct a meaningful query.

3. **Meaningful Results**: "Does the dataset hold enough knowledge coded with the vocabulary I choose?" Knowing the distribution of data in the KB can also be

important for a successful query. A query can be semantically correct but practically less helpful, simply because the coverage of the data is incomplete with regard to the ontology.

4. **User-friendly Interface**: "Can I construct useful and customized queries without using a formal query language?" Even with all the above questions answered, i.e. the users understand exactly what terms to use, casual user might still need help with writing correct formal queries. Even for expert users, an easy interface for frequently used template queries could save a lot of time.

These are the real problems to the Linked Data world and knowledge gaps that users typically have. Even taking an adequate subset of the linked data cloud, we can see wide spread domains from different sources. Then ambiguity becomes a common issue: sometimes a word is used with different senses (e.g. "Bridge" may refer to a structure or a card game), and sometimes the same sense is refined within different domains (e.g. "Person" in a scientific ontology may just refer to "Scientist"). Furthermore, the linked data usually contains errors and is incomplete, which adds to the barrier of understanding the KB. The errors can mislead the users and enlarge the first and the second gaps; and the incomplete coverage of knowledge is exactly the third gap.

Thus our first set of research questions arises: How can we help casual users explore the Linked Open Data (LOD) cloud? Can we provide a more detailed summary of linkages beyond the LOD cloud diagram? Can we help data providers find potential errors or missing links in a multi-source dataset of mixed quality?

Since there are two aspects of a dataset: the ontological terms (classes and properties) and the instances, there are two straightforward ideas to present the KB to users. One way to help users understand these terms is to show the axioms related to each term, however sometimes the axioms are missing or too complex to present in a user-friendly way. Another approach is to examine the instances related to each term. While looking into the related instances one by one provides the most details, it is very time consuming, and there is a risk of getting misled by coincidently looking into some erroneous data. So the questions cannot be answered by only viewing the ontology axioms or only inspecting a small sample of instances. A combined view of both aspects is necessary. There are also two types of linkages: ontological alignment and `owl:sameAs` links between instances. The usability of a multi-source RDF dataset is largely affected by the erroneous or missing links of both kinds in the dataset. Data providers also need a tool, which ideally emphasizes the unlikely facts to help unveil such problems in the dataset.

Our solution is to use tag clouds to display statistical information about the distribution of instances among various ontological terms. A key feature is that each tag cloud is relative to a filter consisting of ontological terms that is dynamically defined by the user. In analogy to traditional Web 2.0 tag cloud systems, an instance is like a web document or photo, but is "tagged" with formal ontological classes, as opposed to folksonomies. Tags are then another name for the categories of instances. We extend the expressiveness and treat classes, properties and inverse properties as tags that are assigned to any instances that use these ontological terms in their triples. The font sizes in the tag cloud reflect

the number of matching instances for each tag. We allow the user to change their focus on a specific subset of instances in the dataset by specifying a combination of ontological terms as the filter on the fly, and then the resulting contextual tag cloud will resize tags to indicate intersection with this context.

We are also curious about how useful this contextual tag cloud system is for the scenarios when the KB is not so complex. In such scenarios, some of the knowledge gaps might be less important. For example:

- If the KB covers **a very focused domain**, and the user has some background knowledge about this domain, then the second gap is less likely to happen since the terms with a specific focused domain usually have clear meanings with little ambiguity. However, for casual users, or someone who is not familiar enough to the terminology in this KB, it is still very important to clarify the terms.

- If the KB has **full-fledged data**, i.e. the closed world assumption is valid, then presumably any semantically meaningful query will be productive and we do not need to worry about the third gap. However it is a bonus if we can provide users some kind of statistics about the dataset with simple user interactions instead of letting the users construct the queries.

In these scenarios, we can usually assume the users are or will quickly become familiar with the ontological terms and how to use them; however there is still the need for an easy interface. On the other hand, we notice that the contextual tag cloud system is not only useful as a tool that familiarize users with the ontological terms, it is also a novel approach

to let users construct certain queries dynamically and then observe interesting patterns from the dataset. For the linked data KB, we present the patterns of ontological terms. For a dataset with focused domain and full-fledged data, we can extend the concept of tags to property values, chaining property values, and functions on such values and then visualize the patterns of values.

In addition to the contextual tag cloud systems, we also investigate approaches to annotate classes in the KB from the natural language aspect. This research direction helps reduce two kinds of gaps:

1. If a user has a particular information need, a keyword search will return the most relevant classes (not necessarily string match).

2. When a user looks at a class, additional explanation (e.g. alternative expressions) will avoid ambiguity and confusion.

One approach we take is to perform Word Sense Disambiguation (WSD) on the label words of classes with the help of external lexicons. We propose a novel WSD approach based on our probabilistic model of word senses of classes' labels, derive the problem formula into small computable pieces, and propose ways to estimate the values of these pieces. For the other approach, instead of relying on external sources, we investigate how to retrieve query-relevant classes by using the annotations of instances associated with classes in the KB. We propose a general framework of this approach, which consists of two phases: the keyword query is first used to locate relevant instances; then we induce the classes given this list of weighted matched instances.

Based upon the above discussion, this dissertation will summarize the accomplished research for reducing the knowledge gaps that prevent users from effectively using the Semantic Web KBs, particularly the large scaled ones with mixed sources and mixed data qualities. I will present the details of design and systems and evaluation results of the developed algorithms. The contributions of this dissertation is:

1. A novel interface that combines aspects of tag clouds and faceted browsing in order to explore Semantic Web knowledge bases consisting of hundreds of thousands of terms. A preliminary user interface study found that 85.7% of novice participants felt the system provided a good means to investigate the data and were able to complete assigned tasks in the same time or less than using a baseline faceted-browsing interface.

2. A scalable infrastructure, in terms of both preprocessing and online computation, of the contextual tag cloud system that can load billions of triples for KBs on LOD. The preprocessing has an almost linear performance with regard to the input size, and can process 1.4 billion triples within 12 hours. We use an inverted index to store the data, and apply effective pruning strategies to avoid unnecessary queries to the index. The online system can respond 90% of the requests to display the first page within 1 second and 97% of the requests within 2 seconds.

3. A demonstration that the interface can also be extended to enable investigation of computer logs with as many as 33 million events. In particular, such application is made possible by a light-weight implicit ontology that discretizes property values

and adds simple hierarchies for date/time and other features. Such a system could be used to facilitate data analyzers in the tasks like monitoring popular trends or suspicious activities and finding out significant differences by comparing patterns.

4. linguistic approaches towards alternative labels for ontological classes including a novel WSD approach based on our probability model for the labels of classes and a two-phase framework that uses textual information of the instances to retrieve relevant classes. The WSD is able to get accuracy of up to a 84.6% while in comparison, the baseline approach only has 64.1% accuracy. The class retrieval method has $\sim$ 20% improvement in Discounted Cumulative Gain scores comparing to the exact string match or synonym expansion approaches.

The rest of this dissertation is organized as follows. Chapter 2 provides background information on the Semantic Web and Linked Data, and also describes related work in the relevant research fields. In Chapter 3, I introduce the overall concept and design of the contextual tag cloud system, and some use scenarios of the system. Following that, in Chapter 4 we will discuss the computation challenges behind the system, and experiments that verify our design choices. Then in Chapter 5, we present a tag cloud system based on property values with the extended concept of tags, and also discuss the new scalability questions after this extension, with experimental data. In Chapter 6, we introduce the user study on the tag cloud system in comparison with another baseline system we implemented. Chapter 7 will cover our WSD algorithm on the labels of classes, and Chapter 8 will cover the two-phase instance-based class retrieval framework, which

both are ways to help users understanding the classes from the natural language aspect, and can be treated as possible extensions to the contextual tag cloud system. In Chapter 9 we will propose some future work, whose detail is out the scope of the dissertation, but still important for extending the current system. We finally conclude in Chapter 10.

Chapter 2

Background

In this chapter, I review important technology and introduce background knowledge related to this dissertation. First, I provide a brief introduction to the Semantic Web technology and Linked Data. After that I will compare several different ideas and system for representing a Semantic Web KB, or an RDF dataset. Then, I will discuss scalable repositories and the use of inverted index of Semantic Web data. Finally, I will introduce relevant works about the relation between natural language words and ontological classes, including WSD on Semantic Web, and how class retrieval is typically done in most existing systems.

2.1 Semantic Web and Linked Data

The design of RDF and OWL by groups of international experts, such as The World Wide Web Consortium (W3C), provided a set of standards for the Semantic Web. Although

RDF is commonly described as a directed, labeled, graph, many researchers prefer to think of it as a set of triples, each consisting of a subject, predicate and object. The subject is the source of an edge, the predicate is its label, and the object is its target. The subject and the predicate are always RDF resources specified by their URIs, but the object can either be a resource or a literal. A literal could be plain (an arbitrary sequence of characters) or it could be adorned with XML Schema datatype information. RDF defines a distinguished property called `rdf:type` to relate a subject to a class. RDF Schema (RDFS) provides some basic semantics for the classes (i.e., objects of `rdf:type` triples) and properties (to be used as a triple's predicate). In particular there are `rdfs:subclassOf` and `rdfs:subpropertyOf`, which allow class and property hierarchies to be defined, and `rdfs:domain` and `rdfs:range` which define the classes of the subjects and objects of triples that use specific properties. The formal semantics of RDFS rules are listed as in Table 2.1, where S indicate the set of original triples. Each rule specifies a kind of inference, and from that we know if such pattern exists in a set of triples, an extra triple can be entailed by this rule. The details and full set of rules can be found in RDF 1.1 Semantics[1].

In comparison with RDFS, OWL adds significant expressiveness for describing the semantics of RDF vocabularies. OWL is based on description logics, a decidable fragment of first-order logic. By using OWL, one could construct various classes by specifying how the instances of this class will use the properties (e.g. the values or the cardinality), or by using intersection, union, or complement of other classes. Properties are more clearly defined in OWL as either `owl:ObjectProperty` or `owl:DatatypeProperty` depending on

[1]RDF 1.1 Semantics W3C Recommendation: `http://www.w3.org/TR/rdf11-mt/`

Table 2.1: Semantics of RDFS Rules

	If S contains:	then S RDFS entails:
rdfs1	any IRI `aaa` in D	`aaa rdf:type rdfs:Datatype .`
rdfs2	`aaa rdfs:domain xxx .` `yyy aaa zzz .`	`yyy rdf:type xxx .`
rdfs3	`aaa rdfs:range xxx .` `yyy aaa zzz .`	`zzz rdf:type xxx .`
rdfs4a	`xxx aaa yyy .`	`xxx rdf:typerdfs:Resource .`
rdfs4b	`xxx aaa yyy .`	`yyy rdf:typerdfs:Resource .`
rdfs5	`xxx rdfs:subPropertyOf yyy .` `yyy rdfs:subPropertyOf zzz .`	`xxx rdfs:subPropertyOf zzz .`
rdfs6	`xxx rdf:type rdf:Property`	`xxx rdfs:subPropertyOf xxx .`
rdfs7	`aaa rdfs:subPropertyOf bbb .` `xxx aaa yyy .`	`xxx bbb yyy .`
rdfs8	`xxx rdf:type rdfs:Class .`	`xxx rdfs:subClassOf rdfs:Resource .`
rdfs9	`xxx rdfs:subClassOf yyy .` `zzz rdf:type xxx .`	`zzz rdf:type yyy .`
rdfs10	`xxx rdf:type rdf:Class`	`xxx rdfs:subClassOf xxx .`
rdfs11	`xxx rdfs:subClassOf yyy .` `yyy rdfs:subClassOf zzz .`	`xxx rdfs:subClassOf zzz .`
rdfs12	`xxx rdf:type` rdfs:ContainerMembershipProperty .	`xxx rdfs:subPropertyOf` `rdfs:member .`
rdfs13	`xxx rdf:type rdfs:Datatype`	`xxx rdfs:subPropertyOf rdfs:Literal .`

whether the object of the property is an instance or a literal value respectively. One could also construct object properties with their inverse properties; or even construct properties with property composition in OWL2[2]. The expressiveness of RDF/OWL allows any person to make any statement on anything, and provides a framework for people to contribute their knowledge.

Among the vocabularies provided by RDFS and OWL, there are some very important ones for the Linked Data: `rdfs:subclassOf`, `owl:equivalentClass` that link the

[2]OWL 2 Web Ontology Language Document Overview: `http://www.w3.org/TR/owl2-overview/`

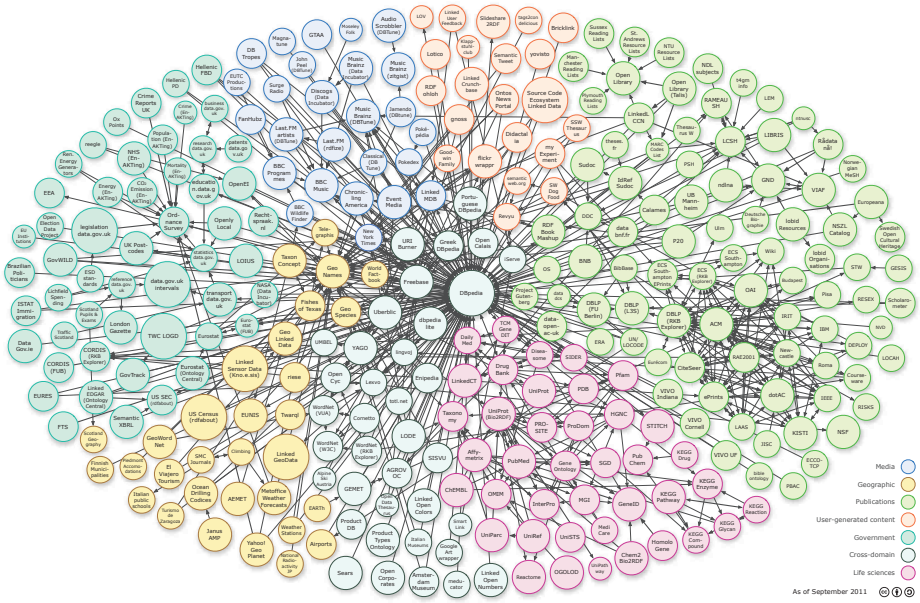


Figure 2.1: Linking Open Data cloud diagram

by Richard Cyganiak and Anja Jentzsch. http://lod-cloud.net/

classes; `rdfs:subclassOf` and `owl:equivalentProperty` that link the properties; and `owl:sameAs` that links the instances. Thanks to these links, people are able to connect and integrate knowledge from distributed sources. Linked data enables a very promising way of sharing and reusing the information and knowledge more efficiently.

> *Linked data describes a method of publishing structured data so that it can be interlinked and become more useful. It builds upon standard Web technologies such as HTTP and URIs, but rather than using them to serve web pages for human readers, it extends them to share information in a way that can be read automatically by computers. This enables data from different sources to be connected and queried.*

Christian Bizer, Tom Heath and Tim Berners-Lee [6]

The Linked Open Data now has tremendous amount of data which is contributed by researchers and organizations that believe the great potential of this framework. Figure 2.1 illustrates the status of the Linked Open Data as in September 2011. The data included in the diagram contains 295 datasets with 31.6 billion triples and more than 500 million links between datasets[3]. These dataset are created for various domains of media, government, life sciences, etc. We have seen more and more people and organizations contributing to the Linked Data. However, on the other side, people, especially the ordinary users, do not get as much benefit of this knowledge base as one would expect. Even expert users are usually very focused on very few sources in the domain they are very familiar with. How to consume the Linked Data is already a very important and attractive research topic in the Semantic Web community. We believe the knowledge gaps are the fundamental problems that prevent people from consuming the Linked Data.

2.2 Visualizing the Semantic Web data

Early researchers used graph representations for browsing Semantic Web data, believing it as a natural choice. But later Karger and schraefel [25] pointed out Big Fat Graphs are not the ideal representation for RDF data because of the problems in terms of the usablity/usefulness . Many recent systems, such as /facet [19], gFacet [18] and BrowseRDF [36], use or extend the faceted browsing idea first presented by Burke et al. [8]. It is

[3]State of the LOD Cloud: `http://lod-cloud.net/state/`

a technique where users can construct a selection query by adding constraints and each new added constraint will update the faceted interface to display further facet options based on the current query selection results. The selection operations often vary in different systems. Most systems support selection on property values, and intersection on selections. For example, BrowseRDF supports existential selection on properties and join selection which is equivalent to property composition, and all these operations on inverse properties. Similarly in gFacet, whose use case is illustrated in Figure 2.2, users can pick a property and specify the property value (datatype or instances with chained facets) to construct a complex query and find matched instances. The interface also has the feature for automatically pinning the positions of node windows to avoid any overlap. For example, if a user add a node as shown in Figure 2.2 (A), it will soon be adjusted to Figure 2.2 (B).

Another type of exploration tools provides summaries of datasets. e.g. Explod [26] provides a summary graph for class and property usage of grouped instances. A simple example illustrated in Figure 2.3 shows how the triples are turned into an Explod class view. The number in the parentheses indicates classes' usage (the size of them), and each group of instances is assigned a block id (shown in square brackets), while the groups are based on common relationships. However such summary information is buried in bracketed labels, making patterns less obvious. When the size of triples grows larger, the graph will also encounter usability issues.

Tag cloud can also be used to summarize the usage and trends of data. Traditional

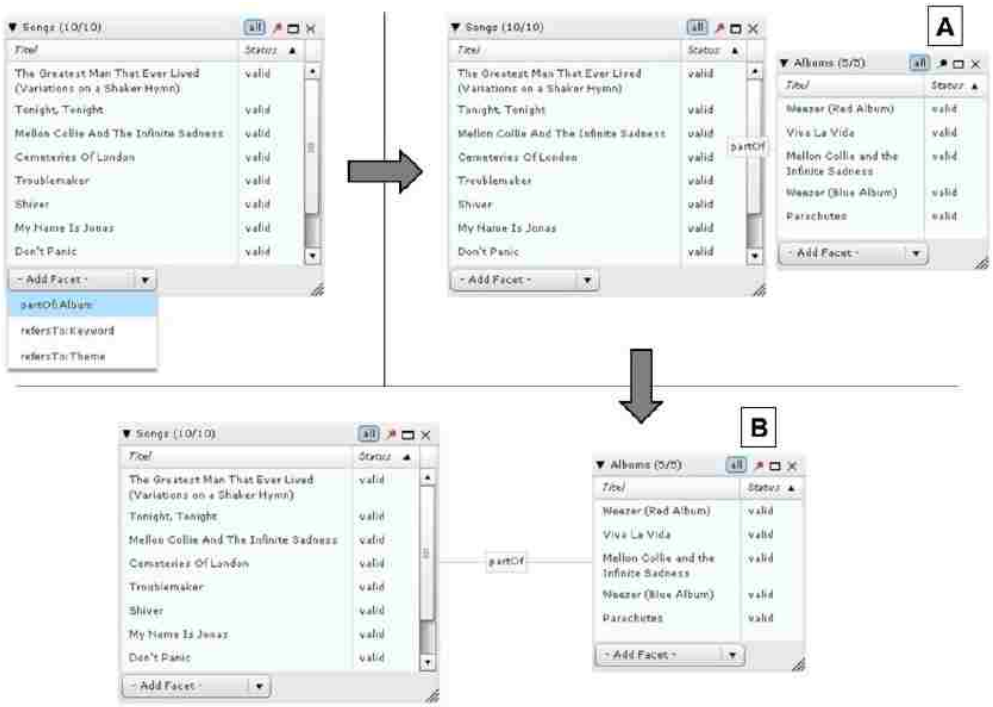


Figure 2.2: The user adds new facets to the graph by selecting them from a menu below the list.

tag cloud interfaces are mostly used for displaying the frequency or popularity of tags in some systems, such as in flickr[4] (illustrated in Figure 2.4) and delicious[5]. Usually tags in these systems are folksonomies assigned to items (like photos or web pages) defined by the users, and the tag clouds visualize the popularity of tags in the dataset.

The TagExplorer [45] application allows for browsing of Flickr photos using the tags. The user can either start browsing using a keyword query or by choosing one of the popular tags. Given the user's query the TagExplorer generates a set of tags related to the query. The tag set is displayed to the user in the tag-cloud paradigm. Additionally, the tag cloud

[4]www.flickr.com
[5]www.delicious.com

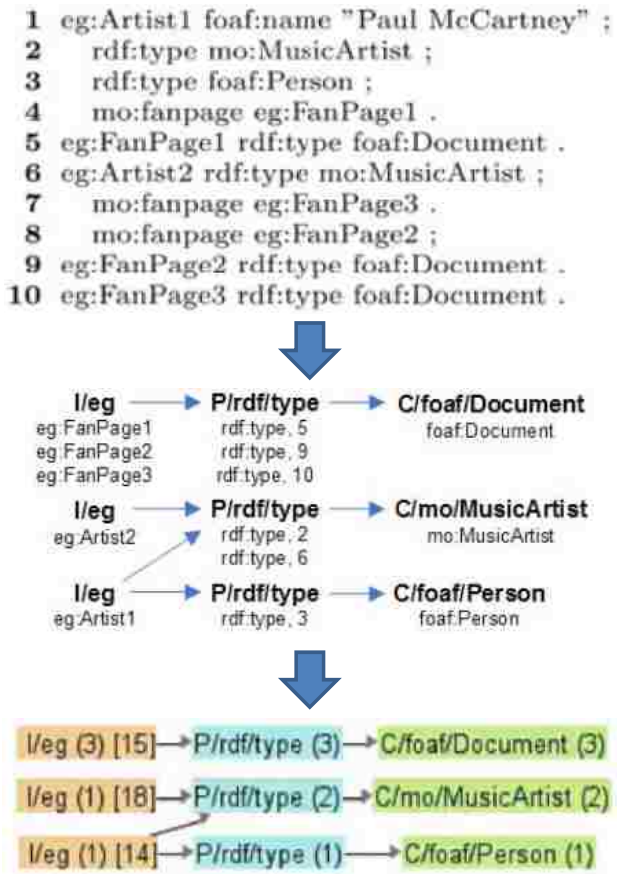


Figure 2.3: The triples are turned into an Explod class view

The size of each class is shown between parentheses, and a unique instance id is shown in square brackets.

is broken up into several parts by grouping together tags that belong to the same syntactic category.

As illustrated in Figure 2.5, the user decides to browse the tag London. In addition to the top photos relevant to the query, the TagExplorer shows a cloud of related tags, such as England, United Kingdom, Southwark, City, Big Ben, London Eye, Thames, party and travel. The tags England, United Kingdom, Southwark, and City are grouped together as

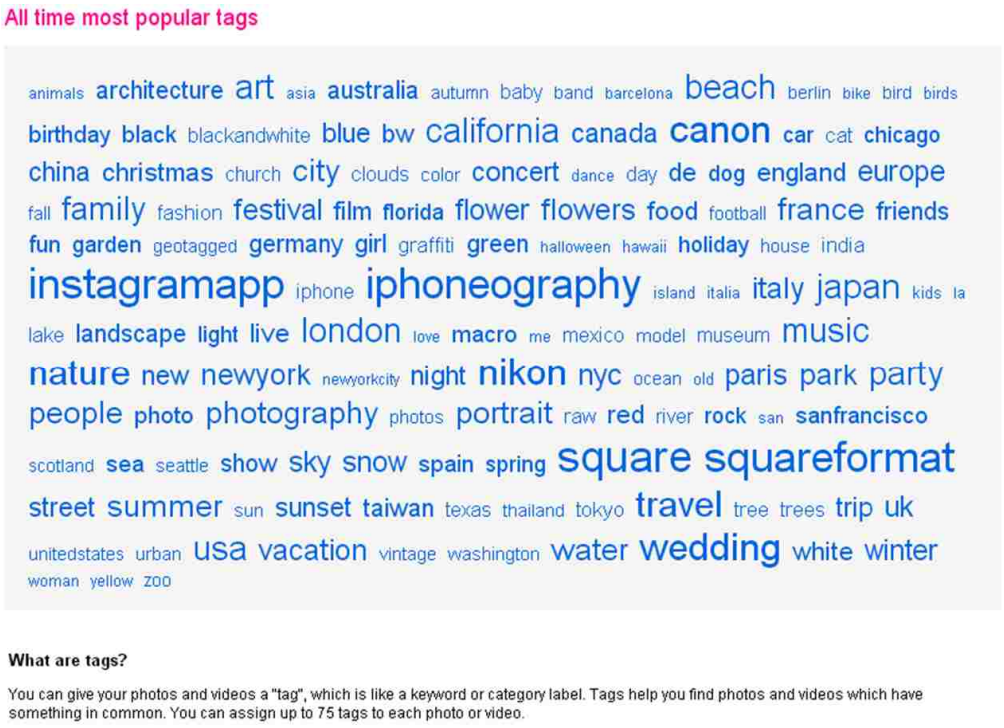


Figure 2.4: Tag cloud of flickr as of May 2014.

locations; the tags Big Ben, London Eye, and Thames are grouped together as subjects; and the tags party and travel are grouped together as activities. This tag interface also has something similar to faceted browsing because the user can refine with additional related tags, and the facets are automatically learned in the preprocessing steps.

2.3 Scalable Repository and Index of the Semantic Web Data

There are many existing systems for storing RDF triples. To compare with general purpose triple stores, Rohloff et al. [42] present a comparison of scalability performance of various

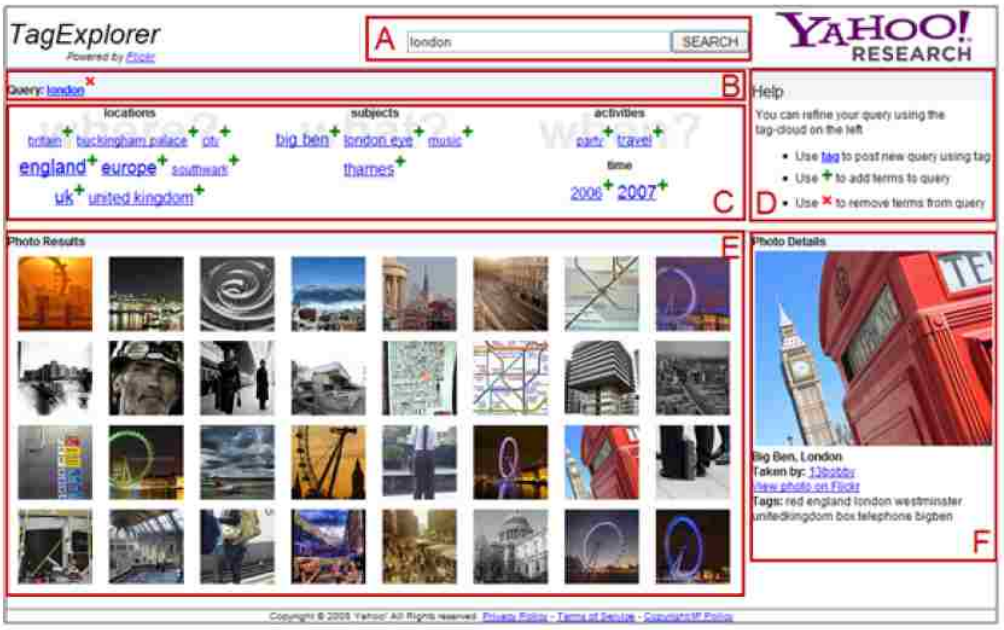


Figure 2.5: TagExplorer showing grouped tags related to the query "london" as well as matched photos in flickr

triple store technologies using the LUBM benchmark [17], and reported that Sesame [7] was the most scalable: It[6] loads around 850M triples in about 12 hours, but it takes more than 5 hours to answer LUBM Query 14, which, similar to the task in the contextual tag cloud systems, requests the instances of a class. Sakr and Al-Naymat [44] survey RDF data stores based on relational databases and classify them into three categories: (1) each triple is stored directly in a three-column table, (2) multiple properties are modeled as n-ary table columns for the same subject, and (3) each property has a binary table. Abadi, et al. [1] explore the trade-off and state the third category is superior to the others on queries assuming a column store is used instead of an RDBMS. However our experiments (in Chapter 4.6) show using an inverted index is much faster for the queries

[6]The experiments were conducted in 2007.

that count instances of intersections of classes/properties. Additionally, we compare our inverted index approach with the state-of-the-art RDF store RDF-3X [34]. The difference in the experiments indicates that a general purpose SPARQL engine is not always the right choice for a Semantic Web system which requires scalable performance on special kinds of queries.

There are also many applications using inverted indices on Semantic Web data. Many of them are Semantic Web search engines. E.g. Sindice [49] and Watson [13] are used to locate Semantic Web documents, while other search engines such as Falcons [9], SIREn [14], and SemSearch [27] are used for locating semantic entities, and thus whether to index labels, URLs, literal values or other metadata might differ between them. Occasionally, question answering systems [50, 51] use inverted indices to help identify entities from natural language inputs, which in some sense is also an entity search engine. Despite the categorization, all the above systems index with keywords because the intended usage is to locate relevant resources based on natural language queries posed by users. Our contextual tag cloud system is very different because the "terms" in our index are no longer keywords but ontological tags. As a result, the index in the tag cloud system is compatible with entailments sanctioned by the ontologies in the data. This is also why we propose our preprocessing steps prior to indexing, which we have not seen in other works.

2.4 Natural Language Words vs. Ontological Classes

Banek et al. [3] stress the importance of WSD in ontology alignment and they recommend WSD as the primary step of ontology integration. They propose their approach of disambiguation on class names by using the names of the related classes in RDFS axioms as context. However, they did not consider the names of properties or names of compound words, and only used a limited subset of axioms in the document. They only reported the experimental results on accuracy of top senses, however, if this WSD component is to be used as part of the integration process, providing the scores of all the possible senses would be more useful for later processes.

WSD techniques and many ideas we use in Chapter 7 are inspired by many previous traditional WSD works, especially the ones that are knowledge based and exploit information from a given lexicon. One category of these approaches relies on the definition of senses. Lesk [28] first invented the gloss overlap algorithm that calculates the overlap between the definitions of two target words. Banerjee and Pedersen [4] developed the extended gloss overlap method by also considering the glosses of other related senses. Another category of approaches uses semantic similarity measures. For example, Resnik [41] and Jiang and Conrath [24] used the notion of information content from corpus statistical information and calculated the similarity distance between senses. A third category of approaches explores the graph structures and tries to find a lexical chain between target words. Hirst and St-Onge [20] introduced the first computational model of lexical chains and counted the number of times the chain changes direction. A comprehensive review of

WSD can be found in [33].

However, most of the previous work define their own scores, based on their own ad hoc heuristics. If we can integrate these heuristics into a more theoretical framework, we may get the combined advantages from different approaches. Also instead of ad hoc scores, probability distributions have clear meanings, and should be easy to reuse. Based on this motivation, we propose our probabilistic approach for WSD in Semantic Web documents in Chapter 7.

A potential use case of the WSD component is in the resource retrieval problem, which we define as: find the best matched resources in the dataset based on the input keywords. Resource retrieval is essential in many systems across various scenarios. For example, Sindice [49] is a state-of-the-art Semantic Web search engine that has an inverted index over resources crawled on the Semantic Web, and allows users to retrieve documents with statements about particular resources. In order to improve instance retrieval, Sindice makes use of inverse functional property values (e.g. email address) as texts that are indexed to retrieve the instances. However, if there are many similar matching instances, it is up to the user to determine if the desired match is not any single instance, but instead a class the represents a collection of instances. We also find that although various strategies are applied in different (controlled) natural language QA systems [5][31][48], all of them implement and integrate some kind of resource retrieval components (to retrieve classes and properties). Most of these systems only use the straightforward strategy for resource retrieval, i.e. exact string match on the *rdfs:label* values, or in the case of Sindice,

on any inverse functional property values.

However, a few QA systems enhance their retrieval component by expanding queries with WordNet, a large lexical database of English. For example, Aqualog [31] uses synonyms for class matching, and Tran et al. [48] mention that they extract synonyms, hyponyms (subclasses) and hypernyms (superclasses) for all the terms. It is not clear whether they only use direct hyponyms and hypernyms, but the drawbacks of matching to indirect hyponyms and hypernyms are easy to see: (1) the need to search for a large number of variations in the query term significantly increases the query time; and (2) when less similar concepts are introduced, precision suffers.

Even when lexicon-based matching sticks to straight synonyms, there are problems. First, in some domain specific KBs, people might use query terms that are not in the lexicon. Second, a synonym might have other meanings as well, and retrieving all occurrences of it can reduce precision. Finally, the ontology creators and KB users may sometimes use words that are not synonyms to refer to the same concept [15] under different circumstances. For example, people might refer to an entity or event (the referent) by one of its features or attributes (the metonym). Some of these problems can be solved if we are able to perform WSD on the labels of resources. However there are still problems that WSD will not help. For example the creator of an academic ontology may use "Person" to name the concept of people at an academic institution; but this concept only consists of "Professors" and "Students". Meanwhile in many cases a partial match is useful. For the keyword query "professor", the *Person* class from the academic ontology may be suitable

as a partial match, even though it is a super class. This is especially true if other constraints in the query restrict results to someone teaching a course or advising a student. Thus alternative approaches for resource retrieval need to be investigated.

Chapter 3

The Contextual Tag Cloud System: Concepts, Features and Use Cases

In this chapter, I will introduce the basic concepts of the contextual system, the functionalities it provides, and use cases of it.

3.1 Concepts of the Contextual Tag Clouds

The idea of presenting data via a tag cloud has been widely used for many systems, particularly the Web 2.0 systems where the contents are mostly from the users and a high-level summary of the contents (or usually folksonomies that categorize or highlight the contents) is quite convenient. An analogous situation exists for the Linked Open Data

in that everyone can contribute their knowledge, therefore we find the tag cloud can be a good fit to the Semantic Web KBs as well. In Table 3.1 we compare our adapted idea to traditional Web 2.0 tag cloud systems. An instance is like a web page document, but is "tagged" with formal ontological classes, as opposed to folksonomies. Tags, are then another name for the categories of instances. We will define tags and contexts in this section first.

Table 3.1: Comparison between traditional Web 2.0 tag cloud and our contextual tag cloud

	Web 2.0	Our System
What is a tag?	A folksonomy defined by users	An ontological term(class or property)
What defines the tag size in the tag cloud?	The count of documents marked by the tag	
What is a document marked by a tag?	A web page tagged by users	An instance associated with the ontological term in the dataset
What happens when a tag is clicked?	Show a list of documents of the tag	Show another contextual tag cloud with this tag added to context

Formally, consider a KB defined by $\mathcal{S}$, a set of RDF statements. Each statement $s \in \mathcal{S}$ can be represented as a triple of subject, predicate and object, i.e. $s = \langle \text{sub}, \text{pre}, \text{obj} \rangle$. In addition to these **explicit triples**, an entailment regime R defines what kind of entailment rules will be applied to the triples. By applying all the specified entailment rules, we can get $\mathcal{S}^{\mathcal{R}}$, a closure of $\mathcal{S}$ which completes $\mathcal{S}$ with the entailed statements. To extend the expressiveness, we include various ways to assign a tag to an instance i:

1. **Class** C, if $\exists \langle i, \texttt{rdf:type}, C \rangle \in S^R$, i.e. by entailment, i is an instance of C.

2. **Property** p, if $\exists\langle i, p, j\rangle \in S^R$, i.e. the instance appears as the subject in one or more triples involving p. Note it does not matter whether j is also an instance or j is a literal value. Thus both `owl:ObjectProperty` and `owl:DatatypeProperty` are valid.

3. **Inverse Property** $p-$, if $\exists\langle j, p, i\rangle \in S^R$, i.e. if the instance appears as the object in one or more triples involving p. Here the property p must be an `owl:ObjectProperty`.

In addition, we find it useful in many scenarios to introduce the **Negation Tag** $\sim t$. While a tag represents that an instance is described by a particular class or property, we use a negated tag to indicate that such a description is missing. This can be useful for inspecting what portions of the data are missing important properties, e.g., how many politicians are missing a political party. We considered three possible semantics for the negated tags:

1. classical negation: Instances have the tag only if the negation of the corresponding concept is logically entailed;

2. negation-as-failure: Instances have this tag if the system fails to infer the regular tag, i.e. it does not have the tag in $\mathcal{S}^{\mathcal{R}}$; and

3. explicit negation: Instances have this tag if they do not explicitly have the positive tag in $\mathcal{S}$.

Classic negation is usually used in the communities related to logic reasoning, such as in First Order Logic. The negation in this case states what must be false given the

information from the dataset. On the other extreme side, the explicit negation only tries to retrieve the statement and return false when it fails to retrieve the exact form of the statement. We soon decide the explicit negation is not an option because it could lead to confusing scenarios where an instance has a regular inferred tag and a corresponding explicit negation tag. Negation-as-failure (NAF) is another well recognized way of defining negation in logic programming [30], and is part of prolog, the widely used, general-purpose logic programming language. Under prolog's NAF inference, what cannot be proved is considered as false statements. This is known as the Closed World Assumption [40] and works well for datasets with complete knowledge. However it is definitely different from many Semantic Web datasets, particularly the LOD, where statements can always be appended to existing datasets, and this is why classic negation is more proper for inference of knowledge in Semantic Web. However, there is another view of NAF, which conveys totally different semantics for negation. Michael Gelfond [16] showed that it is also possible to interpret the negation of something (p) literally as "p can not be shown", "p is not known" or "" "p is not believed", as in autoepistemic logic. The autoepistemic logic is particularly useful for the representation and reasoning of knowledge about knowledge. While propositional logic can only express facts, autoepistemic logic can express knowledge and lack of knowledge about facts. From this view, NAF best fits our requirement and we argue that this is the correct semantics for a system where what is not said is sometimes as important as what is said. Note that the negation tags are virtually assigned to instances, since they can be easily derived by whether their regular tags are assigned.

Let $\mathcal{I}$ be the set of all the instances, $\mathcal{T}$ be the set of all possible regular tags assigned to instances in the dataset, $\mathcal{V}$ be the set of all negation tags, i.e. $V = \{\sim t | t \in \mathcal{T}\}$, and $\mathcal{A} = \mathcal{T} \cup \mathcal{V}$. Given R, we define a function $\text{Tags}_R : \mathcal{I} \to 2^{\mathcal{T}}$ that maps the given instance to all the regular tags assigned to it under R-inference closure. i.e.

$$\text{Tags}_R(i) = \{C | \exists \langle i, \texttt{rdf:type}, C \rangle \in S^R\} \bigcup \{p | \exists \langle i, p, j \rangle \in S^R\} \bigcup \{p - | \exists \langle j, p, i \rangle \in S^R\} \tag{3.1}$$

Note under monotonic logic [1], $R_1 \subseteq R_2 \Rightarrow \text{Tags}_{R_1}(i) \subseteq \text{Tags}_{R_2}(i)$, i.e. if more entailment rules are applied, we will have at least the same set of tags assigned to an instance, if not any more.

The function $\text{Inst}_R : 2^{\mathcal{T}} \times 2^{\mathcal{V}} \to 2^{\mathcal{I}}$ maps the given set of regular tags T and the given set of negation tags V to the set of all instances assigned or virtually assigned with them.

$$\text{Inst}_R(T, V) = \{i | T \subseteq \text{Tags}_R(i) \wedge \neg \exists t (t \in \text{Tags}_R(i) \wedge \sim t \in V)\} \tag{3.2}$$

Since T and V and can be distinguished syntactically, as a short-hand, we will use the following definition:

$$\text{Inst}_R(A) = \text{Inst}_R(T, V) \text{ where } T = A \cap \mathcal{T} \wedge V = A - T \tag{3.3}$$

[1] A logic is monotonic if every thing that is entailed by a KB is entailed by a superset of the KB.

