ABSTRACT

Title of dissertation: USING ONTOLOGIES TO IMPROVE ANSWER QUALITY IN DATABASES

Yu Deng, Doctor of Philosophy, 2006

Dissertation directed by: Professor V.S. Subrahmanian
Department of Computer Science

One of the known shortcomings of relational and XML databases is that they overlook the semantics of terms when answering queries. Ontologies constitute a useful tool to convey the semantics of terms in databases. However, the problem of effectively using semantic information from ontologies is challenging.

We first address this problem for relational databases by the notion of an ontology extended relation (OER). An OER contains an ordinary relation as well as an associated ontology that conveys semantic meaning about the terms being used. We then extend the relational algebra to query OERs. We build a prototype for the OER model and show that the system scales to handle large datasets.

We then propose the concept of a similarity enhanced ontology (SEO), which brings a notion of similarity to a graph ontology. We extend TAX, one of the best known algebras for XML databases, with SEOs. The result is our TOSS system that provides a much higher answer quality than TAX does alone. We experimentally evaluate the TOSS system on the DBLP and SIGMOD bibliographic databases and show that TOSS has acceptable performance.

These two projects have involved ontology integration for supporting semantic queries across heterogeneous databases. We show how to efficiently compute the canonical witness to the integrability of graph ontologies given a set of interoperation constraints. We have also developed a polynomial algorithm to compute a minimal witness to the integrability of RDF ontologies under a set of Horn clauses and negative constraints, and experimentally show that our algorithm works very well on real-life ontologies and scales to massive ontologies.

We finally present our work on ontology-based similarity measures for finding relationships between ontologies and searching similar objects. These measures are applicable to practical classification systems, where ontologies can be DAG-structured, objects can be labeled with multiple terms, and ambiguity can be introduced by an evolving ontology or classifiers with imperfect knowledge. The experiments on a bioinformatics application show that our measures outperformed previous approaches.

USING ONTOLOGIES TO IMPROVE ANSWER QUALITY IN DATABASES

by

Yu Deng

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

Advisory Committee:

Professor V.S. Subrahmanian, Chair/Advisor
Assistant Professor Lise Getoor
Professor James A. Hendler
Professor Dana S. Nau
Associate Professor Donald Yeung

DEDICATION

This dissertation is dedicated to my husband and family.

ACKNOWLEDGMENTS

I want to thank my advisor, Prof. V.S. Subrahmanian, who has given me the best guidance and support throughout my PhD study. I also want to thank Professors Lise Getoor, James A. Hendler, Dana S. Nau and Donald Yeung for being on my dissertation committee, and for their insightful comments and questions.

I would like to thank Dr. Harumi Kuno and Dr. Peter Schwarz for being my mentor during my internship at HP Labs and IBM Almaden respectively, where I advanced my research in schema mapping and semantic similarity search.

My study at Maryland has been a great experience and I would like to thank all of my colleagues. I especially thank Dr. Edward Hung, who often had stimulating ideas. Edward has given a lot of help to me and contributed to the TOSS project. I also want to give my sincere thanks to Octavian Udrea for the project of RDF integration and for having a great time to work with him. Additionally I want to thank T.J. Rogers for his technical support.

Finally, I must thank my family, who always believe in me and encourage me to pursue my academic goals. Most importantly, I thank my husband for being always with me to get through the difficulties in my study. Without the love and support from my family, it is impossible for me to finish my dissertation and obtain the degree of Doctor of Philosophy.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

Introduction

1.1 The Recall Problem

In recent years, people have developed various theories and techniques to handle queries for relational and XML data. For example, relational algebra [1] is one of the most important formalism of the relational data model. For XML, TAX [56] is one of the best known algebras. One problem with TAX and many existing relational DBMSs is that the semantics of terms in databases is not taken into account when answering queries. Thus, even though TAX and the relational DBMSs answer queries with 100% precision, the recall is relatively low.

For example, a TAX query that wishes to find all papers in the DBLP database written by "J. Ullman" will not find bibliographic references to papers by "J.D. Ullman" or by "Jeffrey Ullman." This is because TAX does not use any notion of *similarity* between search terms to answer queries. Likewise, TAX cannot answer queries of the form "Find all papers having at least one author from the US government." Such a query may be useful in order to identify papers that potentially have no copyright restrictions. However, few authors if any will list their affiliations as "US Government." They are more likely to list their affiliations as "US Census Bureau" or "US Army" and so on. As a consequence, TAX will miss these answers. Problems such as these are not really an artifact of TAX's design - rather, they are

caused by a general lack of *lexical semantics* in answering queries. Most commercial relational DBMSs would not perform such reasoning either when answering queries. The net effect of this shortcoming of TAX (as well as most commercial DBMSs) is that such systems have high precision (all the answers to a query are correct) but poor recall (not all the answers that should be returned are in fact returned). As commonly adopted by the IR and DB communities, the precision of an answer is the number of correct results returned divided by the number of results returned and the recall of an answer is the number of correct results returned divided by the total number of correct results that should have been returned. In Chapter 3, we introduce recent works on combining IR techniques and relational/XML database management systems for supporting keyword search, proximity search, relevance ranking, word stemming and thesauri.

The recall problem exists not only in single DBMSs, but also when integrating multiple DBMSs. For relational databases, though the problem of integrating these diverse databases has been studied extensively [111] and many impressive mediator systems have been built, it remains a fact that integration of data at the semantic level remains an open problem. Consider a simple example consisting of industrial parts data. There are numerous companies in the US and elsewhere that maintain stocks of various industrial parts. Two different companies of this kind may maintain part information in varying representations. One company may use the column *Cost* to represent the price of the items and *Shipping* to represent the shipping cost while the other one may use the columns *Price* and *ShipCost*. By looking at the fields, we are not sure whether Cost and Price refer to the same things or the columns Shipping

and ShipCost refer to the same concept. Note that the semantic ambiguities exist not only in schema, but also in instance data. In Chapter 3, we give a review on several well-known mediator systems.