For convenience, we define the frequency of a set of tags $A \subseteq \mathcal{A}$ as

$$f_R(A) = |\text{Inst}_R(A)| \tag{3.4}$$

When the user specifies a *context* $A \subseteq \mathcal{A}$, he actually constructs a class expression in description logic (except for the negation tags), using a very simple syntax. For example, the context $\{\texttt{eg} : \texttt{Town}, \texttt{eg} : \texttt{mayor}\}$ is the same as $\texttt{eg} : \texttt{Town} \sqcap \exists \texttt{eg} : \texttt{mayor}.\top$. Then the context defines a narrowed scope of instances to be further investigated and the next tag cloud is presented within this dynamically specified scope of instances.

Note that for all the definitions above, the entailment regime R is also a variable to the functions, since we can choose to include subsumption inference and domain/range inference. To investigate the impact of different R, we can generalize various entailment rules into tag subsumptions. Tag t_1 is a sub tag of tag t_2 if and only if the entailment regime requires $\text{Inst}_R(\{t_1\}) \subseteq \text{Inst}_R(\{t_2\})$. For example, this sub tag relation includes RDF subclasses/subproperties plus the ones entailed by the domain/range axioms: If $\langle p$, `rdfs:domain`, $C\rangle$ and $\langle p$, `rdfs:range`, $D\rangle$, then p is a sub tag of C and $p-$ is a sub tag of D. We use the notation $a_1 \sqsupseteq_R a_2$ or $a_1 \sqsubseteq_R a_2$ for $a_1, a_2 \in \mathcal{A}$ to denote that a_1 is a super/sub tag of a_2 under entailment regime R respectively.

In our implementation, we have two specific sets of rules: R_{Sub} for entailment of class/property subsumption relations(including equivalence relations, which can be treated as two-way subsumption relations) (`rdfs5`, `rdfs7`, `rdfs9` and `rdfs11` in Table 2.1); and R_{DR} for property domain/range entailment (`rdfs2`, `rdfs3` in Table 2.1). We also

support the combination of these two sets, leading to four distinct entailment regimes $\mathcal{R} = \{\emptyset, R_{Sub}, R_{DR}, R_{Sub} \cup R_{DR}\}$.

3.2 System Feature and Use Cases

We introduce the features and use cases of our system using the dataset provided for the Billion Triple Challenge 2012, which is a significant subset of the Linked Open Data. This complex dataset contains 1.4 billion triples, from which we extract 198.6M unique instances, and assign more than 380K tags to these instances.

The initial tag cloud has context $A = \emptyset$ or semantically $A =$`owl:Thing`, and the tags in the cloud reflect the absolute sizes of instances related to each tag. We put classes and properties into two separate views, so that users will not treat a property called "author" (which may have domain Publication) as a class name by mistake. To emphasize that difference, we also add an icon with "C" or "P" in front of each tag. If a tag is clicked, it will be added to the current context, and then a new tag cloud will be shown for the updated context. A user can add/remove any tags to/from the context, and explore any custom defined collection of instances. A user can also switch to Instance View to get a list of instances that match the context and investigate their triples.

A user can also change the inference regime, which by default is R_{Sub}, the subsumption inference. Usually we can expect tags to become larger when more inference is introduced. If R entails that a set of tags are equivalent, we choose a canonical tag to group them under. We display a $\equiv$ after the canonical tag to indicate this; clicking it will display the

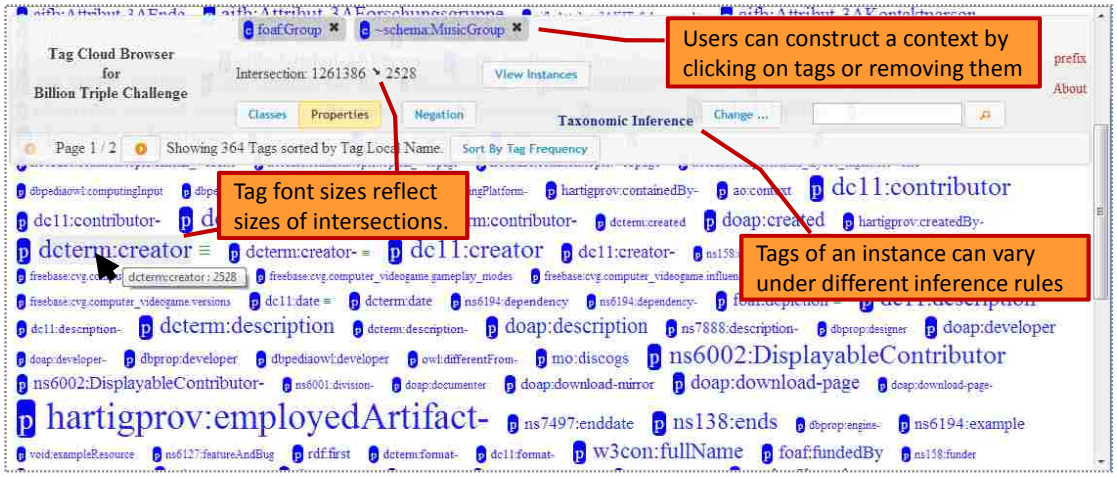


Figure 3.1: Property Tag Cloud with contexts `foaf:Group` and ~`schema:MusicGroup`.

equivalent tags. Also for any tag cloud, we can turn on the negation mode, and then the tag sizes indicate how many instances do not have this tag under the current context and inference level. A negation tag can be also added to the context, which mathematically means the relative complement. For example in Fig. 3.1, the property tag cloud with context `foaf:Group` and ~`schema:MusicGroup` shows us the common property usages of instances of `foaf:Group` that are not instances of `schema:MusicGroup`.

Given the context $A_0 \in \mathcal{A}$ and the entailment regime R, the contextual tag cloud presents a list of regular tags $[t_1, \ldots, t_n]$ with various font sizes $[fs_1, \ldots, fs_n]$ that reflects the instance sizes of types $[f_R(A_0 \cup \{t_1\}, \ldots, A_0 \cup \{t_n\}]$. The font sizes for a tag t_i in the tag cloud is

$$fs_i = (FS_{\text{MAX}} - FS_{\text{MIN}}) \frac{\log f_R(A_0 \cup \{t_i\})}{\log f_R(\emptyset)} + FS_{\text{MIN}}$$

where the max and min font sizes are denoted as FS_{MAX} and FS_{MIN}. In practice we use $FS_{\text{MAX}} = 48$ px and $FS_{\text{MIN}} = 12$ px. The max font FS_{MAX} indicates $f_R(\emptyset)$, i.e. the total number of instances, which provides a universal scale to the font sizes over all contextual tag clouds. Alternatively, in some cases, we can also make the size of the current context, i.e. $f_R(A_0)$ map to the max font, which makes the sizes on the current page larger especially if the context is a very specific one. We use log functions[2] on the f_R so that the tag cloud shows differences of tags in orders of magnitude. The sizes of tags in the LOD usually vary from a wide range, so if we do not use log functions the tag cloud page will be dominated by a small number of largest tags.

With the BTC dataset, a challenging problem for UI design is how we can show so many tags in the tag cloud. A straightforward idea is to show tag clouds in pages. To help users locate specific tags in the tag cloud, we initially sort the tags by their local names alphabetically. When the system receives a request (context T and inference R), it will process tags in the same alphabetic order, and then stream out whatever is available for the requested page. If the user chooses to browse tags alphabetically, then the streaming of results is generally able to stay ahead of the user by pre-fetching results for tags on subsequent pages. Instead of browsing, a user can also search for tags by keywords. We index the local name, `rdfs:label` and `rdfs:comment` (if it exists) for each tag to support such keyword search. The retrieved tags will then be shown in the tag cloud sorted by their relevance to the keyword with their frequencies under the current context and inference regime. In addition, we provide sorting by tag frequency as another option, so that users

[2]The base of log does not matter because of the division.

can easily see the most popular tags under the current context and inference. However, we have to wait until all the frequencies are computed to enable this sort option. For some contexts, it can take a few minutes for the overall computation of thousands of pages of results. We show a progress bar of the computation and the estimated time left; and before the frequency sort is enabled, users can still browse by alphabetical order or search with keywords.

We believe our system can be used for multiple purposes. Here we shall briefly present four scenarios that describe how a user can explore the BTC dataset.

Choose the right terms for SPARQL. A user wants to build a SPARQL query on lakes, but does not know what classes about lakes are available. The user starts with a keyword search "lake", and is presented with a tag cloud as illustrated in Figure 3.2. The tag cloud contains all tags that match the keyword, and is sorted by the relevance scores to the query. Then the user finds that `dbpediaowl:Lake` is the largest tag, which indicates that it contains the most instances. After picking this class, the user wonders what property to use for querying the area of a lake. Then by searching again with keyword "area", the user is presented with the contextual tag cloud with keyword-matched tags whose sizes reflect the intersection of the instances of `dbpediaowl:Lake` and the tags, as illustrated in Figure 3.3. It turns out `dbpediaowl:areaTotal` is the best choice of the property.

Learn interesting facts. A casual user tries a keyword search on "Manhattan". The

Figure 3.2: The tag cloud search result page for a user's query "lake".

Figure 3.3: The tag cloud search result page for a user's query "area" with context `dbpediaowl:Lake`.

tag cloud is shown as Figure 3.4. There are classes of parks, streets, etc. located in Manhattan. However, it also has the slightly unexpected class `yago:ManhattanProjectPeople`; with her interest piqued, the user adds this to the context to explore in more detail. In

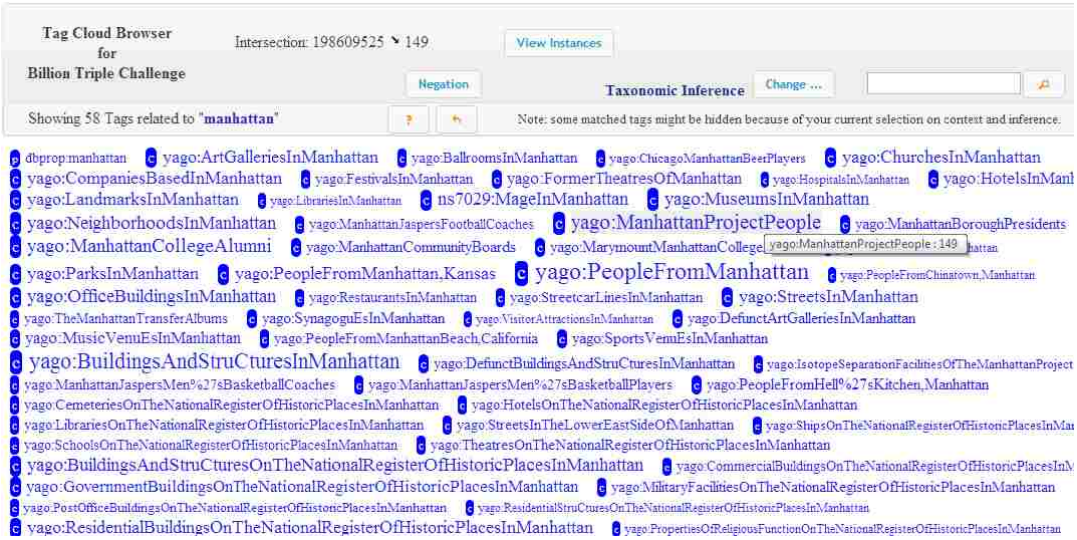


Figure 3.4: The tag cloud search result page for a user's query "Manhattan".

the resulting tag cloud, the user finds various categories for such people, and then searches again for "scientist" to find out which scientists are involved. Then surprisingly there is a tag `freebase:computer.computer_scientist`. The user is intrigued, because she did not know that any computer scientists were involved in the effort to build the first atomic bomb. By adding that tag and switching to the Instance View, she finally learns that this scientist is John von Neumann.

Detect Co-reference Mistakes. Sometimes when two tags have a small unexpected intersection, it is due to an error, rather than an interesting fact. For example, the tag `yago:BritishComputerScientist` has one common instance with a very small tag `dbpediaowl:MusicalArtist`. By adding this tag and looking into the triple details in the Instance View (as shown in Figure 3.5), we can see the two `dbpediaowl:abstract` values clearly refer to two different people who have the same name but different birth

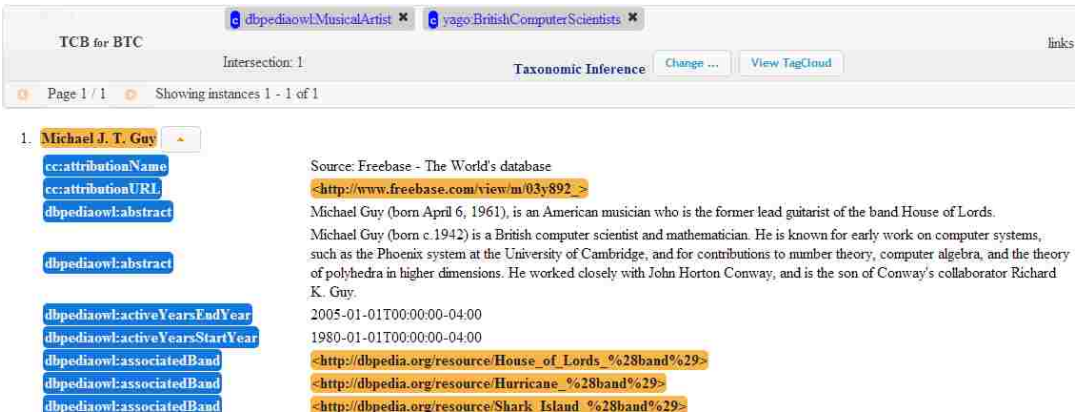


Figure 3.5: The instance view reveals the co-reference mistake.

We find different birth year and occupations in the only instance for context {`yago:BritishComputerScientist`, `dbpediaowl:MusicalArtist`}.

years (1941 and 1962) and occupations (American musician and British scientist), and are mistaken as the same person in the dataset.

Examine ontological errors. Under inference R_{DR}, a user finds that `foaf:Person` appears in the tag cloud of context `dbpediaowl:Software`, implying that some people are software, or vice versa! If the user changes the inference to $R_{\emptyset}$ or R_{Sub}, this error will disappear. So that means there must be something wrong with the domain-range inference. We know that if there is a property claimed as having `foaf:Person` as it domain, then any instance using this property will be classified as the instance of this class. With this hypothesis in mind, the user adds both `foaf:Person` and `dbpediaowl:Software` to the context, selects the property view and inference R_{DR}, and sorts the properties by frequency. Then the top tag is `foaf:homepage`, which has all the instances in the current context (by hovering the mouse over the tag, we can see the frequency of this tag). This is very suspicious, and by clicking on the "P" icon before `foaf:homepage`, the user can see

Figure 3.6: Examining ontological errors.

The first property `foaf:homepage` in the property view implies class `foaf:Person`.

(in Figure 3.6) that `foaf:Person` is an inferred super tag of this tag, and that causes the error. By checking the raw ontology we find that although the domain of `foaf:homepage` is `owl:Thing` in the `foaf` schema, two other sources in the BTC dataset make the claim that the domain is `foaf:Person` and `foaf:Agent` respectively.

To visualize the patterns, our contextual tag cloud system combines the idea of faceted browsing with the traditional tag cloud visualization technique. Our current system is similar to faceted browsing systems in the sense that our contextual tag cloud idea also has the feature that the new tag cloud is generated based on the previous selected tags. In comparison with the faceted browsing systems we introduced in Section 2.2, our contextual tag cloud system has less expressiveness than BrowseRDF since it supports only existential, inverse existential and intersection, and does not support any operations on the values of properties. However on the other hand, it is potentially more scalable. To the best our knowledge, there are no scalability experiments for such systems, however the scalability issue can be questionable due to the fact that a facet selection is a query to the dataset and intersection is a typically a costly operation in SQL. While none of the 3 papers stressed the scalability issue, our system particularly focuses on limited expressiveness, i.e. the

existence of properties and classes, and by optimizing the infrastructure, provides a more scalable performance over large datasets. Meanwhile we want to emphasize, except for the same flavor of adding constrains on the fly and the comparison of expressiveness and scalability, our system has different purposes compared to faceted browsing systems. Our system aims at revealing the patterns of co-occurrence of ontological terms and familiarizing the users with the KB while the faceted browsing systems mostly help users find specific instances that meets some criteria. In comparison with the traditional tag cloud systems, while tags in them are folksonomies, tags in our system are ontological terms, or property values, and thus have precise semantics enabling inference on the tags. Also most tag cloud systems only provide a top level tag cloud, and our system provides tag clouds of dynamically defined subsets of instances. TagExplorer is similar to our contextual tag cloud idea in the sense that faceted browsing and tag cloud browsing are combined. However, they classifies folksonomies in the preprocessing and group them by facets in the resulting tag cloud, while our tags are ontological terms and the facets are used as filters to generate dynamic tag clouds based on the filter.

Chapter 4

Scaling the Contextual Tag Cloud System

Since our goal is to display the frequency of all tags given a context A, our main challenge is to compute $f_R(\{t\} \cup A)$ for $\forall t \in \mathcal{T}$ efficiently. There are two ways to approach this problem: (1) ensure efficient calculation of $f_R(A)$ for any A; and (2) prune unnecessary calls of $f_R(\{t\} \cup A)$. To achieve this, we need to correctly structure the repository and develop an efficient preprocessing step. In this section we will first solve these problems for the situation where there is only a single entailment regime R. Then we will discuss how to "infer" relations between tags and instances, and how to determine co-occurrence between tags under tag inference. At the end we will provide some experimental results to verify our design choices.

4.1 Preprocessing

Our preprocessing is shown in Figure 4.1, where the dashed boxes are input or intermediate data and the solid ones are data results for the online system, and the detailed steps are as follows.

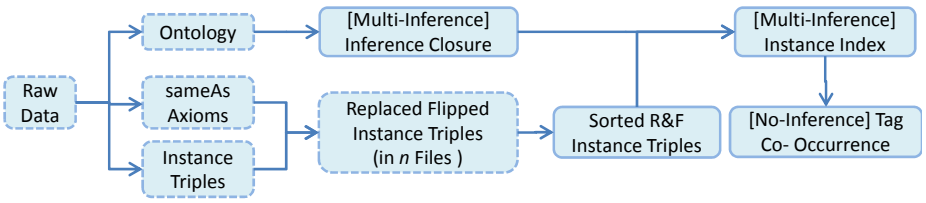


Figure 4.1: Preprocessing for the tag cloud system

1. **Split the Triples.** The raw triple files are parsed and split into three triple files (one triple per line): the ontology file which includes specific properties (e.g., `rdfs:subClassOf`) or classes (e.g., `owl:Class`), the `owl:sameAs` (instance equivalence statements) file, and the file of remaining instance triples. It is not a trivial task in order to exactly find the ontological axioms (and nothing else) because ontology triples that are part of complex expressions don't necessarily contain `owl:` or `rdfs:` namespaces. However the simple approach is good enough for our system since we do not apply complex logic inference. Note in different scenarios, this step can be simplified or complicated. This step can be skipped if the ontology and sameAs files are provided separately. However, if any possible sub property of `owl:sameAs` under the given entailment might exists, the extraction of sameAs axioms should be postponed after the closure of the ontological axioms (i.e. the next step) has been

computed.

2. **Inference Closure.** The ontology is processed into a closure set of sub-tag axioms for the given entailment regime (or regimes). In practice, for each entailment regime, we have a pair of maps for sub tag relations and super tag relations. Then we go through the ontology and for each tag subsumption statement we update the maps of the relevant entailment regime as follows. If the current ontological statement is equivalent to $t_1 \sqsubseteq_R t_2$, we update the sub tag map by adding all the existing sub tags of t_1 to the sub-tag-relation sets of all the existing super tags of t_2, and update the super tag map by adding all the existing super tags of t_2 to the super-tag-relation sets of all the existing sub tags of t_1. Also at the end of the process, we trim the maps of $R_{Sub} \cup R_{DR}$ by removing what is already in the maps of R_{Sub} or R_{DR}, and the trimmed maps will only contain the tag subsumption relations that can only be inferred by applying both R_{Sub} and R_{DR}. By removing the redundant information in the last map, we save 87.7% space. When we want to retrieve the content of $R_{Sub} \cup R_{DR}$, we can always reconstruct it by union the three maps on the fly.

As the result of this process, the closure is then responsible for two functions: $\text{sub}_R(a)$ and $\text{super}_R(a)$ which respectively return the sets of sub/super tags of tag $a \in \mathcal{A}$ under inference R. Notably, although the functions can take either a regular or a negation tag as input, in implementation, we only need to compute the closure for class tags

and non-inverse properties, since for an inverse property $p-$,

$$\text{sub}_R(p-) = \{p' - | p' \in \text{sub}_R(p)\} \tag{4.1}$$

$$\text{super}_R(p-) = \{p' - | p' \in \text{super}_R(p)\} \tag{4.2}$$

Also the results for the negation tags can be computed because $t_1 \sqsubseteq_R t_2 \Leftrightarrow \sim t_1 \sqsupseteq_R \sim t_2$. Given $t \in \mathcal{T}$,

$$\text{sub}_R(\sim t) = \{\sim t' | t' \in \text{super}_R(t)\} \tag{4.3}$$

$$\text{super}_R(\sim t) = \{\sim t' | t' \in \text{sub}_R(t)\} \tag{4.4}$$

3. **Replace, Flip, and Split the Instance Triples.** We use the well-known union-find algorithm [11] to compute the closure for `owl:sameAs` statements, and pick a canonical id for each `owl:sameAs` cluster. In this process, for each statement $\langle a, \texttt{owl:sameAs}, b\rangle$, we find the sets containing a and b, and union them together. Then for the instance triples, we replace each instance with its `owl:sameAs` canonical id (if any). Note that at this step we merge triples of all ids in the same set, and after this integration, there is no way to correct any results after this step if we later find any false sameAs statement unless we restart from this step. If the object of the triple is also an instance, we add an additional flipped triple to the intermediate file, i.e., if the triple is $\langle i, p, j\rangle$, the flipped one is $\langle j, p-, i\rangle$. By this means, we can find all the regular tags (particularly inverse property tags) of an instance i by simply

looking at the triples with i as a subject. Note by duplicating the object property statements, the output can have up to twice as the original triple size. In order to index an instance, we need to first group all of its triples together. To do this, we first output the triples into n files based on the hashcode of their subjects, so that we keep the information of an instance in the same file while making each file relatively small.

4. **Sort the n Triple Files.** We use merge sort on each "replaced and flipped" file generated from the last step, so that triples with the same subject instance are clustered together. Note that by splitting the triples into n files, we gain benefits from two sides: (1) sorting each file becomes faster (and since we only need to group triples with the same subject, we do not need to merge the sorted files); (2) we can sort in parallel (either with multiple machines or with multiple threads). We use these sorted files together with the given inference closure to build an inverted index of the instances.

5. **Index the Sorted Files.** The inverted index is built with tags as indexing terms and each tag has a sorted posting list of instances with that tag. This means given a "type" defined by a set of tags we can quickly find all the instances by doing an intersection over the posting lists. Also, since we use negation as failure, we do not need to index negation tags; their size can be calculated from its complementary tag. i.e. $f_R(\{\sim t\} \cup A) = f_R(A) - f_R(\{t\} \cup A)$. Given a type defined by context $A \in \mathcal{A}$, which can be represented as $\{t_1, t_2, \ldots, t_n, \sim s_1, \sim s_2, \ldots, \sim s_m\}$, the instances

defined by this context can be retrieved by a boolean IR query:

$$t_1 \texttt{ AND } t_2 \texttt{ AND } \ldots \texttt{ AND } t_n \texttt{ AND NOT } s_1 \texttt{ AND NOT } s_2 \texttt{ AND NOT } \ldots \texttt{ AND NOT } s_m$$

At the time of indexing each instance, we materialize all the tags that are entailed based on our previously computed entailment closure. Note that for different entailment regimes, we have different set of posting lists, which increases the disk space. However, we will justify this choice in Section 4.3. Also note that the tags we use here in the index are in fact integer ids instead of their original URIs, and we will introduce the details about the tag id mappings later in this section. Meanwhile we add other fields to facilitate other features in our tag cloud system: (1) labels of instances, to display human-readable instance names in the instance view; (2) sameAs sets, to allow users to inspect all the ids in the sameAs set of an instance; and (3) file pointers to the raw file, to locate the file block where the system is able to load the triples starting with the given instance as its subject.

6. **Compute Co-occurrence Matrix.** To help prune unnecessary tags when computing the conditional distribution of tags under any given context T, we precompute the **Co-occurrence Matrix** for all the tags. Define M_R as a $|\mathcal{T}| \times |\mathcal{T}|$ symmetric boolean matrix, where $M_R(x, y)$ denotes whether tags t_x and t_y co-occur, i.e. $M_R(x, y) = (f_R(\{t_x, t_y\}) > 0)$. We will discuss different approaches for computing this matrix next, then introduce the pruning benefit from this matrix in Section 4.2, and later discuss how to efficiently compute this matrix for different entailment regimes in Section 4.3.

There are three ways to generate the Co-occurrence Matrix M_R. We can roughly estimate the execution time of each method from how much index access (the functions Tags_R, f_R, and Inst_R) is needed. Assume on average a tag has d instances and an instance has e tags. The cost of $\text{Inst}_R(\{t_x, t_y\})$ (or $f_R(\{t_x, t_y\})$) is estimated as $c_1 d$, because the intersection needs to simultaneously walk through both sorted posting lists. The cost of $\text{Tags}_R(i)$ is estimated as $c_2 e$. Here, c_1, c_2 are constants given the dataset and the environment.

1. **Traverse all the instances.** For each instance $i \in \mathcal{I}$, we get all of its tags $\text{Tags}_R(i)$, and then for each pair of tags $(t_x, t_y) \in \text{Tags}_R(i) \times \text{Tags}_R(i)$, set $M_R(x, y)$. This method has $|\mathcal{I}|$ iterations and takes $|\mathcal{I}| c_2 e$.

2. **Traverse pairs of tags.** For any pair of tags $(t_a, t_b) \in \mathcal{T} \times \mathcal{T}$, if $f_R(\{t_a, t_b\}) > 0$, set $M_R(x, y)$. This method has $|\mathcal{T}|^2/2$ iterations and takes $c_1 d |\mathcal{T}|^2/2$.

3. **Traverse tag instances.** For each tag $t_x \in \mathcal{T}$, we get all of its instances $\text{Inst}_R(\{t_x\})$, and then set occurrences for all tags in them. For $i \in \text{Inst}_R(t_x)$, for any tag $t_y \in \text{Tags}_R(i)$, set $M_R(x, y)$. This method has $d|\mathcal{T}|$ iterations and takes $c_2 e d |\mathcal{T}|$.

There is one problem with the estimations above: we ignored the cost of setting M. The first approach can repeatedly set the same cell. However, for a large scale dataset, the full matrix may not fit in memory, and thus updating random cells becomes more costly due to the lack of disk locality. In contrast, for the second and the third approach, they both only need to set each $M_R(x, y)$ once. Specifically the third approach calculates cells

row by row, and both the second and the third approach can stream out the results since each cell is set at most once. When choosing between the second and the third approach, we pick the third one if the ratio $r = \frac{c_1 d|\mathcal{T}|^2/2}{c_2 e d|\mathcal{T}|} = \frac{c_1|\mathcal{T}|}{2c_2 e} > 1$. Note both c_1 and c_2 can be easily estimated by experiment, and c_2 is usually one to two orders of magnitude larger than c_1. In general, if the size of all the tags is small enough to hold the full matrix in memory, then use the first approach; otherwise, if we find in the dataset that each instance usually uses a very small portion of all the tags (e.g. less than 1%), the third approach is preferred than the second. In a multi-source cross-domain dataset such as the BTC dataset, instances usually have very few tags from other domains, e.g. a musician instance will seldom use tags from domains like e-Government or life sciences; thus we use third approach.

This matrix provides a function for each regular tag $t_x \in \mathcal{T}$ to return all the regular tags that co-occur with it in at least one instance. i.e.

$$\mathrm{CO}_R(t_x) = \{t_y | M_R(x, y) = \text{true}\} = \{t_y | t_y \in \mathcal{T} \wedge f_R(\{t_x, t_y\}) > 0\} \tag{4.5}$$

Note there are different ways to store this matrix. If we treat the relationship of tag co-occurrence as a graph, we can see all the graph representations are applicable to this problem. In practice, this matrix is usually very sparse for most interlinked datasets, and "adjacency lists" [12] are generally preferred because they efficiently represent sparse graphs. An adjacency list representation of a graph is a collection of unordered lists, one for each vertex in the graph. Each list describes the set of neighbors of its vertex. Thus

in our adjacency lists, for each tag, we maintain a list of tags that co-occur with that tag.

To reduce the space cost (and thus the loading cost) in the above steps, we apply two kinds of mappings to the URIs of tags before we build the index and the co-occurrence matrix. We first replace the full URIs of the ontological terms with their qualified URIs (*prefix:local_name*), where the prefix is mapped from the namespace of the URI. The namespace is first looked up via a namespace lookup service [1], so that the well-known namespaces will be replaced by their typical prefixes, e.g. `http://xmlns.com/foaf/0.1/` to `foaf`. However, if no record is found, we will use an automatically generated one, such as `ns1`, `ns2`, etc., and record the mappings. Then we map these qualified URIs to integer IDs, so that we have a minimal cost for storing all tag related information. The integers are specially designed to facilitate many features of the system:

- **Integer Ranges.** We split the range of integers into a few sections, so that for any given integer, we can quickly know whether it is a regular or negation tag, and whether it is a class, a property, or inverse property. In our implementation, we use the first bit of a positive integer to indicate whether the tag is class(0) or a property(1), and use the second bit to indicate whether a property tag is an inverse property(1). Thus in this implementation, we limit the total number of classes to `Integer.MAX_VALUE`/2, and the total number of properties to `Integer.MAX_VALUE`/4.

- **Mapping Relations.** The integers are also assigned in a way so that: from the ID of a property we can quickly compute the ID of its inverse property or vice versa

[1] `http://prefix.cc/`

($id_1 = id_2 + \text{INV\_MASK}$); from the ID of a regular tag we can quickly get the ID of its negation tag or vice versa ($id_1 = -1 - id_2$).

- **Sorted Order.** To help sorting alphabetically in the system, we ensure that within the same regular tag type (class, property, or inverse property), the order of IDs is in accord with the order of the local names of the tags.

Proposition 1. *If t_1 is a sub tag of t_2 under R, any tag that co-occurs with t_1 under R should also co-occur with t_2 under R. i.e.* $\text{CO}_R(t_1) \subseteq \text{CO}_R(t_2)$ *if* $t_1 \sqsubseteq_R t_2$.

Proof. $\forall t_x \in \text{CO}_R(t_1)$, we know $\exists i \in \mathcal{I}, \{t_1, t_x\} \in \text{Tags}_R(i)$. Thus we know t_2 is also assigned to this instance i, because t_2 is a super tag of t_1 under R-entailment, and thus $\{t_2, t_x\} \in \text{Tags}_R(i)$. By definition of CO_R, we know $t_x \in \text{CO}_R(t_2)$. So we have proven that $\forall t_x(t_1 \sqsubseteq_R t_2 \wedge t_x \in \text{CO}_R(t_1) \rightarrow t_x \in \text{CO}_R(t_2))$. i.e. $t_1 \sqsubseteq_R t_2 \rightarrow \text{CO}_R(t_1) \subseteq \text{CO}_R(t_2)$ □

4.2 Online Computation

Given a context $A \in \mathcal{A}$ and entailment regime R, the online computation will return all the $f_R(\{t\} \cup A)$ for every tag $t \in \mathcal{T}$. With our index, we can simply issue an IR query for each t that counts all the instances with all tags in A and t, which returns the number of total hits for a boolean `AND` query (or `AND NOT` for negation tags). Note that the underlying system compares the posting lists of all tags in the query, and because A is the common part among this series of queries, the intersected posting list can be shared among queries. Thus increasing $|A|$, i.e., the number of tags in the context, may simplify the queries by

generating a shorter posting list for A. A quality IR system can answer a count query within a few milliseconds, but since we have hundreds of thousands of tags, we need to focus on how to reduce the number of queries.

There are two special cases of the f_R results, which we want to know without issuing f_R queries:

1. **Always-Occur** i.e. $f_R(\{t\} \cup A) = f_R(A)$. If t is a super tag of any tag t' in A, then $t \sqcap t' = t'$, adding t to T does not change the instance set and thus does not change f_R, i.e.

$$\forall t' \in A, \forall t \sqsupseteq_R t', f_R(\{t\} \cup A) = f_R(A) \tag{4.6}$$

2. **Never-Occur** i.e. $f_R(\{t\} \cup A) = 0$. If there is any negation tags in the context A, there will be no instance in this context that is also assigned its regular tag or any sub tag of this regular tag, i.e.

$$\forall \sim t' \in A, \forall t \sqsubseteq_R t', f_R(\{t\} \cup A) = 0 \tag{4.7}$$

Ideally we want to skip every tag in both special cases. However, the above rules are both entailed from axioms, and will only prune a small amount of the tags. However in practice, there are many tags that never co-occur in the same instance, even though there are no axioms stating this disjointness (in some cases, this may not be a necessary condition, but instead a property of the current state of the world). Thus we find more approaches to resolve this.

For convenience, we let $T = A \cap \mathcal{T}$, i.e. all the regular tags in the context A. Since we do not have any further optimization for the negation tags in A, in the following discussion of pruning algorithms, we only deal with the context input $T \in \mathcal{T}$, and leave the pruning algorithms for more general cases as future work. We define $Z_R(T) = \{z | f_R(\{z\} \cup T) = 0\}$, in other word the set of tags that never occur in combination with T. Let CL be the candidate list of regular tags and each candidate $t \in CL$ will need a query $f_R(\{t\} \cup T)$ to be issued. We propose three different pruning approaches to make CL as short as possible.

1. **Use the Co-occurrence Matrix (M).** Given T, $\bigcap_{t' \in T} \mathrm{CO}_R(t')$ has (and not necessarily only has) all the tags $\{t | f_R(\{t\} \cup T) > 0\}$. When $|T| = 1$, it returns only the co-occurring tags and prunes all the $Z_R(T)$. When $|T| > 1$, it returns a super set of the co-occurred tags, because the returned tags are only known to pairwise co-occur with any tag in T, but are not guaranteed to co-occur with all tags in T in the same instance.

2. **Use the previous tag cloud cache (C).** We can show $\mathrm{Inst}_R(\{t\} \cup T) \subseteq \mathrm{Inst}_R(T)$,

$$\mathrm{Inst}_R(\{t\} \cup T) = \{i | \mathrm{Tags}_R(i) \subseteq \{t\} \cup T\} \subseteq \{i | \mathrm{Tags}_R(i) \subseteq T\} = \mathrm{Inst}_R(T) \quad (4.8)$$

   As a result, the set of co-occurred tags given context $\{t\} \cup T$ is also a subset of that given T. Thus if we cache the previous tag cloud, which has the same context T except for the most recently added tag, we can get another super set of the co-occurred tags for context T. This relies on the tag cloud application scenario: it is

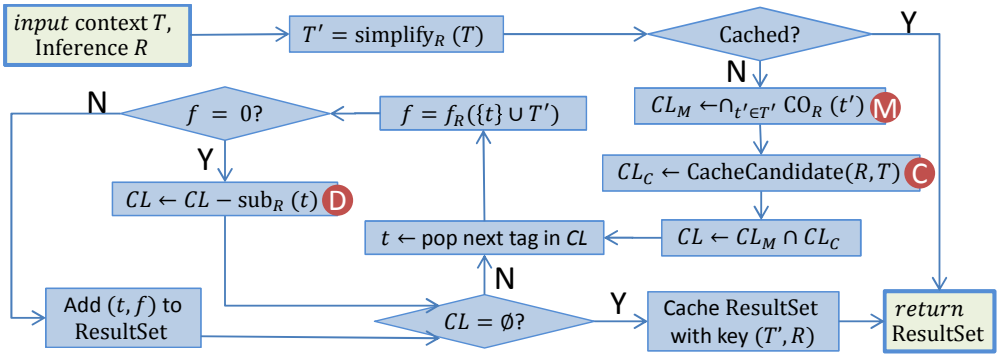


Figure 4.2: Pruning for Online Computation

very likely that the current request is from a user adding a new tag to the context. However we believe it can be applied to any scenarios involving a depth-first search of the context space.

3. **Dynamic update (D).** When computing $f_R(\{t\} \cup T)$ for all the candidate tags from the above two approaches, if we find $f_R(\{t_x\} \cup T) = 0$, we know $\forall t_y \in sub_R(t_x), t_y \in Z_R(T)$, and these tags can be ignored in further computation. This approach can be optimized if we sort the list of tags such that sub tags always follow super tags. However, our tag cloud system does not use this optimization because it needs to stream results alphabetically.

The online computation works as shown in Fig. 4.2, where the pruning steps are marked with red circles. First, the input context T will be simplified (under R-Inference) to its semantic-equivalent T' so that any redundant super tags will be removed and any equivalent tag will be changed deterministically to a representative tag. The key idea of this algorithm is the *coverage* of a tag. Since each tag in the context is a filter, the

functionality of filtering by a super tag is *covered* by it sub/equivalent tag, and thus the redundant super tags are to be removed. This simplification process should ensure the following properties:

- any original tag in T should be covered by some tag in T', i.e.

$$\forall t \in T \exists t' \in T'(t' \sqsubseteq_R t) \tag{4.9}$$

  Another way of expressing this is: If we union the tags in T that are covered by each tag in T', we should see all tags in T are covered. i.e.

$$\bigcup_{t' \in T'} (T \cap \text{super}_R(t')) = T \tag{4.10}$$

- after simplification, any tag in T' should have an equivalent tag in T, i.e.

$$\forall t' \in T' \exists t \in T(t \equiv_R t') \tag{4.11}$$

- after simplification, any tag in T' should not be covered by another in T', i.e.

$$\neg(\exists t_1, t_2 \in T'(t_1 \neq t_2 \wedge t_1 \sqsubseteq_R t_2)) \tag{4.12}$$

The first two properties ensure T and T' are equivalent, and the third ensures T' is most simplified. One implementation of this function is shown in Algorithm 1. The

algorithm finds the set of tags that are covered by some other tag in the context. If there are equivalent tags, the algorithm will consider the latter tags as covered by the first equivalent tag. Also note in this algorithm, the context is modeled as a list[2] (not a set) of tags because the order of tags matters: it reflects the sequence of a user's action. So we need to sort the tags in T' to ensure the deterministic result.

Algorithm 1 Algorithm for Simplifying Context T

```
1: function simplifyContext(Tag[] T, Entailment R)
2:     Set<Tag> C ← ∅                    ▷ Initialize the set of tags that are already covered
3:     for int i ← 1 to T.length−1 do
4:         if T[i] ∉ C then
5:             Tag s ← T[i]                          ▷ s gets the lowest sub tag of T[i] in T
6:             for int j ← i + 1 to T.length do
7:                 if T[j] ∉ C then
8:                     if s ⊑_R T[j] then
9:                         C ← C ∪ {T[j]}                ▷ Ignore super/equiv tags of s
10:                    else if s ⊐_R T[j] then                ▷ Found a lower one
11:                        C ← I ∪ {s}                        ▷ The old s is covered
12:                        s ← T[j]
13:                    end if
14:                end if
15:            end for
16:        end if
17:    end for
18:    Tag[] T' ← ∅                              ▷ Initialize the simplified context
19:    for Tag k in T\C do                ▷ For all the tags that are not covered in T
20:        Set<Tag> E ← {t|t ≡_R k}                    ▷ The equivalent tags of k
21:        T'.push(min(E))                ▷ Use the smallest id as the representative
22:    end for
23:    sort(T')                          ▷ Sort by tag id, to ensure deterministic result
24:    return T'
25: end function
```

Then after the context simplification, the system checks whether this semantic-equivalent request (with context T') has been kept in cache for direct output. If not, the system will

[2]We assume that array index starts at 1

get candidate lists CL_M from approach M using T' and CL_C from approach C using T. Then we use the intersection $CL = CL_M \cap CL_C$ as the candidate list for queries and keep updating it using approach D.

Proposition 2. *Using simplified T' in approach M will get the same candidate tags as using T. i.e.*

$$T' = simplifyContext(T) \rightarrow \bigcap_{t' \in T'} \mathrm{CO}_R(t') = \bigcap_{t \in T} \mathrm{CO}_R(t) \tag{4.13}$$

Proof. By Equation 4.10, we can rewrite $\bigcap_{t \in T} \mathrm{CO}_R(t)$ by grouping tags based on what tag in T' covers them. i.e.

$$\bigcap_{t \in T} \mathrm{CO}_R(t) = \bigcap_{t' \in T'} \Big(\bigcap_{t \in T \cap \mathrm{super}_R(t')} \mathrm{CO}_R(t) \Big) \tag{4.14}$$

Then for each $t' \in T'$, from Equation 4.11, we know that $\exists s \in T, t' \equiv_R s$. Also this $s \in T \cap \mathrm{super}_R(t')$. By Proposition 1, we know

$$\forall t \in T \cap \mathrm{super}_R(t'), \mathrm{CO}_R(s) \subseteq \mathrm{CO}_R(t) \tag{4.15}$$

Since s is a sub tag of all other $t \in \mathrm{super}_R(t')$, then $\mathrm{CO}_R(s)$ must be a subset of all other $\mathrm{CO}_R(t)$. Thus, the intersection of $\mathrm{CO}_R(s)$ with the $\mathrm{CO}_R(t)$ is simply:

$$\bigcap_{t \in T \cap \mathrm{super}_R(t')} \mathrm{CO}_R(t) = \mathrm{CO}_R(s) \tag{4.16}$$

And since $t' \equiv_R s$,

$$\mathrm{CO}_R(t') = \mathrm{CO}_R(s) = \bigcap_{t \in T \cap \mathrm{super}_R(t')} \mathrm{CO}_R(t) \tag{4.17}$$

Thus we know

$$\bigcap_{t \in T} \mathrm{CO}_R(t) = \bigcap_{t' \in T'} (\bigcap_{t \in T \cap \mathrm{super}_R(t')} \mathrm{CO}_R(t)) = \bigcap_{t' \in T'} \mathrm{CO}_R(t') \tag{4.18}$$

□

By removing super tags, we can avoid unnecessary intersection of lists when computing the candidates. On the other hand, the cache approach needs the original T in order to get the previous context; subsequently, this previous context is simplified for cache lookup.

4.3 Supporting Different Entailment Regimes

In our implementation, we have two specific sets of rules: R_{Sub} for sub class/property entailment and R_{DR} for property domain/range entailment. The combination of these two sets leads to four distinct entailment regimes $\mathcal{R} = \{\emptyset, R_{Sub}, R_{DR}, R_{Sub} \cup R_{DR}\}$.

From the raw dataset, we get only $\mathrm{Tags}_\emptyset$, the tags of each instance with no inference applied. In order to implement Tags_R, Inst_R and CO_R for different R, we can either materialize them so that they serve as independent repositories; or we can always do the inference on-the-fly. We first discuss how to represent the three functions under R by combining the $R = \emptyset$ versions (i.e., with no inference) with the tag subsumption functions super_R and sub_R. After that we will discuss the design choice regarding materialization.

By adding inference, an instance will be assigned with the super tags of its explicit tags, and a tag will be assigned to all instances of its sub tags. i.e.

$$\mathrm{Tags}_R(i) = \bigcup_{t' \in \mathrm{Tags}_\emptyset(i)} \mathrm{super}_R(t') \tag{4.19}$$

$$\mathrm{Inst}_R(A) = \mathrm{Inst}_R(T, V) = \bigcap_{t \in T} \bigcup_{t' \in \mathrm{sub}_R(t)} \mathrm{Inst}_\emptyset(t') - \bigcup_{v \in V} \bigcup_{v' \in \mathrm{sub}_R(v)} \mathrm{Inst}_\emptyset(\sim v') \tag{4.20}$$

where $T = A \cap \mathcal{T}$ are the regular tags in A and $V = A \cap \mathcal{V}$ are the negation tags in A (thus $\sim v'$ is the regular tag of the negation tag v').

From Eq. (4.19), we know that t is a tag of instance i under R if and only if at least one sub tag of t under R is assigned to i under $\emptyset$. i.e.

$$t \in \mathrm{Tags}_R(i) \Leftrightarrow \exists t' \in \mathrm{sub}_R(t), t' \in \mathrm{Tags}_\emptyset(i) \tag{4.21}$$

Then from the equation above and the definition of CO_R, we can also see the relation between a pair of co-occurring tags under R. If tag s co-occurs with tag t under R, we can imply that at least one sub tag of s and one sub tag of t under R should co-occur

under $\emptyset$.

$$
\begin{aligned}
& s \in \mathrm{CO}_R(t) \Leftrightarrow \exists i \in \mathcal{I}, s \in \mathrm{Tags}_R(i) \wedge t \in \mathrm{Tags}_R(i) \\
\Leftrightarrow \quad & \exists i \in \mathcal{I}, \exists s_x \in \mathrm{sub}_R(s), \exists t_y \in \mathrm{sub}_R(t), s_x \in \mathrm{Tags}_\emptyset(i) \wedge t_y \in \mathrm{Tags}_\emptyset(i) \\
\Leftrightarrow \quad & \exists s_x \in \mathrm{sub}_R(s), \exists t_y \in \mathrm{sub}_R(t), s_x \in \mathrm{CO}_\emptyset(t_y) \qquad (4.22)
\end{aligned}
$$

For convenience, we define

$$
\mathrm{super}_R^{\cup}(T) = \bigcup_{t' \in T} \mathrm{super}_R(t') = \{t | \exists t' \in T, t \in \mathrm{super}_R(t')\} \qquad (4.23)
$$

which includes all the super tags of any tag in the given set T under R. And similarly,

$$
\mathrm{CO}_R^{\cup}(T) = \bigcup_{t' \in T} \mathrm{CO}_R(t') = \{t | \exists t' \in T, t \in \mathrm{CO}_R(t')\} \qquad (4.24)
$$

which includes all the co-occurring tags of any tag in the given set T under R. Then we compute $\mathrm{CO}_R(t)$ from Eq. (4.22).

$$
\begin{aligned}
\mathrm{CO}_R(t) &= \{s | \exists s_x \in \mathrm{sub}_R(s), \exists t_y \in \mathrm{sub}_R(t), s_x \in \mathrm{CO}_\emptyset(t_y)\} \\
&= \{s | \exists t_y \in \mathrm{sub}_R(t), \exists s_x \in \mathrm{CO}_\emptyset(t_y), s \in \mathrm{super}_R(s_x)\} \\
&= \mathrm{super}_R^{\cup}(\{s_x | \exists t_y \in \mathrm{sub}_R(t), s_x \in \mathrm{CO}_\emptyset(t_y)\}) \\
&= \mathrm{super}_R^{\cup}(\mathrm{CO}_\emptyset^{\cup}(\mathrm{sub}_R(t))) \qquad (4.25)
\end{aligned}
$$

In our implementation, as shown in Fig. 4.1, we materialize Tags_R for all 4 entailment regimes, thus we do not need to compute Eq. (4.20) for online computation. However we only precompute $\text{CO}_\emptyset$ and use Eq. (4.25) at online computation. We made our design choices based on two reasons. First, how much slower will it be if not materialized? Both Eq. (4.20) and (4.25) include union and intersection of sets or posting lists, however the lists of instances in Eq (4.20) are usually much larger than the list of tags in Eq. (4.25), and using Eq. (4.20) significantly increases the execution time compared to the materialized index. Second, how important is the runtime performance? As in our scenario, for each tag cloud (or conditional distribution) given T, CO_R is only called once, however Inst_R is called for each tag from the candidate set.

Also note Eq. (4.25) can be used for either online computation of CO_R or precomputation if it is materialized. Building the co-occurrence matrix M_R is a time consuming step (see Fig. 4.4). We should avoid repeating it four times for four inference regimes. Instead, we only need to build $M_\emptyset$, which is the easiest because each instance has the minimal number of tags, and the co-occurrence for all the other inference regimes can be computed based on Eq. (4.25).

4.4 Implementation Details

In this section we discuss some implementation details that are important to the system. The system is deployed as shown in Figure 4.3. On the user side, the web browser will always first load a page frame by a JSP page. However, the frame nearly contains no

specific data that is dependent on the KB except for the total number of instances in the KB, and the number of instances filtered by the current context (since these two requests can be completed in milliseconds and the results are used at multiple places in the page). Then the javascript code will initiate a few different kinds of AJAX (Asynchronous JavaScript and XML)[3] requests to the web server. These AJAX requests will respectively interact with some back-end JSP. Those JSP will never be directly seen by normal users, but instead return some JSON (JavaScript Object Notation)[4] objects to the frame JSP, where some javascript code will modify the content to the page, such as rendering the data into a tag cloud. When processing the AJAX requests, those back-end JSP do not run any real computation tasks, but instead call functions to compiled Java classes on the web server. Also the Java classes on the web server side only act as a data client, who actually communicates with another server, the data server via our customized socket protocol. The data server is in fact the part that runs all the algorithms that we discussed in Section 4.2. We separate the data server from the web server based on the fact that our online computation requires significant computational resources and that the web server usually supports other services that we do not wish to be adversely affected by the Tag Cloud Browser. Meanwhile such separation also make things easy for debug and system maintenance. Now we shall discuss in details how the tag cloud is shown via various requests.

For the contextual tag cloud page, after the frame is loaded, two AJAX requests will

[3]AJAX Wikipedia page: `http://en.wikipedia.org/wiki/Ajax_(programming)`
[4]Introduction to JSON: `http://json.org/`

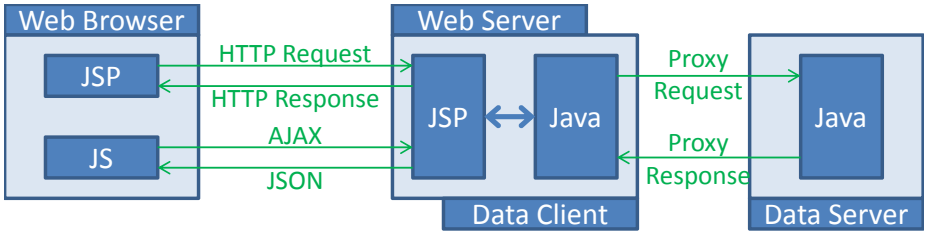


Figure 4.3: System architecture and major techniques for contextual tag cloud system.

first be issued. One will fetch the context information, which also include the change in the number of instances if any one tag is removed from the context. The other will start the computation of the tag cloud for the current page. Then two more repeated requests will follow the start request. One will fetch whatever data is currently available for the current page to display, and decide whether to repeat itself by checking from the result whether the complete flag for the current page is set. Also the next page button is enabled after this flag is set and a "more" flag is set. The other request following the start request will check the status for all pages, i.e. it will check whether all the candidates have been used to issue a f_R query, and based on the progress, report information (such as the estimated time to finish) in the result. Similarly there is a flag indicating whether all pages are completed. If they are completed, the sort-by-frequency option will be enabled; otherwise we will repeat this status request after a short interval (which we set as 1-3 seconds).

Meanwhile on the data server side, we simplify the context in the request and use that as the key to interact with the cache. We implement the cache in two layers. The first layer is the LRU (Least Recently Used) in-memory cache, i.e. when the cache is full (the number of stored keys exceeds the limit) we discard the least recently used (requested) items first.

Then the second layer, i.e. the disk based cache is used when an item is removed from memory: we extract the list of tag-frequency pairs, which is the most time consuming part of the results, as one array of tag ids and another array of their corresponding frequencies and serialize these two integer arrays into a file. When a start request is received, the data server will first check if its result is in-memory, if not then it will check if it is available in disk. If neither exists, we will create an in-progress cache item and put it in memory cache. A thread is then started to process this request and push any tag with non-zero frequency to the cache. So when there is a request for a specific page of the tag cloud, we can directly find the in-progress cache item, and compute the beginning index and the end index in the array (since we have a fixed number of tags per page), and return any available ⟨tag, frequency⟩ pair in that range. Meanwhile we maintain a queue for the candidates (CL in Figure 4.2), the queue's initial size, and the start time of the request, and by using these records, we are able to estimate the percentage of the progress, time to finish, etc. in order to answer the status request.

Theoretically we can provide the same paging feature for the search tag cloud. However currently in our implementation we do not use cache for the search feature, which means we do not have the streaming or paging feature on the search results. Instead, a keyword query will only return the result page with up to 500 matched tags, after the page is complete. We find this limit is quite enough for most queries because we sort the results by relevance score, and usually the results become very irrelevant after 100 tags. This is partially due to the relatively straightforward search algorithm we currently use. For a

keyword search, we only consider the `rdfs:label`, `rdfs:comment` and the local names of the classes and properties. We implement a parser for the local names to deal with the camel case (e.g. `d:SoccerPlayer` has capital letters "S" and "P" for writing compound words "Soccer Player"), which is frequently seen in them. Then we index the stemmed words from these labels, comments or local names. Each word in the index has a posting list of tag ids, which are the same as we use in the instance index. Thus for any search request, we first get a list of tag ids from the keyword match, and then a candidate list from the Co-Occurrence Matrix approach. We go through the first list and ignore any tag that is not in the second, until we reach the end of the first list or we reach the page limit of the search result page.

4.5 System Scalability Analysis

Our system is implemented in Java and we conducted all experiments on a RedHat machine with a 12-core Intel 2.8 GHz processor and 40 GB memory.

In order to test the performance of our preprocessing approach, we apply it to all five subsets of the BTC 2012 dataset, as well as the full dataset. The statistics are listed in Table 4.1.

Fig. 4.4 illustrates how long each step of preprocessing takes for each subset, among which the sort step is multi-threaded (6 threads in our experiment). The Multi-Inference step is not included in the figure since it is too short (41s for the full set) compared with other steps. In general the sorting step and the steps that involve a full scan of the

Table 4.1: Statistics of Triples in the subsets of BTC 2012 dataset

Set Name	Total	Ontology Triples	SameAs Triples	Instance Triples
`rest`	22 M	54.7 K	734 K	∼22 M
`freebase`	101 M	0	8975 K	92 M
`dbpedia`	198 M	1.8 K	22,818 K	175 M
`timbl`	205 M	1,260.1 K	340 K	203 M
`datahub`	910 M	466.0 K	4,490 K	905 M
full set	1,437 M	1,782.6 K	37,357 K	1,397 M

dataset, such as Replace/Flip and index, are the most substantial. Each step is related to certain factors of the dataset provided in Table 4.1. E.g. the time for inference is related to the number of tag subsumption axioms, which is correlated with the number of ontology triples; the time for union-find on sameAs is related to the number of SameAs triples; and most of the other steps are related to the number of instance triples. Despite the differences in the portions of different kinds of triples, we also plot the time/space for datasets against their numbers of total triples in Fig. 4.5, which shows the scalability of our preprocessing approach. The reported disk space includes both the index and the no-inference co-occurrence matrix ($M_\emptyset$), and is dominated by the index, which usually takes $> 90\%$ of the total space. We can see the time is quite linear with the total number of triples, because most of the major steps are linear w.r.t. the number of triples. All these major steps require a full scan of the triples, and might be slightly complicated by the complexity of the entailment results. Among these steps, the most tricky one is the sort step, which in theory should have the time complexity of at least $O(n \log n)$. However, note that in our approach, we always split the triples into n small files and do not need to merge the sort results of them. So we can fix the average file size of each file, i.e. set

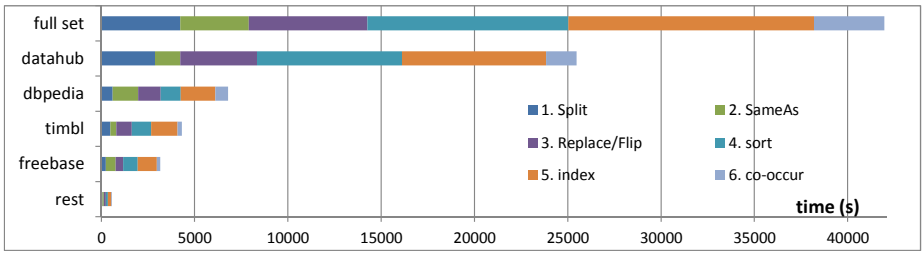


Figure 4.4: Time for steps of preprocessing various datasets

$c = \frac{N}{n}$ as a constant, where N is the total number of input triples and n is the number of files we want to split. Then if we assume, roughly speaking, each file has the same size, sorting each file will have a constant time cost and the total time is linear w.r.t. n, and thus linear w.r.t. N. However, we should also be aware that the assumption can be false in some cases. Since all triples with the same subject must be assigned to the same split file, if the total number of triples of some subject is large enough to make its file significantly larger than the average, then we would not expect to see this linearity trend continue. For the extreme case,the whole set contains only a single subject, and the split result will be a single file the same size as the input, and this file must still be sorted using a $n \log n$ algorithm.

The space has the same trends as the time, however is slightly less correlated to the total number of triples, since many different triples might only contribute to a single tag in the index. For example, there might be 1000 triples saying a `foaf:Person` `foaf:knows` 1000 different people, however these triples only contribute a single property tag to this person. This is exactly what happens in the `timbl` subset, and explains why we see `timbl` has slightly more triples than `dbpedia` but needs less time/space.

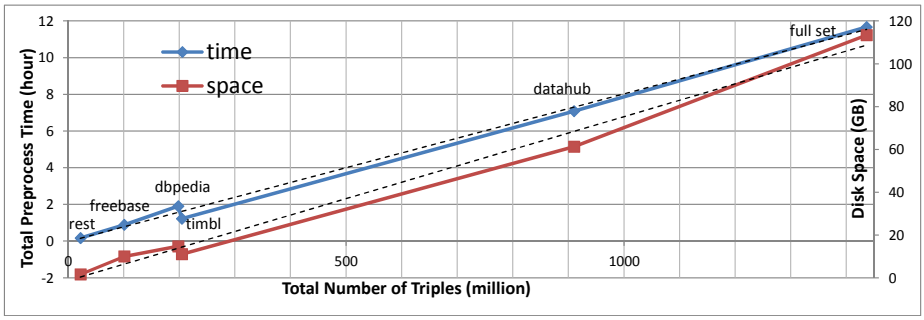


Figure 4.5: Preprocessing: Time/Space - Total Triples

We then test the response time of $f_R(\{t\} \cup T)$ queries, i.e. how long it takes to count the instances of tag t with context T by querying the index. To ensure a random but meaningful context T, i.e. $\text{Inst}_R(T) \neq \emptyset$, we randomly pick an instance i and get a subset (size of 6) from its tags $\text{Tags}_\emptyset(i)$ as $[t_{i,1}, t_{i,2}, \ldots, t_{i,6}]$. Thus the six tags in this array are known to co-occur under all entailment regimes. We generate 100 such arrays using different i. Additionally, we pick a set S of 10000 random tags. Starting from[5] $k = 1 \ldots 6$, we use the first k tags in the arrays as contexts T, and we measure the average time of $f_R(\{s\} \cup T)$ for all $s \in S$. While S might overlap with some T, it does not impact the query time since we issued the same f_R queries without removing redundant query terms. By doing this, we can compare the average query time for different contexts T because they are intersected with the same tags; and we can compare the difference when adding more tags to contexts because as k increases, each array will provide a more "strict" context then before. We also change $R = \emptyset, R_{Sub}, R_{DR}, R_{Sub} \cup R_{DR}$ to examine the impact of different inference. The average time per 10K queries grouped by $|T|$ is shown in Fig. 4.6.

[5]The initial tag cloud ($|T|$=0) is precomputed and cached, thus we do not test it here.

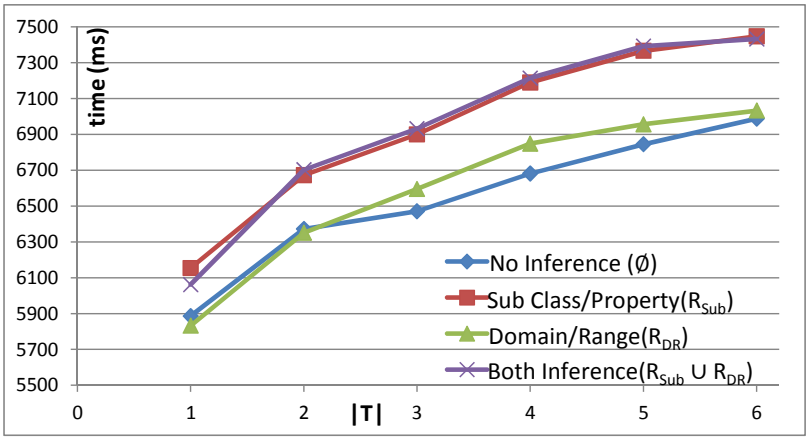


Figure 4.6: Average time for 10K queries as context T grows for each entailment regime.

In average, it takes 0.6∼0.7 milliseconds for a single f_R query. The time slightly increases (sub-linear) when we add more tags to context. It takes longer if R has more inference rules due to longer posting lists of tags in the index. As we expect, since there are fewer tags added to each instance from domain/range inference, we find the curves for R_{DR} and $\emptyset$ are close, while R_{Sub} and $R_{Sub} \cup R_{DR}$ are nearly identical.

For the different subsets, we test the response time of $f_R(T)$ with random T. Since freebase does not have any ontology axioms, we choose $R = \emptyset$. For each dataset, we generate 500K random queries for $|T| = 1, \ldots, 5$ (100K each), and record the average time for every 1000 queries. In Table 4.2, we report the average time for 1000 random queries on each dataset, as well as possible impacting factors such as the number of triples/instances/tags and the average length of posting lists (PList) in the index (i.e. on average, how many instances have each tag). We can see that the numbers of triples/tags do not directly impact query time, but the numbers of instances are very correlated. When

Table 4.2: Average time per 1000 queries over datasets

Dataset	Time	Triples	Instances	Tags	PList
rest	257ms	22.3M	4.1M	3K	6581
freebase	270ms	198.1M	23.5M	308K	3661
timbl	326ms	204.8M	22.7M	12K	11616
dbpedia	341ms	101.2M	31.1M	29K	445
datahub	972ms	910.1M	122.0M	33K	22644
full	1256ms	1436.5M	198.6M	378K	2986

the numbers of instances are similar (timbl and freebase), huge difference in the average lengths of PLists can also impact the time.

We also test how well our system does for pruning candidate tags under the most complex inference $R = R_{Sub} \cup R_{DR}$. Using the approach above, we generate 100 arrays of length 6 from $\text{Tags}_R(i)$, by changing the length of sub arrays we get 600 random T. As we discussed in the previous section, there are three approaches: by co-occurrence matrix (**M**), by previous cache (**C**), or by dynamic update (**D**). By each combination of approaches, we can count how many f_R queries are finally issued, and see how many queries are pruned. Note there is always some pruning due to super tags of tags in contexts. When using approach C, we always assume the previous cache is available.

The average number of pruned tags is shown in Fig. 4.7. There are $|\mathcal{T}|$ =389K tags in total however most tags only co-occur with a few other tags. Pruning usually saves us unnecessary queries. We can see when $|T|$ increases any approach will generally prune more tags because more tags in T means a more constrained context. Among the three approaches, M in average prunes more tags, and enabling the other two approaches in conjunction with M only provides less than 1% more pruning (thus we do not show the

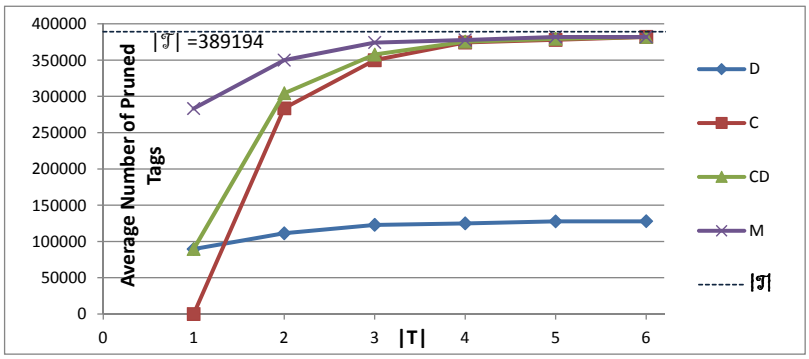


Figure 4.7: Average Number of Pruned Tags

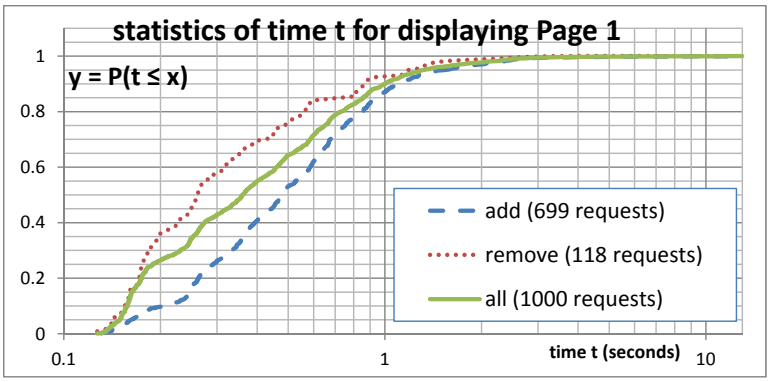
overlapping curves for combinations MC, MD and MCD). This justifies the preprocessing for the co-occurrence matrix. C also has good pruning except that when $|T| = 1$, the cache of $|T| = 0$ is a list of every tag and C will not help. However, in the tag cloud scenarios, $|T| = 1$ is important as it will decide the response after the user's first click. Also in practice, the history cache might not always be available (e.g. a user adds t_1, t_2, t_3 and then removes t_2). So its availability is a concern although it requires no preprocessing. The time cost for CO_R is not a key concern to our system. The average time for the above test set is $1.1s$ with all approaches enabled. However running this pruning saves ~300K f_R queries or in average $0.6ms \times 300K = 180s$ for each tag cloud. For the above 600 T, we have an average time of $8.8s$ per tag cloud, with max of $48.8s$. Thanks to the paging and streaming features in our interface design, the first 200 tags in the tag cloud page almost always show within 2 seconds, which we consider an acceptable responsiveness.

Lastly we test with end-to-end web requests. We create a random browser model, with 0.6 probability to add a tag to the context (*add request*), 0.2 probability to remove a tag from the context (*remove request*), and 0.2 probability to start over with empty context

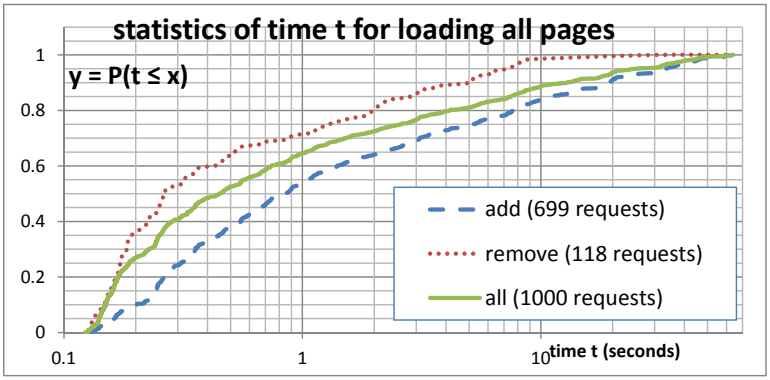
(*start-over request*). In order to simulate more realistic requests, when the context size is small, we bias in favor of adding tags. Also when trying to add a tag, we give the more frequent tags in the tag cloud a higher chance to be selected. Using this model, we randomly generate a series of 1000 requests on R_{Sub}, including 699 add requests, 118 remove requests, and 183 start-over requests (note when the context has only one tag, we count the remove request as start-over), with an average context size of 2.53. We record the time spent on displaying all the tags (up to 200 tags per page) in Page 1, and that on finishing computation for all the pages (that is also when the "Sort by Frequency" feature is enabled). In Fig. 4.8(a) and Fig. 4.8(b) we show the percentage of requests that can be finished within x seconds. For displaying Page 1, 90% of all the requests can be finished within $1s$, and 97.7% within $2s$. Among these requests, start-over is the fastest (not shown) since it just returns the cached results. On average, remove requests are faster than add requests since they are more likely to have a shorter context or be cached due to previous add requests. However as shown in Fig. 4.8(b), the time spent on loading all the pages can be as long as a minute and vary more uniformly. This signifies the importance of streaming results and displaying them in pages. Also, this justifies the decision to defer the frequency sort option.

4.6 Comparison to Other RDF Repositories

We use the inverted index as our RDF triples, although there are many existing alternative repositories for general-purpose RDF data management and query answering. From



(a)



(b)

Figure 4.8: Browser-Side Response Time for Random Requests: (a) Time for displaying Page 1; (b) Time for loading all pages.

the aspect of formal queries to RDF dataset, $f_R(A)$ is actually a special kind of query template. Thus most available RDF repositories should be able to answer this kind of query. However, the choice depends on the scalability on two aspects: most importantly the online computation time for f_R, and meanwhile an affordable preprocessing time. In this section, we will compare our system to other systems with regard to this aspect.

Our first attempt is to build a system involved an relational database, with proper optimizations applied. Firstly, we did not consider holding the KB with a triple table,

otherwise our task would require expensive join operations over multiple selections on the giant table. On the other hand, the decomposed storage mode seems more promising: the triples are inserted into n two-column tables (each of which is much smaller than a single triple table) where n is the number of unique properties (including `rdf:type`) in the data. In each of these tables, the first column contains the subjects that define that property and the second column contains the object values for those subjects. However the join operation on selections on tables is inevitable for our task, thus we made several simplifications and optimizations, in order to minimize the cost of the DB approach. For each $t \in \mathcal{T}$ we create a table with a single indexed column, *id*, an integer that represents each unique instance of tag t. Thus $f_R(T)$ can be computed by joining the tables of classes and properties. For faster table joins, we use a dictionary that maps all strings (either full URI or its qualified name) to integer ids, and use the ids in the property tables. However this might require maintenance of the dictionary and frequent look-ups slow down the process. Given that the task is only collecting summary information, we do not have to record the real URIs, but only need to know whether an instance has appeared before when processing a new triple. So if we reuse the first three steps in the IR approach, we can just assign an auto increment id to each new instance (if it is different from the last one) while reading through the sorted file line by line. Then this auto id can be inserted into tables corresponding to the tags in its cluster of triples. Similarly, the superclass and super property inference is materialized at this step. In practice, if the insertion is done line by line there is much waste in the overhead cost of DB operations. Thus instead, we

first generate the script file of insertions, and run it in a batch.

We choose DBPedia 3.6 as our dataset. Specifically, we load two raw data files, i.e. Ontology Infobox Types and Ontology Infobox Properties, together with the ontology. Comparing to the BTC 2012, DBPedia is not a multi-source interlinked dataset, but still a good subset of the Linked Open Data: (1) it plays a central role in the Linked Data world; and (2) it has two important features of multi-source linked datasets: an ontology with broad scope and a large scale of data. There are in total 2,446,683 instances in 27,221,328 triples (including the flipped ones), which we turned into 1952 tags. We simplify the preprocessing steps in Section 4.1, and only keep the necessary steps (2-5), with the only entailment regime R_{Sub}.

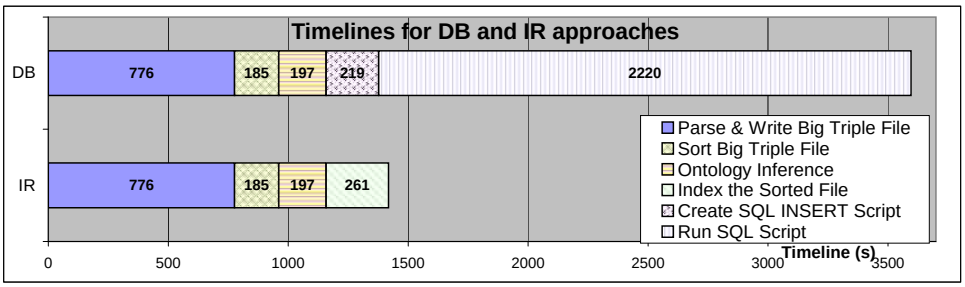


Figure 4.9: Timeline for loading data with DB and IR approaches

We use MySQL 5.0.21 and Lucene 3.3.0 as the underlying DB and search engine. In Fig. 4.9 we illustrate the process of both approaches for loading the DBPedia dataset and compare the timeline between them. Using the IR approach, most of the time is the cost of parsing and writing the file. However, comparing between the DB and the IR approaches, we find that the step for running the SQL script itself is already more than the total time of the IR approach.

To compare the time cost of f_R by both approaches, we conduct another experiment: run $f_R(\{t_x, t_y\})$ for a comprehensive set of combinations pairwise $t_x, t_y \in \mathcal{T}$. There are 1952 tags, and thus we call f_R for 1,904,176 ($= \frac{1952 \times 1951}{2}$) times. It turns out that it takes more than 2 hours if we issue so many queries to DB, but only less than 8 minutes if we search the index. This shows the great advantage of the IR approach compared to the DB approach. In conclusion, we find the DB approach is not as efficient as using an inverted index for this specific application purpose, both in terms of load time (8X slower) and online query time (18X slower).

We also compare the response time of this specific kind of queries with RDF-3X, a state-of-the-art SPARQL engine that "outperforms its previously best systems by a large margin" [34]. It takes 9 hours and 11 minutes to load the full BTC dataset into RDF-3X. Note that this loading does not include any kind of inference, sameAs closure/replacement, nor co-occurrence computation as we do in our preprocessing. Similar to the previous experiment, for context size $|T| = 1, \ldots, 5$, we randomly pick 50 (10 of each) contexts, and this time we measure how long it takes for both systems to compute the full contextual tag cloud without pruning. i.e. for a given T, we compute $f_\emptyset(\{t\} \cup T)$ for $\forall t \in \mathcal{T}$. We use $R = \emptyset$ because RDF-3X does not explicitly do any inference. The comparison results are shown in Table 4.3. In addition to the average execution time of both systems, we also list the Average/Maximum/Minimum Differences, which shows how much faster our system is compared to RDF-3X, with respect to an average query, its best query and its worst query. Note, the times in this table are longer than those in Fig. 4.6, because we are

Table 4.3: Comparison on Time Cost for Computing Full Tag Cloud (No Pruning)

$\|T\|$	Avg. Time Ours	Avg. Time RDF-3X	Avg. Diff.	Max Diff.	Min Diff.
1	65.8 s	887.6 s	13.5 X	93.2 X	1.71 X
2	84.9 s	516.7 s	6.09 X	15.6 X	2.87 X
3	90.7 s	721.2 s	7.95 X	20.6 X	4.56 X
4	92.8 s	1030.8 s	11.1 X	30.8 X	6.24 X
5	110.3 s	1359.7 s	12.3 X	33.4 X	4.44 X
All	88.9 s	903.2 s	10.2 X	93.2 X	1.71 X

issuing ∼380K queries as opposed to 10K. It is clear that our system always outperforms RDF-3X. Averaging across all queries in our test set, our system is 10 times faster than RDF-3X. The differences are more pronounced when $|T|$ increases, although both systems have a sub-linear increase in query execution time as $|T|$ increases. There are two outliers of the Max/Min trends. When $|T| = 1$, the Max Diff. occurs when $f_\emptyset(T) = 49,584,018$, which is the largest set of instances specified by the context in our test set. When $|T| = 5$, the Min Diff. occurs when $f_\emptyset(T) = 143$, which is the smallest set of instances specified by the context in our test set. It is possible that the smaller sizes of instances specified by the context lead to more efficient joins in RDF-3X, allowing it to approach our system's performance.

The key point to recognize here is that one-size-fits-all triple stores are not always the best solution for scalable applications. By choosing a carefully constrained user interaction method, we are able to design a specialized infrastructure that can meet our performance requirements. That said, we posit that the systems capable of performing voluminous tag intersections can be used not just for supporting user interfaces, but for data mining and anomaly detection as well.

Chapter 5

Preliminary User Study on Contextual Tag Cloud System

We proposed the contextual tag cloud system as an easy and straightforward way for users to explore large scale linked data. The system, as well as the idea it demonstrates, has been well-received during several Semantic Web conferences since 2011. Unlike most of the other available tools for Semantic Web datasets, this system does not require a lot of the background knowledge regarding the Semantic Web such as ontological axioms or SPARQL syntax, yet conveys significant useful information about a large, complex dataset while supporting real-time exploration by users. We hypothesize that this will be a practical tool for non-expert users to access knowledge bases with thousands of terms and millions of instances, such as Linked Open Data. In this chapter we want to focus on the particular questions related to usability: How well will non-expert users accept this

tool? Will the system help them explore a large scale dataset and find useful information from it? We start with the study design and continue with analysis on the results from the study.

5.1 Study Design

We want to understand whether average users could easily explore a large scale Semantic Web dataset by using our proposed system. However there is not a straightforward way to evaluate this. We address two main factors which make it difficult to design a reasonable user study.

The first reason is that we have not seen any other system like the Contextual Tag Cloud system (referred to as TC in this chapter) that is available (directly accessible or downloadable for our deployment) for users to explore a large scale Semantic Web dataset. When a dataset is small, it is easy to customize and design a system for users to explore it; and those specific systems are not considered for comparison. Also we do not consider any system that requires SPARQL as the input; nor any system that simply does information retrieval as a search engine, which would not help users with understanding the structural information. Due to the lack of a comparison system, we implement another interface, namely the Hierarchical Faceted Term List (referred to as HL in this chapter), which supports faceted browsing of Linked Data, but does not include any features of tag clouds. Both systems hold the same dataset, i.e. the BTC 2012 dataset. We shall introduce the HL system later in this section.

The second problem is how to define the tasks for the users when they use the system during the study. Usually when a user study is designed, we want to tell the users what is the task, i.e., explain the purpose of the task for the investigation. We have a hypothesis we want to test, the task must be measured in a way that can be used to test the hypothesis. The task, is also set as what typically the real world users want to do with the system. However, given that the system is still a prototype with a lot of room for extension, we find it really hard to specify what exactly the typical purpose should be when it is used in the real world. If we want to evaluate the system's use as an easy exploring tool, it is unclear how to specify an appropriate task whose result can be objectively quantified. Everyone can learn different things after exploring the dataset, but how shall we measure how well and how much the user learns from exploring the dataset with a specific tool? We shall discuss how we choose the tasks later in this section.