We address the recall problem by introducing the concept of an ontology to capture inter-term lexical relationships. In Chapter 2, we give a review on ontologies and ontology languages. In Chapter 4, we define a graph-based ontology and propose the notion of an ontology extended relation (OER). An OER contains an ordinary relation as well as an associated ontology that conveys semantic meaning about the terms being used.

We extend the relational algebra to query OERs. Furthermore, we show how multiple ontologies may be merged, which is imperative when executing queries over multiple ontology extended relations. We describe an implementation of the OER model and show (via experiments) that the system scales to handle large data sets.

We have made a similar effort on XML database systems. But differently, we investigate how to use both similarity metrics and ontologies to answer queries correctly. In Chapter 5, we introduce the TOSS system, which extends and enhances the semantics of TAX (a tree algebra for XML databases) so that the resulting system returns high quality answers. We show how to generate a *similarity enhanced ontology*, a concept that brings a notion of similarity into a integrated ontology. We describe a prototype implementation of TOSS on top of the Apache Xindice XML database system [115]. Our experiments conducted on the SIGMOD XML proceedings data set[96] and the DBLP data set[31] show that the TOSS system consistently outperformed ordinary XML query engines in terms of answer quality.

1.2 Ontology Integration

It is a common case that distributed data sources may be associated with different ontologies, but on similar topics. These associated ontologies provide semantical information about the data. When semantically integrating such data and performing queries over multiple ontology extended data sources, we may need to merge multiple ontologies. On the other hand, since the adoption of "Resource Description Framework" (RDF) as a web recommendation by the World Wide Web Consortium, there has been growing interest in using RDF for expressing ontologies about a diverse variety of topics. As more and more ontologies emerge about the same topics, there is also a growing need to integrate these ontologies.

There are some initial approaches to merging ontologies in the literature. The initial pioneering work of [80] showed that ontology merging is an important problem. In another important paper, [24] develops a model theoretic basis for merging ontologies assuming they are in description logic. [98] uses natural language and formal concept analysis methods to merge ontologies using a concept lattice which is explored and transformed by user interactions. Meanwhile, there are many works on finding relationships between ontologies, e.g. [21, 75]. We will give details on the related work in Chapter 3.

Meanwhile, extensive research has been performed on integrating logical theories from multiple sources. In Chapter 3, we introduce these research efforts motivated by problems arising in areas such as cooperative information systems, multidatabases, multi-agent systems and distributed expert systems. These works may

be applied to ontology integration. However, due to the special properties of some ontologies, especially the underlying graph models, it is often possible to find a more efficient method to integrate them, as we show in Chapter 4 and Chapter 6.

In Chapter 4, we provide a definition of a graph-based ontology and study the problem of integrating ontologies under a given set of interoperation constraints. We formally define the notion of canonical witness to the integrability of a set of ontologies under such constraints. We have established a theory that a set S of ontologies is integrable if and only if the canonical witness is a witness to the integrability of the ontologies in S. We further provide an efficient algorithm to compute the canonical witness given a set of ontologies and their interoperation constraints.

We present our approach on integrating RDF ontologies in Chapter 6. Our framework allows relationships to be not only between terms, but also allows complex Horn Constraints, the Horn clauses that specify semantic relationships among terms in ontologies. Note that our approach is rooted in the novel concept of an integration witness and in graph theoretic methods. We also show the correctness and complexity proofs of our approach as well as the experimental results on both synthetic and real ontologies.

1.3 Ontology-based Semantic Similarity Measures

Finding interoperation constraints between ontologies is an important step for integrating them. Since most existing approaches focus on the structures of ontolo-

gies, how to find such constraints at the semantic level is a challenging problem. In Chapter 7, we introduce our work on ontology-based semantic similarity measures which are applicable for identifying similarity relationships between two terms, two sets of terms and two individuals.

Our measures are also useful for finding similar objects given a description of a target in a domain. For example, one may wish to find patents similar to a given patent, or subjects similar to a hypothetical "ideal" subject for a clinical trial, or gene products (e.g. proteins) similar to a given gene product. When merging data from multiple sources, it is often important to efficiently identify duplicate objects.

There have been some approaches on string matching, object matching and entity resolution in the literature. For example, Cohen et al. [28] conduct extensive experiments to compare a set of string metrics for matching names and records, Zhou et al. [116] present a framework for supporting object matching in integrating heterogeneous data, Hernandez and Stolfo [49] study the problem of merging data and identifying distinct entities in large databases, Bhattacharya and Getoor [18] show an iterative deduplication algorithm to identify similar entities, and Dong et al. [35] propose a couple of algorithms for supporting similarity search for web services. However, none of the above research groups has considered to use taxonomies (ontologies) to find semantically similar objects, e.g., those objects that are highly related, but not matched by traditional IR metrics.

Taxonomies have been a useful tool for classification. They also provide a means of determining how similar one such individual is to another. In its simplest form, a taxonomy defines a hierarchical grouping of individuals into ever more

specific classes. Two individuals share the properties of the most specific grouping that includes both of them, and the degree to which the two individuals are similar depends on the location of this class in the hierarchy. The lower in the hierarchy, the more similar the individuals are. For example, two apples of the same species are more similar than apples of different species, and an apple is more similar to another apple of any species than it is to an orange. Based on this intuitive idea of similarity, various authors, e.g. [94, 114, 66], have defined different ways to compute a numeric value for measuring the similarity of objects. We will give details about these works in Chapter 3.

However, previous ontology-based similarity measures may not be applicable for complex cases as follows. Firstly, it is frequently the case that a class of individuals may specialize the properties of more than one parent class, i.e., taxonomies are often DAG-structured. Furthermore, taxonomies often evolve, as new specialized groupings are formed and older ones are reorganized. Even with an unchanging taxonomy, the classification of a particular object may evolve as more is learned about it, or users of the taxonomy may disagree as to how it should be classified. Thirdly, real taxonomies tend to be quite large, and the sets of objects they are used to classify are often very large. Lastly, in practice, an object is often associated with a set of terms (instead of one term) in a taxonomy.