5.1.1 Comparison System: Hierarchical Faceted Term List

Since we did not find any similar systems that are available online for users to explore a large scale Semantic Web dataset, we decided to implement an unbiased baseline system for comparison with our proposed TC system. We think there are two major advantages of the TC system: it is responsive and it requires very little prior knowledge. To make sure the other system has similar performance, we decide to mostly change the user interface part, and reuse most of the infrastructure functions we already have.

The TC system, as we have introduced before, can be considered as a combination of the tag cloud browsing and the faceted browsing interfaces. Either kind of interface, we

believe, provides some good aspects for exploring large scale datasets. Thus we want to see, through this study, what if we did not apply the combination, i.e. what if we only use one aspect or the other? We quickly found that the system becomes very trivial if it only supports tag clouds: The system will only need the precomputation for counting instances of each class, and then present a static page of the statistics on those counts. There will be very few interactive use cases, and thus users will not be able to get as much information as from the TC system.

Thus we decide to implement a faceted browsing system. We design and implement the faceted browsing system similar to the online shopping sites, which faceted browsing systems are best known for. A screenshot is illustrated in Figure 5.1. The tags are still the same as we defined for the TC system, however, instead of the various sized tag cloud, each tag is alphabetically listed on the left-hand sidebar, with a square bracket and a number after it indicating the relative size of this tag to the selected facets (i.e. the context in TC). We use the same function $f_R(T \cup \{t\})$ (defined in Chapter 3) to compute the relative size of tag t with context T and apply the same pruning algorithms to avoid unnecessary queries when generating the tag list. Like TC, the tags are also displayed in two separate views, classes and properties. The tag search feature provides a list of matched tags in the same order as the TC system. However the search result has a different view: there will be a pop-up window showing the list of tags using a consistent font size, but the user will still be able to see the relative count of each tag from the following number in square brackets. In the main area of the page, we list all the instances that match the current

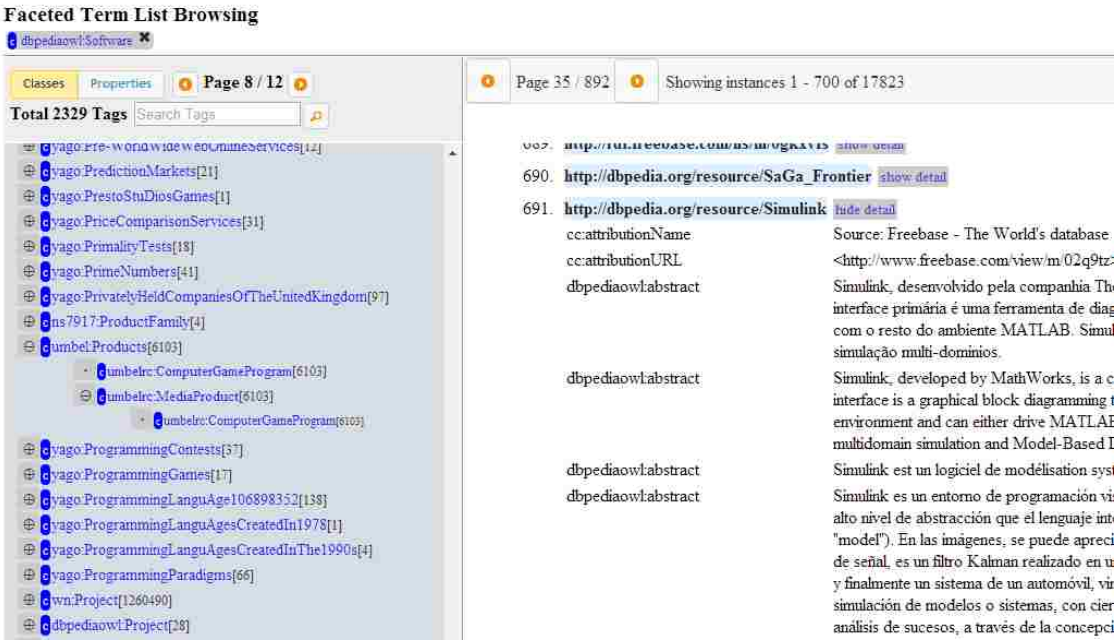


Figure 5.1: Class Hierarcical Faceted Term List with contexts `dbpediaowl:Software`.

context, and also allow users to expand the details of each listed instance which shows the raw triples about it.

We also implement an extra feature in addition to a typical faceted browsing system: the type hierarchy information. Since a Semantic Web dataset usually contains a taxonomy, we think it useful to represent the hierarchy of tags in order to clarify the semantics for users and help them locate information of interest. When a tag t_1 is subsumed by another tag t_2, t_1 will only be found in the sub folder of t_2. As illustrated in Figure 5.1, `umbelrc:ComputerGameProgram` is subsumed by `umbelrc:MediaProduct`, and both are subsumed by `umbel:Products`, and thus they are shown as a tree of depth 3. Also this hierarchical display can hide unnecessary tags from the page, which reduces potential distractions to users. Currently we use a naive way to decide whether a tag needs to be

shown. Given a context T, we will show each tag that co-occurs with the context and for which any of the following is true:

- The tag t is a top-level tag. A top-level tag is a tag that is not a proper sub tag of any other tag. i.e. $\nexists t' \in \mathcal{T}, t \sqsubset_R t'$.
- The tag t is a super tag of a tag in T. i.e. $\exists t' \in T, t \sqsupseteq_R t'$. Let S denote the set of tags that qualify this criteria. We need to define the criteria with S.
- The tag t is a direct sub tag of a tag in S, i.e. $\exists t_1 \in S\{t \sqsubset_R t_1 \wedge \nexists t_2(t_2 \sqsubset_R t_1 \wedge t \sqsubset_R t_2)\}$

When a tag is shown, we precede it with a "+" if there is no proper sub tag shown under it; otherwise a "-" indicates that proper sub tags are already displayed. A user can always click the "-" to fold the sub tree of a tag. However, when the user clicks on a "+", it will expand and show any direct sub tag that co-occurs with the context, but if there is none, a "." will replace the "+".

5.1.2 Tasks in the Study

We want to specify the tasks to the participants so that they would use the system to answer questions similar to those that typical users might have. We reviewed the four proposed use cases in Section 3.2, including choosing the right terms for SPARQL, finding interesting facts, and detecting potential errors in instance co-reference or ontological alignment; and from them we tried to see which use cases can be cast as a proper task for the study.

The first use case is to use the system to choose the right terms for SPARQL. We propose this use case as the most promising usage of the TC system. However, during the study or in the real world, we shall not expect that the participants or average users already understand the concepts of SPARQL before they start to use the system. A practical way is to give a tutorial and tell the participants that we usually want to choose the terms that are used in the most instances when no other clue is available for us to decide which candidate term is better for queries. We illustrate the set of instances with Venn diagram and use a made up example about the "movie" classes and the "directedBy" properties. Then the task is stated as

> **Task 1.** Given a pair of keywords for the class and the property as the query purpose, can you find a matched class and a matched property from some source (specified by their namespaces), so that using this pair of terms (the best combination) would retrieve most of the instances in the dataset?

We also need to decide the pairs of keywords. There are a few considerations when picking the keywords. Firstly the topic should not require any specific background knowledge, i.e. it should not be in life science domains, nor about academic publications, given that the participants could be a freshman without such knowledge. Also because we will have multiple pairs, we want the topic of each pair to be different from those of all other pairs. The most important reason is that we do not want a participant to acquire any information in the task of one pair and have this information bias the subsequent task of another pair. Also, isolating topics makes it easier to analyze the user logs and see which

topic each request is related to. We list the final pairs and the best combinations we have for the study as follows. Note that in this experiment we only consider the syntactic matches, so that the tasks are more straightforward to the participants and the less open questions make it easier to analyze the results. However in the future we should also consider involving synonyms as answers.

- **Task 1.1** Keyword for Class: "Company", for Property: "location"

  Best Combination: {`dbpediaowl:Company`, `dbprop:location`} (19208 instances)

- **Task 1.2** Keyword for Class: "Scientist", for Property: "award"

  Best Combination: {`dbpediaowl:Scientist`, `dbpediaowl:award`} (3072 instances)

- **Task 1.3** Keyword for Class: "Town", for Property: "population"

  Best Combination: {`dbpediaowl:Town`, `dbpediaowl:populationTotal`} (23104 instances)

- **Task 1.4** Keyword for Class: "Actor", for Property: "starring"

  Best Combination: {`yago:Actor109765278`, `dbpediaowl:starring-`} (15225 instances)

After considering the second use case of "learning interesting facts", we decided that it is not specific enough to investigate with a user study. Although we believe the TC system provides users an easy but flexible way to explore the dataset and learn something, it is hard to quantify whether their findings are interesting and valuable, in fact, interesting and valuable are subjective criteria themselves. So instead, we want to provide participants

the opportunity of interesting findings in other tasks.

The other two use cases are "detecting errors", one involves co-reference errors and the other ontological errors. We believe that non-expert users will find it difficult to analyze errors in order to determine what the reason of the error is. However it might not be hard for the users to find something that sounds improbable as long as they are presented with a interface that they clearly understand what the data indicates. We give a made up example of overlapping between instances of movies and book, and suggest that the reason is because some instances of movies are considered the same as their original novel books. This task is stated as

> **Task 2.** Given a class as the context, browse all the overlapping class tags in the system, can you find something that is incompatible, i.e. it is impossible that something is both an instance of the context class and an instance of the tag?

Again we follow the criteria we use for Task 1 to pick the context, and we also want the total number of overlapping class tags to be shown in 10 - 15 pages. We finally decide two contexts: **Task 2.1** `lgv:Stadium`; **Task 2.2** `dbpediaowl:College`. We shall discuss more with these two sub tasks in Section 5.3.

In order to compare the TC and HL systems, we have each participant to equally use both systems, which also means that each system will be equally used across participants. Also we want each question to be equally answered using either system, so that we can compare and see which system is more effective for completing certain tasks. We fixed the

order of the questions shown to the participants but randomized the order of systems to be used. To ensure the order does not change after refreshing the page, we used a pseudo random order determined by each participant's login user name.

5.1.3 Procedures of the User Study

Before the user study, both Professor Jeff Heflin and I completed the National Institutes of Health (NIH) Web-based training course "Protecting Human Research Participants" certified by the Office of Extramural Research in NIH. We also submitted materials for this research study to IRBNet, and the IRB approved this project (IRBNet ID: 594046-1, Local Board Reference #: 14/176 T) in the EXEMPT category so that the project is exempt from continuing IRB review according to federal regulations.

The study was done in groups, with two participants in each. They were briefly informed about the purpose, the content and the estimated time of the study before they came to the study. At the beginning of the study, they were presented with a consent form (see Appendix A), which clearly states the steps of the study, the potential risks of the study, and also their rights and privacy in the study. The formal study began after participants signed the forms.

The first part was a general tutorial about the related concepts in this experiment, which took 15 - 20 minutes. There were two categories of background contents to be covered in the tutorial: the user interface and the basics of Semantic Web. Even for most of the non-expert computer users, they probably have already used systems with either of the two interfaces, tag cloud browsing and faceted browsing; however they may not know

the formal names for these concepts. The tutorial would remind them of the semantics underlying the representation of the interfaces and the purpose of such design, and help reduce the learning curve of using the two systems. Given that participants may have no background knowledge about Semantic Web or structured knowledge bases, we wanted them to understand the basic concepts, the purpose and the real world scenarios of the Semantic Web, which motivates this study. Then we did a few live demos using both the TC and HL systems. The main focus points of the demo were the basic concepts of the interfaces, the actions and results of modifying the facets (or contexts), the keyword search feature, and the means to inspect specified instances. Although there are a lot of other features, we did not cover those that are irrelevant to this study. Then we also showed the participants the web page where they would answer the questions of the tasks and survey. Finally we demonstrated solving an example question of each task (not from the questions they needed to answer).

After the tutorial, each participant was assigned pseudo randomly to a system for each question in the two groups of tasks. After they finished all the tasks, they were asked a few survey questions (see Appendix B), including 11 Likert scale [29] (single select) questions and 3 short free-form questions. The last 3 text questions are just places for participants to comment on both systems and the study. Each Likert question is a statement with five options: Strongly Agree, Agree, Neutral, Disagree, and Strongly Disagree. The first 3 of the 11 Likert questions are about the participant's experience in using computers and using similar interfaces, and how well they understand the tutorial. The other 8

are actually 4 pairs of questions about the user's opinion on each system: (1) how easy was it to become familiar with the system; (2) how easy was it to finish Task 1 with the system; (3) how easy was it to finish Task 2 with the system; and (4) whether they think the system is a good tool for exploring the dataset. By using Likert scale questions and avoiding stating any direct comparison statements between the two systems, we believe we minimized the bias from the wording or implied attitude from the questions.

Using this procedure, we asked two participants for a trial of the study, and then formally got 14 other participants to complete this study. All the participants are students from Lehigh University: 2 of the 14 are undergraduate students, 4 of the 14 are women, and 6 of the 14 are computer science majors.

5.2 Analysis on Results of Task 1

We compare how well participants completed the four subtasks of Task 1 when using both systems. The answers are considered invalid if any term in the selected combination does not match to the original meaning and purpose of the question. For example, in Task 1.1, where we wanted the participants to find a term for "company", some participants choose properties like `dbprop:companyType` or `dbprop:companyLogo-`. These terms, although partially matches for the keywords, do not have the same semantics as the query purpose (also they would not retrieve as many instances). Thus we count these answers as invalid combinations, no matter how many instances they can retrieve.

We want to quantify how good each valid participant answer is. For each question,

there is a best combination of terms which will retrieve the most number of instances that matches the purpose of the query. Also for each valid participant answer, we record how many instances it will retrieve. We define the coverage as the ratio between the counts of instances retrieved by a valid answer and the best combination [1]. We compute the average coverage ratio of the valid answers by questions and systems, and report result in Table 5.1. For each subtask and each system, the number of attempts in the first row is the number of participants who answer that question with that system, followed by the number of valid participant answers in the second row, and the ratio between the second row and the first row as the third row. In the last row we report the average of the coverage ratios which we compute only for the valid answers. Note the final column provides adjusted results for Task 1.4, which is an ideal interpretation of the participants' answers, as will be explained later.

Table 5.1: How well do participants finish Task 1 with both systems?

	Task 1.1		**Task 1.2**		**Task 1.3**		**Task 1.4**		**Task 1.4***	
	HL	TC	HL	TC	HL	TC	HL	TC	HL	TC
# of Attempts	8	6	7	7	6	8	7	7	7	7
# of Valid Answers	5	4	7	7	3	4	6	6	6	6
% of Valid Answers	0.63	0.67	1	1	0.5	0.5	0.86	0.86	0.86	0.86
Avg. Valid Coverage	0.63	0.58	0.89	1	0.66	1	0.49	0.47	0.99	0.93

Generally speaking, we find that there is no obvious difference in whether a participant will find a valid answer by using both systems. We think that is because whether a

[1] We assume that all such instances are valid, even though it is likely that some terms have erroneous instances. If we assume that such errors are distributed uniformly, this metric still records the best combination.

participant could provide a valid answer is mostly related to how well they understand the general task purpose (or the way of querying structured datasets). Also from the result we can see the TC system sometimes has significantly better average coverage of the valid answers. In Task 1.2 and Task 1.3, TC gets full coverage, which means participants can always find the best combinations by using TC in these two questions, as long as they correctly interpret the purpose of the questions. In Tasks 1.1 and 1.4, TC users have slightly lower coverage than HL users. We investigate the two answers of Task 1.1 via the TC system that do not get full coverage, and find that both answers are the same: user chose `yago:Company108058098` instead of `dbpediaowl:Company` as the term for "company". The former company class has 5656 instances in the dataset while the latter one has 33747. The difference in the counts of instances is obvious; however we find that in the TC interface, the font sizes for these two tags are 28.3 px and 31.6 px respectively due to the log function mapping from instance counts to font sizes. When these two tags are not placed side by side, the difference in font sizes is really hard to discriminate by eye, especially when the former has more characters which makes it seem to take more space. It might be that when the participants use the TC system for the first time, they assume any difference in the instance counts should be easily reflected by font sizes, or they do not know that they can get the exact count numbers when they hover the mouse on the tags. However, in comparison, participants using HL sometimes select tags with significantly fewer matches, e.g. one participant chose `ns6998:Company` (which has only 775 instances) in Task 1.1 using the HL system.

Task 1.4 is a relatively tricky question. The best combination {yago:Actor109765278, dbpediaowl:starring-} has 15225 instances. Since we did not cover the concept of inverse properties in the tutorial, we believe most of the participants do not know the difference in semantics between dbpediaowl:starring- and dbpediaowl:starring. We saw quite a few of answers without the inverse notation, and we are not sure whether they found the right one but omitted the "-" when filling in the form. In the dataset, both dbpediaowl:starring- and dbpediaowl:starring have some instances of actors as its subjects, although the number of the former is much greater than that of the latter (15225 vs. 6 instances). That means there are data supporting either direction of the usage, and we consider both are valid answers for the keyword "starring". Thus we report two columns for Task 1.4, where the original one, has a much lower coverage ratio than the adjusted one. That is because in the adjusted results, we assume that participants who answered dbpediaowl:starring actually meant dbpediaowl:starring-. There is another tricky point of Task 1.4 because the "greedy algorithm" will fail. When a user searches for "actor", the largest class freebase:film.actor has 26067 instances, which is larger than yago:Actor109765278 (24760 instances). However, if the user chooses this "local optimal" class, the combination of {freebase:film.actor, dbpediaowl:starring-} will only have 12175 instances. In fact, the only two answers by TC system that do not get full coverage (after the inverse property adjustment) are exactly {freebase:film.actor, dbpediaowl:starring-} resulted from the local optimal actor class.

We also compare how long it took the participants to answer each question with both

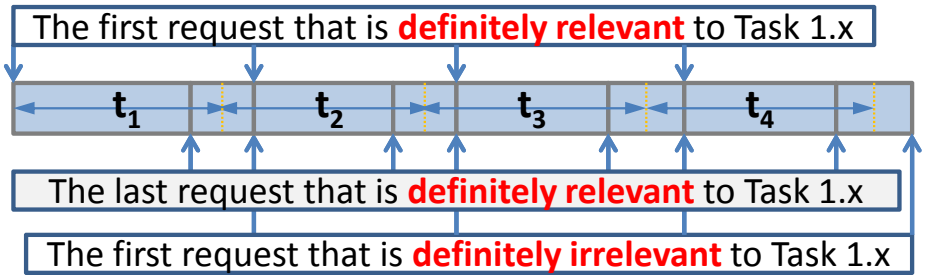


Figure 5.2: Estimating time spent on each question of Task 1.

systems. We used web request logs to estimate the time spent on each task question. When the request contained some keywords, or tags related to the keywords in any of the tasks, we are very sure that it is related to a specific task, and thus we can certainly mark them. However for other requests, we were not sure whether it is relevant to a specific task, especially when this request is between two clusters of requests for two adjacent tasks. Thus we simply apportion half of the ambiguous time interval to each of the two tasks. As illustrated in Figure 5.2, we mark the first and last clearly relevant request of each task question and estimate that each tasks ends exactly halfway between its last clearly relevant request and the first clearly relevant request of the next task. Similarly, we assume that the next task begins at this point.

We plot the box-and-whisker diagram for time spent on each question by each system in Figure 5.3. The lower red boxes indicate the range between the first quartile and the median, the upper blue boxes indicate the range between the median and the third quartile, and the lower bar and upper bar indicates the min and max time. We can see that participants usually spent less time to complete a task using TC in Task 1 except for Task 1.2. HL is slightly better than TC in Task 1.2, and we find the times by the two

systems are very close in this subtask. We also found that one of the time records, by a participant who uses TC, may be estimated with a higher error, because the ambiguous time interval (transition time) between Task 1.2 and 1.3 by that participant is the highest among all all participants and tasks. The average length of the ambiguous time intervals is 32.7 s, while the highest is 109 s.

We also checked the points that seem to be outliers, especially focused on the following three points: 580.5 s in Task 1.1 with HL, 319.5 s in Task 1.3 with TC, and 252.5 s in Task 1.4 with TC. It turns out that there is nothing unexpected. All the request logs show that the participants were doing something related to their tasks. Also, we note that these outliers are not due to our approach to estimating task time: If we do not include the ambiguous time interval in the estimation, i.e. we only look at the time between the first and the last relevant requests, the times are 574 s, 309 s, and 241 s respectively. We believe that is just because of the browsing habits of the participants, and in fact the longest times in Task 1.3 and Task 1.4 with TC come from the same participant. Additionally, the participant who spent the most time in Task 1.1 with HL, also uses the second longest time in Task 1.4 with HL. In order to investigate the times and also take the differences between participants into account, we plot the total time spent on each system by each participant in Figure 5.4. It shows us that 9 out of 14 (64.3%) participants spent less time in total when they use TC. But also it indicates that some users might find HL a more effective way for them to complete Task 1. Note that we have mapped the original usernames to user ids, which does not reflect the sequence of participating this

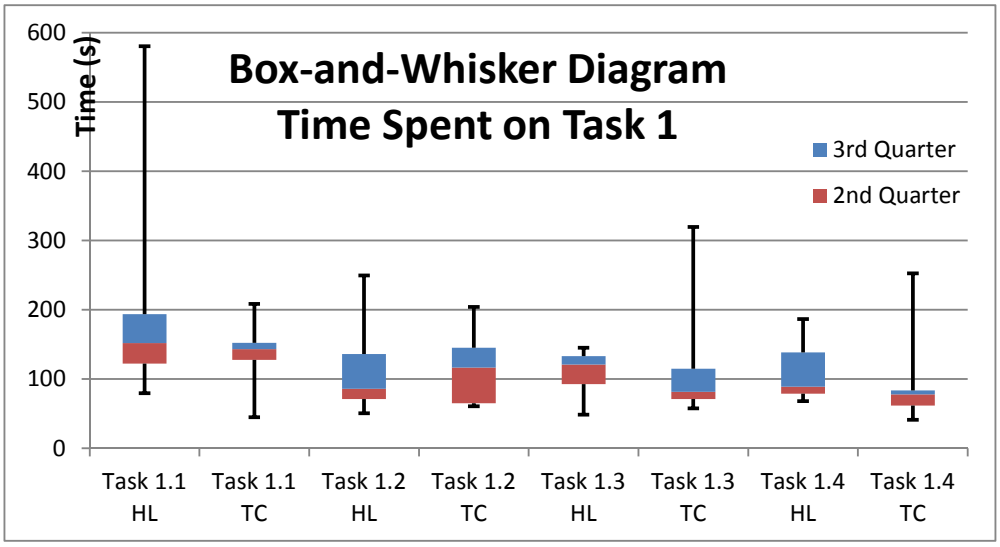


Figure 5.3: Statistics of Time on each question of Task 1.

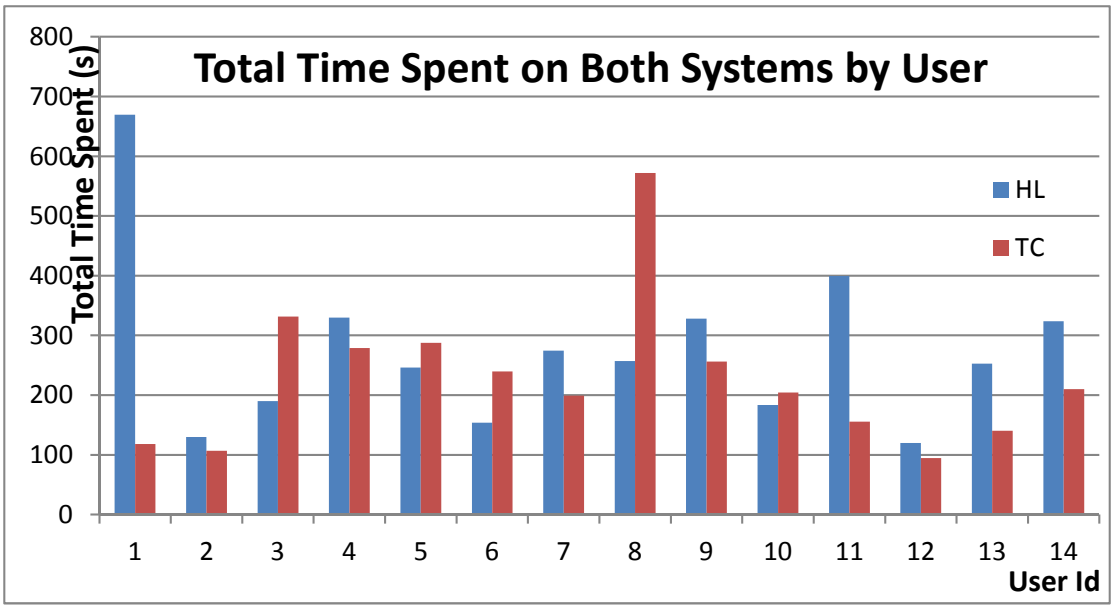


Figure 5.4: Total time spent on each system by all participants.

study.

Another trend from Figure 5.3 is that participants spend less time from Task 1.1

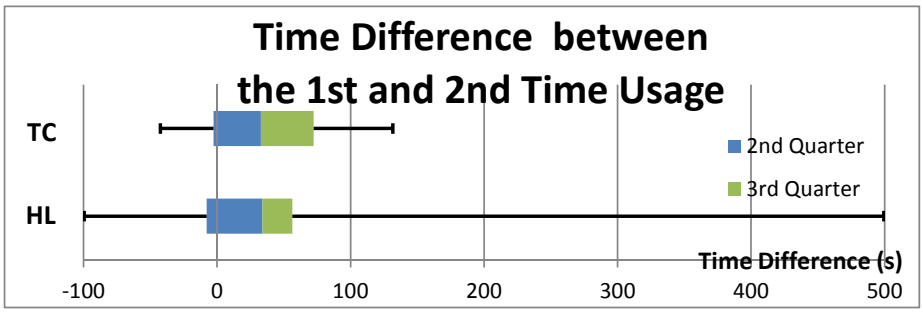


Figure 5.5: Statistics of time reduced when a participant uses a system for the second time.

to Task 1.4. We hypothesize that this improvement could be attributed to participants getting familiar with both the tasks and the tools. Based on this observation, we compute the difference of time that a participant spent the first time they used a system and the second time they used the same system. We find that 10 out of 14 participants improve their completion time the second time they use HL, and 10 out of 14 for TC as well. So usually there is an improvement, and the time gets reduced. We plot another box-and-whisker diagram in Figure 5.5, which summarizes these differences. Although we find HL has a larger value in the max, which is due to the "suspicious outlier" we mentioned before, TC is better at the min value, the first quartile, the median, and the third quartile. That suggests people are likely to get more improvement when they use TC for the second time.

We used a one-tailed t-test to verify whether our hypotheses are statistically significant: (1) The total time spent on TC is less than that on HL; and (2) The improvement on TC is larger than that on HL. However, the p-value for them are 0.175 and 0.306 respectively. This means under the current size of samples, we are not able to prove our hypotheses. For future work, we need more participants in order to see if those are true.

5.3 Analysis on Results of Task 2

There are two contexts for the Task 2, {`lgv:Stadium`} and {`dbpediaowl:College`}. Before the study, we recognized that identifying errors is a much more difficult task for non-expert users, and it is well known that people might sometimes disagree on the Ontology models. Thus we told the participants when they started Task 2 that the answers would be very subjective, and they did not need to provide a perfect and complete list of errors. They only need to record any classes that they think are weird and suspicious.

We observed that most participants felt unconfident about some tags during this task, and they tended to skip a few following tags after they saw some tag that made them hesitant or frustrated. We also noticed that the native English speakers (5 participants including 2 in the trial) were obviously quicker at reading the tags and thus usually quicker at completing this task.

We examined all the answers, and checked if a participant had found any true errors. Again, the errors can be very subjective, some can be very controversial and we only consider the ones that are definitely incorrect. For example, in displays for the stadium class, there are classes of teams, which are mistakenly considered the same as their stadiums. In the display for the college class, we find classes such as websites, companies, and person. After examining all the answers, we find 7 of 14 (50%) participants found some true errors in Task 2.1, and 14 of 14 (100%) found some true errors in Task 2.2. We think one reason to explain why Task 2.2 is answered better is that there are more errors in the college class than in the stadium class, and another reason is in Task 2.1 there are many false

errors (the ones that seem to be an error but in fact are not) that distract participants' effort. We will analyze the false errors later in this section.

We also examined how well the participants completed Task 2.1 with both systems[2]. Ten participants used TC and four used HL for Task 2.1. We find 4 of 10 (40%) succeeded in finding some true errors using TC, and 3 of 4 (75%) succeeded using HL. Although the sample size is small, we think the results are reasonable in some sense. We also get some comments submitted by the participants which explain the problems with using TC for this task.

> "The problems is the errors may always occur in those tags that is quite small. If I want to find those errors as many as I can, that is not a good experience."

> "When the words are smaller it hurts your eyes to search through a list that is not indented in the same manner."

> "For task group 2, tag cloud based scheme is awkward. It is difficult to locate a detailed stuff based on words flushing the whole screen."

While TC makes larger tags more noticeable to users, it also makes the smaller tags less likely to get noticed. Also because of the layout, users may fatigue more easily while reading through the various sized tags on a page. In contrast, the tags in HL are well aligned, and users can quickly go through the list that start with some string if they

[2]This analysis was not done for Task 2.2, as both system resulted in successful task completion for all users.

think these tags can be skimmed for the task. For example, there are a lot of tags like "UniversityIn..." and users can usually go through the list more quickly.

We do not report the time spent in this task for several reasons. First, the questions in this task are open questions, and participants were allowed to stop at any time they felt they had produced sufficient answers. During the study, we also found that when they felt frustrated or tired, some participants would move on to the next task or step, while some others would take a short break and then resume the task. We also noticed that a participant was very likely to be affected if he/she found the other participant in the same group had completed all the questions and submitted the form.

Now we discuss the false errors with examples from participants' answers. The first category are the classes that are too abstract. For example, classes such as `schema:Thing`, `pos:SpatialThing`, `foaf:Agent` are very frequently recorded as errors. Usually experts or users with experience of Semantic Web will find it very natural to have these abstract classes in the high levels of the ontology hierarchy. However, from this study we find that most non-expert users, including participants with computer science background, will have difficulty understanding these classes without proper tutorials. The second category are the classes that are designed in unusual ways. For example, we find some classes from `freebase` like `freebase:common.topic`, `umbel:Attributes`, `gml:_Feature` are also very frequently chosen. Few people understand what these classes mean, but also feel uncertain to claim those are wrong. These classes can be very dependent on the domain where their ontology are designed for, however we feel it will cause misunderstanding to

most users even including some experienced Semantic Web users. The third category are the classes that represent categories of related topics. For examples there are many classes from `yago` that use the names of athletic teams as the local name of that class, such as `yago:SamsungLions` and `yago:ChicagoWolves`. These classes are usually considered as errors, if the participants did not realize that those are the team names representing their stadiums.

We wonder whether improving the interface could help reduce any misunderstandings. We think representing the hierarchy of classes is useful for clarifying the meaning of abstract classes. Although subsumption relationship were indicated in both systems, we noticed that participants were more likely to understand the folder-like representation for the hierarchy, compared with the gray-colored tags shown in the tag cloud which were used to indicate super tags of the context. Another idea is that probably we should develop some algorithm to discover or some syntax to denote those abstract classes, and hide them from the non-expert users.

5.4 Analysis on Survey Questions

There are 4 pairs of Likert-scale questions about the two systems. We plot the answers from all 14 participants in Figure 5.6. The answer ranges from Strongly Agree (SA) to Neutral (N) to Strongly Disagree (SD). From the pairwise comparison, we can see TC usually has a higher acceptance rate (i.e. more people agree with the statements) than HL, except the group of familiarity. The HL may seem more familiar to most users as the

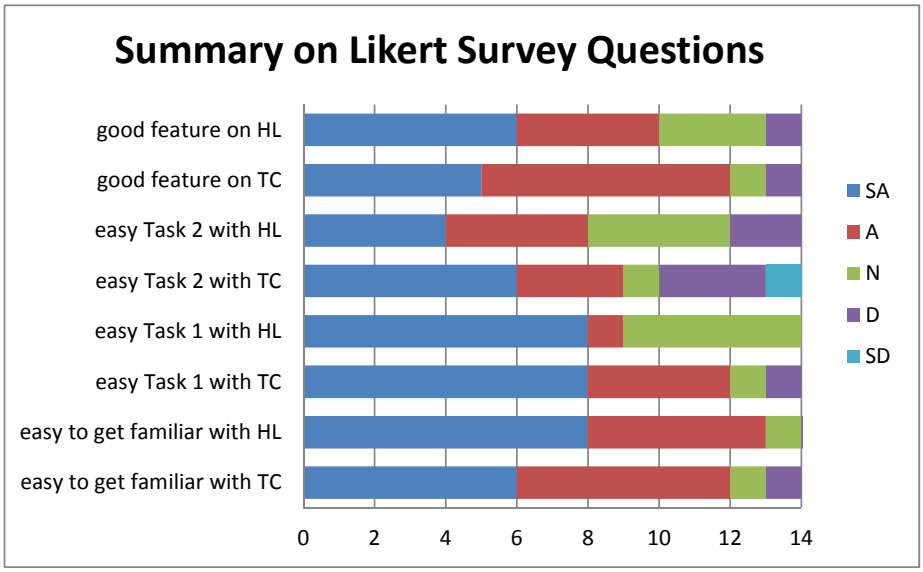


Figure 5.6: Pairwise comparison on Likert scale answers for both systems.

faceted browsing style is widely used in online shopping sites. We can also see that TC is most accepted as the tool for Task 1, which corresponds with our analysis on the objective results of Task 1. On the other side, TC also has a higher rate for disagreement on the statements, which makes it a more polarizing system. That is reasonable because some users may enjoy novelty, while others may take time to adjust to new paradigms. Among the questions, we see that TC gets most negative remarks on Task 2, and in Section 5.3 we have discussed possible reasons why TC is not as good as HL in Task 2.

We compare whether people prefer one system to another in each pair of questions. Let SA = 5 and SD =1, the difference in scores by TC and HL ranges from 4 to -4, where 4 indicates greatly in favor of TC and -4 indicates greatly in favor of HL. We plot the count of preference in Figure 5.7. Generally speaking, there is no evidence that one system is more subjectively favored than the other.

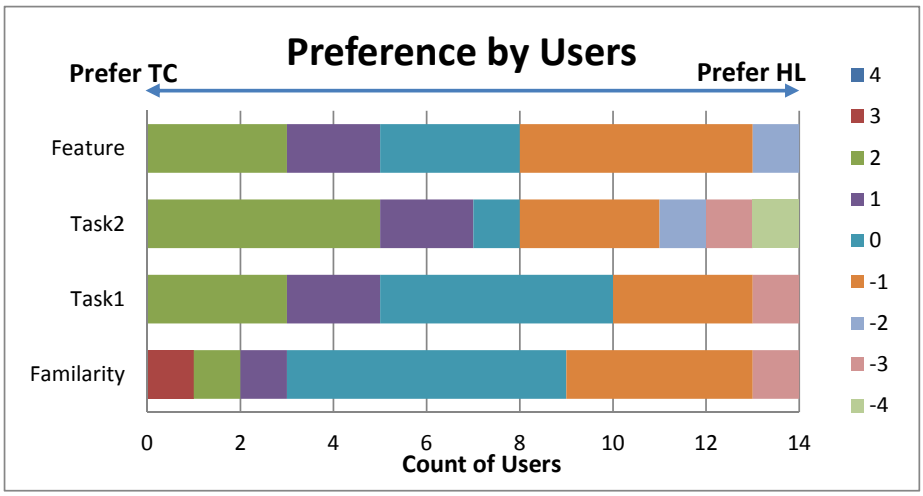


Figure 5.7: Count of Users group by system preference by combining scores in the pairwise Likert questions.

Is less time spent on one system the main reason why the participant prefers that system in Task 1? In order to check whether this is true, we compute the correlation between them. For each participant i, we compute the difference in total time spent on both system: $d_i = t_{HL,i} - t_{TC,i}$, and the difference in the Likert scores: $p_i = scr_{TC,i} - scr_{HL,i}$. However, surprisingly, we compute the correlation coefficient, and find $r = -0.44$. This means the two events have a moderate inverse correlation, and it is more likely to see a participant prefer the system on which he/she spent more time!

5.5 Discussion and Future Work

Although this study did not accomplish everything we had hoped, we feel we can still draw some initial conclusions. The most important one is that we proved that our tag cloud system can be used by non-expert users. The system we proposed is still a very immature

prototype system. Prior to this study, we only had anecdotal evidence that academic people can use it with minimal instructions. This study demonstrates that non-expert users are able to access a Semantic Web dataset and successfully complete some typical tasks as in the real world: they are able to find proper terms to form queries; and they are able to contribute to the Semantic Web community by suggesting potential errors. Since we were unable to find a similar tool that is available to explore a large scale Semantic Web dataset, we built a baseline system. By comparison with a baseline system HL, we also showed that the tag cloud system is accepted by 85.7% of users as a good tool for exploring a linked dataset.

There are many places we can improve for this study in the future. First of all, we need more participants. Currently we have only 14 participants, which is good enough to show the hints of trends. However, given that in the study each task often involves many parameters which divides the data points into many groups, we have only 3-4 in each group. Thus we can hardly make any statistically significant statement. Also the design of the study can be improved. We think Task 1 is a good start, but we want the participants to explore further based on their choice of combinations. Task 2 should be modified so that participants have more options to explore the data while still keeping a clear purpose. We think the current Task 2 requires too much time and effort which makes participants get tired, and this problem should also be resolved. Meanwhile, we should also keep searching for other similar systems and use that for comparison. It will provide participants more heterogeneous interfaces for participants to experience and then decide

which interface is better.

Chapter 6

Extending the Contextual Tag Cloud System

We believe we have a very useful system for users to explore the Semantic Web KB based on Linked Open Data. One of the key feature it provides is that users are able to observe the data distribution based on a set of ontological terms that they specify on-the-fly, which helps reveal the co-occurrence or patterns between different terms. The contextual tag cloud system provides a very user-friendly way for constructing template queries. We have discussed how the system could be use to explore merged collections of diverse data sets such as Linked Data. Could this same approach be beneficial for more focused data sets where the users are already familiar with the schema? In particular, can the system be used to detect interesting pattern or identify anomalies in such datasets? In this chapter, we discuss how to extend the tag cloud to a specific domain that does not use Semantic

Web data, and showcase some use scenarios. Then with experiments, we discuss whether the infrastructure for the ontological tag cloud still fit for this new dataset.

6.1 Extended Tags and Use Cases

Our scenario of interest is finding suspicious activity in the computer logs of a large enterprise. We use various synthetic datasets created by CERT [1] for use by the DARPA Anomaly Detection at Multiple Scales (ADAMS) effort. The dataset is based upon realistic data models of a large enterprise and contains 33 million events of 1000 users activities during 17 months[2]. The data is provided as logs (comma separated values, i.e., CSV) about computer usage, including logons, file accesses, web accesses and email communications. We noticed that for a specific kind of usage logs (as shown in Figure 6.1), the events almost always have the same set of properties, and showing the existence of a property is not very useful to users. Instead, the values of the properties contain more interesting information and potential patterns. So we define a tag as a property-value pair, i.e. $t = \langle p, v \rangle$, and extend the concept of tags to the following categories.