In Chapter 7, we present a pragmatic approach to the use of taxonomies to identify similarity relationships between terms and objects. We have defined two taxonomy-based similarity measures to take the above cases into account. The first, *holistic similarity*, is a new information-theoretic similarity measure that is more

general and well-founded than prior art. The second, *generic holistic similarity*, adapts our holistic similarity measure to cope with ambiguity, as introduced by an evolving taxonomy or classifiers with imperfect knowledge. To the best of our knowledge, this is the first attempt to consider this latter problem. We also describe a scalable implementation of our measures that is tightly integrated with an object-relational database, and we evaluate our approach by applying it to an object-matching problem from bioinformatics for which the correct answers are known *a priori*. The results show that our new measures are more successful than those previously reported, and that our implementation scales well for large taxonomies and data sets.

Chapter 2

Review of Ontologies

According to Thomas Gruber [41], "an ontology is an explicit specification of a conceptualization". Here, a conceptualization describes the classification of entities in the universe and how they are related. An ontology defines a set of representational terms to name the entities and associates text and axioms with the terms to constrain their interpretation.

In a more practical point of view, ontologies provide a shared and common understanding of a domain that can be communicated between people and heterogeneous and distributed application systems [61]. They are (meta)data schemas, providing a controlled vocabulary of concepts, each with an explicitly defined and machine processable semantics [72]. Ontologies play increasingly important roles in the areas of Data Integration, Information Retrieval and Knowledge Sharing and Reuse, etc. In 2001, Berners-Lee et al. [15] proposed the concept of Semantic Web and described a promising picture for this future generation of World Wide Web. Ontologies are used in this picture to resolve semantic ambiguities between different databases. Since then, this picture has triggered a strong interest on ontology research in the Semantic Web community.

In this chapter, we give a brief introduction to several well-known or widely-used ontologies/ontology languages, which include graph-based ontologies, RDF,

SHOE, OWL, DAML+OIL and Description Logics.

2.1 Graph-based Ontologies

Graph-based ontologies cover various kinds, ranging from simple catalogs, IS-A hierarchies, to complex graph systems, such as WordNet. These ontologies are closely related to two early knowledge representation systems, semantic networks and frame systems.

In 1960's, Quillian proposed semantic networks [90], where each concept is represented by a node (called type node) in a graph. The meaning of each type node is implied by an associated structure, which consists of "token nodes" interconnected by different kinds of links. These links are further labeled with semantic relations, such as conjunctive and disjunctive relations. In addition, it is the token nodes that connect a type node to other type nodes, because each token node is linked to a type node out of the structure that the token node belongs to.

Later, Minsky introduced frame systems [79]. A frame is a named data object with a set of slots that represent properties of the object. Each slot can specify the conditions its assignments must meet. A assignment of a slot can be an object, or a pointer to another frame. Meanwhile different frames can share slots in order to represent different ways of using the same information. Although frame systems seem to have complex structures, it has been shown that those systems are isomorphic to semantic networks.

Recent years, many graph-based ontologies have been built for specific appli-

cation domains. They more or less inherit some features from semantic networks and frame systems. WordNet, as an example, is one of them.

WordNet is a lexical reference system which organizes lexical information by semantic relations [78]. These relations include synonymy, antonymy, hyponymy, and meronymy, etc. In WordNet, English nouns, verbs and adjectives are grouped into synonym sets, each of which represents one lexical concept. For example, {board, plank} and {board, committee} designate two different meanings of board. Semantic relations, like pointers, link the synonym sets. It is important to note that the three categories, nouns, verbs and adjectives, have different semantic organizations in WordNet. For instance, nouns are organized into hierarchical trees while the organization of adjectives is more or less a n-dimensional hyperspace. WordNet has been widely used in information retrieval systems. A couple of semantic similarity measures have been defined with regard to such a large lexical database. We will come back to this in Chapter 3.

We also propose a simple notion of a graph-based ontology [19] which is a partial mapping from a set of relations to a set of hierarchies. We formally define the concept of "canonical witness", based on which we provide an efficient algorithm to merge such ontologies under a set of interoperation constraints. The details are given in Chapter 4. Note that our notion of ontology generalizes the ontologies in RDF, DAML+OIL and OWL, of which the underlying models are graphs.

2.2 RDF

Resource Description Framework (RDF) is a W3C recommendation endorsed by approximately 400 companies. It is a foundation for processing metadata and it provides interoperability between applications that exchange machine-understandable information on the Web [64]. Lassila et al. [64] introduce a model for representing RDF metadata as well as a syntax for encoding and transporting the metadata. Their work is clarified and updated by W3C RDF Primer [74]. Brickley et al. [22] introduce RDF's vocabulary description language, RDF Schema (RDFS). Hayes [46] specifies a precise semantics and corresponding complete systems of inference rules for RDF and RDFS.

The basic elements of RDF are triples with the form (subject, predicate, object). RDF is property-centric. RDF Schema defines vocabulary to describe classes, properties and other resources.

2.2.1 Syntax of RDF

RDF defines a model for describing relationships between resources in terms of uniquely identified properties (attributes) and values. RDF has become the standard for the description and exchange of metadata on the Web.

The underlying model for RDF is a labeled directed graph where nodes are either *resources* or *literals*. The graph is defined by a set of *triples*, statements of the form (subject, predicate, object), where subject is a resource, predicate is the edge label and object is either a resource or a literal. The basic elements in RDF data

model are as follows:

- *Resource*: A resource is anything identified by a URI reference (URI plus optional anchor ids) and described in terms of simple properties and property values.

- *Literal*: A literal may be plain or typed. A plain literal is a self-denoting string combined with an optional language tag. A typed literal is a string combined with a datatype URI reference. It denotes the member of the identified datatype's value space obtained by applying the lexical-to-value mapping to the literal string.