C0. Explicit Class. In the original dataset, we may find existing well-defined classes, if instances are categorized explicitly. If $\langle i, \text{type}, C \rangle \in \mathcal{S}$, where $\mathcal{S}$ is the set of all the triples in the dataset, we assign tag $t : \langle type, C \rangle$ to i. This is the same as our definition of class tags in Section 3.1.

[1] `http://www.exactdata.net/`, `http://www.cert.org/`
[2] Project dataset: R3V1

C1. Existential. Similar to $\exists P.\top$ and $\exists P.C$ defined in OWL, we define tags based on the property usages. If $\langle i, P, j\rangle \in \mathcal{S}$ we assign tag $t : \langle P, \top\rangle$ to i, and this is the same as our definition of property tags in Section 3.1. If in addition we know j is also an instance and $\langle j, \text{type}, C\rangle \in \mathcal{S}$, we also assign $t : \langle P, C\rangle$ to i.

C2. Discrete Enumerated Literal Categories. If the literal values of a property P is from a fixed set, each value, combined with P can be a category. If $\langle i, P, \text{“}L\text{”}\rangle \in \mathcal{S}$ we assign tag $t : \langle P, \text{“}L\text{”}\rangle$ to i.

C3. Continuous Literals. Some literals have values from infinite sets (e.g. real numbers) or there are too many to enumerate (e.g. timestamps). We can define a function f_P that maps the continuous value space to an enumerable value set. i.e. if $\langle i, P, \text{“}L\text{”}\rangle \in \mathcal{S}$ we assign tag $t : \langle f_P, f_P(L)\rangle$ to i, where the function name is used as the property of the tag. Sometimes P may have multiple functions that divide the space in different way or into different granularities.

C4. Text. Some literals (e.g. sentences or paragraphs) are too many to enumerate, but can be represented by a set/list of enumerated parts (e.g. terms, keywords, or topics). We define function *tokens* that tokenizes such literals into parts, and assign tag $t : \langle P', \text{“}L\text{”}\rangle$ to i, if $\langle i, P, o\rangle \in \mathcal{S} \wedge \text{“}L\text{”} \in tokens(o)$. Semantically, the new property P' is different from P, and can be treated as a composition property $P' = P \circ hasToken$ if we treat the text o as an instance with property “$hasToken$”.

We list some representative attributes and the way we extract tags from them and correspond each item below to our categories of tags:

userID, filename and URL. Here, we directly use their values as tags. Linking back to $C2$ in our data model, for an event i with user ID *user001*, a tag from its *userID* attribute will be $\langle$*userID*, "user001"$\rangle$, where the attribute *userID* becomes the property P;

date. Our date information contains the date and time of an event, e.g., for event i, we could have the following value as its *date*: *12/30/2010 09:20:23AM*. In this case, instead of directly using the attribute and its value as tags, we give a more fine-grained classification of time, including *MonthYear, TimeofDay, DayofWeek, Daytype and Hour*. Corresponding back to $C3$ in our data model, for the above given date, we then have the following tags assigned to event i: $\langle$*MonthYear*, "Dec2010"$\rangle$, $\langle$*DayofWeek*, "Thursday"$\rangle$, $\langle$*TimeofDay*, "Morning"$\rangle$, etc.; and here tag properties *MonthYear*, *DayofWeek* and *TimeofDay* are the mapping functions on the date, and tag values *Dec2010*, *Thursday* and *Morning* are the functions' values. Time information is continuous and a typical timeline may be good in showing the activities along the time; however, timelines will not sufficiently display recurring at multiple scales, while our way of tagging time provides a better possibility to explore recurring activities.

content. *content* refers to the actual file content, web page content and email content. We tokenize (simply by white spaces and punctuation) the content and treat each distinct token as a tag. This type of tag is what we call *Text* ($C4$) in our model, where the property P here is *content* and "L" refers to the individual tokens.

Comparing to the interface for browsing Linked Data, we add two new concepts to

group the the tags: a set of tags that represent a common facet constitutes a **block** and a collection of blocks that describe a particular dimension of the data constitutes a **tab**. Each tag $t = \langle p, v \rangle$ should belong to one and only one block, and typically the block collects tags with the same value for p. e.g. tags $\langle$*DayOfWeek*, "Sunday"$\rangle$ and $\langle$*DayOfWeek*, "Monday"$\rangle$ both belong to block *DayOfWeek*. Blocks like *DayOfWeek*, *MonthYear* appear under the tab "time". Figure 6.2 shows a snap-shot of the interface.

In Figure 6.1, we list all the mappings from the original attributes in the logs to the blocks and the tabs in the interface. Note some attributes are used for multiple blocks, for example, the date attribute is mapped to all the blocks in the Time tab, by various discretization functions. In addition to those general categories we mentioned above, we also apply a few special mappings. For example, "duration" is computed by the difference of time between two adjacent log-on and log-off activities; by comparing the sender and recipient to the user's email address, we can tag whether the email event is sending or receiving an email. Also note we do not have C1 mappings in it, because we only have the types of log events in this dataset. If we integrate more external knowledge, we can also have meaningful C1 category tags: e.g. if we have categories of websites, we can add tags for Web events to indicate what kind of website is involved in the event. We list some scenarios below for demonstrating the interface and use cases.

Monitoring popularity trends We can determine the most/least popular items in the data set. For example, sales logs could be used to understand customer preferences and analyze shopping trends. As in our dataset, from Figure 6.3 we find that the duration

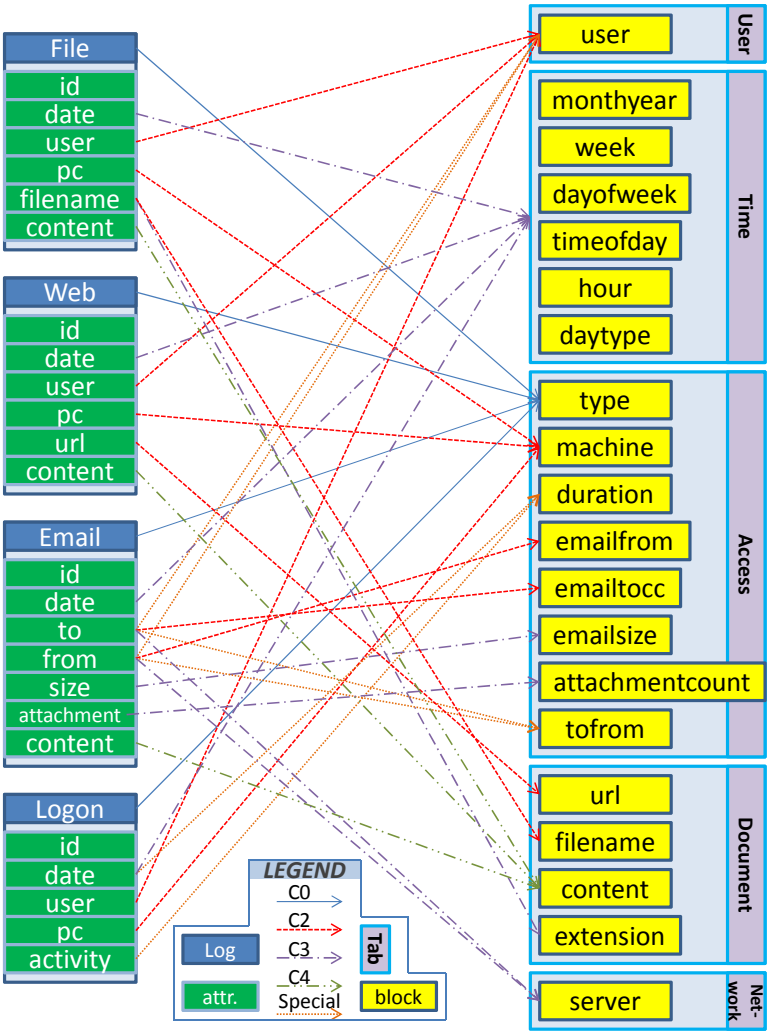


Figure 6.1: Mappings from attributes in logs to blocks and tabs in the interface.

for logon events in the night tends to be either very short (10 to 30 mins) or long (4-8 hours). Another example (in Figure 6.4) is that we find that the most events in the evening is web access.

Monitoring suspicious activity Due to the mission critical nature of data networks and systems of an organization, it is necessary to protect these systems from both internal

Figure 6.2: Tag cloud interface with multiple constraints showing the "Access" tab consisting of 5 blocks.

Figure 6.3: Duration of Night Logon Events

and external threats. Thus different types of logs such as email records, VOIP activity logs, Internet usage logs, etc. could be retained and monitored periodically. For example in the log data, we consider the set of email activities on Saturday nights. This set itself

Figure 6.4: Types of events in the evening.

is not suspicious, but a reasonable hypothesis is that it may include some anomalous activity, since the actors will attempt to conceal their behavior from coworkers, possibly by conducting operations during atypical work hours. Thus by examining tags in different blocks, we are able to inspect potential threats: Are there any suspicious keywords in the content? What are the emails of the contact people? Is anyone bcc'ed? How many attachments are in the email? All these questions can be answered by exploring our system, and if we find any suspicious tags, we can simply add that tag to the context and continue to inspect the resulting tag cloud. Figure 6.2 shows a step in this inspection process.

Profiling a user When we select a particular user as a constraint, we are actually presenting the profile of a user, with multiple aspects. In our dataset, we can answer questions like: Whom does he frequently exchange emails with? Does he access the Web more in the afternoon? Does he usually work at night? e.g. as shown in Figure 6.5 we find that the user "LKF0122" has no events recorded in the evening or night, has no file access, and the only machine he logged on is "PC-2051".

Comparing two patterns One might want to compare two scenarios to find the differences in pattern. For example, our system can also be used to visualize the evolution

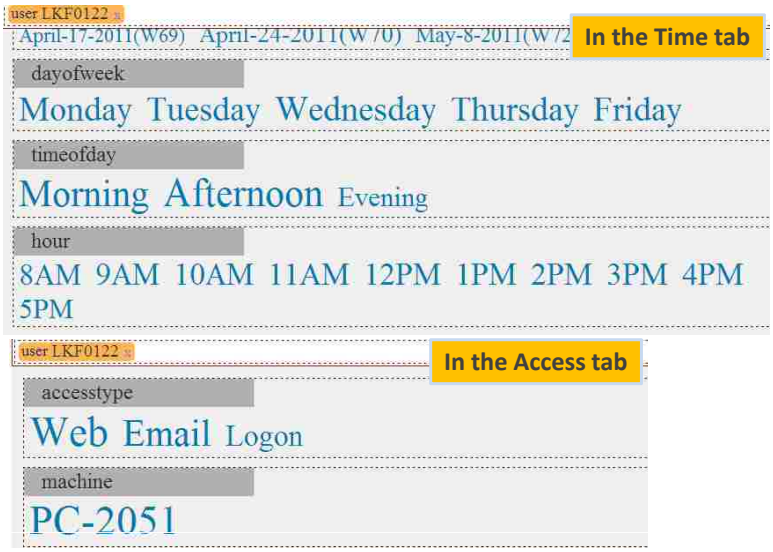


Figure 6.5: Profiling user "LKF0122"

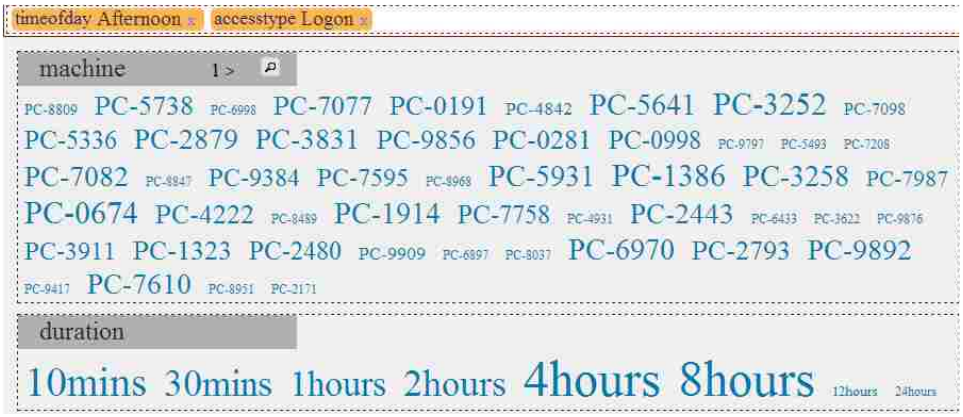


Figure 6.6: Duration of Afternoon Logon Events

of the terms and hence the topic of interest among the users. As in above-mentioned examples, we can compare between users, compare between access patterns. e.g. we can compare the duration of logons in the afternoons (Figure 6.6) with those in the nights (Figure 6.3). We can see that although there are still many short logons (less than 1 hour), the largest ones are the 4-8 hours. We think the comparison on the tag clouds of similar

contexts are usually interesting and can reveal important patterns (or changes in patterns). So we also implemented a comparison view. Figure 6.7 illustrates the comparison between (1) the activities of user "CSF0929" at any time, and (2) the activities of the same user in July 2010. The comparison tag cloud combines the two regular tag clouds and shows any tag that is in either of them. The font size of each tag indicates the absolute value of the difference for this tag between the two tag clouds, and the font color indicates whether the difference is positive or negative (the color of the tag corresponds with the context bar to indicate the more frequent side). This experimental feature currently supports customizable definition of the difference. In this example, the difference is defined as the change of the percentage that a tag takes in its context w.r.t. the number of events. From the comparison we can see, user "CSF0929" has more activities in the evenings and nights and in weekends in July 2010, comparing to his usual activities.

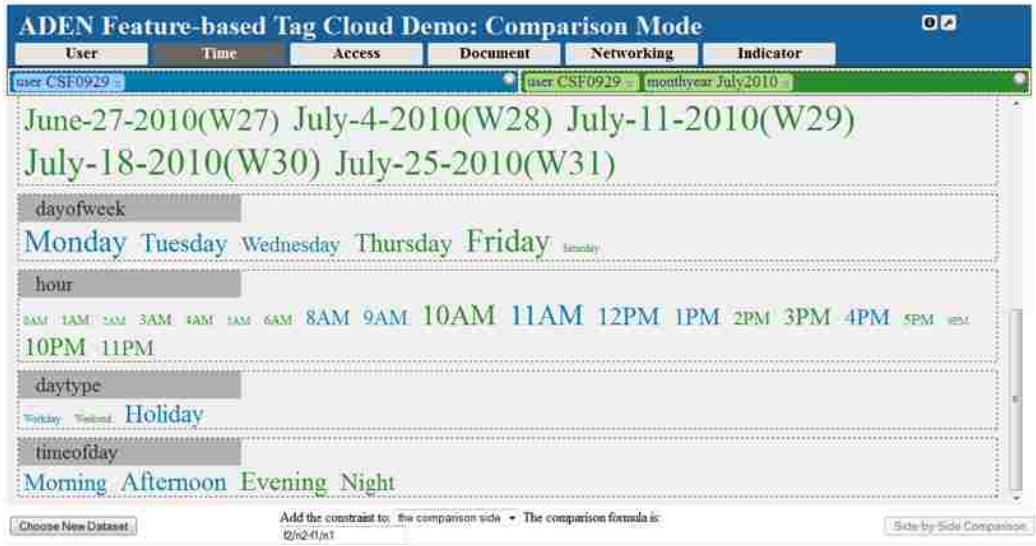


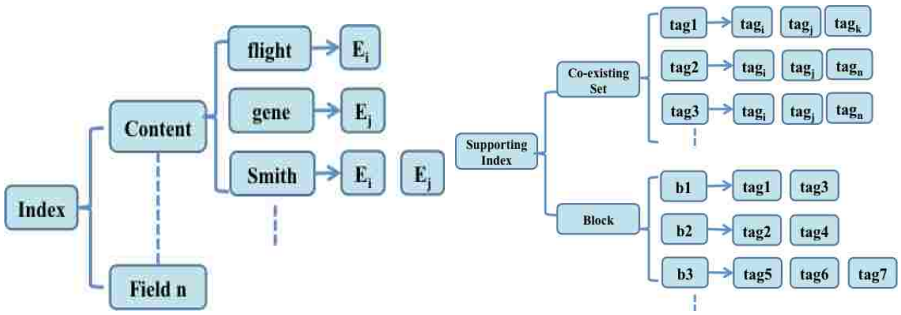
Figure 6.7: The comparison view for user "CSF0929" to check whether his activities in July 2010 is consistent with those in other months

6.2 Required Infrastructure Changes

The previous section has proposed several changes to the system. To support the characteristics of the new dataset, the modified definition of tags, and the block/tab nature of the interface, we need to modify the infrastructure accordingly in order to provide responsive system.

The first difference is how we index the events (i.e. the instances) and how we store the Co-occurrence Matrix. Since the request now is based on each block, the index and the matrix are optimized to be built based on blocks too. Our indexing structure is shown in Figure 6.8(a), where for each block (such as "Content") we have a corresponding indexing field, and in each field we index each possible value (such as "flight") with a posting list of events that have the ⟨block, value⟩ tag. The Co-occurrence Matrix is also built as an inverted index called Supporting Index as in Figure 6.8(b). There are two retrieval fields in the Supporting Index: Co-existing Set (*tagset*) and Block. In the Co-existing Set field, each indexing tag has a posting list of tags that co-occur with that tag. In the Block filed, each indexing block has a posting list of tags that appear in that block. By introducing the block names as either retrieval fields or indexing terms in a retrieval field, we can easily get the tags of a block by specifying the block name from both the Event Index and the Supporting Index.

We defined ontological axioms for tag entailment, which speed up both the preprocessing and the online computation. When building the supporting index, naively, we could issue the following conjunctive query to our event index:"p_1=v_1 AND p_2=v_2" for every



(a) Event Index Structure (b) Supporting Index Structure

Figure 6.8: Index Structures

pair of $\langle p_1, v_1\rangle, \langle p_2, v_2\rangle \in \mathcal{T}$. However we note that some combinations will never result in co-occurrences. Therefore, we carefully picked 30 tags (e.g. $\langle$*AccessType*, Email$\rangle$) or *tag groups* (e.g. $\langle$*Content*, *$\rangle$), and created a 30×30 *disjoint matrix*, as illustrated in Figure 6.9, in which the cell (i, j) indicates the disjointness (1 means they are disjoint) between the ith and jth tags (or groups). Disjointness was manually determined based on semantic conditions of the blocks and tags, resulting in 13% of the cells being marked as disjoint relation. For example, the tag $\langle$*AccessType*, Web$\rangle$ (f in Figure 6.9) is disjoint with the tag group $\langle$*Filename*, *$\rangle$ (l in Figure 6.9) because no files can co-occur with a web access in an event. Note, the matrix is symmetric, but the diagonal is not always disjoint. In particular, multi-values properties such as content, server-names (extracted from emails recipients and senders), contacts, and e-mail to/cc/bcc are not disjoint with themselves. When building the supporting index, the disjoint matrix is used to prune unnecessary queries to the event index. This disjointness is also used for pruning online computation. Because we can see some tag groups (i.e. blocks) are disjoint with some other tags or tag

groups, when any of these disjoint ones are in the context, we can directly ignore that block, without even querying the supporting index.

Besides the disjointness axioms, we also use the Supporting Index (for the same purpose as the Co-occurrence Matrix in Section 4.2) to reduce the number of tags that will require frequency (f_R) queries. Given block B and context $T = \{t_1, t_2, ...\}$, we issue a boolean query "block=B `AND` tagset=t_1 `AND` tagset=t_2 ..." to the supporting index, and the result will be a set of tags in block B that are likely to co-occur with the context. This pruning strategy is called **Precomputed Candidate Set (PCS)**, since the result from the precomputed supporting index is a set of candidates.

In some cases, we find that the context is actually very selective, and there are only a few instances that match the context. Instead of issuing queries from the candidate set, we can directly count the tags that appear in these matched instance. Notably, this will be inefficient when |Inst(T)| is very large and $|B|$, i.e. the number of unique tags in this block, is relatively small. Thus in practice, we use the following **Conditional Instance Processing (CIP)** rule: if $|B|/|Inst(T)| > \alpha_B$; otherwise, we use the naive approach which issues a conjunctive query for every tag t in the dataset. Here α_B is a constant parameter for each block B. In practice we let $\alpha_{Content} = 10$, and $\alpha_B = 100$ for all the other blocks. The Content block is specially handled because it is a multi-value block, accessing each instance would add counts to multiple tags at the same time. We did the estimation of α_B in a very experimental manner, and we think further investigation and a more theoretical estimation method on this is necessary for future work.

CIP and PCS are efficient in some different cases, so we combine these two approaches into a third one (**CIP+PCS**): we use the precomputed candidate set and apply a modified conditional instance processing rule. On every request, if $|T| > 1 \wedge |cand_{PCS}(B,T)|/|I_{sub}(T)| > \alpha_B$, we process matched instances; otherwise we use supporting index to get a smaller candidate set. Note that the key difference between this combined approach and the CIP only approach is whether we use the precomputed candidates from the supporting index or use all the tags from a given block as candidates. Also because PCS is guaranteed to provide all the co-occurring tags if $|T| = 1$, we do not consider to use CIP in this case.

6.3 Experiments

In order to test the online response time of our system, we generate random requests to simulate real users' behavior. Initially we let context $T = \emptyset$. We pick a random non-empty block B as the block that the user is interested in, and measure the computing time for that B. Then we pick a random tag $t \in B$ and add it to T. We repeat the random procedure of selecting B and t until $|T| = 5$ (i.e. 5 random web requests), as a sequence of a user's behavior. In total, 1000 random sequences (i.e. 5000 requests) are generated as the test set for evaluating our system. The time for each request given T and B is recorded for each of the three systems (CIP, PCS and CIP+PCS). Note that the time we recorded is only for the system to calculate the tags and co-occurrence frequencies, and does not include the time for web browser to render the data as a tag cloud ($< 0.2s$). Also

in real scenarios, when a user views a tab, there are actually multiple requests of blocks issued to the server.

In Table 6.1 we provide a comparison of systems by statistics on selected subsets of requests. In addition to the average and worst case time, we also define **responsiveness rate (rr)** as

$$rr = \frac{\text{number of requests with response time } \leq 0.8s}{\text{total number of requests}} \tag{6.1}$$

We choose $0.8s$ because after factoring $0.2s$ of rendering time, this is the limit for the user's flow of thought to stay uninterrupted according to Nielsen [35]. We can tell from the full test set, in general, PCS is better than CIP and CIP+PCS is the best. When $|T|$ increases from 0 to 4, the average times for both CIP and CIP+PCS decrease. This is because $|\text{Inst}(T)|$ usually decreases when $|T|$ becomes larger, and instance processing is very efficient when $|\text{Inst}(T)|$ is very small. However for the other cases, the supporting index provides more benefit. We also list a few subsets of requests selected by the requested blocks. It is clear to see that the blocks have more impact on response time than $|T|$. Consider block *Content* with $|B| = 88665$ and block *DayOfWeek* with $|B| = 7$. It is obvious that the blocks with smaller sizes require fewer queries to the event index and thus lead to shorter execution times. We also inspect the worst cases for the systems. For CIP, the worst case ($20.6s$) happens when $B = Content, |T| = 4$. $|\text{Inst}(T)| = 4623$ and $|B| = 88665$, so it chooses to process instances because there are too many tags to test in B. However from the supporting index, there are only 2290 candidates, the other two systems just test tags from this much smaller set, and thus are much faster for this request.

For PCS, the worst case ($11.0s$) happens when $B = Content, |T| = 3$. $|\text{Inst}(T)| = 1$ but the candidate set provided by the supporting index has 20063 tags, in this case, issuing count queries is not as efficient as CIP when it only has to process the single matched instance. So CIP+PCS is a balanced system that performs well on both cases when only CIP or PCS does well. On average, CIP+PCS has response times well under $1s$ and the worst case for most blocks is $< 2.5s$ except the User block. However, we find it is still not perfectly tuned. e.g. for blocks DayOfWeek and TimeOfDay, a perfectly tuned CIP+PCS system should have a worst case time close to that of PCS. So one area for future work is to find better ways to let CIP+PCS choose its strategy more wisely. Note that this combined strategy might also be useful for the LOD dataset, because it also has many situations where the context selects a small number of instances that could be used to quickly determine the candidate set. However, since the co-occurring matrix is usually sufficiently selective and therefore fast enough, without future experiments, it is unclear whether *on average*, the overhead of choosing strategies is worth the benefit of a more efficient choice.

The previous experiment shows that our system is more efficient if we use a supporting index however at the expense of consuming more time in the preprocessing step. The question is, is the supporting index worth the effort? We provide two sides of problem to answer this question.

Preprocessing Scalability. Comparing to the preprocessing steps in Section 4.1, there are a few differences: there is no split step because the data has no ontology or

Table 6.1: Statistics of Time for Random Request Test on Systems.

We compare mean response times (μ), responsiveness rates (rr) and maximum response times (max) on selected subset of requests. All the times here are in milliseconds.

Request Selection [count]	CIP			PCS			CIP+PCS		
	μ	rr	max	μ	rr	max	μ	rr	max
all [5000]	180.5	95.8%	20579	120.7	97.4%	10966	101.6	98.6%	5513
$\|T\| = 0$ [1000]	234.7	99.7%	3471	172.3	99.9%	826	207.6	100.0%	792
$\|T\| = 1$ [1000]	218.2	91.5%	7919	117.9	96.5%	1337	121.6	96.6%	1353
$\|T\| = 2$ [1000]	170.6	94.3%	7677	61.6	98.9%	5385	87.2	98.1%	5513
$\|T\| = 3$ [1000]	130.6	96.4%	11124	85.6	97.6%	10966	55.3	98.6%	2442
$\|T\| = 4$ [1000]	148.4	97.0%	20579	165.9	94.2%	9026	36.1	99.5%	1426
$B = Content$ [511]	463.4	88.5%	20579	529.4	82.8%	10966	227.9	95.9%	2442
$B = DayOfWeek$ [421]	97.8	97.9%	2424	43.2	100.0%	797	60.7	99.3%	2042
$B = TimeOfDay$ [482]	76.4	98.1%	2294	53.6	99.8%	964	67.9	99.6%	2208
$B = User$ [435]	200.2	92.4%	5747	108.5	95.6%	5385	109.7	95.9%	5513

sameAs; there is no flip step because the event is alway what we focus on; the sort step can be ignored because information about each event is already together in the log. So for this dataset, the preprocessing just includes the index/co-occur steps. In this experiment, our preprocessing step includes building the event index only (*Event*) and the supporting index as well (*Event + Supporting*) by querying the event index to get all tag pairs that have ever co-occurred. We show the runtime and index size for the two different approaches in Figure 6.10. We use logarithmic scale for both axes (runtime in seconds and index size in MB).

For runtime, we can see that building this supporting index requires one to two orders of magnitude more time since we need to query the event index to see if every pair of tags have ever co-occurred. This is very time-consuming when the event index is large. Although we adopted the disjoint matrix for optimization, it still took nearly 6.32 hours to finish building the pair of indices for 10 million log events. As for index size, when we increase the number of events, the size of the *Event* index grows faster than the other and the differences between *Event* and *Event+Supporting* become less substantial. This is because the size of our supporting index depends on the total number of tag pairs that have ever co-occurred. As the number of events continues to grow, the event index size grows rapidly since we continue to add more documents to the posting list of the terms in the index; however, the total number of co-existing tag pairs grow at a much slower pace because there are a fixed number of tags in our entire dataset and they can be sufficiently covered in a decent number of events. Note that the index structure here is different from

the one designed for the LOD, and different information (and also with different encoding methods) is stored in them, so we do not compare the disk usage here with the disk usage for the LOD system.

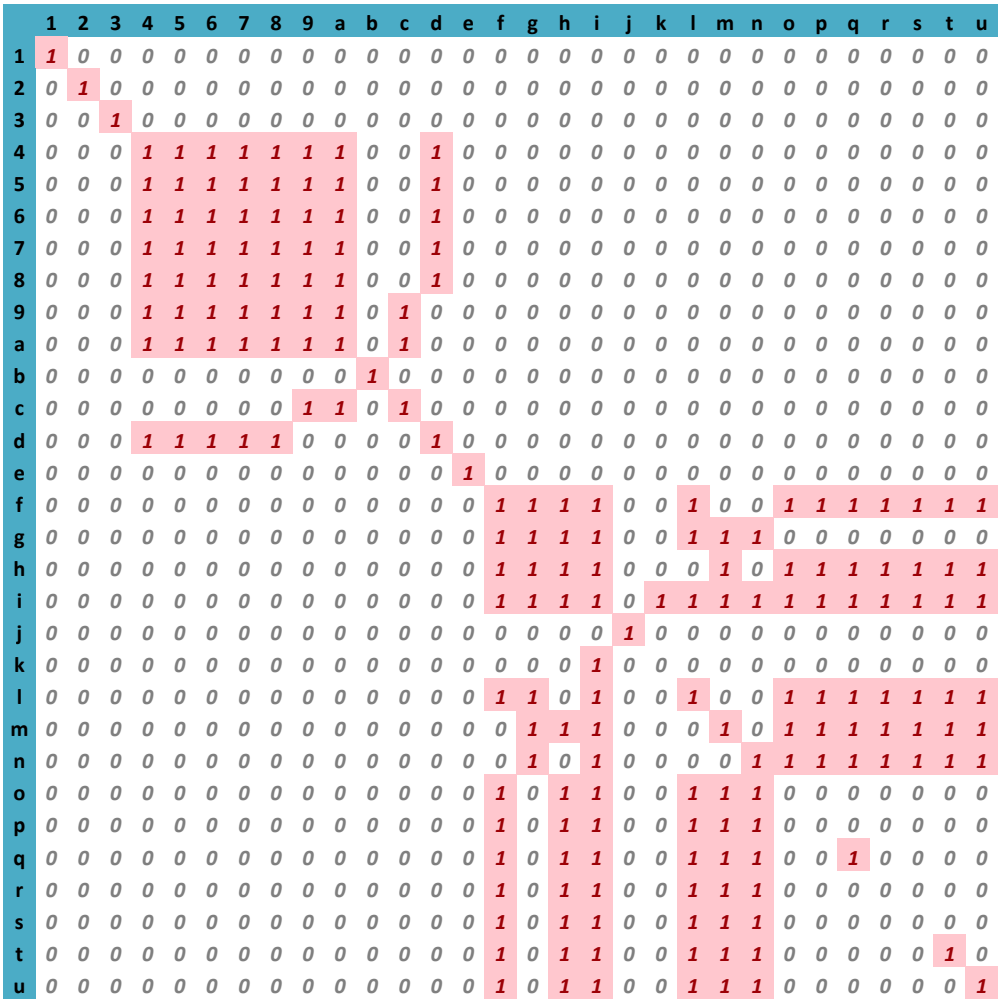
The Effectiveness of the Supporting Index. As we just discussed, building the supporting index adds a certain level of complexity in both runtime and disk space. Here, we briefly discuss how helpful the supporting index in terms of how many tag pairs it actually prunes as shown in Table 6.2. As we increase the number of events to be indexed,

Table 6.2: Pruning Capability of the Supporting Index

Number of Events	1K	10K	100K	1M	10M
|Tags in event index|	14,183	37,886	69,520	104,428	227,861
|Supporting index tag pairs|	1,679 K	7,328 K	27,638 K	96,556 K	292,178 K
Pruning Percentage	99.17%	99.49%	99.43%	99.11%	99.44%

the number of tags starts to grow and the same is true for the number of tag pairs in our supporting index. For all event sizes, the supporting index always gives a good pruning percentage calculated as $Pruning\ Percentage = 1 - \frac{|\text{Supporting Index Tag Pairs}|}{|\text{Tags in Event Index}|^2}$.

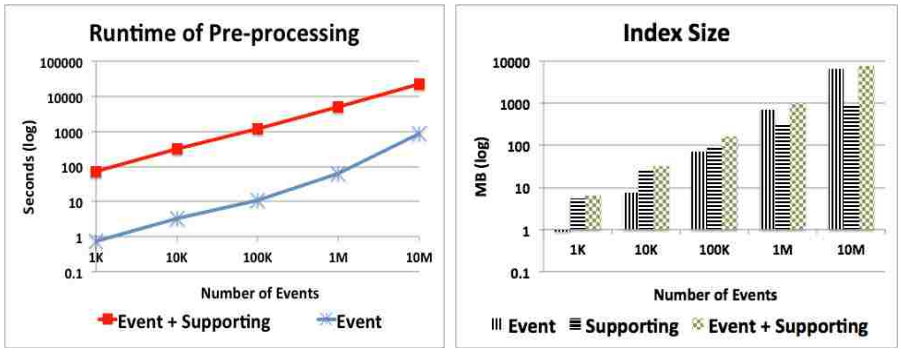
In conclusion, we find that building the supporting index is time-consuming, but its growth rate slows as the total amount of events grows; also we find that the supporting index provides a significant benefit for the pruning. The bottom line is building the supporting index is good for the system scalability.



	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
b	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
c	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
d	0	0	0	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
e	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
f	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	0	0	1	1	1	1	1	1	1
g	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	1	1	1	0	0	0	0	0	0	0
h	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	1	1	1	1	1
i	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
j	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
k	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
l	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	1	0	0	1	1	1	1	1	1	1
m	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0	1	1	1	1	1	1	1
n	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	1	1	1	1	1	1	1
o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	1	0	0	0	0	0	0	0
p	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	1	0	0	0	0	0	0	0
q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	1	0	0	1	0	0	0	0
r	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	1	0	0	0	0	0	0	0
s	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	1	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	1	0	0	0	0	0	1	0
u	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	1	1	1	0	0	0	0	0	0	1

Figure 6.9: Disjointness between tags or tag groups.

The tags are: 1=⟨*User, **⟩, 2=⟨*Department, **⟩, 3=⟨*MonthYear, **⟩, 4=⟨*DayofWeek, Monday*⟩, 5=⟨*DayofWeek, Tuesday*⟩, 6=⟨*DayofWeek, Wednesday*⟩, 7=⟨*DayofWeek, Thursday*⟩, 8=⟨*DayofWeek, Friday*⟩, 9=⟨*DayofWeek, Saturday*⟩, a=⟨*DayofWeek, Sunday*⟩, b=⟨*TimeofDay, **⟩, c=⟨*DayType, Weekday*⟩, d=⟨*DayType, Weekend*⟩, e=⟨*DayType, Holiday*⟩, f=⟨*Access, Web*⟩, g=⟨*Access, Email*⟩, h=⟨*Access, File*⟩, i=⟨*Access, Logon*⟩, j=⟨*Machine, **⟩, k=⟨*Content, **⟩, l=⟨*Filename, **⟩, m=⟨*URL, **⟩, n=⟨*Extension, **⟩, o=⟨*ServerNames, **⟩, p=⟨*Contacts, **⟩, q=⟨*EmailFrom, **⟩, r=⟨*EmailToCC, **⟩, s=⟨*EmailBCC, **⟩, t=⟨*EmailSize, **⟩, u=⟨*EmailAttachmentCount, **⟩



(a) Preprocessing Runtime (b) Preprocessing Disk Space

Figure 6.10: Preprocessing Scale-up Test

Chapter 7

Word Sense Disambiguation on Labels of Classes

Understanding the meaning of classes is a very important part for users to get familiar with the Semantic Web KB. When presented with a new dataset, users tend to assume that local names and/or `rdfs:labels` are sufficient for knowing the meaning of classes and properties. If they are unsure, they might look at the `rdfs:descriptions`, but it is rare that users will use ontological axioms to interpret ontological terms. Any attempt to interpret terms based on names and labels must take into account that the constituent word forms are sometimes ambiguous, and Word Sense Disambiguation (WSD) may be required for clarification. In addition to helping humans understanding the classes, WSD can also work as a component in a Question Answering system or any system that allows users to identify classes by natural language keyword input.

The typical WSD is the process of identifying which sense of a word (i.e. meaning) is used in a sentence, when the word has multiple meanings. In addition to traditional WSD, WSD on class labels involves two new problems: the role of ontological axioms in determining the context of WSD, and how to compute results that are meaningful to subsequent processes such as formulating a SPARQL query. Unless confidence is very high, a good intermediate result should not be a single top sense. Some WSD processes may produce a rank ordering of possible meanings, but then there is no information about about the confidence that the i-th meaning is more likely than the $(i + 1)$-th. Internal scores provide more information, but if these scores are not normalized consistently, it is impossible to know if the best score is a meaning that can be relied on with high confidence or not. A probability distribution for each ambiguous word provides the most complete information. As probabilistic models have proved successful in many other fields, we propose a novel WSD approach by using a probability model and calculating the distribution as the score results.

7.1 Utilizing WordNet

WordNet is a widely used lexicon for the English language. It groups English words into sets of synonyms called *synsets*, provides short, general definitions, and records the various semantic relations between these synsets. For convenience, in this chapter we define $\mathcal{T}$ the set of terms and $\mathcal{S}$ the set of synsets in WordNet, and describe several functions in WordNet as follows.

A *term* in $\mathcal{T}$ is a word or a phrase that can be found in WordNet. A term may have multiple senses, i.e. WordNet provides a function *syn* which takes a term as input and outputs a set of synsets of this term. i.e.

$$syn : \mathcal{T} \rightarrow 2^{\mathcal{S}} \tag{7.1}$$

Inversely, one could also find the *word forms* of a given synset by the function $wordForm$. , i.e.

$$wordForm : \mathcal{S} \rightarrow 2^{\mathcal{T}} \tag{7.2}$$

A term bound with a synset of it is a *word sense*. The gloss of a synset is the definition of this synset in WordNet. Let $\mathcal{D}$ be the set of all definitions in WordNet. The function *gloss* takes a synset as input and produces a list of words in the glossary of it as output, which is a human-readable definition of the synset. , i.e.

$$gloss : \mathcal{S} \rightarrow \mathcal{D} \tag{7.3}$$

A synset is related to other synsets. WordNet defines a set $\mathcal{E}$ of relation or edge types between synsets. For example, for a noun synset, there are hypernyms (super class), hyponyms (sub class), part holonyms (part of), etc. Note for a given synset the available edges is a subset of $\mathcal{E}$. A function $relEdge$ returns the available edges of a synset, i.e. $relEdge : \mathcal{S} \rightarrow 2^{\mathcal{E}}$. Given a synset and an edge type, we could get the synsets related via this type, which is a subset of the set $\mathcal{S}$ of all synsets in WordNet. A function $relSyn$

provides such information, i.e. $relSyn : \mathcal{S} \times \mathcal{E} \rightarrow 2^{\mathcal{S}}$. Besides the edges defined in WordNet, we add a new type of edge "DONE" which links every synset to *null*. The usage of this edge will be discussed later.

In addition, WordNet provides the statistical information for synsets and terms. A function $tagCount : \mathcal{T} \times \mathcal{S} \rightarrow \mathcal{N}$ tells the frequency (integer) of a word sense (i.e., a <term, synset> pair) against the text corpus used for WordNet[1]. From the corpus statistics, we can estimate different types of probabilities. We assume the probability of a given word sense is in proportion to the frequency that a term is used with that meaning in a given corpus.. Thus, the probability that some sense S is the meaning of a given term T is the ratio of the frequency of that word sense against the total frequency of that term, i.e.

$$P(S = s|T = t) = \frac{tagCount(s,t)}{\sum_{s_i \in syn(t)} tagCount(s_i,t)} \tag{7.4}$$

Similarly, the probability that one tends to use a term T for a given sense S is as follows.

$$P(T = t|S = s) = \frac{tagCount(s,t)}{synTC(s)} \tag{7.5}$$

where the function $synTC$ of a synset s counts the frequency by summing the tag counts of word senses including all possible variations on the word form of S, i.e.

$$synTC(s) = \sum_{w_i \in wordForm(s)} tagCount(s, w_i) \tag{7.6}$$

[1]We set the tag count of a word sense equal to 1 if it is 0 in WordNet to avoid a 0 prior probability for this word sense.

Equations (7.4) and (7.5) are probabilities w.r.t. the relations between terms and senses, which reflect our language habits. In some scenarios, we care more about the *concept frequency*, that is, how frequently we refer to an instance of a given concept, regardless of how frequently we might use the exact synset of it. For example, the term "Hominidae" is very rare in real world use, but the concept of this term is frequently encountered because it is a generalization of the concept of "human". Thus we define the function cf for counting the concept frequency of a synset S: for nouns it is the size of the set $\mathcal{HS}$ that consists of its direct and indirect hyponyms(subclasses); otherwise it is the same as the $synTC$ of S, i.e.

$$cf(S) = \begin{cases} \sum_{S_s \in \mathcal{HS}} synTC(S_s) & \text{, if } S \text{ is a noun} \\ synTC(S) & \text{, otherwise} \end{cases} \tag{7.7}$$

Equations (7.4) and (7.5) are the *corpus probability formulas* we get from statistics against a given corpus. A domain specific corpus, if available, can provide much better prior knowledge. Equation (7.4) gives us a prior probability distribution for the meaning of a given term, without considering the context. Our goal is to provide a better estimation of such probabilities with contextual information.

7.2 Probability with Context

With contextual information, we can provide better probability distributions for the meaning of terms. In this section, we formally define the problem probabilistically, apply some

assumptions, and break down the problem into small computable pieces.

In a Semantic Web document, there are many *URI resources*. A URI resource could be either a class, a property, or an instance that we want to match to something else. Each URI resource has various *associated texts*, such as *rdfs:label*, *rdfs:comment*, or even a parse result of its URI, which are the syntactic information sources. An associated text can be further split into zero or more WordNet *terms*. There might also be words that cannot be found in WordNet, which we ignore here.

To avoid discussion of minutiae, let us simplify the problem by only considering disambiguation within a single ontological document. In real world applications, we might also want to consider a set of ontological documents that contain mappings and other alignment axioms; or even consider the whole KB so that the associated texts of instances are also considered and the RDF triples related to instances are available as clues for disambiguation. Without loss of generality, such a KB, or a set of documents can be viewed as a virtual document.

Now we formally define our problem. Given an ontological document O, and a WordNet term T appearing in the associated text of an RDF resource U,we want to find the probability that this T means the sense S_0, i.e. $P(S_0|O, U, T)$. The condition consists of three parts: the ontology, the URI resource, and the term we want to disambiguate. The discrete random variable S_0 has a domain of $\mathcal{S}$ and stands for the event that T means S_0. All the possible senses are exclusive and exhaustive, thus the sum of all possible senses

should be 1, i.e.

$$\sum_{s \in syn(T)} P(S_0 = s|O, U, T) = 1 \tag{7.8}$$

While T constrains the possible values of S_0, O and U are actually the context that have effect on the distribution. Equation (7.8) defines an ideal probability distribution without information loss, however we have to make some simplifications to estimate it.

Researchers in WSD usually simplify the condition part to some context. The context in theory could be anything, such as very rich structured data. It is basically whatever we want to know from the document in the process of WSD. In traditional WSD, the context is usually defined as a bag of words. We follow this tradition and also define the context in our problem as a bag of terms $W_1, W_2...W_n$, and replace O and U with these terms. i.e.

$$\mathbf{P}(S_0|O, U, T) \approx \mathbf{P}(S_0|T, W_1, W_2, ..., W_n) \tag{7.9}$$

In traditional WSD, such a bag of words is usually the neighboring words of the target word in the free text document. A window size is set to decide how many words around the target word are included. We shall define the *axiom distance* which is similar to the window size for selecting context.

We first define the set of *relation triples* as all the explicitly stated triples in the document, excluding the ones that use a term from the RDF/RDFS/OWL namespace as its subject or object. Based on these relation triples, we can draw an undirected graph, the *relation triple graph*: each node stands for a unique RDF resource, (which can be a

blank node in RDF graphs), and every two nodes, including the properties, that appear together in at least one relation triple are connected with an undirected edge. For any two URI resources, we define the axiom distance as the number of edges in a shortest path connecting them in this relation triple graph [2]. An example is given in Figure 7.1.

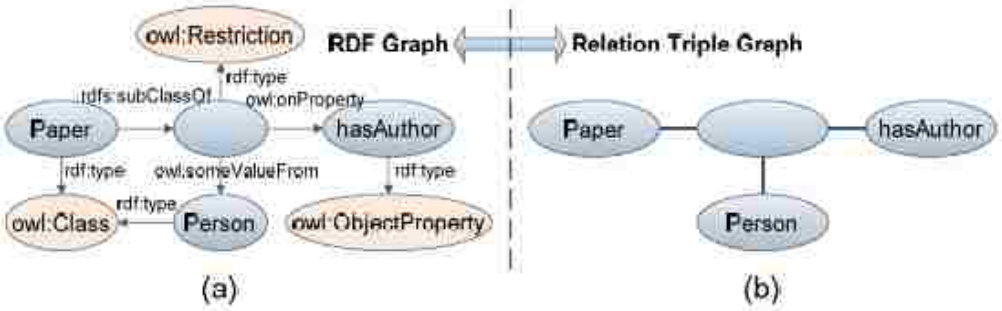


Figure 7.1: An example of axiom distances.

There are three URI resources in this example: Paper, hasAuthor, and Peron. The axiom distance between any two of them is 2.

We define the function *context* as follows. It takes three arguments. The first is the term T we want to disambiguate, the second is the URI resource U of which the label contains T, and the third is an integer that indicates the maximum axiom distance. The output is a bag of words that appears in the labels of URI resources within axiom distance of d of U excluding the ones identical to T. There are two special cases of this function. If we set the axiom distance $d = 0$, it means we only consider the terms that appear in U's label. If $d = \infty$, we consider the labels of every URI resource within the connected RDF graph that contains U. [3]

[2]We consider elements within the same parseType Collection to be connected to an anonymous node, and the distance between any two elements in such collections is 2

[3]In practice, there might also be resources that have no axioms in the ontology. The context selection problem in such cases is out of the scope of this paper.

Banek et al. [3] use a very similar way to find context, but they only consider class names in 4 kinds of axioms, i.e. subclass, superclass, domain and range. Our definition tries to obtain more context words by considering all the axioms and all URI resources with associated texts. It is worth pointing out that our approach of finding context uses the structure in RDF graphs, but does not use the semantics in it. This makes our approach still a syntactic matching process, and thus is not redundant with any subsequent semantic based matching processes that may occur.

Once the context is defined, we can further derive the formula in Equation (7.9) to computable parts as follows.

$$\begin{aligned} P(S_0|T, W_1, ..., W_n) &= \frac{P(W_1, ..., W_n, S_0|T)}{P(W_1, ...W_n|T)} \\ &= \frac{P(W_1, ..., W_n|T, S_0) \cdot P(S_0|T)}{P(W_1, ...W_n|T)} \\ &= \frac{1}{P(W_1, ...W_n|T)} \cdot P(S_0|T) \cdot \prod_{i=1}^{n} P(W_i|S_0, T) \end{aligned} \tag{7.10}$$

We first apply Bayes' rule, then apply the naive Bayes assumption that the occurrence of each W_i in the bag is conditionally independent with others given the disambiguation target word sense, and we have Equation (7.10). We can interpret this equation as follows. The probability before derivation is the chance that term T has the sense S_0 when a bag of words $W_1, ..., W_n$ co-occur in the context. In the resulting formula, $P(W_1, ...W_n|T)$ is the probability that the bag of words co-occur given that T occurs. Since Equation (7.8) holds, this part is just a normalization factor for estimating the probability, so we do

not need to calculate it if we intend to calculate the entire distribution. $P(S_0|T)$ is the corpus probability in Equation (7.4). The product of $P(W_i|S_0, T)$ for all $1 \leq i \leq n$, is the co-occurrence of W_i given the word sense (S_0, T). It can be interpreted as the probability that each term W_i is mentioned when people attempt to define something referred by the target word sense. It tells the relatedness between a term and a word sense. While using WordNet, the condition that a word sense is given is almost the same as the condition that its synset is given, because from the aspect of calculation, we get almost the same information from a synset as from the synset and its word form[4]. Thus we can use the approximation as follows.

$$P(W_i|S_0, T) = P(W_i|S_0) \tag{7.11}$$

While diverse approaches of estimating $P(W_i|S_0)$ may be chosen, again we follow the most common one in traditional WSD: the relatedness between synsets. Now we try to transform and relate $P(W_i|S_0)$ to $P(S_y = s|S_0), s \in syn(W_i)$, where S_y is the meaning of W_i.

The intuition of estimating relatedness between synsets is that more information from WordNet can be utilized if we investigate synsets. $P(W_i|S_0)$ is the probability that W_i is used in the ontology as part of the definition of S_0. $P(S_y|S_0)$ is the probability that the person thinks of the synset S_y of W_i when attempting to define S_0. We can model the cognitive process with the Bayesian Network reflecting the causal relationships [37] as

[4]they provide almost the same information about relations to other synsets or terms except the "antonym" relation, which we do not use in our algorithm.

follows.

$$S_0 \rightarrow S_y \rightarrow W_i$$

Bayesian networks are Directed Acyclic Graphs whose nodes represent random variables in the Bayesian sense, and the edges represent conditional dependencies; nodes that are not connected represent variables that are conditionally independent of each other. A causal network is a Bayesian network with an explicit requirement that the relationships be causal. We assume the conditional independence based on the following modeling of how the W_i appear in the context of S_0. The person first has a synset S_0, or say a concept, in mind. This S_0 leads the person to think of another synset S_y, with some probability, in the purpose of defining or explaining this concept in the ontology. At last this S_y is represented with the term W_i by this person. Many synsets can appear given S_0 with some probability, however only the synsets $S_y \in syn(W_i)$ have some probability to cause W_i. Note here S_y is a hidden variable with discrete values. Following the Bayesian Network rules, i.e. the assumptions of conditional independence, we have the following equation.

$$\begin{aligned} P(W_i|S_0) &= \frac{\sum_{\forall s} P(S_0)P(S_y = s|S_0)P(W_i|S_y = s)}{P(S_0)} \\ &= \sum_{s \in syn(W_i)} P(S_y = s|S_0) \cdot P(W_i|S_y = s) \end{aligned} \tag{7.12}$$

$P(W_i|S_y)$ is the corpus probability in Equation (7.5). The probability $P(S_y|S_0)$ reflects relatedness between synsets.

7.3 Estimation of Relatedness between Synsets

In order to find the relatedness between two synsets S_0 and S_y, we may need to explore the synset graph in WordNet because S_0 and S_y might not be directly related but are indirectly related via other synsets. Thus, we start the *synset expansion* from the given synset S_0 with the goal of finding chains to S_y. In such an expansion, the process that people think of more synsets starting from S_0 is also simulated, thus we propose a model and algorithm that estimates $P(S_y|S_0)$.

7.3.1 Synset Expansion Model

We model the expansion as steps of exploration to neighbors in the synset graph from the given synset S_0, with probabilities of deciding which synset to choose at each step. A step of expansion consists of two decisions. First it chooses the WordNet relation type for this step. Some relation types such as "hypernym" have higher probabilities than others, because the connections to other synsets often pass their hypernyms. For example, the synset *cat#n#1* is connected to *paw#n#1* via its hypernym *feline#n#1*. $P(E_1|S_0)$ denotes such probability. In the real world, this reflects the probability that one thinks of a WordNet relation type E_1 when he tries to think about expansion of a synset S_0 in order to define it. The event of deciding a type of WordNet relation edge given the current synset has exclusive and exhaustive values, i.e.

$$\sum_{e \in relEdge(S_0)} P(E_1 = e|S_0) = 1 \tag{7.13}$$

The second decision of expansion continues with a synset that follows the selected relation edge, and a related synset is selected with some probability, $P(S_1|S_0, E_1)$. This can be viewed as the probability that one thinks of a synset S_1 when he tries to think about a synset related to S_0 with a given type of relation E_1. Similarly, the event of deciding the synset following the relation edge we have chosen also has exclusive and exhaustive values, i.e.

$$\sum_{s \in relSyn(S_0, E_1)} P(S_1 = s|S_0, E_1) = 1 \tag{7.14}$$

Following Equation (7.13) and (7.14), we can derive $P(S_y|S_0)$ at the first step of expansion as follows.

$$\begin{aligned}
& P(S_y|S_0) \\
= & \sum_{E_1 \in relEdge(S_0)} P(S_y, E_1|S_0) && (7.15) \\
= & \sum_{E_1 \in relEdge(S_0)} P(S_y|S_0, E_1) \cdot P(E_1|S_0) && (7.16) \\
= & \sum_{E_1 \in relEdge(S_0)} \sum_{S_1 \in relSyn(S_0, E_1)} P(S_1, S_y|S_0, E_1) \cdot P(E_1|S_0) && (7.17) \\
= & \sum_{E_1 \in relEdge(S_0)} \sum_{S_1 \in relSyn(S_0, E_1)} P(S_y|S_0, E_1, S_1) \cdot P(E_1|S_0) \cdot P(S_1|S_0, E_1) && (7.18)
\end{aligned}$$

Equation (7.15) and (7.17) are derived by marginalization. Equation (7.16) and (7.18) can be derived by reforming the conditional probability. Equation (7.18) is the result of first step expansion. $P(S_y|S_0, E_1, S_1)$ shows that we expand the synset S_0 to its 1st level neighbors, if we can further find the relatedness between S_1 and S_y, we know that S_0 and S_y are somehow indirectly related via S_1. Then we can continue the expansion at the

second level between S_1 and S_y.

We define a *chain* after the l-th expansion as $C_l = S_0, E_1, S_1, ...E_l, S_l,\ l = 0, 1, ...$ We now show the formula for the (l+1)-th expansion in general case. Note that when $l = 0$, the expansion is the same as above. Deriving the formula is similar to Equation (7.15)-(7.18).

$$\begin{aligned} & P(S_y|C_l) \\ = & \sum_{E_{l+1} \in relEdge(S_l)} \sum_{S_{l+1} \in relSyn(S_l, E_{l+1})} P(S_y|C_{l+1}) \cdot P(E_{l+1}|C_l) \cdot P(S_{l+1}|C_l, E_{l+1}) \end{aligned} \tag{7.19}$$

Equation (7.19) suggests a recursive algorithm for calculating the relatedness probability for two different synsets. We should also define the exit of recursion, i.e. at some step we should stop expanding the chain C and assign some value to $P(S_y|C)$. We implement it by adding an edge type "DONE" with a small probability at each step. This "DONE" edge expands the last synset S_l of the current chain C_l and links to *null* with probability 1. It indicates we want to force the expansion of this branch to stop and see the direct relatedness between synset S_l and S_y. We further make the assumption that only the last synset in the stopped chain affects the probability.

$$P(S_y|C_l, E_{l+1} = \text{DONE}, S_{l+1} = null) = P(S_y^D|S_l) \tag{7.20}$$

$P(S_y^D|S_l)$ is the probability that S_y is directly referred given S_l. We use S_y^D to denote the event that S_y is directly related, which also has discrete values. We shall discuss estimation of *direct relatedness* later. Note that the chain here in our probabilistic model is very different with the Markov chains. We do not make any independence assumption during the expansion.

Once we have that defined, we have a finite set of chains $\mathcal{CS}$ to be expanded, then we can rewrite $P(S_y|S_0)$ as follows in a simpler way.

$$P(S_y|S_0) = \sum_{c \in \mathcal{CS}} P(S_y|C = c)P(C = c|S_0) \tag{7.21}$$

$P(C = c|S_0)$ is the probability of the chain c that starts with S_0. Let L be the total number of steps of expansion in c (i.e. the length of c), c_l be the sub chain at the l-th expansion, e_l and s_l be the edge and synset selected at the l-th step. Then the current chain can be viewed as the result of a series of actions of adding e_l and s_l to the chain $(1 \leq l \leq L)$. So the probability is the product of the probabilities of them. i.e.

$$P(C = c|S_0) = \prod_{l=0}^{L} P(e_{l+1}|c_l) \cdot P(s_{l+1}|c_l, e_{l+1}) \tag{7.22}$$

Equation (7.21) provides another way of understanding the nature of our model. We can also model a Bayesian Network causal graph that leads to Equation (7.21).

$$S_0 \rightarrow C \rightarrow S_y$$

One intuition of exiting the expansion is that we should stop expansion if the chain is too long. It is unlikely that one synset will remind people of another synset if this is only a distant indirect relation. Mathematically, it means the chain has a very low probability, i.e. $P(C|S_0) < \epsilon$. In this case, we ignore the further expansions that are unlikely to happen, and force the chain to stop expansion by adding a DONE edge with probability 1 after it. i.e.

$$P(S_y|C_l) = P(S_y|C_l, \text{DONE}, null) = P(S_y^D|S_l) \tag{7.23}$$

Another problem is cyclic chains. Mathematically we have no problem in computation, because cycles make the probability of the chain $P(C|S_0)$ decrease and as the length of the chain approaches infinity, its probability approaches 0. However in reality, we believe people tend to avoid such cyclic thinking in their mind when they try to associate synsets. Thus we remove those expansions that lead to cycles from the possible expansion branches. We shall discuss how we decide possible edges for expansion later in Section 7.3.3.

There are three probabilities we shall estimate: the direct synsets relatedness probability $P(S_y^D|S_l)$, the conditional edge expansion probability $P(E_{l+1}|C_l)$, and the conditional synset expansion probability $P(S_{l+1}|C_l, E_{l+1})$. We first introduce our simple estimation on $P(S_{l+1}|C_l, E_{l+1})$. We assume the probability that C_l is expanded to S_{l+1} in edge E_{l+1} is decided by $cf(S_{l+1})$ in Equation (7.7), for those synsets that are targets of that edge E_{l+1}. In consistency with Equation (7.14), we have the normalized estimation.

$$P(S_{l+1}|C_l, E_{l+1}) = \frac{cf(S_{l+1})}{\sum_{s_e \in \mathcal{AS}} cf(s_e)} \tag{7.24}$$

In the real world, this equation implies that people are more likely to think of synsets that are frequently met. The set $\mathcal{AS}$ is the set of available synsets that are in the set of $relSyn(S_l, E_{l+1})$ but not in the current chain C_l.

7.3.2 Estimation of Direct Relatedness between Synsets

$P(S_y^D|S_l)$ tells the direct relatedness between synsets. Here we introduce three different ways to estimate it. A straightforward idea is that we can consider the case that S_y is the same as S_l. Thus we have the first estimation.

$$P_1(S_y^D|S_l) = 1 \text{ iff. } S_y = S_l \text{ , 0 otherwise} \tag{7.25}$$

However in practice this may not perform well, because it is useful only if we find a chain connecting two synsets S_0 and S_y. Since WordNet does not provide every possible connection between synsets in its synset graph, merely depending on finding explicit chains often fails. For example, there exists no reasonable chain between *person#n#1* and *name#n#1*, which we know are somehow related.

One approach to overcome this problem is using the gloss in WordNet. If some word form T_y of S_y happens to appear in the gloss of S_l, it is evidence that they are related, or mathematically $P(S_y^D|S_l) > 0$. This gives us an approach to estimate $P(S_y^D|S_l)$ with $P_2(S_y^D|S_l)$ based on gloss.

$$P_2(S_y^D|S_l) = \max_{T_y \in wordForm(S_y)} p(T_y, gloss(S_l)) \cdot P(S_y|T_y) \tag{7.26}$$

There are two parts in the equation. The first part $p(T_y, gloss(S_l)$ is a function that tells the *portion* of T_y in the gloss of S_l, which will be defined soon. The second is the corpus probability in Equation (7.4), which is the probability that T_y means S_y. We try to match all possible word forms of S_y to the gloss and use the max likelihood. In practice, every word in either the word form or the gloss is stemmed before matching. This allows for small word variations and even enables the match across different part-of-speech. Thus it increases the chance that matches are found, but may lower the precision.

An easy definition of the function p could be the ratio between the numbers of words, however this makes the estimation of $P_2(S_y^D|S_l)$ biased towards common senses, because the senses that have common word forms are much more likely to be matched in the gloss. Thus we use the inverse document frequency(idf)[5] to determine the importance of the words and compute the portion.

$$p(T_y, gloss(S_l)) = \begin{cases} \frac{idfSum(T_y)}{idfSum(gloss(S_l))} & \text{, if } T_y \in gloss(S_l) \\ 0 & \text{, otherwise} \end{cases} \tag{7.27}$$

We record the document frequency of all the stemmed terms in the gloss of all synsets in WordNet. $idfSum$ is the function that sums the idf of words given a bag of words.

The gloss based estimation helps find the missing relatedness between synsets in WordNet, but still it does not find all. Also we do not want the relatedness between synsets at any time to be 0, a very small probability is better. Note these direct relatedness probabilities are the leaves in every branch of the expansion. If $P(S_y^D|S_l)$ is 0 for every

[5]If a word appears in n documents, then for this word, $idf = 1/n$.

chain, then according to Equation (7.20) and (7.21), $P(S_y|S_0) = 0$. If $P(S_y|S_0) = 0$ for every sense S_y of W_i, then according to Equation (7.12) and (7.10), $P(W_i|S_0) = 0$, thus $P(S_0|T, W_1, ..., W_n) = 0$. Thus the whole disambiguation is very sensitive to the selection of context, which is not desirable. Thus we define $P_3(S_y^D|S_l)$, the smooth approach of estimating $P(S_y^D|S_l)$.

$$P_3(S_y^D|S_l) = \frac{synTC(S_y)}{\sum_{\forall S} synTC(S)} \tag{7.28}$$

This simply means the conditional probability is the same as the probability of encountering the synset in the corpus.

With theses three approaches, we have three estimators for $P(S_y^D|S_l)$. We use a simple linear combination method as follows.

$$P(S_y^D|S_l) = \sum_{k=1}^{3} a_k P_k(S_y^D|S_l) \text{ , where} \sum_{k=1}^{3} a_k = 1 \tag{7.29}$$

There are some heuristics for deciding the weights a_k. The larger a_k is, the better we trust that estimation. Think of one extreme case when $a_3 = 1$. It will always return the same value for the expansion given different S_0, thus it can not disambiguate at all. Making $a_1 > a_2 > a_3$ seems reasonable, because the estimations from 1 to 3 become less reliable. However, as the chain grows longer, the first two approaches gain larger estimation errors, and thus become less reliable. So we make these weights the functions of the current chain probability. The values of both a_1 and a_2 decrease to some small non-negative values as the chain probability goes down. The functions can be defined very differently. If we assume

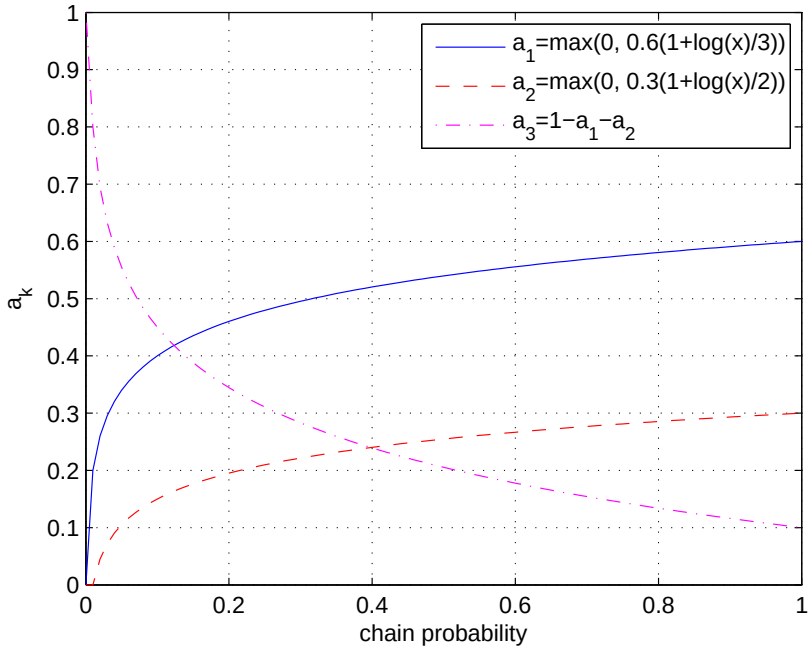


Figure 7.2: An example group of functions. The x-axis is the chain probability. During expansion, the probability goes down, the values of a_k change from right to left.

that the second estimation is more likely to accumulate errors, we can make a_2 decrease with a faster rate as the chain probability goes down. a_3 becomes dominant as both a_1 and a_2 decrease. This can be interpreted as when the chain becomes long enough, the estimation that any synset can be related becomes more accurate. An example group of functions is given in Figure 7.2. Note that the curves should be read right-to-left, because the chain probability decreases as the expansion goes on. We choose log functions because the chain probability changes approximately exponentially step by step, and we want the change of a_k to be approximately linear w.r.t. the steps.

7.3.3 Estimation of Conditional Edge Expansion Probability

$P(E_{l+1}|C_l)$ can be interpreted in the real world as the probability that one thinks of a relation type E_{l+1} given the current chain C_l in mind. We can predefine the weights for different types, that is $weight(E)$ for every relation type E. Then we can estimate $P(E_{l+1}|C_l)$ by normalizing the weights of available edges.

$$P(E_{l+1}|C_l) = \frac{weight(E_{l+1})}{\sum_{E \in \mathcal{ES}} weight(E)} \tag{7.30}$$

The normalization is required by Equation (7.13). $\mathcal{ES}$ is the set of available edge types given C_l. It is a subset of edges of the last synset S_l in C_l. To avoid cyclic chains, we prevent edges in $\mathcal{ES}$ from linking S_l to some synset that is already in the chain.

$$\mathcal{ES} = relEdge(S_l) \cap \{E | \exists S \in relSyn(S_l, E) \text{ and } S \notin C_l\} \tag{7.31}$$

Following Hirst and St-Onge's idea [20] to consider the "number of times the chain changes direction", we can modify Equation (7.30) to "encourage" the chain to keep its direction, which means the adjacent edge types in the chain are the same. The function aug boosts the weight if the chain keeps the direction. Note, that in order to ensure that the distribution sums to one, we must also include aug in the denominator.

$$aug(E_l, E_{l+1}) = \alpha > 1 \text{ if } E_l = E_{l+1} \text{ , 1 otherwise} \tag{7.32}$$

$$P(E_{l+1}|C_l) = \frac{aug(E_l, E_{l+1})weight(E_{l+1})}{\sum_{E \in \mathcal{ES}} aug(E_l, E_{l+1})weight(E)} \tag{7.33}$$

7.4 Preliminary Experiments and Discussions

We test our approach on the dblp ontology[6], which is is adapted from the XML schema[7] of the DBLP Computer Science Bibliography[8], and has 22 classes and 23 properties. The disambiguation targets are the 39 ambiguous noun terms (i.e. each term that has more than one synset) from *rdf:label* in the ontology. The ground truth is gained by collecting online votes from 5 students who are familiar with the ontology. The base line that our approach is compared to is the corpus probability in Equation (7.4). We compare two things: (1) the accuracy, i.e. the percentage that the top sense is correct; and (2) the probabilities of the correct sense.

In this preliminary experiment, we cannot demonstrate the results of every possible combination of parameters. Instead, we set the axiom distance level for context $d = 1$, the constant-direction augment factor $\alpha = 1.5$, use a typical stop list for idf and the predefined weights for every type of relation, and use the function group of a_k which combines the direct relatedness estimation as follows.

$$\begin{cases} a_1 &= \max[0, a_{10}(1 - \log_\epsilon p)]; \\ a_2 &= \max[0, a_{20}(1 - \log_{10\epsilon} p)]; \\ a_3 &= 1 - a_1 - a_2 \end{cases} \tag{7.34}$$

[6]http://swat.cse.lehigh.edu/resources/onto/dblp.owl
[7]http://dblp.uni-trier.de/xml/dblp.dtd
[8]http://www.informatik.uni-trier.de/ ley/db/

Table 7.1: Accuracy Results

(a) $(a_{10}, a_{20}) = (0.5, 0.4)$

ϵ =?	10^{-2}	10^{-3}	10^{-4}	10^{-5}
accuracy	71.8%	76.9%	82.1%	84.6%

(b) $\epsilon = 10^{-4}$

(a_{10}, a_{20}) =?	$(0.5, 0.4)$	$(0.9, 0)$	$(0, 0.9)$
accuracy	82.1%	79.5%	79.5%

,where p is the chain probability, a_{10} and a_{20} are the initial values for a_1 and a_2 respectively when $p = 1$.

We compare the accuracy when the chain probability threshold ϵ, and a_{10}, a_{20} change in Table 7.1. In the WSD community, researchers usually use two naive baselines to evaluate their systems: (1) randomly select a sense from all the senses for the target word, and this baseline is usually compared with unsupervised approaches; (2) always select the top sense which is most frequently used for the target word, and this baseline is usually compared with supervised approaches. In contrast to our unsupervised approach, the accuracy from WordNet top sense is 64.1%. In Table 7.1.(a), we can see the accuracy becomes better when we lower the threshold ϵ. This is what we expected, because theoretically the chain gets more expanded and thus gather more information. However, the lower ϵ would also need more computation time. In Table 7.1.(b), we set a moderate threshold $\epsilon = 10^{-4}$ (which also requires moderate running time), and compare the effects by different direct relatedness estimators. When we set a_{10} or a_{20} equal to 0, it means we do not use that estimator of direct relatedness probability ($P_1(S_y^D|S_l)$ and $P_2(S_y^D|S_l)$ respectively). Only relying on one of them can also get good accuracy, but not as good as combining them.

In order to evaluate the probability distributions, we define the *Distribution Candidate Ratio (DCR) Test* as the ratio between the probability of the correct discrete value (from ground truth) and the probability that is the highest *among all other* possible values. Formally, let $D = [p_1, p_2, \ldots, p_n]$, where $\sum_{i=1}^{n} p_i = 1$ be the probability distribution to test, p_c be the probability for the correct value. Then $p_m = \max_{i \neq c} p_i$ is the highest among all other possible values. We define $DCR = \frac{p_c}{p_m}$. This ratio test can be used to evaluate any distribution of discrete value events. If the correct one is not the highest probability in the distribution, this ratio is less than 1 and tells the closeness to candidacy; if the correct one is the highest probability, this ratio is greater than 1 and tells how well this event is distinguished from the others.

Now we examine the probability distribution of our results by DCR Test. Here we use the setting $\epsilon = 10^{-4}, d = 1, (a_{10}, a_{20}) = (0.5, 0.4)$. In Figure 7.3 we contrast the ratios of distribution by our approach and WordNet (WN) corpus probability. The terms are sorted by the highest probability output by our approach. From this result we have two findings. First, for most of the cases, our approach has better DCR test result. Our approach either makes the top sense correct when the WN result gets it wrong, or makes the correct sense more distinguished from the others than the WN results. Second, we were somewhat surprised that some of the distributions have one dominate probability which is higher than 99%. This could be good for the correct ones showing the confidence of the judgment: the ones with extreme dominant probabilities (also having a very high ratio) are very likely to be correct, because it is usually the sign of finding strong relatedness

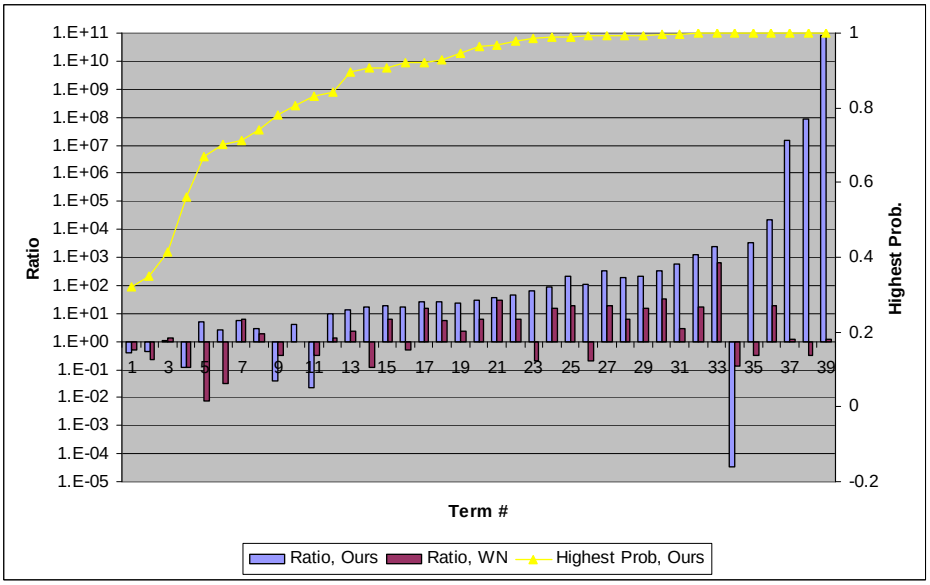


Figure 7.3: An example ratio of distribution between the correct one and the other top one. Terms are sorted by the highest probability output by our approach.

between the target term and the contexts. For example, term #39 "publication" has the highest dominant probability, because most of the context around it are all related to the correct sense (since the ontology is about publications).

However, there are also exceptions that our result is worse than WN in the ratio test. Term #34 is "master". The correct sense is *master#n#8* which means "someone who holds a master's degree from academic institution", and the top one in our result is *master#n#5* which means "an original creation (i.e., an audio recording) from which copies can be made". The context words, such as "publication", are found related to *master#n#5*, because they are hyponyms (sub classes) of *creation#n#1*. On the other hand, there is little relatedness between context and *master#n#8* found by our approach. However, to a human reader, it is clear that the word "academic" in the gloss shows some

relatedness to context words. Term #9 and #11 are both "title" appearing at different places in the ontology. A similar problem occurs with them. Our approach cannot find the relatedness between the correct sense *title#n#2* and context. However in its gloss, the phrase "literary composition" can be easily related to context words such as "publication" by humans. So all these mistakes our approach makes shows that our challenge is to find efficient ways of estimating the relatedness between synsets, as in Section 7.3.2, and also to minimize the impact of errors.

Currently we have not tried to optimize execution time. The whole process can be very time-consuming when ϵ is small and d is large. The average time for each pair of synsets in $P(S_y|S_0)$ is 278 ms for $\epsilon = 10^{-3}, d = 1$.

Although this WSD algorithm still needs further research and more experiments, we think the ideas can be expanded to be useful in many Semantic Web applications. One of the most important use case is for retrieving classes. If we have the probability distributions of senses for classes, and also we have the distributions of senses for a users input, we can develop an algorithm to compute the expected distance (i.e. weighted by probabilities) of senses between the input and each class in the KB, and then present users a ranked list of classes that are relevant to the users input. This would help improve recall by retrieving matches of synonyms or hypernyms/hyponyms, and also limit the loss of precision by combining information from multiple probability distributions.

Chapter 8

Class Retrieval using Instance Texts

An important problem in using KBs is how to translate natural language queries to the appropriate ontological terms. In this chapter we address the **resource retrieval** problem, which is the task to find the best matched resources (classes, properties, or instance) in the KB given a keyword-based query. Existing tools typically use simple string matching, although some expand the matching by using lexicons like WordNet. We believe that leveraging usage information from the KB can improve retrieval quality better than referencing external lexicons.

Our intuition comes from the observation that in many scenarios, humans learn what a named class refers to by examining some of its instances. For example, when we see a class named "Person", if after examining several random instances of it we find out

all of them are scientists, we have an idea that this class *Person* may mainly refer to researchers. Now consider another example: a class named "Cat". If the instances include species of tigers, leopards, etc. then we know that it refers to felines in general, and not just the typical house cat. Similarly, a resource retrieval component can also obtain more information about a class by identifying patterns in the textual properties of an instance, and using this information to improve retrieval quality. This approach will also have benefits when the KB does not have a sufficient concept for the query. Consider the "Cat" example above in a KB that does not have the class "Tiger": when users query "tiger", the retrieval component knows the class *Cat* covers the query topic best.

We focus on the problem of **class retrieval** using instance texts, and propose a general framework of this approach, which consists of two phases: the keyword query is first used to locate relevant instances; then we induce the classes given this list of weighted matched instances.

8.1 Class Retrieval Framework

Formally, we define the class retrieval problem as: given a natural language query q, return a set of ⟨class, score⟩ pairs $\{\langle C_i, scr_i\rangle\}$, where $C_i \in \mathcal{C}$, and the score scr_i is how well C_i matches q. The naive approach is to define scr_i as the string similarity between the label (l_{C_i}) of C_i and query q:

$$scr_i(q) = StrSim(l_{C_i}, q) \tag{8.1}$$

In practice we implement this method **SL** with the Scaled Levenstein distance metric (edit distance similarity). By utilizing external lexicons, we further define a function $expand(q)$ that derives a set of terms (synonyms, hypernyms, or hyponyms etc.) related to q, and combine string similarities (also using the SL distance) as scr_i by choosing the max value of similarity to the query among all the possible variations of expanded word forms. i.e.:

$$scr_i(q) = \max_{q' \in expand(q)} StrSim(l_{C_i}, q') \tag{8.2}$$

This method **WN** is implemented with WordNet as the lexicon.

We propose a two-phase class retrieval framework, as shown in Figure 8.1. In the first phase, the query q is matched to instances' texts (which we shall soon define), instead of directly matched to classes' labels. This step can be done with a standard IR query; and a set of ⟨instance, IR score⟩ pairs $RS = \{\langle I_j, r_j \rangle\}$ are returned. Given a set of weighted matched instances, the problem in the second phase is then how to induce the class represented by these instances. Then in the process of computing scr_i we may take RS and the *rdf:type* relations (i.e. a property that denotes the class of an instance) in the KB as extra information (they are constant for a given KB). We shall further discuss different implementations of this function in the next section.

Defining instances' texts is crucial and fundamental to the framework. The idea that using texts from instances improves class retrieval is based on the assumption that if we have collected sufficient instance texts, the common terms among these instances are very likely to be indicative of the class (as we shall soon discuss for the "Fern" example in

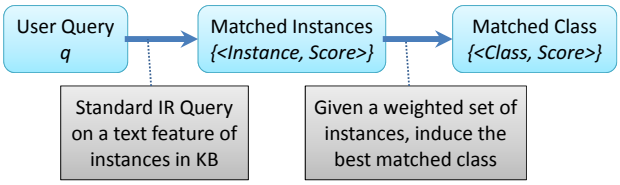


Figure 8.1: Two-phase Class Retrieval Framework

Table 8.1). We generalize the definition of instance texts as a function on an instance, as well as the given KB. We need to point out that there may not be a universally applicable methodology to define the function due to wide variations of KBs. However, as illustrated in Figure 8.2, we list the following ones that often work well: (1) **annotation properties** such as *rdfs:label* and *rdfs:comment*. These properties are widely used in many KBs: the *rdfs:label* values (T_{i1}) are the names of instances and *rdfs:comment* usually provides a human-readable description. (2) **properties with high discriminability/coverage**. Song and Heflin [46] proposed an automatic approach that computes the discriminability/coverage scores in a given KB and finds a list of properties such as job title/name/address, etc. We can further manually (or develop some algorithms to) select some property (such as p_2 in the figure) from this list to get the text values (T_{i2}). (3) **external links**. Different types of external knowledge can be used, e.g. WWW search, or *owl:sameAs* links, to get more texts (T_{i3}) about the instance. (4) **refining existing texts**, e.g. using some function f_1 to extract key terms (T'_{i1}) from the value of some textual property of an instance.