- *Property*: A property is a resource that represents a specific aspect, characteristic, attribute, or relation used to describe resources. The set of properties is a subset of the set of resources.

- *Statement*: A statement is composed of a subject, a predicate and an object in an ordered manner. It is also called a triple. With a statement, a specific resource (the subject) is associated with a property (the predicate) and a value of that property (the object). Specially, a statement can be transformed into a resource with a URI reference using reification, a mechanism in RDF for describing statements.

Example 1 *The bottom half of Figure 2.1 shows an RDF example for artifact auctions. The top half shows the corresponding RDF schema. As illustrated in the figure, the resource r1, with URI http://www.artist.net#guyrose, is associated with sev-*

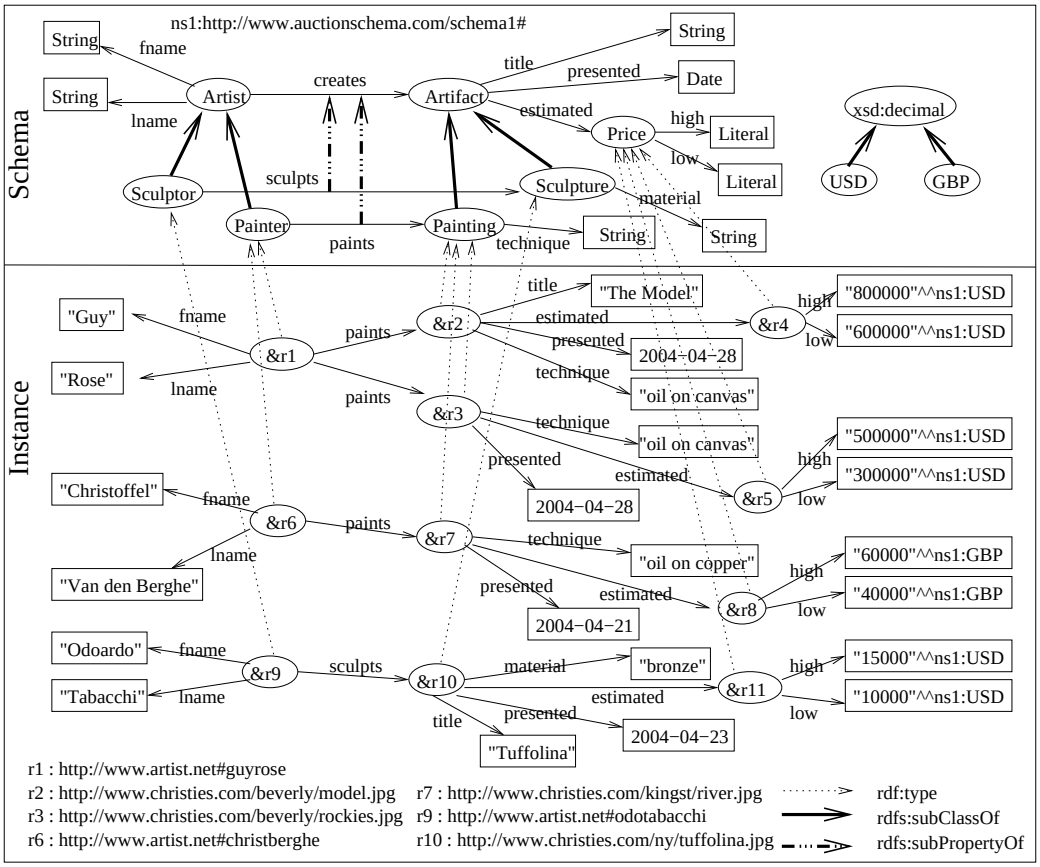


Figure 2.1: An RDF example describing artifact auctions in Christie's auction house

eral properties, e.g. <ns1:fname> with URI http://www.auctionschema.com/schema1♯fname (ns1 is the namespace http://www.auctionschema.com/schema1♯). The value of <ns1:fname> for the resource r1 is the plain literal "Guy". Furthermore, the statement (<http://www.artist.net♯guyrose>, <ns1:fname>, "guy") associates the resource r1 with the property <ns1:fname> and the value "Guy".

In contrast to an object-oriented system, RDF is property-centric instead of object-centric, although people might think RDF resources can be analogous to objects and RDF properties are similar to attributes. RDF Properties are themselves resources, and are independent from classes. The relationships between RDF resources and

properties are not tightly restricted by schemas.

2.2.2 Semantics of RDF

RDF is an assertional logic, in which each triple expresses a simple proposition. An RDF graph is a set of RDF triples and a ground RDF graph is one with no blank nodes (anonymous resources without URI references).

Hayes [46] gives very detailed discussion about RDF semantics. For reader's convenience, we present the important concepts of Graph Instance, Merge of RDF Graphs and Interpretation here. Interested readers may refer to [46].

Definition 1 (Graph Instance [46]) *Suppose M is a mapping from a set of blank nodes to some set of literals, blank nodes and URI references. Then any graph obtained from a graph G by replacing some or all of the blank nodes N in G by M(N) is an instance of G.*

A proper instance of a graph is an instance in which a blank node has been replaced by a name, or two blank nodes in the graph have been mapped into the same node in the instance. An RDF graph is lean if it has no instance which is a proper subgraph of the graph.