We also show the traditional approach in Figure 8.2, which expands the original text (L_i) of a class, and tries to match with any relevant words (DL_{i1}, DL_{i1}) via external

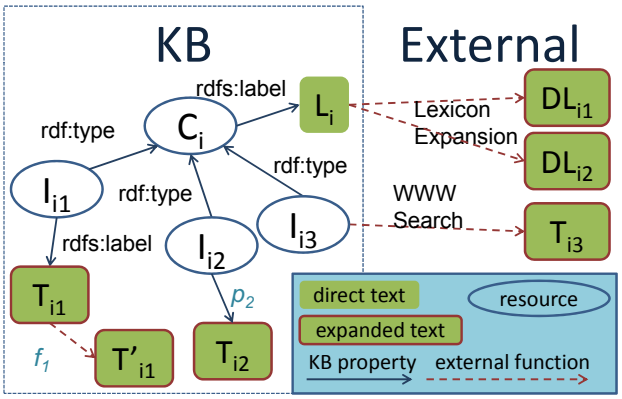


Figure 8.2: Expanded Texts of a Class

lexicon. In fact, Figure 8.2 illustrates that all the resources and literals are connected in a huge graph. Comparing our instance-based proposal to lexicon expansions, we see the common purpose is to find more texts connected to class C_i. However, they face different uncertainties. While the links within the KB are statements and the nodes in the graph are always semantically related, the external links are based on the (possibly ambiguous) syntactical forms of terms, and may lead to texts with no semantic connections to the class. If the expansion goes further from the original term, the uncertainty becomes higher and more noise is introduced. On the other hand, although we have certainty that the instances' texts are connected to the instances and the instances are connected to the class, it is unclear how well the instances' texts can represent the class.

In this chapter, we use DBPedia 3.7 [2], an RDF dataset of structured information extracted from Wikipedia, as *the* KB to test our ideas and run the evaluation experiments. We chose DBPedia for mainly two reasons. First, it contains a large amount of data, and most resources have the property *rdfs:label*, which provides a human-readable version

of a resource's name, and the property *rdfs:comment*, which provides a human-readable description of a resource. This enables us to study contributions from various annotations. Although our work does not require a KB to have these properties, generating annotations from scratch will add extra factors that complicate our analysis. The second reason is that it includes various kinds of classes that cover several different domains. As we shall see soon, annotations of instances of different classes have different features. Since DBPedia has a wide range of topics described at various levels of detail, it allows us to evaluate how our algorithm responds to diverse conditions. For convenience, we use "*d*" as the prefix of DBPedia's naming space, e.g. *d:Book* for *http://dbpedia.org/ontology/Book*. DBPedia also has many links to external ontologies or sources. Some of them, e.g. YAGO[1] types, may also be useful for retrieval. However in this chapter, we ignore such interlinks.

For example, we find that after removing the stop words from labels of all the 683 instances of *d:Fern*, the top 15 frequent terms appearing in these labels (as shown in Table 8.1) are mostly the hyponyms of "fern". We also observe a similar phenomenon for many other classes such as *d:EducationalInstitution*, when the name of each individual instance usually contains the category name, which can be an alternative to the label of (a subset of) its class. e.g. the label "Chesterton Community College" of an instance indicates the category name "Community College". However, the labels of other classes are less informative. For example, the labels of *d:Person*, *d:Film*, and *d:Song* fail to present high rank relevant terms for classes because the titles and names seldom contain relevant terms to the class.

[1]http://www.mpi-inf.mpg.de/yago-naga/yago/

Table 8.1: The top 15 frequent terms in the labels of instances of `d:Fern` after removing stop words

Rank	Term	Frequency
1	cyathea	137
2	asplenium	36
3	blechnum	25
4	polypodium	22
5	polystichum	16
6	dryopteris	16
7	equisetum	14
8	adiantum	14
9	pellaea	13
10	cheilanthes	13
11	tectaria	8
12	fern	8
13	botrychium	8
14	diplazium	7
15	woodsia	6

On the other hand, values of *rdfs:comment* (basically the content of the Wikipedia article) often contain useful terms for class retrieval, but due to their length they contain many irrelevant terms too. However, the oft-repeated terms in these values are often closely **related** to the class. For example, in the comments of *d:Film*, the top terms not only include similar concepts such as synonyms like "movie", and sub categories like "comedy" and "drama"; but also include contextual terms such as "starring", "directed", etc. To enhance the chance that the selected texts accurately reflect the class, we introduce a third text type by refining the comments with simple string manipulations.

We take advantage of the Wikipedia custom, which usually starts its paragraph with a very descriptive sentence about the basic information of each instance page in the form:

⟨label of the instance⟩ ⟨to-be verb⟩ ⟨rest of the sentence⟩.

We implement a straightforward approach to split[2] the comment values and extract only the third part of the first sentence as the third type of texts. We remove the label of the instance because the labels can introduce noise as we discussed in *rdfs:label*. It turns out that we successfully extract these fragments out of 96.5% of the comments in the KB. Note that although this approach is not generally applicable to every instance in the LOD dataset, it can be still useful in many cases for retrieving classes in other subsets, given that DBPedia is a central node in the LOD diagram, with many ontological alignment axioms and instance sameAs links.

Thus in total we have three types of texts: labels, comments, and fragments of comments. We use a traditional IR inverted index and index each of the KB instances with these types of texts into corresponding indexing fields respectively. The index of each field has a term list that contains all the unique terms in the texts of that type; and each term will be associated with a posting list with all instances that contain this term in this field. Given a query q and a specified text feature (a single field or any weighted combination of them), a set of ⟨instance, IR score⟩ pairs can be retrieved via standard IR means; and the first phase in our framework is done.

8.2 Inducing Classes from Instances

From the first phase we have a set of ⟨instance, IR score⟩ pairs $RS = \{\langle I_j, r_j \rangle\}$ as the results. For convenience, we define $\mathcal{I}_q = \{I_j | < I_j, r_j > \in RS\}$ the set of all the instances

[2]The program tries to find the first to-be verb and splits the sentence based on that.

in RS. The task of the second phase is to assign an appropriate score scr_i for each C_i in the KB. While the first phase can rely on a standard IR approach, we have more choices in how to induce a class in the second phase. In this section, we cast it in three different ways as discussed below.

8.2.1 Additive Value Function

We start from the most straightforward intuition: if an instance I_j of a class C_i is returned, the IR score r_j associated with I_j should somehow contribute to scr_i, the score of C_i. In utility theory, the influence of multiple attributes can be represented by an additive value function as long as we assume mutual preferential independence holds between the attributes. An additive value function is simply a multi-attribute function that is the sum of a set of single attribute value functions. Inspired by this idea, we define the additive value function (AVF) score of a particular class C_i as:

$$scr_i(q) = \sum_{I_j \in C_i \cap \mathcal{I}_q} T(r_j) \tag{8.3}$$

In this formula, we take the score of each instance in both the class C_i and the matched instances I_q, apply a normalization/transformation function T, and then simply apply a naive summation of each transformed value.

We can define T with simple functions. For example, define $T_0(r_j) = 1$ if $r_j > 0$, otherwise 0, which means every instance "votes" for its classes. Or set a threshold ϵ and let $T_\epsilon(r_j) = r_j$ if $r_j \geq \epsilon$ or 0 otherwise. If ϵ is set as the n-th greatest value in $\{r_j\}$, it

means we only consider the top n matching instances.

Note that in a KB, $I_j \in C_i$ can be either explicit or entailed by ontological axioms. An instance thus can belong to multiple classes even in single-inheritance ontologies. This suggests that the score (or vote) $T(r_j)$ from each instance I_j should be apportioned among every class it belongs to. Thus a modified version is

$$scr_i(q) = \sum_{I_j \in C_i \cap \mathcal{I}_q} T(r_j) \cdot f(I_j, C_i) \tag{8.4}$$

where $f(I_j, C_i)$ factors how the vote from I_j is divided. Let nc_j be the total number of classes I_j (explicitly and implicitly) belongs to i.e. $nc_j = |c|KB \models I_j : c|$, then we can naively equally divide $T(r_j)$ by setting $f(I_j, C_i) = 1/nc_j$, i.e.

$$scr_i(q) = \sum_{I_j \in C_i \cap \mathcal{I}_q} \frac{T(r_j)}{nc_j} \tag{8.5}$$

However, this does not reflect the general intuition that the more specific classes should be more favored. For the extreme case, `owl:Thing`, in theory, contains all the instances, and thus it will get the max score among all the possible classes. However returning `owl:Thing` does not provide any useful information as well. So instead, we should somehow penalize the larger classes in the KB. Let $f(I_j, C_i) = \frac{1}{|C_i|}$, where $|C_i| = |\{x|KB \models x \in C_i\}|$ denotes the size of instances entailed in C_i, we have

$$scr_i(q) = \sum_{I_j \in C_i \cap \mathcal{I}_q} \frac{T(r_j)}{|C_i|} \tag{8.6}$$

8.2.2 Instances as IR Queries

Another option is to consider the problem as another IR problem by treating each class as an IR document, and indexing each class with its instances' IDs as its content. In the index, each instance has a posting list of classes it belongs to. Then the problem in the second phase of our framework is cast as Boolean retrieval given a long query with query terms $\mathcal{I}_q$ to this index. We consider the basic tf-idf (short for term frequencyinverse document frequency) [32] approach. In its classic definition, the tf-idf is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. The tf-idf value increases proportionally to the number of times a word appears in the document, but is offset by the frequency of the word in the corpus, which helps to control for the fact that some words are generally more common than others. In our case, the tf-idf is useful for normalizing the scores of instances that belong to multiple classes. i.e.

$$scr_i(q) = \sum_{\forall I_j \in \mathcal{I}_q} tf(C_i, I_j) \cdot idf(I_j) = \sum_{I_j \in C_i \cap \mathcal{I}_q} idf(I_j) \tag{8.7}$$

An instance is either a member of a class ($tf = 1$) or not ($tf = 0$), thus the tf merely denotes whether $I_j \in C_i$ is true (explicitly or by entailment) for this KB. Furthermore if we consider that query terms are not equally weighted, again we apply the transformed

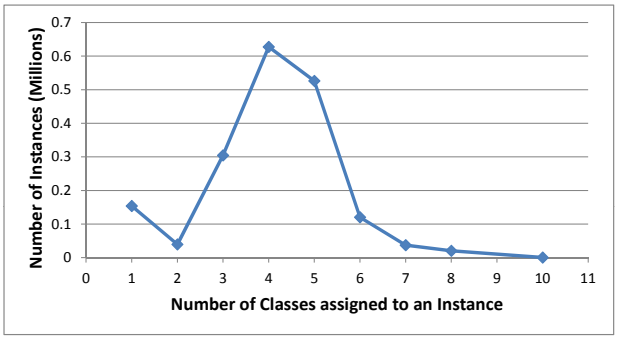


Figure 8.3: Distribution of Number of Classes - Number of Instances

scores $T(r_j)$ associated with each I_j as the boost factor[3], and then we have

$$scr_i(q) = \sum_{I_j \in C_i \cap \mathcal{I}_q} T(r_j) \cdot idf(I_j) \tag{8.8}$$

The idf is usually defined as $idf(I_j) = \log \frac{N_C}{df(I_j)}$, where N_C is the total number of documents (in our case classes). The number of classes is often relatively small in Linked Data datasets, for example, in DBPedia, $N_C = 319$. The log function is taken to scale the huge difference among document frequencies of terms ($df(I_j)$). However in our case, $df(I_j)$ is just the total number of classes I_j belongs to, in other words, $df(I_j) = nc_j$. Figure 8.3 illustrates that in DBPedia for most of the instances $nc_j = 3 \sim 5$. Note that we count both the explicit and entailed classes of an instance, except *owl:Thing*, which is the top concept and should be assigned to any instance. Since nc_j does not change in orders of magnitude, for most of the time we get $idf(I_j) = 1.80 \sim 2.02$, thus we can approximately

[3]In many standard IR approaches, the boost factor is used to adjust the weight of a specific word to emphasize/de-emphasize it in the query or in the document collections

treat $\log N_C/nc_j \approx \alpha$ as a constant. Then

$$scr_i(q) = \sum_{I_j \in C_i \cap \mathcal{I}_q} T(r_j) \cdot idf(I_j) \approx \sum_{I_j \in C_i \cap \mathcal{I}_q} \alpha \cdot T(r_j) \tag{8.9}$$

is mathematically approximate to AVF in Equation (8.3). To emphasize the difference, we define idf' without the log, then

$$scr_i(q) = \sum_{I_j \in C_i \cap \mathcal{I}_q} T(r_j) \cdot idf'(I_j) = \sum_{I_j \in C_i \cap \mathcal{I}_q} \frac{N_C \cdot T(r_j)}{nc_j} \tag{8.10}$$

which is directly proportional to Equation (8.5), and thus produces an equivalent rank-ordering of classes. In practice, the basic tf-idf approach above is usually tuned with various normalization factors, and we will use a state-of-the-art IR tool in the experiment to evaluate this approach to scoring.

8.2.3 Ontology Alignment Problem

We first discuss the relations between a query and a class in the KB. Imagine a virtual class Q based on the query q. Q basically means anything that is *closely related* to the query q. Thus an instance with texts containing q is considered to be explicitly declared to have type Q. Note that in the real world, it is very unlikely to have a system class interpreted as "everything about q", thus we do not expect any class C_i from the KB s.t. $C_i = Q$. On the other hand, we define another virtual class D_q, which is the concept that directly corresponds to the query need represented by the term q. $D_q \sqsubset Q$. Often this D_q

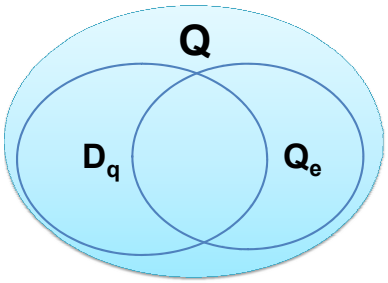


Figure 8.4: Relation between Q, Q_e and D_q

can be aligned to some class C in the KB. We show the Venn Diagram in Figure 8.4.

However D_q is not directly observable in the KB; and even Q is not fully observable. We can only detect a subset Q_e consisting of "explicit" instances of Q. Since Q is anything closely related to q, instances that have texts containing synonyms or hyponyms of q should also be considered as implicit instances. Thus $D_q \sqcap Q_e$ is the part that is both desirable and observable. For example, if $q =$"movie", the target class $D_q =$*d:Film*. Some instances of *d:Film* have "movie" in their texts, and thus are instances of $D_q \sqcap Q_e$, while others using "film" instead of "movie" are implicit members of Q. The other classes in Q can be movie actors, movie songs, etc. that are disjoint with, but also closely related to D_q. In other examples, Q may also include classes that are totally irrelevant to D_q due to the ambiguity of q. We can define two ratios:

$$\alpha_1 = P(D_q|Q_e) = \frac{|D_q \sqcap Q_e|}{|Q_e|} \tag{8.11}$$

$$\alpha_2 = P(Q_e|D_q) = \frac{|D_q \sqcap Q_e|}{|D_q|} \tag{8.12}$$

where α_1 is the likelihood that an instance with text q is an instance of D_q (related to the ambiguity and usage of q), and α_2 is the likelihood that an instance of D_q has q in its texts (related to the variations of expression of D_q). These two ratios are variables of the query and the KB. But for a given query q and KB they are fixed, and so is the ratio between them. The ratios can be some important factor for our estimation on the probabilities, however, currently we make a naive assumption that the ratio is similar on different queries, and leave the study on how to decide these ratios based on the statistics of the KB for future work. Here we simply define the ratio as a constant which is also the ratio between D_q and Q_e.

$$\alpha = \frac{\alpha_1}{\alpha_2} = \frac{|D_q|}{|Q_e|} \tag{8.13}$$

Under a naive assumption of uniform distribution of D_q in Q_e and uniform distribution of Q_e in D_q, we assume that given any sample of Q_e, we can infer the size of the corresponding sample of $D_q \sqcap Q_e$, and then infer the size of the corresponding sample of D_q. We can further interpret it in an extreme way: every time we find an instance of Q_e, we estimate that it is expected to represent α instances of D_q.

From the first phase, the result set $\langle I_j, r_j \rangle$ actually returns a set of instances that are likely to be instances of Q_e, where the relevance score r_j indicates such likelihood. So we can first apply a function $T_p(r_j)$ which is the probability that I_j is an instance of Q_e. If in addition, we take the ratio α into consideration, and finally we can define a transform function $T_\alpha(r_j) = \alpha \cdot T_p(r_j)$. Then $T_\alpha(r_j)$ can be interpreted as the expected number of

instances of D_q that I_j represents. The total size of D_q is estimated as

$$|D_q| \approx \sum_{\forall I_j \in \mathcal{I}_q} T_\alpha(r_j) \tag{8.14}$$

Based on the uniform assumption, we can estimate that every instance that is in C_i and matches q implies α instances of $D_q \sqcap C_i$. i.e., :

$$|D_q \sqcap C_i| \approx \sum_{I_j \in C_i \cap \mathcal{I}_q} T_\alpha(r_j) \tag{8.15}$$

With these estimated sizes, many existing approaches in instance-based ontology alignment can be applied. e.g. the the most commonly used Jaccard (**Jcd**) approach [47] is defined as:

$$scr_i(q) = \frac{|D_q \sqcap C_i|}{|D_q \sqcup C_i|} = \frac{|D_q \sqcap C_i|}{|D_q| + |C_i| - |D_q \sqcap C_i|} \tag{8.16}$$

There are also measures for ontology matching based on information theory. e.g. The Pointwise Mutual Information (**PMI**) approach [10] measures the reduction of uncertainty that the annotation of one concept yields for the other. Mathematically, it is the log of the ratio between the probability of their coincidence given their joint distribution and the probability of their coincidence given only their individual distributions:

$$scr_i(q) = \log \frac{P(D_q \sqcap C_i)}{P(D_q) \cdot P(C_i)} = \log \frac{\frac{|D_q \sqcap C_i|}{N}}{\frac{|D_q|}{N}\frac{|C_i|}{N}} = \log \frac{|D_q \sqcap C_i| \cdot N}{|C_i| \cdot |D_q|} \tag{8.17}$$

where N is the total number of instances in the KB. PMI is maximized when $D_q \sqsubseteq C_i$ or

$D_q \sqsupseteq C_i$ is true. Other measures in this category include log likelihood ratio, information gain, etc. A comprehensive comparison of measures in instance-based ontology alignment can be found in [22].

We also propose an alternative approach. First we can consider $P(C_i|D_q)$ which represents the probability that an instance has type C_i if we already know it has type D_q.

$$scr_i(q) = P(C_i|D_q) = \frac{|D_q \sqcap C_i|}{|D_q|} = \sum_{I_j \in C_i \cap \mathcal{I}_q} \frac{T_\alpha(r_j)}{|D_q|} \tag{8.18}$$

Note that scr_i is calculated for each class C_i given query q, and $|D_q|$ is just a normalization factor. So the output rankings of C_i using this approach is the same as that of AVF in Equation (8.3) when we assume $T_\alpha(r_j)$ is proportional to $T(r_j)$. From one perspective, we evaluate each $C_i \in \mathcal{C}$ with some metric on how well C_i matches virtual class D_q. Then Equation (8.18) evaluates each candidate C_i with the *virtual* recall of D_q, i.e., the percentage of D_q instances that were identified by C_i. We can also use the *virtual* precision: the percentage of C_i instances that are actually in D_q, i.e.

$$scr_i(q) = precision(C_i, D_q) = \frac{|D_q \sqcap C_i|}{|C_i|} = \sum_{I_j \in C_i \cap \mathcal{I}_q} \frac{T_\alpha(r_j)}{|C_i|} \tag{8.19}$$

which, except for scaling for α and normalization, is identical to Equation (8.6) (the AVF with bias towards more specific classes). In combination, we propose the *virtual* F-Measure approach (**FM**):

$$scr_i(q) = \frac{2 \cdot \frac{|D_q \sqcap C_i|}{|C_i|} \cdot \frac{|D_q \sqcap C_i|}{|D_q|}}{\frac{|D_q \sqcap C_i|}{|C_i|} + \frac{|D_q \sqcap C_i|}{|D_q|}} = \frac{2 \cdot |D_q \sqcap C_i|}{|D_q| + |C_i|} \tag{8.20}$$

Table 8.2: Summary of Class Induction Formulas. The $\propto$ indicates the two formulas are proportional to each other.

	Section 8.2.1			Section 8.2.2		Section 8.2.3	
	Eq(8.3)	Eq(8.5)	Eq(8.6)	Eq(8.9)	Eq(8.10)	Eq(8.18)	Eq(8.19)
Eq(8.3)	$\equiv$			$\propto$		$\propto$	
Eq(8.5)		$\equiv$			$\propto$		
Eq(8.6)			$\equiv$				$\propto$
Eq(8.9)	$\propto$			$\equiv$			
Eq(8.10)		$\propto$			$\equiv$	$\propto$	
Eq(8.18)	$\propto$				$\propto$	$\equiv$	
Eq(8.19)			$\propto$				$\equiv$

8.2.4 Summary of Formulas

In this section we have introduced three ways of viewing the problem of inducing classes from instances. Although initiated from different viewpoints, we find that some approaches are proportional to each other, and thus are mathematically equivalent with regard to the output rankings of classes, although some are approximately proportional because of various assumptions. We see that an approach proposed in one view gets backed up by another view. However we still need to point out the difference behind these similar formula. The variability hidden in the formula is the transform function. Table 8.2 summarizes the similar formulas in the three views.

8.3 Evaluation

In this section, we first introduce our experimental setup, and then discuss the results of our proposed approaches.

8.3.1 Experiment Setup

We begin with a discussion of how evaluate the results of a class retrieval system. In traditional IR, a document is either relevant or not, but classes can be generalizations of each other, and intuitively more specific classes should be better matches than very generic classes because the generic ones usually fail to represent the user's interest and thus return many results that are useless to the user. We continue using the previous notation, assuming the query term q represents a virtual concept D_q. We discuss different categories of matches to D_q that can be specified in the ground truth. Within each case, we want to clarify the goal of the retrieval, i.e. which classes are worth retrieving. With this goal in mind, we define a function $rel(C_i, q)$ that indicates the degree of relevance of a retrieved class C_i to the query q.

1. **Equivalence Match**: $\exists C_b$ in the KB, $C_b = D_q$. In this case, returning exactly C_b is the only goal. e.g. $q =$"movie", $D_q = C_b =$*d:Film*. We define $rel(C_i, q) = 1$ if and only if $C_i = D_q$ and 0 otherwise. In addition, to better distinguish the difficulty of query, we call an Equivalence Match a **Syntactic** match if the matched class C_b has label $l_{C_b} = q$, otherwise a **Synonym** match because C_b and q represent (almost) the same concept and usually the label of C_b is a synonym of q.

2. **Partial Match**: Sometimes $\neg\exists C_b = D_q$ for a query q. However we may find classes very close to the virtual query class D_q. e.g. given $q =$"composer" return $C_b =$*d:MusicalArtist*; or given $q =$"physicist" return $C_b =$*d:Scientist*. In both examples, the classes are the superclasses of the query class D_q, but are the best results the

system can return, because they are the most specific *upper bounds* of D_q. A class C_u is *an* upper bound of D_q in the KB if and only if $\exists C_u$, $C_u \sqsupset D_q$, and $\neg\exists C'_u$, s.t. $C_u \sqsupset C'_u \sqsupset D_q$. In other words C_u is a most specific subsumer of D_q in the KB. Similarly, we can define the *lower bounds* of D_q. A class C_l is *a* lower bound of D_q in the KB if and only if $\exists C_l$, $C_u \sqsubset D_q$, and $\neg\exists C'_l$, s.t. $C_l \sqsubset C'_l \sqsubset D_q$. Note there can be multiple upper bounds or lower bounds. Suppose the total number of upper bounds and lower bounds of a query are u and l respectively. We define three types of partial matches:

- **Superclass Match** if $u \geq 1$ and $l = 0$. We assign $rel(C_i, q) = 0.8/u$ if C_i is one of u upper bounds of D_q; otherwise it is 0. Thus, a superclass match is not as relevant as an exact match, and the more possible upper bounds there are, the less relevant any single one is. Note that if no other upper bound exists, in theory *owl:Thing* is the upper bound, however we do not count that as a Superclass Match.

- **Subclass Match** if $u = 0$ and $l \geq 1$. We assign $rel(C_i, q) = 0.8/l$ if C_i is one of l lower bounds of D_q; otherwise it is 0.

- **Bounded Match** if $u \geq 1$ and $l \geq 1$. We assign $rel(C_i, q) = 0.4/u$ if C_i is one of u upper bounds of D_q; assign $rel(C_i, q) = 0.4/l$ if C_i is one of l lower bounds of D_q; otherwise it is 0.

The $rel(C, q)$ values can be viewed as the expected utility of translating q into C for subsequent usage. We set these values based on rough estimations of the likelihood that such a match can be a satisfactory search result for a user, or the likelihood that such a

match can be used to formulate a SPARQL query to get useful answers. Note that we actually define very strict goals for the retrieval task. In all three cases, we are asking what the best possible match could be for the query. Thus the superclasses of the upper bounds or subclasses of the lower bounds do not get a partial relevance score. We also debated whether we should give partial relevance scores to the overlapped classes. An overlapped class is a class that shares *some* instances with the query class D_q, but is not a super class nor a sub class of D_q. The benefit from retrieving an overlapped class C_x is really determined by the ratio of the common part of C_x and D_q. Since we need human judgment to produce ground truth for the evaluation, in order to increase the intra-human agreement as well as reduce human effort, we simply treat all overlapped classes as irrelevant.

For our evaluation, we call a query q a qualified query if and only if $\exists C_x$ in the KB, s.t. $rel(C_x, q) > 0$. At first, we tried to collect qualified queries by extracting terms from target web pages: we visited the web sites that contain general purpose questions/answers, such as Ask.com, and went through the question archives in order to extract useful query words from them. However, we found a low yield because either the domain of the questions was irrelevant to the classes in DBPedia, or the nouns in the questions were simply proper nouns which could not be considered as a match to any class. Finally we chose to make use of the interlinks from DBPedia to another dataset, namely the WordNet dataset. The links are statements each of which claims an instance has a WordNet type that specifies the synset that categorizes this instance. e.g.

⟨ *dr:Gzip, dp:wordnet_type, wn:synset-software-noun-1*⟩

which says that Gzip is a software, and the term "software" used here should refer to the meaning of the first noun definition of "software" in WordNet. Thus we extract this term "software" as an original (**e.O**) query term. There were 425,008 unique instances connected to 124 unique synsets in the interlinks. We then removed those links involving instances that are not classified to any class in DBPedia, and that resulted in 339,041 instances that are classified to 252 unique classes and linked to 113 unique synsets. In order to provide various terms, we also take advantage of the specified sense of the term, and expand the query set by the adding the synonyms (**e.S**) and hypernyms (superclasses of terms) of the original terms from WordNet. For hypernyms, we use not only the direct (level 1, **e.H1**) hypernyms, but also hypernyms of level 2 (**e.H2**) and level 3 (**e.H3**). We did not expand the query in the hyponym (subclass of term) direction, because we think in that direction, we would usually find only Superclass Matches for the expanded queries. However, we are able to get various types of Matches if we expand in the hypernym direction. In addition, we also avoid terms that are rarely used in the real world. Since WordNet provides tag counts (number of occurrences of words with specific meaning in a corpus), for each synonym or hypernym expansion, we pick up to the three most frequently used word forms with tag counts > 10 of that sense. This expansion process resulted in 184 candidate queries.

We implemented an interface and asked three native English speakers to provide their judgment on whether an ontological class is a super/equivalent/sub/overlapped/disjoint

concept to the query. The interface always infers for the relevant classes when the user specifies his judgment on one class: If the user specifies a class as the subclass of the query, all the subclasses of that class are marked as subclasses as well. The super class and equivalent class follow the same inference. We also made an assumption that frequently holds in DBPedia: different branches are disjoint with each other, and thus when a super class is specified or inferred, all the other branches are automatically marked as disjoint. Note the automatic markup feature reduces the burden of participants and also ensures a consistent judgment (it is impossible to mark C_1 as super class and C_2 as sub class while $C_1 \sqsubset C_2$ in the KB).

Using these judgments, our system automatically categorized the queries into the five matching types, i.e. Syntactic match, Synonym match, Superclass match, Subclass match, and Bounded match. We now discuss the algorithm for combining the judgments and deciding what match it is based on the combined judgment.

We model the judgment as a set of beliefs each participant holds for a given query, and the belief could be that one class is a super/equivalent/sub class of the query. Since we assume disjointness in different branches, and assume the participant provided their judgment to the best of their knowledge, we can simply use the closed-world assumption [40], and consider anything that cannot be inferred from one's belief to be against his belief. Then we provide two methods for beliefs w.r.t. inference. The *complete*() method completes the set of beliefs by adding all entailed ones, and in the opposite direction, the *simplify*() method reduces the beliefs to the minimum set of beliefs that could infer any

belief in the current set. Note that we always simplify the judgment before we store it or report it, because the simplified one saves space and is easier to understand.

We present the algorithm for combining various judgments of a given query in Algorithm 2 which works as follows. We first complete the set of beliefs in each judgment, and then count how many people are in favor of each belief. By majority vote we decide what to be kept in the combined judgment. Lastly we simplify the combined result and return it. From the simplified results, we can classify the query by its simplified combined judgment. If there is a belief for equivalent class, we check the word form and decide if it is a Syntactic match or Synonym match. If there are only super classes or sub classes, it is a Superclass match, or a Subclass match. If there are both, then it is a Bounded match. If there is nothing, it fails to obtain intra-judge agreement.

Algorithm 2 Algorithm for Combining the Judgments in $jList$

```
1: function combineJudgments(Judgment[] jList)
2:     for Judgment j in jList do
3:         j.complete()            ▷ Complete the set of beliefs by adding all entailed ones
4:     end for
5:     Map<Belief, Integer> countMap ← ∅        ▷ Declare a map to count each belief
6:     for Judgment j in jList do
7:         for Belief b in j.getBeliefs() do
8:             addCount(countMap, b)
9:         end for
10:    end for
11:    Judgment cj ← ∅        ▷ Declare the combined judgment initialized with no belief
12:    for Belief b in countMap.keySet() do
13:        if countMap.get(b) > jList.size()/2 then        ▷ Majority vote for each belief
14:            cj.addBelief(b)
15:        end if
16:    end for
17:    cj.simplify()                                ▷ Simplify beliefs to the minimum set
18:    return cj
19: end function
```

After the combination of judgments, there were only 4 queries that lacked intra-judge agreement and we dropped them from the query set. Note that there are two approaches we applied to maximize the intra-human agreement in this algorithm:

- We use a majority vote (e.g. at least two out of three people should agree) on the judgment of each class query pair. If we only keep the ones that everyone agrees on, we will have 23 queries that lack intra-judge agreement.

- We extend the judgment to super/sub classes to see if there is anything that can be agreed in the hierarchy. For example, for C_i and D_q, if one says $C_i \sqsubset D_q$ and another says $C_i = D_q$, we consider $C_i \sqsubset D_q$ as their agreement. If we do not apply such inference, we will have to drop 16 queries.

Proposition 3. *Combining a list of consistent judgments will result in a consistent judgment using Algorithm 2. The consistency here we discuss means that we do not have any* $C_1 \sqsubset D_q \sqsubset C_2$ *in the belief set of a judgment if* $C_1 \sqsupset C_2$ *in the KB.*

Proof. If we have $C_1 \sqsubset D_q$ in the combined judgment, it means we have the majority of votes for $C_1 \sqsubseteq D_q$. For each of these vote, we know after inference with $C_1 \sqsupset C_2$ by calling the *complete*() method, each judgment should also have $C_2 \sqsubset D_q$. Thus the majority should have $C_2 \sqsubset D_q$. On the other hand, $D_q \sqsubseteq C_2$ in the combined judgment means the majority should have $C_2 \sqsupseteq D_q$. Thus we have at least one judgment that has both $C_2 \sqsubset D_q$ and $C_2 \sqsupseteq D_q$, which means it is an inconsistent judgment. □

A summary of the data set is presented in Table 8.3. This table categorizes the queries

Table 8.3: Summary of Query Dimensions

	All	e.O	e.S	e.H1	e.H2	e.H3
All	180	106	13	28	17	16
Syntactic	68	53	3	7	1	4
Synonym	26	15	2	5	2	2
Super	42	26	6	7	3	0
Sub	12	0	0	3	5	4
Bounded	32	12	2	6	6	6

by two dimensions: how a query is generated (the columns) and the ground truth of the query (the rows). From the table we can see that the query set we generated has a good covering on different match types. In total there are 94 exact matches and 86 partial matches. Note that we have e.S which uses synonyms in WordNet to expand queries, and we have Synonym matches which are often the matches of expanded query to class labels. However these two synonym-related features do not highly correlate. Interestingly, we find 3 terms from e.S have Syntactic matches: their original queries match the synonyms of class terms and the expanded queries become the same terms as the class labels. By hypernym expansions (e.g. e.H1), we find that although we get more general query terms, we sometimes get exact/subclass matches to other classes in the KB. All these features indicate the diversity in our generated set.

To best evaluate the top-k retrieved classes, we use Discounted Cumulative Gain (DCG) [23] as our metric. i.e.

$$\mathrm{DCG}_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2 i} \tag{8.21}$$

The gain (i.e. measure the usefulness) is accumulated from the top of the result list to the bottom with the gain of each result discounted at lower ranks. It is particularly useful for us to evaluate our system because we assume a list of matched results, instead of the top one, are returned to a user or to a subsequent component. When a query is issued, the top $p = 10$ matched classes are returned, and by using the average of relevance scores from human judgment, we get the DCG score for this query. There is another similar metric nDCG (normalized-DCG). nDCG sorts documents of a result list by relevance, producing an ideal DCG, and normalizes the DCG of the retrieval results by this ideal DCG. We choose DCG without normalization because we want to weight each match by its rank and utility, and to emphasize that there is a difference between an exact match and a partial match. Meanwhile we also present the average ideal DCG score to show the best possible performance a system can achieve based on human judgment.

8.3.2 Experimental Results

We use Additive Value Function (AVF) from Equation (8.3) , an IR method (Luc) based on Equation (8.8), Jaccard (Jcd) from Equation (8.16), F-Measure (FM) from Equation (8.20), and Pointwise Mutual Information (PMI) from Equation (8.17) introduced in Section 8.2 as our test systems. The IR algorithm uses the state-of-the-art IR system Lucene 3.5, which uses a combined Boolean model and Vector Space Model scoring method. We say the Luc approach is "based on Equation (8.8)" because although it is similar, the weighting and normalization factors are much better tuned for standard IR tasks (detailed in the javadoc of *org.apache.lucene.search.Similarity*). All of the above systems use

instance texts. As we introduced in Section 8.1, the texts we chose are labels (L), comments (C) and fragments of comments (F) which we extracted by using the Wikipedia custom. We also implemented two baseline systems, a Scaled Levenstein approach (SL in Equation (8.1)) and a WordNet approach (WN in Equation (8.2)), that only use class labels. To avoid extra factors complicating the analysis, we define the transforming function as $T(r_j) = r_j$, i.e. we directly use the IR scores from results in the first phase, and we set a constant ratio $\alpha = 1$ in Equation (8.13). Our future work includes studying the effect of different transforming functions and values of α on different queries.

Figure 8.5 shows the overall comparison of average DCG scores among the combination of the systems and text fields, in contrast with the baselines and the ideal DCG score. Among different systems, we find that Jcd, FM, and Luc have a better performance than the others, and it suggests that if we use Jcd, FM, or Luc, our proposed idea of using instance texts can provide better class results than the syntactic matching approaches on class labels. A straight-forward AVF is always worse than the WN baseline, and if it uses just labels, it is even worse than the SL baseline. For different text fields, we can see that F is the best feature in general, although C is almost as good. Using L seems to be less helpful than the other two fields. That is because, as we discussed in Section 8.1, labels of instances do not usually provide useful terms that refer to the class of that instance. However, it is still useful if we manipulate it with a right approach. In this experiment, we did not try to use multiple fields as a text feature. However we have noticed the fact that in some cases one field provides more useful information but in other cases introduces

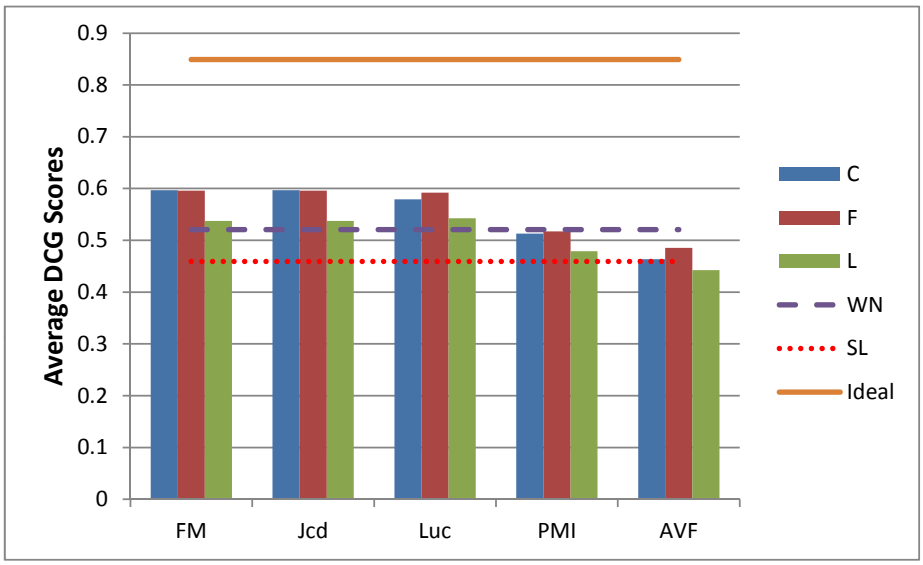


Figure 8.5: Comparison on the Overall Query Set

more noise than the other. We believe by combining these different fields, we can expect improvement for the class retrieval problem. We will study this in our future work. Also, interestingly, we find the coincidence that FM and Jcd always return the same rankings of classes for all the queries in this experiment. After reflection, we realized that since we did not transform the IR score r_j, we always underestimated $|D_q \sqcap C_i|$ in Equation (8.15); and thus in Equation (8.16) $|D_q \sqcap C_i|$ is usually negligible when compared to $|D_q| + |C_i|$; the typical value of $\frac{|D_q \sqcap C_i|}{|D_q|+|C_i|} = 10^{-5} \sim 10^{-4}$ in our test set, thus it it makes Equation (8.16) highly similar to Equation (8.20). Since there is no significant difference between FM and Jcd, in the rest of the chapter we only present the results of FM.