Definition 2 (Merge of RDF Graphs [46]) *If the graphs in the set have no blank nodes in common, then the union of the graphs is a merge; if they do share blank nodes, then it is the union of a set of graphs that is obtained by replacing the graphs in the set by equivalent graphs that share no blank nodes.*

Definition 3 (Interpretation [46]) *A simple interpretation I of an RDF vocabulary V is defined by:*

1. *A non-empty set IR of resources, called the domain or universe of I.*
2. *A set IP, called the set of properties of I.*
3. *A mapping IEXT from IP into the powerset of IR x IR i.e. the set of sets of pairs <x,y> with x and y in IR .*
4. *A mapping IS from URI references in V into (IR union IP).*
5. *A mapping IL from typed literals in V into IR.*
6. *A distinguished subset LV of IR, called the set of literal values, which contains all the plain literals in V.*

2.2.3 RDF Schema

RDF-Schema (RDFS) extends RDF with special class and property constructs that may be used to describe classes, properties and other resources. The important constructs in RDFS are as follows:

- rdfs:Resource. All RDF resources are instances of rdfs:Resource.
- rdfs:Class, the class of resources that are RDF classes.
- rdfs:Literal, the class of literal values. rdfs:Literal is an instance of rdfs:Class and a subclass of rdfs:Resource.

- rdfs:subClassOf, an instance of rdf:Property that is used to state that all the instances of one class are instances of another. It is a transitive relationship w.r.t. class hierarchy.

- rdfs:subPropertyOf, an instance of rdf:Property that is used to state that all resources related by one property are also related by another. It is a transitive relationship w.r.t. property hierarchy.

- rdf:type, an instance of rdf:Property that is used to state that a resource is an instance of a class.

- rdfs:domain, an instance of rdf:Property that is used to state that any resource that has a given property is an instance of one or more classes. For example, a triple of the form (P rdfs:domain C) states that the resources denoted by the subjects of triples whose predicate is P are instances of the class C.

- rdfs:range, an instance of rdf:Property that is used to state that the values of a property are instances of one or more classes. A triple of the form (P rdfs:range C) states that the resources denoted by the objects of triples whose predicate is P are instances of the class C.

Besides the above constructs, RDFS also defines a vocabulary for RDF container classes and properties, RDF collections and reification, etc. Please refer to [22] for details.

As pointed out by Staab et al. [97], while support for modeling of ontological concepts and relations has been extensively provided in RDF(S), the same cannot

be said about the modeling of ontological axioms since RDF(S) has relatively weak expressiveness. They propose an approach to model axioms in RDF(S) while the core semantics of RDF(S) is re-used and the semantics is preserved between different inferencing tools.

2.3 SHOE

In 1995, Hendler's group proposed the SHOE language at the University of Maryland, and later they refined the language and experimented its use [71, 48, 47]. SHOE, which stands for Simple HTML Ontology Extensions, supports knowledge acquisition by augmenting the Web with tags that provide semantic meaning. To eliminate the possibility of contradictions between agent assertions, the designers of SHOE carefully chose some features for the language. For example, SHOE does not permit logical negation, nor the specification of disjoint classes.

Syntactically, elements of SHOE can be described in HTML extended with additional semantic tags, or in XML. A SHOE ontology has both an identifier and a version number, where all ontologies with the same identifier are different versions of the same ontology. For example [1],

```
<ONTOLOGY ID="university-ont" VERSION="1.0">
```

declares an ontology called "university-ont" whose version is 1.0. In addition, SHOE allows for ontology inclusion, which helps to reuse ontologies.

A SHOE ontology may contain a number of elements, in which categories,

[1] All the examples of the SHOE language are from [47].

relations, and inference rules are important components. A category (or class) is a set of objects that share some common properties. The $is-a$ relation is commonly used to group categories together. For example,

```
<DEF-CATEGORY NAME="Faculty" ISA="Person">
```

defines a category called "Faculty", which is a subcategory of "Person". In SHOE, a relation is equivalent to a n-ary predicate with zero or more arguments, which are typed and explicitly ordered. For instance, a relation of "advises" can be defined as follows:

```
<DEF-RELATION NAME="advises">
    <DEF-ARG POS="1" TYPE="Faculty">
    <DEF-ARG POS="2" TYPE="Student">
</DEF-RELATION>
```

where "Faculty" and "Student" are its two arguments. SHOE also supports inference rules, each of which consists of a set of antecedents and a set of consequents. Notice that the axioms in SHOE are restricted to Horn clauses since SHOE's semantics is based on datalog. The following example defines a rule that the head of a Department is a Chair:

```
<DEF-INFERENCE>
    <INF-IF>
        <RELATION NAME="g.headOf">
            <ARG POS="1" VALUE="x" USAGE="VAR">
```

```
            <ARG POS="2" VALUE="y" USAGE="VAR">
        </RELATION>
        <CATEGORY NAME="Department" FOR="y" USAGE="VAR">
    </INF-IF>
    <INF-THEN>
        <CATEGORY NAME="Chair" FOR="x" USAGE="VAR">
    </INF-THEN>
</DEF-INFERENCE>
```

The subclauses enclosed by <INF-IF> and </INF-IF> tags are antecedents and the ones enclosed by <INF-THEN> and </INF-THEN> are consequents.

2.3.1 Semantics of SHOE

A set of functions has been defined to translate different constructs of the SHOE syntax to concepts in the logical model. For example, $res : Url \rightarrow R$ maps a uniform resource locator from the set Url to a specific resource in R. Via these functions, each tag in the SHOE language can be mapped into a set of assertions. For detailed description, please see [47].

2.4 DAML+OIL

DAML+OIL [29], a semantic markup language for Web resources, provides modeling primitives similar to frame-based languages. It was built from the original DAML ontology language as well as the language components of OIL, the Ontology

Interface Layer defined by Horrocks et al. [50]. Figure 2.2 shows a fragment of a weather ontology in DAML+OIL.

A DAML+OIL ontology consists of zero or more headers, followed by zero or more class elements, property elements, and instances. The headers contain version information and imports elements. As shown in Figure 2.2, the <daml:Ontology> element contains the version information. A class element, <daml:Class>, contains the definition of an object class. It refers to a class name (a URI), such as "UnitAbbreviation" in the example ontology, and may contain other elements, like <daml:disjointWith>. A daml:Property element refers to a property name (a URI), e.g., "hasUnitAbbreviation", and may contain other elements, like <rdfs:domain> and <rdfs:range>. There are two kinds of instances: class instances and property instances. For example, in Figure 2.2, celsiusMap is an instance of class UnitAbbreviation. [29] provides details about syntax of DAML+OIL and [106] gives the model-theoretic semantics for DAML+OIL.

2.5 OWL

OWL [13, 88], the Web Ontology Language, is a semantic markup language for publishing and sharing ontology on the World Wide Web. It is a W3C recommendation and provides three subsets of language constructs with different features: OWL Full, OWL DL and OWL Lite.

OWL Full allows free mixing of OWL with RDF Schema, but it is not decidable although it gains the highest expressivemess in the three sublanguages of OWL. OWL

```
<rdf:RDF
  xmlns="http://www.csd.abdn.ac.uk/research/AgentCities/WeatherAgent/weather-ont.daml"
  xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

  <daml:Ontology rdf:ID="metarOnt">
    <daml:versionInfo>
      $Id: metar1.1.daml,v 1.12 2002/01/23 16:11:17 ggrimnes Exp $
    </daml:versionInfo>
  </daml:Ontology>

  <daml:Class rdf:ID="UnitAbbreviation">
    <rdfs:label>Unit Abbreviation</rdfs:label>
  </daml:Class>

  <daml:Property rdf:ID="hasUnitAbbreviation">
    <rdfs:label>Abbreviation</rdfs:label>
  </daml:Property>

  <UnitAbbreviation rdf:ID="celsiusMap">
    <hasUnitAbbreviation>C</hasUnitAbbreviation>
    <unitPropertyName rdf:resource="#celsuisTemperature"/>
  </UnitAbbreviation>

</rdf:RDF>
```

Figure 2.2: A fragment of a weather ontology in DAML+OIL

DL is a subset of OWL Full and was designed to support the existing Description Logics. Note that OWL DL maintains decidability while still has high expressiveness. OWL Lite was targeted for easy implementation and providing users a functional subset of constructs. Besides abiding by all the restrictions of OWL DL, it has further requirements for some constructs. Figure 2.3 shows a fragment of an example OWL DL ontology [13].

It is important to note that the meaning of an OWL ontology is solely determined by the underlying RDF graph. Patel-Schneider et al. [88] provide a thorough analysis in the model-theoretic semantics for OWL.

2.6 Description Logics

As introduced in [8, 25], Description Logics are a family of knowledge representation languages that can be used to represent the knowledge of an application domain in a structured and formally well-understood way. They allow for modeling an application domain in terms of objects, classes and relationships between classes, and for reasoning about them. The important notions of a given domain are described by concept *descriptions*, which are expressions built from atomic concepts and roles. A knowledge base in DLs is formed by two components, called TBox and ABox. A TBox expresses intensional knowledge about classes and relations and an ABox expresses extensional knowledge about individual objects. Furthermore, DLs are equipped with a formal and *logic*-based semantics.