We also compare the systems in different dimensions of queries. Figure 8.6 shows the comparison on match types of queries. We compare the four systems on the same field F, to see how each system performs against queries with different matching types. There are

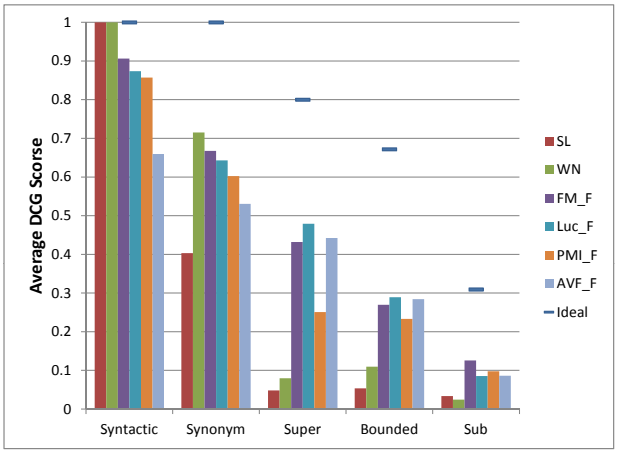


Figure 8.6: Comparison on the Match Type Dimension

several things we want to point out. First we want to explain the change of average ideal scores. There are two reasons why an ideal score becomes small: (1) there are many partial matches, which get partial scores; (2) a partial match has too many classes as its bounds. The upper bounds usually have only one class for each query, however there could be many for the lower bounds for a query. Although we set the sum of rel scores of these bounds to 0.8, the ideal DCG score, however, will sum the discounted scores to less than 0.8 if there are ≥ 3 bounds. This is why we see exact matches have 1.0 and Superclass matches have 0.8 as their ideal scores, while Subclass matches have very a small ideal scores and Bounded match is between Superclass match and Subclass match. Secondly, we inspect the performance of baseline systems. As we can expect, SL works perfectly on Syntactic matches, and sometimes finds matches in Synonym match queries thanks to the partial string match. e.g. "character" matches to *d:FictionalCharacter* and "official" matches to *d:OfficeHolder*. However it has difficulty finding partial string matches in other match

types, and the score drops dramatically. Similar to SL, WN is perfect at Syntactic matches, and is very good at Synonym matches because of the lexical expansion on queries to match to class labels. For Superclass matches and Bounded matches, we find that such lexical expansion continues benefiting WN; however for Subclass matches, it makes WN worse than SL: when we expand the query too much, we reduce the precision while increasing the recall. Also, we want to point out that the experiment is a little biased towards WN, mainly because we generate the query terms using WordNet. Lastly we compare the performances of proposed systems on different match types. We can see that FM is the best on Exact and Subclass matches, however Luc and AVF become better for Superclass and Bounded matches. By examining Equation (8.3) of AVF, we can see that AVF has no factoring for classes. While each instance adds some utility to each of its classes, the more general classes are more likely to get larger scores. Thus AVF always favors general classes, and effectively finds upper bounds (usually general classes) for Superclass and Bounded matches. On the other hand, FM has a factor of $|C_i|$ that penalizes general classes, thus it is less likely to get upper bounds in these cases. Luc, however has a moderate factoring on $|C_i|$, which is encoded in its normalization of document sizes (which in our case is $|C_i|$), and thus exhibits good performance over different match types.

To further inspect how the systems perform on queries that match to general or specific classes, we divide the query set in a third way. We define the depth of a class as the shortest path length from this class to the top class *owl:Thing* in the class hierarchy graph. For each query, we calculate the average depth for all the bound classes (or the exact class if

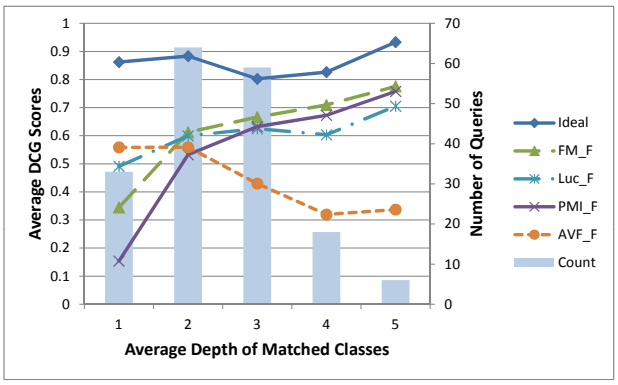


Figure 8.7: Comparison on the Average Depth of Matched Classes

it is an exact match), and round to the nearest integer. As a result, general queries have a small average depth, while specific queries have a larger average depth. We compare FM, Luc, PMI, and AVF on field F in Figure 8.7. We can see that each system has a clear trend of performance as the average depth increases. Most systems achieve better class retrieval results if the query becomes more specific, while only AVF favors the general queries. In other words, FM, Luc, and PMI are leaning towards returning specific classes in the KB because $|C_i|$ all appear in their formula as a factor discounting the general classes. While such discounting is desirable for specific queries, we wonder whether in some systems it is over-discounting. From the figure we can see that Luc is most robust with the change of generality of queries, while PMI is most sensitive to it.

Based on our results, we find a nearly 20% improvement is achieved when comments or comment fragments are used as the instance text and the FM or Jcd approach is used for class induction. Also, we believe the results can be further improved based on our two phase instance text based class retrieval framework. There are a few directions for future

work: new approaches for inducing the class, alternative text fields (or combination of text fields), and the impact of various of transforming functions.

We believe this instance-based class retrieval can be used in applications with the following characteristics: there are multiple user communities with different terminology for the same concept, or the conceptual model of the ontology is dissimilar from that of many users. If this is the case, and there is instance data that has text fields of sufficient length, then our approach is likely to provide some benefit. So the LOD can be a very appropriate dataset for this approach, or at least retrieval of classes from DBPedia and its interlinked datasets can be improved. For example, we can integrate this component into the contextual tag cloud system in order to get better search results (with a much higher recall) which would also include classes with alternative names, or even partially matched classes (super/sub classes). However, meanwhile, how to scale this approach to the LOD is also another interesting challenge for future work.

Chapter 9

Future Work

In this chapter, I propose important and interesting topics that are worthy of investigation beyond this dissertation. We have successfully applied the idea of contextual tag clouds to two datasets that are very different and adapted various optimizations based on the features of the dataset. However, we wonder whether we are able to apply this to further scenarios as well. Thus the ongoing research effort will focus on how to generalize the contextual tag cloud and underlying algorithms.

9.1 Generalized Theory: Boolean Attribute Co-occurrence Histogram

In order to make the algorithms more broadly applicable, we consider the more general problem of calculating Boolean Attribute Co-occurrence Histogram, which focuses on the

computation of conditional distributions of boolean attributes and some useful mathematical operation on the distributions. Here we provide a proposal of the formalization of the problem.

Let $\mathcal{I}$ be the set of all the data record items, and $\mathcal{T}$ be the set of all the boolean attributes (tags). For $t \in \mathcal{T}, i \in \mathcal{I}$, $t(i)$ and $\sim t(i)$ denote item i has a true/false value on t respectively. For convenience, we also define $\mathcal{S}$ as the set of all the false-value attributes. i.e. $\mathcal{S} = \bigcup_{\forall t \in \mathcal{T}} \{\sim t\}$. Let $\mathcal{A} = \mathcal{T} \cup \mathcal{S}$.

Given a set $A \subseteq \mathcal{A}$, a query $q : 2^{\mathcal{A}} \to 2^{\mathcal{I}}$ is a function that selects items that match the conjunction of all the given attributes, i.e.

$$q(A) = \{i | \wedge_{a \in A}\, a(i)\} \tag{9.1}$$

Given a set $I \subseteq \mathcal{I}$, a histogram $h(I) = [(t_1, f_1), \ldots, (t_n, f_n))]$ outputs a **list** of $(t, f) \in \mathcal{T} \times \mathbb{N}^0$, where $\forall t_j \in \mathcal{T}$, $f_j = |\{i | i \in I \wedge t_j(i)\}|$. We define the function for co-occurring histogram of boolean attributes A, $hq(A) = h(q(A))$.

By default, each element (t_j, f_j) in this list is ordered using the natural ordering of $\mathcal{T}$, i.e. alphabetically. Given the order of t, the list can be easily represented as a $|\mathcal{T}| \times 2$ tag-frequency matrix, where the first column is the tag id (1 to $|\mathcal{T}|$) and the second column is the corresponding frequency of the tag.

Based on the formalization of the histogram result, we can further define operations on the histogram (assuming it is represented as H), such as

- $scale(H, func)$, where $func$ is a function applied to each frequency in H.

- $combine(H_1, H_2, func)$, where the frequencies of H_1 and H_2 corresponding to the same terms are combined via a function $func$. In either histogram, missing terms are treated as frequency = 0. This can be useful in performing comparative analysis, e.g. by using subtraction.

- $sort(H)$, the result of which is sorted by descending frequencies.

- $choose(H, \{t_1, t_2, \ldots, t_n\})$, which creates a histogram that consists only of specified elements t_1 through t_n.

- $frame(H, i : k)$, which selects the i through k elements from H. This can be used in paging the tags.

- $filter(H, n, m)$, which selects only elements with frequencies in the range of $[n, m]$.

- $topK(H, k)$, which selects the top-k most frequent elements from H.

Notably, some of the listed functions can be implemented as a combination of other functions, e.g. $topK(H, k) = frame(sort(H), 0 : k - 1)$.

Let $\mathcal{R}$ be the relation set between all the attributes, including both subsumption (e.g. $t_1 \sqsubseteq t_2$) and disjointness (e.g. $t_1 \sqsubseteq \sim t_2$). We can define Histogram Axioms such as:

Axiom 1. The histogram of query attribute set A can be computed via combining (summing the frequencies at each tag) the two histograms of query attribute set A union any regular tag and A union its negation tag. i.e.

$$hq(A) = \text{combine}(hq(A \cup \{t\}), hq(A \cup \{\sim t\}), g(x, y) = x + y) \tag{9.2}$$

Axiom 2. For any two attribute sets A and A', if for any attribute in A' we can find a sub attribute of it in A, then by comparing the histogram of the two sets, we will find no frequency for the first will be larger than the corresponding frequency for the second. i.e.

$$(\forall a' \in A', \exists a \in A \text{ s.t. } a \sqsubseteq a') \Rightarrow \text{combine}(hq(A), hq(A'), I_{x>y}(x, y)) = \emptyset \tag{9.3}$$

Axiom 3. The scale functions in the combine function can be simplified and combined into a single function.

$$\text{combine}(\text{scale}(H_1, g_1), \text{scale}(H_2, g_2), g_0) = \text{combine}(H_1, H_2, g_0(g_1(x), g_2(y))) \tag{9.4}$$

These axioms will be useful for optimizing the online computation algorithms, which is the topic of the next section.

9.2 Improve Pruning Algorithm for Online Computation

In Section 4.2, we introduced three ways of pruning unnecessary tag queries: by using the Co-occurrence Matrix, by Using the previous tag cloud cache, or by dynamic update. If we reconsider the approaches, we can actually treat the Co-occurrence Matrix as a special kind of cache, with two major differences:

- **Context**. The Co-occurrence Matrix is built for all the regular tags, while the cache context could be a combination of regular tags and negation tags. Also the size of

the context for the Co-occurrence Matrix is always 1, while the cache context can be any positive integer.

- **Stored Information**. The Co-occurrence Matrix only has the tags whose frequency with the context tag is greater than 0 (in other words, the co-occurred tags), however, the frequency is not stored. In comparison, in the cache, we always store the frequencies. We decide to store only the tags in the Co-occurrence Matrix, because we want to minimize the disk cost for this large matrix, which also means for online computation we need less time to load the matrix on the fly.

However, in fact, we can actually implement the cache file in the way that no extra time is needed if we do not need to know the co-occurring frequencies: we can first store the array of tags and then the corresponding array of frequencies sequentially. If we do not want the frequencies, we just load the first array without reading the remaining part of the cache file.

On the other hand, we should think what is the advantage of storing those co-occurring frequencies. First of all, the frequencies can be directly used as output, if the request is exactly the context of the cache (or semantically equivalent). Secondly, the frequency of each tag also indicates an upper bound which can be a potential point for optimization. Formally, if we have the cache for context A, and now we get a request for $A \cup \{a\}$, then we know that adding a tag to the argument of f_R will result in an equal or less value. i.e.

$$\forall t \in \mathcal{T}, f_R(A \cup \{t\}) \geq f_R(A \cup \{a, t\}) \tag{9.5}$$

Previously, we only used this rule for the special case:

$$f_R(A \cup \{t\}) = 0 \Rightarrow f_R(A \cup \{a, t\} = 0 \tag{9.6}$$

which in other words, means that the co-occurring tags for a more restricted context can only be a subset of the co-occurring tags of a less restricted context. Now we can think about the optimization for the general case. The upper bound can be used when we issue a boolean IR query. When doing the intersection of the posting lists, we can stop the intersection process if the current size of the intersection already reaches the upper bound.

Another interesting problem is, when we have multiple cache files available, and each of them has a context that is a subset of the context of the request. We might be able to prune some tags by loading any combination of these cache files and finding the intersection. The more cache files we load, the better chance we will prune more tags, but there is also an additional cost for loading each cache. Do we prefer loading some combination of the cache so that we get the most benefit with regard to the total cost? Note that the algorithm for deciding which cache files to load should also be timed for the total cost. Can we find a smart algorithm to select the most efficient cache files? Only via experiments can we answer the question.

9.3 Extend Contextual Tag Cloud to Property Value Spaces

In Chapter 6, we have discussed how we extended the contextual tag cloud system to visualize the computer logs. We think this is a good practice for evolving our contextual tag cloud idea, because in addition to the ontological usages, we also investigated how to properly show the values of properties as tags, which greatly improve the query ability of the system. However in that system, we manually mapped certain value spaces, and defined blocks and tabs for clustering tags. So one of our future work should focus on how to automate this process, or minimize the manual effort, and eventually we would like to have a contextual tag cloud system for LOD on value spaces, which supports any standard "star query" (or a converted one).

Star queries are "SPARQL queries that contain a graph pattern forming a star, with a single center vertex (that is usually a variable), and one or more edges emanating from this center vertex to variable or constant vertexes", which are "fairly common in SPARQL workloads" [21]. The contextual tag cloud interface can be easily adapted as the templates for these queries: In addition to the current design of choosing classes and properties, for each selected property in the context, a user can be provided with a dynamically generated tab (besides the class tab and property tab), where a tag cloud is shown for the value space of that property. Ideally that value space is also dynamically computed, based on the current context, and it can also be refined in some cases. For example, for the `eg:birthYear` property, a user is first shown tags that specify the century; then after picking a tag "1900-1999", the user is shown a refined tag cloud consisting of tags "1900s,

1910s, $\cdots$", lastly after picking a tag "1980s", the refined tag cloud shows "1980, 1981, $\ldots$".

While the ideal use case sounds very desirable to us, and requires only a few straightforward changes in the interface, we face many challenges on the implementation side.

The first question is how to automatically define the tags. Although we may get some information from the definition of some data type properties, we cannot assume the instance triples follow the definitions. So it is more practical to go through all the values of a given property and decide what data type it is, while also allow for outliers (e.g. most values are numbers but occasionally there are values like "N/A" or "null"). After knowing the type of the data, we should choose a corresponding strategy to apply to the values. For numbers, we should apply algorithms to allocate meaningful buckets (ranges) based on the min, max, mean, and/or standard deviation of the values, and use these buckets as discrete tags. For strings, we should tokenize them, and cluster similar ones into the same tag. We should also consider assigning multiple tags for each property value in order to facilitate refinement of contexts.

Also there are a lot of questions about the infrastructure. We are introducing thousands to hundreds of thousands of tags, and there are various relations between them: e.g. tags of the same property should be grouped, the refining tags are organized in a hierarchy, the tags of strings may have similarity scores. Thus we need to reconsider the data structure and organization of the tags. Most likely, we will need a more complex data structure and split the storage of tags in different places with some shared reference schema. This

also raises a question about how the Co-Occurrence Matrix might need to change. For so many tags in different categories, we probably do not want to compute or record pairwise co-occurrence for all the tags, but instead only compute the pairs that provide us the most benefit for online computation.

We understand much work is required to achieve this goal. So in practice we should also plan intermediate steps. For example, we can consider to reduce or simplify some features, or we can first deploy a system on a smaller subset, such as DBPedia as a proof-of-concept.

9.4 Apply the Infrastructure to Other Research Fields

We believe the scalable infrastructure in our contextual tag cloud system, should not be used only for the user interface. The computational problem under the system is closely related to some fundamental statistics of the dataset. Thus we believe the system can also be used to help any other algorithms that might need to perform many computations on the statistics that we can provide. Here we have two scenarios as examples.

Association Rule Learning. Association rule learning [38] is a popular method for discovering interesting co-occurring relations between variables in large databases. The most well-known algorithm is as follows: (1) use the Apriori algorithm to find all frequent itemsets, i.e. attributes that occur together, with frequency $\geq \epsilon$; (2) for each itemset I, divide $I = X \cup Y$, where $X \cap Y = \emptyset$, accept any rule $X \Rightarrow Y$ if the confidence $count[I]/count[X]$ is above the threshold. The Apriori algorithm can be implemented

with our system because an itemset is like a set of attributes or a context in our tag cloud, and the function for the frequency of a given itemset is exactly our f_R function. Also the frequent itemsets of size $n+1$ are built based on the frequent itemsets of size n, where our cache and related algorithms can be helpful.

Feature Selection in Decision Trees. The goal of a decision tree [43] is to predict the value of a target variable based on several input variables. When creating a decision tree (generating the rules), at each node of the tree, the feature selection algorithm chooses the attribute of the data that most effectively splits its set of samples into subsets enriched in one class or the other. In many popular feature selection algorithms, such as C4.5 [39], the splitting criterion is the normalized information gain, and the attribute with the highest normalized information gain is chosen to make the decision. If we only consider boolean attributes, our system fits the problem very well. At each node of the tree, we can treat the node as a tag cloud with tags in it path as the context, and after applying some functions on the frequencies, we get the scores (normalized information gain) for each tag in the tag cloud, and then the tag with the highest score should be selected. Since the entropy formula is basically a function over the conditional probabilities, our system can be very fast at providing these statistics.

Chapter 10

Conclusion

We are on the verge of an explosion of RDF data. Billions of facts have been created in the RDF format, and there are even more to be come available. The Semantic Web technology provides a great framework to allow different people contribute their knowledge, or link their knowledge to others'. While more and more Semantic Web Knowledge Bases are built either from the interlinked datasets or from a self-contained domain expert system, we put our focus onto how such a volume of valued knowledge should be consumed, not only by the experts, but also by every casual user. We analyzed and summarized the reasons why a user might not be able to use a Semantic Web KB, and make our efforts towards reducing the knowledge gaps via two aspects.

The first direction is a system for exploring a Semantic Web KB. We proposed the contextual tag cloud system. The contextual tag cloud system is a novel tool that helps both casual users and data providers explore the LOD dataset: by treating classes and

properties as tags, we can visualize patterns of co-occurrence and get summaries of the instance data. From the common patterns users can better understand the distribution of data in the KB; and from the rare co-occurrences users can either find interesting special facts or errors in the data. The main challenge is how to provide a responsive system on a large dataset such as the BTC dataset which contains more than 1.4 billion triples. In addition to the interaction design, we implemented the infrastructure with an inverted index and three pruning approaches, as well as a scalable preprocessing approach. In the experiments, we demonstrated an affordable preprocessing approach, which has almost linear time and space costs and loads the BTC dataset in ∼12 hours; we showed that using the inverted index takes 0.6-0.7 millisecond for each count query for the BTC dataset, and is on average 18X faster than relational DB based storage and 10X faster than a state-of-the-art general purpose RDF repository; we proved the effectiveness of our pruning approaches which usually saves us 80-90% of unnecessary count queries; we showed that 90% of the randomly generated requests from an end user can get the response within 1 second, 97% within 2 seconds, thereby justifying our design choices. We also extend the idea and the concept of tags, by defining more categories such as property values as tags. We applied the contextual tag cloud system to the problem of the computer usage logs, and revisited the infrastructure of the system, where we proposed different strategies for processing queries, and by experiments we showed that the new combined query strategy reduces response time by 16% on average and by as much as 50% on particular queries. We also conducted a preliminary user study on the contextual tag cloud system. We are

pleased to see that the non-expert participants were able to use the system to access a Semantic Web dataset and successfully complete a few typical tasks as in the real world, and 85.7% of these novice participants felt the system provided a good means to investigate the data. We believe this is a good sign that we are filling the knowledge gaps between users and Semantic Web KBs. However, we think there is still a long way to go for the system. We think a generalized abstraction of the tag cloud system is essential for applying this idea more widely. Querying the values of properties would be a huge upgrade to the current system. Reducing displayed tags and showing only the interesting ones is also a very important and desirable feature for a really practical product. In order to achieve these, we are likely to encounter more challenges on computational problems for the infrastructure, and also data mining or machine learning techniques will be involved for deciding which tags to hide. By formalizing the algebra and axioms of the tag cloud system, we expect such problems can be resolved both theoretically and practically in the future.

The other direction is linking the ontological classes to natural language expressions. We first proposed a novel approach for WSD on the labels of classes with our probability model. We constructed a probabilistic model of the WSD task, and derive a formula for calculating the sense distribution, and then propose approaches of estimating each term in this formula. Our preliminary experiments show our approach can achieve a 84% accuracy and also make the correct senses distinguished from other senses in the result distribution. Alternatively we addressed the keyword based class retrieval problem and

solved the problem without the help of external lexicons. Unlike traditional approaches that directly match the query to the labels of classes, we proposed a two-phase framework that utilizes the texts from instances to improve class retrieval, and by experiment we find a nearly 20% improvement in DCG scores is achieved when comments or comment fragments are used as the instance text for our proposed instance-based approach comparing to the baseline systems. The future work includes improvements on both algorithms, and potentially integration of the algorithms to the contextual tag cloud system as the add-ons of linguistic annotation or the advanced search feature.

The Semantic Web has been evolving for over 10 years, and the related techniques have been widely applied to many real world scenarios. While there has been a debate on whether Semantic Web is *the* future of the Web, no one can deny that the era of Big Data has come, nor would anyone deny the great need of integrating various information in order to extract useful information from it. Among all the efforts for easier information integration, Semantic Web is probably the most influential one supported by world wide communities. Meanwhile, we are also aware of other sibling frameworks of representing structured data (graph data). For example, Schema.org, a lightweight framework, provides a shared markup vocabulary that makes it easier for web masters to decide on a markup schema and get the maximum benefit for their efforts, which is another popular way for representing structured data and is supported by search engines including Bing, Google, Yahoo! and Yandex to improve the display of search results. We want to emphasize here that all the approaches we have discussed in this dissertation do not heavily depend on

the Semantic Web specific environment, and can be easily adapted to other frameworks.

While more knowledge providers and more researchers devote their effort to building the ultimately semantically interlinked web of knowledge, we would like to see more people, not just the academics, to consume and benefit from the knowledge. We understand the challenges due to the nature of the diversity and dynamics of LOD, and we are aware of the imperfect quality of the data. We are not going to resolve all the difficulties that people have when they try to use a Semantic Web KB. However, we believe the completion of this dissertation is moving towards the right direction, bringing more people closer to the world of Semantic Web.

Appendix A

Consent Form

CONSENT FORM

Study on User Interfaces for Exploring Large Scaled Linked Data

You are invited to be in a research study of User Interfaces for Exploring Large Scaled Linked Data. You were selected as a possible participant because you are a Lehigh student over 18 years old. We ask that you read this form and ask any questions you may have before agreeing to be in the study.

This study is being conducted by: Xingjian Zhang, under the direction of Prof. Jeff Heflin, Department of Computer Science and Engineering.

Purpose of the study

The purpose of this study is: to evaluate the usefulness of the user interface that we proposed for exploring large scaled linked data datasets. The dataset includes general facts such as the name, birth year, and active years of an athlete, or the properties of a

location, of a company, etc.. However, the information is so mixed, and there are just too many different terms to express those facts. A challenge is how to familiarize people with the dataset and help them decide what terms are more frequently used so that they can construct queries that are likely to have answers. In order to study whether our proposed interface would be helpful for this purpose, we ask the participants to use two web applications for certain browsing tasks. We will record the time and the answers for the tasks performed by each participant, as well as collect surveys of the comments on the applications.

Procedures

If you agree to be in this study, we would ask you to do the following things:

1. A general tutorial ($\sim$ 15 min) about the related concepts in this experiment, such as different types of user interfaces, the concepts, purpose and statistics of the real world linked data, and the features of the test systems. Then we ask them to make their best attempts to accomplish a set of tasks by using both systems.

2. The first group of tasks (10 - 15 min) is to find the best combinations of terms that would return the most results. For example, if the question is the areas of lakes, there are hundreds of terms (from different data sources) that describe the concept lake, and hundreds to describe the property area, and the task is to find which combination would provide the most information to the user.

3. The second group of tasks (10 - 15 min) is to find potential errors in the dataset

within a given context. For example when examining the software instances in the dataset, there is an error that some software products are considered the same as the company who produce it.

4. The survey questions ($\sim$ 5 min) are about your experience in using the system, such as how useful you feel about various features of the systems in finishing the tasks.

More details will be given in the tutorial and the instruction sheet.

Risks and Benefits of being in the study

Possible risks:

First, participants will be subjected to minimal risk, similar to sitting at a desk and working on a computer; Second, participants may also feel frustrated when they try to accomplish the task or wait for the response.

The benefits to participation are:

Participants will understand more about the linked data, which could be helpful for their future study and work in computer science major. The study may also contribute as the prove of the usefulness of a novel user interface for the real world linked data. When linked data get more popular and easier to use, everyone will benefit from this large amount of shared knowledge on the web.

Compensation

You will receive no monetary compensation but will be provided with pizza and soft drinks.

Confidentiality

The records of this study will be kept private. In any sort of report we might publish, we will not include any information that will make it possible to identify a subject. Research records will be stored securely and only researchers will have access to the records.

Voluntary Nature of the Study

Participation in this study is voluntary: Your decision whether or not to participate will not affect your current or future relations with the Lehigh University. If you decide to participate, you are free to not answer any question or withdraw at any time without affecting those relationships.

Contacts and Questions

The researchers conducting this study are: Xingjian Zhang and Prof. Jeff Heflin. You may ask any questions you have now. If you have questions later, you are encouraged to contact Xingjian Zhang at PL117, 610 758 4235, xiz307@lehigh.edu or Prof. Jeff Heflin at PL330, 610 758 6533, heflin@cse.lehigh.edu.

Questions or Concerns

If you have any questions or concerns regarding this study and would like to talk to someone other than the researcher(s), you are encouraged to contact Susan E. Disidore at (610)758-3020 (email: sus5@lehigh.edu) or Troy Boni at (610)758-2985 (email: tdb308@lehigh.edu) of Lehigh Universitys Office of Research and Sponsored Programs. All reports or correspondence will be kept confidential.

You will be given a copy of this information to keep for your records.

Statement of Consent

I have read the above information. I have had the opportunity to ask questions and have my questions answered. I consent to participate in the study.

Signature: _____ Date: _____

Signature of Investigator: _____ Date: _____

Appendix B

Survey Questions

1. I am a computer user with experience more than web browsing and document editing:

   ◯ Strongly agree

   ◯ Agree

   ◯ Neutral

   ◯ Disagree

   ◯ Strongly disagree

2. I am a computer user with experience more than web browsing and document editing:

   ◯ Strongly agree

   ◯ Agree

   ◯ Neutral

   ◯ Disagree

◯ Strongly disagree

3. Before this study I have already used some tag cloud browsing systems (like flickr) and faceted browsing systems (like Amazon.com):

◯ Strongly agree

◯ Agree

◯ Neutral

◯ Disagree

◯ Strongly disagree

4. After the tutorial I am aware of the purpose and the basic concepts of the linked data, and the need of exploring tools of it:

◯ Strongly agree

◯ Agree

◯ Neutral

◯ Disagree

◯ Strongly disagree

5. It is easy to get familiar with the concepts and functionality of System 1 (Tag Cloud):

◯ Strongly agree

◯ Agree

◯ Neutral

◯ Disagree

◯ Strongly disagree

6. It is easy to get familiar with the concepts and functionality of System 2 (Term List):

   ◯ Strongly agree

   ◯ Agree

   ◯ Neutral

   ◯ Disagree

   - I feel the first group of tasks (finding best combinations) are easy to accomplish via System 1 (Tag Cloud):

     ◯ Strongly agree

     ◯ Strongly disagree

   ◯ Agree

   ◯ Neutral

   ◯ Disagree

   ◯ Strongly disagree

7. I feel the first group of tasks (finding best combinations)are easy to accomplish via System 2 (Term List):

◯ Strongly agree

◯ Agree

◯ Neutral

◯ Disagree

◯ Strongly disagree

8. I feel the second group of tasks (finding errors) are easy to accomplish via System 1 (Tag Cloud):

◯ Strongly agree

◯ Agree

◯ Neutral

◯ Disagree

◯ Strongly disagree

9. I feel the second group of tasks (finding errors)are easy to accomplish via System 2 (Term List):

◯ Strongly agree

◯ Agree

◯ Neutral

◯ Disagree

◯ Strongly disagree

10. In general, I think System 1 (Tag Cloud) is a good way of visually organizing data and supports a lot of functionalities for exploring data:

    ◯ Strongly agree

    ◯ Agree

    ◯ Neutral

    ◯ Disagree

    ◯ Strongly disagree

11. In general, I think System 2 (Term List) is a good way of visually organizing data and supports a lot of functionalities for exploring data:

    ◯ Strongly agree

    ◯ Agree

    ◯ Neutral

    ◯ Disagree

    ◯ Strongly disagree

12. Please provide your general comments (pros and cons) on System 1 (Tag Cloud) (you can also emphasize your keywords with #):

13. Please provide your general comments (pros and cons) on System 2 (Term List) (you can also emphasize your keywords with #):

14. Please provide your general comments (pros and cons) on this study:

Bibliography

[1] Abadi, D., Marcus, A., Madden, S., Hollenbach, K.: SW-Store: a vertically partitioned DBMS for Semantic Web data management. The VLDB Journal 18(2), 385–406 (2009)

[2] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.: Dbpedia: A nucleus for a web of open data. In: 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference. pp. 722–735 (2007)

[3] Banek, M., Vrdoljak, B., Tjoa, A.M.: Word sense disambiguation as the primary step of ontology integration. In: Proceedings of the 19th international conference on Database and Expert Systems Applications. pp. 65–72. Springer-Verlag, Berlin, Heidelberg (2008)

[4] Banerjee, S., Pedersen, T.: Extended gloss overlaps as a measure of semantic relatedness. In: International Joint Conference on Artificial Intelligence. pp. 805–810 (2003)

[5] Bernstein, A., Kaufmann, E., G?ring, A., Kiefer, C.: Querying ontologies: A controlled English interface for end-users. In: 4th International Semantic Web Conference and 2nd Asian Semantic Web Conference. pp. 112–126 (2005)

[6] Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. International Journal on Semantic Web and Information Systems (IJSWIS) 5(3), 1–22 (2009)

[7] Broekstra, J., Kampman, A., Van Harmelen, F.: Sesame: A generic architecture for storing and querying RDF and RDF schema. In: ISWC. pp. 54–68 (2002)

[8] Burke, R.D., Hammond, K.J., Young, B.C.: Knowledge-based navigation of complex information spaces. In: Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI). pp. 462–468 (1996)

[9] Cheng, G., Ge, W., Qu, Y.: Falcons: searching and browsing entities on the Semantic Web. In: WWW. pp. 1101–1102 (2008)

[10] Church, K.W., Hanks, P.: Word association norms, mutual information, and lexicography. Computational Linguistics 16(1), 22–29 (March 1990), `http://dl.acm.org/citation.cfm?id=89086.89095`

[11] Cormen, T., Leiserson, C., Rivest, R., Stein, C.: Chapter 21: Data structures for disjoint sets. In: Fagerberg, J., Mowery, D., Nelson, R. (eds.) Introduction to Algorithms (Second ed.). MIT Press (2001)

[12] Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Section 22.1: Representations of graphs. In: Introduction to Algorithms, pp. 527C–529. McGraw-Hill Higher Education, 2nd edn. (2001)

[13] d'Aquin, M., Motta, E.: Watson, more than a Semantic Web search engine. Semantic Web Journal 2(1), 55–63 (Jan 2011)

[14] Delbru, R., Campinas, S., Tummarello, G.: Searching web data: an entity retrieval and high-performance indexing model. Journal of Web Semantics 10(0) (2012)

[15] Fan, J., Porter, B.: Interpreting loosely encoded questions. In: 19th National Conference on Artifical Intelligence. pp. 399–405 (2004)

[16] Gelfond, M.: On stratified autoepistemic theories. In: Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1. pp. 207–211. AAAI'87, AAAI Press (1987)

[17] Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics 3(2), 158–182 (2005)

[18] Heim, P., Ziegler, J., Lohmann, S.: gFacet: A browser for the web of data. In: International Workshop on Interacting with Multimedia Content in the Social Semantic Web (IMC-SSW'08). vol. 417, pp. 49–58 (2008)

[19] Hildebrand, M., van Ossenbruggen, J., Hardman, L.: /facet: A browser for heterogeneous Semantic Web repositories. In: 5th International Semantic Web Conference(ISWC'06). vol. 4273, pp. 272–285 (2006)

[20] Hirst, G., St-Onge, D.: Lexical chains as representations of context for the detection and correction of malapropisms. In: Fellbaum, C. (ed.) WordNet: An electronic lexical database, pp. 305–332. The MIT Press, Cambridge, MA (1998)

[21] Huang, J., Abadi, D.J., Ren, K.: Scalable sparql querying of large rdf graphs. Proceedings of the VLDB Endowment 4(11), 1123–1134 (2011)

[22] Isaac, A., Van Der Meij, L., Schlobach, S., Wang, S.: An empirical study of instance-based ontology matching. In: 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference. pp. 253–266 (2007)

[23] Järvelin, K., Kekäläinen, J.: Cumulated gain-based evaluation of IR techniques. ACM Trans. Inf. Syst. 20, 422–446 (October 2002)

[24] Jiang, J.J., Conrath, D.W.: Semantic similarity based on corpus statistics and lexical taxonomy. Proceedings of the International Conference on Research in Computational Linguistics (1997)

[25] Karger, D.R., schraefel, m.c.: The pathetic fallacy of RDF. In: 3rd International Semantic Web User Interaction Workshop (2006)

[26] Khatchadourian, S., Consens, M.: Explod: Summary-based exploration of interlinking and RDF usage in the linked open data cloud. In: 7th Extended Semantic Web Conference(ESWC'10). pp. 272–287 (2010)

[27] Lei, Y., Uren, V., Motta, E.: Semsearch: A search engine for the Semantic Web. Managing Knowledge in a World of Networks pp. 238–245 (2006)

[28] Lesk, M.: Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In: SIGDOC '86: Proceedings of the 5th annual international conference on Systems documentation. pp. 24–26. ACM, New York, NY, USA (1986)

[29] Likert, R.: A technique for the measurement of attitudes. Archives of psychology 22(140), 1–55 (1932)

[30] Lloyd, J.W.: Foundations of Logic Programming. Springer-Verlag New York, Inc., New York, NY, USA (1984)

[31] Lopez, V., Pasin, M., Motta, E.: Aqualog: An ontology-portable question answering system for the Semantic Web. In: 2nd European Semantic Web Conference. pp. 546–562 (2005)

[32] Manning, C.D., Raghavan, P., Schütze, H.: Scoring, term weighting, and the vector space model. In: Introduction to Information Retrieval. Cambridge University Press, New York, NY, USA (2008)

[33] Navigli, R.: Word sense disambiguation: A survey. ACM Comput. Surv. 41(2) (2009)

[34] Neumann, T., Weikum, G.: The RDF-3X engine for scalable management of RDF data. The VLDB Journal 19(1), 91–113 (2010)

[35] Nielsen, J.: Usability engineering. Morgan Kaufmann, San Francisco (1993)

[36] Oren, E., Delbru, R., Decker, S.: Extending faceted navigation for RDF data. In: 5th International Semantic Web Conference(ISWC'06). pp. 559–572 (2006)

[37] Pearl, J.: Causality: Models, Reasoning, and Inference. Cambridge University Press, New York, NY, USA (2000)

[38] Piatetsky-Shapiro, G.: Discovery, analysis and presentation of strong rules. In: Piatetsky-Shapiro, G., Frawley, W.J. (eds.) Knowledge Discovery in Databases, pp. 229–248. AAAI Press (1991)

[39] Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)

[40] Reiter, R.: On closed world data bases. In: Gallaire, H., Minker, J. (eds.) Logic and Data Bases, pp. 55–76. Springer US (1978)

[41] Resnik, P.: Using information content to evaluate semantic similarity in a taxonomy. In: International Joint Conference on Artificial Intelligence. pp. 448–453 (1995)

[42] Rohloff, K., Dean, M., Emmons, I., Ryder, D., Sumner, J.: An evaluation of triple-store technologies for large data stores. In: On the Move to Meaningful Internet Systems Workshop. pp. 1105–1114. Springer (2007)

[43] Rokach, L., Maimon, O.: Data Mining with Decision Trees: Theroy and Applications. World Scientific Publishing Co., Inc., River Edge, NJ, USA (2008)

[44] Sakr, S., Al-Naymat, G.: Relational processing of RDF queries: a survey. ACM SIGMOD Record 38(4), 23–28 (Jun 2010)

[45] Sigurbjornsson, B.: TagExplorer: Faceted browsing of flickr photos. Tech. Rep. YL-2010-005, Yahoo! Research (August 2010)

[46] Song, D., Heflin, J.: Automatically generating data linkages using a domain-independent candidate selection approach. In: 10th International Semantic Web Conference. pp. 649–664 (2011)

[47] Tan, P.N., Steinbach, M., Kumar, V.: Introduction to Data Mining, (First Edition). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2005)

[48] Tran, T., Wang, H., Rudolph, S., Cimiano, P.: Top-k exploration of query candidates for efficient keyword search on graph-shaped RDF data. In: 25th International Conference on Data Engineering. pp. 405–416 (2009)

[49] Tummarello, G., Delbru, R., Oren, E.: Sindice.com: Weaving the open linked data. In: ISWC/ASWC, pp. 552–565 (2007)

[50] Unger, C., Bühmann, L., Lehmann, J., Ngonga Ngomo, A.C., Gerber, D., Cimiano, P.: Template-based question answering over RDF data. In: WWW. pp. 639–648 (2012)

[51] Walter, S., Unger, C., Cimiano, P., Bär, D.: Evaluation of a layered approach to question answering over linked data. In: ISWC, pp. 362–374 (2012)

Vita

Xingjian Zhang was born in Liaoning, China, on June 5, 1983, and grew up in Hubei until finishing high school in 2001. He studied at Tsinghua University, Beijing, between 2001 and 2007, and got his Bachelor and Master Degree in the Department of Automation. He first became interested in the Semantic Web while working on information integration for his master thesis. He then enrolled in Lehigh University, Bethlehem PA to do PhD research with advisor Professor Jeff Heflin on Semantic Web technologies. His research mainly focuses on improving user interaction with Semantic Web data. He designed the Contextual Tag Cloud system and the resulting system won the Billion Triple Challenge at the International Semantic Web Conference in 2012.