An important problem for a DL system is the trade-off between the expressivity

```
<owl:Class rdf:about="#MusicDrama">
  <owl:equivalentClass>
    <owl:Class>
      <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Opera"/>
        <owl:Class rdf:about="#Operetta"/>
        <owl:Class rdf:about="#Musical"/>
      </owl:unionOf>
    </owl:Class>
  </owl:equivalentClass>
</owl:Class>

<owl:Class rdf:about="#Opera">
  <rdfs:subClassOf rdf:resource="#MusicDrama"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasLibrettist" />
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinality>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Figure 2.3: A fragment of OWL ontology

of DL and the complexity of its inference. Current DL systems can employ very expressive DLs while still providing powerful inference tools. These strengths have made DLs a popular choice for representing ontologies.

Note that our notion of graph-based ontology cannot generalize the ontologies in Description Logics. In addition, since OWL DL is based on Description Logics, it cannot generalize OWL DL either.

Chapter 3

Review of the State of the Art

3.1 Using IR Techniques to Improve Answer Quality

In recent years, it has become more and more common to have unstructured text embedded in or connected to structured data in database systems. For example, a database about books may have unstructured text about readers' comments and a database about patients may be associated for each patient record with detailed symptoms stored as text. This has motivated the research on supporting keyword search in databases.

On the other hand, since they may not know the detailed content or structure of a database, users tend to be interested in (and often satisfied with) the results that do not exactly match the query, but are highly relevant. Furthermore, they often hope to have results ranked according to their relevance to the query.

As these changes and query patterns have attracted more and more attention from both academia and industry, many research efforts have emerged to combine IR techniques and relational/XML database management systems to query unstructured text embedded in (or connected to) structured data and to support keyword search, proximity search, relevance ranking, word stemming and thesauri in database systems. In this section, we introduce some of the works in keyword search and relevance ranking for relational and XML database systems respectively. Note that some

of these systems have incorporated ontological information into query answering.

3.1.1 IR and RDBMS

DBXplorer [2] and DISCOVER [52] are two early systems to support keyword search in relational database systems. But they do not support IR style ranking. Commercial relational database management systems have provided keyword search capabilities over single columns [30, 85, 76]. They use standard IR techniques to rank query results.

However, several research groups have found that using standard IR ranking functions has a few shortcomings. For example, traditional IR functions miss records that do not contain the keywords, but are closely related to them. Furthermore, the traditional IR semantics are unable to meaningfully sort the resulting objects according to their relevance to the keywords [9]. Instead, they [9, 53] have adapted PageRank [23] technology for ranking query results in databases.

Balmin et al. [9] develop the ObjectRank system which models databases as labeled graphs and applies authority-based ranking to keyword search. The idea is to first identify a base set where each object contains the keywords. Then authority originates at the objects in the base set and flows to objects according to their semantic connections. ObjectRank makes use of schema information when computing the scores. In addition, it considers domain-specific link semantics and can rank results with regard to specific keywords. However, it needs to materialize the database graph which poses large space overhead. Although it incorporates certain

semantics, the system cannot support arbitrary semantic relationships such as is-a and part-of. Furthermore, it relies on the base set which still requires exact match.

Note that the above research efforts do not consider answering a keyword query by joining tuples from multiple relations (which is called free-form keyword search). Hristidis et al. [51] adapt IR-style document-relevance ranking strategies to the problem of precessing free-form keyword queries over RDBMSs. Different from previous research efforts, their solution for ranking query answers is based on existing search capabilities in RDBMSs as well as IR ranking techniques. In addition, they can handle queries with not only AND, but also OR semantics. They have also introduced several top-k query-processing algorithms and compared the performance of the algorithms with regard to different parameters.

Different from the above work, we do not consider keyword search or relevance ranking in our HOME system. Instead, our goal is to extend the relational algebra with ontological information such that it is feasible to integrate heterogeneous relational databases at the semantic level. We provide details about the HOME system in Chapter 4.

3.1.2 IR and XML DB

To the best of our knowledge, Florescu et al. [36] were the first to combine XML query processing and keyword search capabilities. They extend the XML-QL query language by adding a predicate called "contains", which has four arguments: an XML element variable, a word, an integer expression and a boolean expression.

The integer expression limits the depth at which the word is found within the element. The boolean expression over the set of constants {*tag_name*, *attribute_name*, *content*, *attribute_value*} imposes a constraint on the location of the word. For example, contains($A, "Database", 3, any) is used to search for the elements containing the string "Database" with a search depth no more than 3. The authors also discussed how to process such keyword queries using a relational database management system. However, their work is limited to single keyword search. They do not consider other cases, such as multiple keywords with AND/OR semantics.

Amer-Yahia et al. [4] propose TeXQuery, a full-text search extension to XQuery. Instead of exploring only a few primitives at a time to support full-text search, they provide a rich set of fully composable primitives including Boolean connectives, phrase matching, proximity distance, word stemming and thesauri. They introduce an underlying data model called $FullMatch$ to guarantee that full-text search primitives can be closed under the model. The $FullMatch$ model also helps to integrate the primitives seamlessly into the XQuery language. An interesting feature of TeXQuery is that it provides an interface to incorporate terms from thesauri, such as synonyms. However, the system does not support other kinds of queries (such as join) with regard to thesauri.

Note that the above systems do not consider relevance ranking for query results. We introduce a couple of research efforts to support ranking in XML databases.

The TAX algebra by Jagadish et al. [56] is a formal theoretical basis for semistructured data. It proposes the concept of a pattern tree to query semistruc-

tured data sources. However, ontologies are not used in TAX and TAX does not support similarity search. Based on TAX, Al-Khalifa et al. introduced a bulk algebra TIX [3] to integrate information-retrieval style query processing into a traditional pipelined query evaluation engine for an XML database. TIX uses a scored pattern tree instead of a pattern tree as in TAX. User-defined score functions are given to some nodes in a scored pattern tree to specify how to compute scores of nodes using IR-style conditions. By extending existing operators in TAX and introducing new operators, an IR-style query can be expressed in TIX to find relevant results, with weighting and ranking support. However, they only support traditional IR methods (i.e., exact keyword match) for scoring such as term frequencies. Hence, they cannot incorporate lexical relationships into query answering. As a result, TIX may miss those instances that are highly related to a query but do not contain those keywords in the query. Compared to TIX, our work on TOSS uses similarity enhanced ontologies to support both lexical and similarity relationships between terms in databases. In an ontology, users can define arbitrary relationships that they are interested in. In addition, TOSS provides mechanisms to resolve semantic ambiguities when answering queries across heterogeneous databases which TIX does not consider.

Guo et al. [42] present the XRANK system that supports ranking results for keyword search queries over XML documents. The ranking is at the granularity of an XML element and the algorithm for computing the ranking is based on the PageRank [23] technology. Different from previous work, they support two dimensional proximity search by considering both the keyword distance and ancestor distance.

They also propose new inverted list index structures to efficiently evaluate XML keyword search queries.

Theobald and Weikum [100] present an interesting work on relevance ranking in XML databases. They consider a similarity operator $\sim$ which takes ontological information into account. They also propose similarity metrics (with regard to ontologies) and special index structures to evaluate XML queries with similarity search conditions. Different from our work, they only consider similarity relationships (including synonyms, hypernyms and hyponyms), but do not support other interterm lexical relationships such as *part-of*. Our TOSS system supports both interterm lexical relationships and similarity metrics defined in the IR literature. Another difference is that their work is based on an XML query language while our work is based on an XML algebra. Furthermore, we support semantic integration of XML databases but they focus on relevance ranking.

3.2 Mediator Systems

Wiederhold [110, 111] proposed the important concept of a mediator - a program that would integrate data from multiple heterogeneous information sources. The first efforts of mediator systems were SIMS [6], HERMES [70] and TSIMMIS [37]. Subsequent efforts included the DISCO system [101].

SIMS [6] is targeted at efficiently mapping a query at the domain level into a set of subqueries to individual information sources. A planner is used to select the information sources to be used in answering a query and also to order the

subqueries in a more efficient way. Their domain model is a simple form of ontology which is used to reformulate an original domain-model query into a query in terms of database concepts. This step of reformulation is important to find out the related information sources. However, different from our work, their focus is not on defining an ontology-extended algebra. They define a set of operators, some of which are similar to our operators in selection conditions. They use constraints to specialize concepts while we use ontology itself.

Similar to SIMS, most of these efforts focused on developing query languages, optimization algorithms or caching strategies with regard to the system performance. However, the above efforts do not take into account lexical semantical relationships between terms occurring in databases such as those shown in previous examples. They do not have the ability to automatically reason that "US Census Bureau" is *part-of* "US Government" and "Maryland" is *part-of* the "USA", and so on. Other such relationships between terms include *isa* (e.g. duck *isa* bird) or *affects* (e.g. rain *affects* visibility).

3.3 Ontology Merging and Alignment

Wiederhold et al [112, 73, 81] at Stanford University have proposed an ontology algebra for manipulating ontologies from different domains. Their algebras consisted of logical statements [73] using a LISP style syntax. In their later work (combined in their ONION system), they represent ontologies with a graph-based model and provide a set of operators to articulate multiple ontologies into a unified

ontology using both logical rules and functional rules. However, they do not consider *interoperation constraints* in [112, 73], which we think are important when merging ontologies. Although a similar concept was considered in [81], their integrated ontologies were not concise. Furthermore, [112, 73, 81] only merges together ontologies without directly addressing the impact of how the query algebra is affected. And, no concept of similarity search is involved in these works.

[21] extends the syntax of OWL by adding constructs to express relationships between multiple OWL ontologies, but don't actually tell us how to merge ontologies together using these relationships. [75] describes a tool called Chimaera that finds taxonomic "areas" for merging as well as a list of similar terms. Noy and Musen [84] propose the PROMPT algorithm to semi-automatically merge and align ontologies. Based on an initial matched terms, the PROMPT algorithm iteratively executes user's selected operation, checks for any conflicts and makes a list of suggestions to user. Different from previous tools, the PROMPT approach provides good guidance to the user in the process of ontology alignment. Notice that the output of the ontology alignment tools can be used as interoperation constraints for our ontology integration algorithm in Chapter 4.

Dou et al. [33] proposed an approach (based on ontology merging and automated reasoning) for ontology translation, which is useful for datasets translation, ontology extension generation and querying through different ontologies. They built an inference engine to process assertions and queries in semantic translation. Different from ontology alignment, ontology translation requires human experts to specify semantic differences between ontologies.

On the other hand, extensive research has been performed on integrating logical theories from multiple sources (e.g., [65, 26, 40, 14, 62, 63, 67, 99, 12, 11, 10].) These studies have been motivated by problems arising in areas such as cooperative information systems, multi-databases, multi-agent systems and distributed expert systems.

Baral et al. [12, 11, 10] have studied the combination of multiple knowledge bases under a set of integrity constraints when each of the knowledge bases is a normal Horn logic program, a first order theory and a default logic theory respectively. Their idea is to obtain a maximally combined knowledge base with respect to the union of knowledge bases that is consistent with the integrity constraints.

Subrahmanian [99] presents a uniform theoretical framework, based on *annotated logics*, for amalgamating multiple knowledge bases when these knowledge bases (possibly) contain inconsistencies. In the framework, individual knowledge bases are transformed into new annotated logic programs and amalgamated under a new axiom scheme.

Based on epistemic logic, Liau [65] provides a modal logic framework for reasoning about multi-agent belief and its fusion. Two strategies are considered for the cautious merging of beliefs. In addition, the author proposes two logics for these two strategies respectively. An important concept of this framework is the different degrees of reliability for the agents.

Different from the above efforts, we study the problems of integrating graph ontologies and RDF ontologies in Chapter 4 and Chapter 6 respectively. We have provided efficient algorithms to integrate the ontologies based on their special prop-

erties.

3.4 Object Matching

The problem of object matching is important in data integration. It helps to determine whether object representations in different databases refer to the same entity and helps to remove duplicates when combining information from different sources. There has been a large body of work in this aspect. Object matching may be referred to as entity resolution, object identification, and record linkage, etc. We review some of them in Section 3.4.1.

3.4.1 Entity Resolution

Entity resolution has received significant attention from both industry and academia, due to its wide use in many problems, such as deduplication in data integration and reference reconciliation in Personal Information Management. The approaches in the literature can be divided into two kinds: single class resolution and multiple class resolution.

Earlier work in the literature explored various methods to resolve entities in a single class that has a number of attributes. Many similarity metrics have been proposed to match strings or textual values, for example, *Levenstein distance*, *Monge-Elkan distance*[82], *Jaro metric*[57], and token-based distance like *Jaccard similarity* and *cosine similarity*, etc[28]. [28] provides an experimental comparison for these string metrics. Meanwhile, a lot of work has been focused on matching records where

each record has a set of attributes and each attribute can be matched by those string metrics or some domain-specific similarity measures. For example, Hernandez and Stolfo [49] proposed a efficient merge/purge process to compute the transitive closure over multiple independent passes, each of which uses a different key to sort the data such that similar records are in a small neighborhood. Their system also includes a declarative rule language to define equivalence of two values.

Recent research efforts have made intensive use of dependency relationships between different class resolution decisions. Pasula et al. [87] have defined a generative model in the context of citation matching. Their model is based on a relational probability model and adopts an approximation method based on Markov chain Monte Carlo. In [86], Parag and Domingos proposed a multi-relational approach to allow information to propagate from one candidate match to another via their common attributes. Their systems makes a collective decision for all the candidate pairs instead of making a decision for each pair seperately.

Bhattacharya and Getoor [18, 16, 17] have proposed various approaches for entity resolution. In [18], an interative algorithm for deduplication was proposed to identify authors. Their main idea is to make use of coauthor relationships and iteratively identify additional potential co-references as more evidence is collected. In [16], they regarded entity resolution as a clustering problem and defined similarity measures to group the references. In [17], they proposed a generative probabilistic model to make collective decisions for resolving entities. In contrast to previous approaches, they introduced two hidden variables for each reference instead of one decision variable for each duplicate pair. In addition, they presented an unsupervised

method for determing the number of entities.

Dong et al. [34] describe an algorithm to reconcile references (entities) which belong to multiple related classes and each reference may have very few attribute values. The major application of their algorithm is in Personal Information Management. In their algorithm, they build dependency graph to represent the dependency between a pair of similarities, which may be the similarity of a pair of references or the similarity of a pair of attribute values. By enriching the references while propagating similarity decisions from node to node in the dependency graph, their algorithm makes a continuous loop between computing similarities and matching decisions until a fixed point is reached.

None of the above efforts has considered to use ontologies for matching objects. Ontologies constitute a useful tool to convey the semantics of terms in databases. For example, a "blue" shirt is considered to be similar to a "navy" shirt, and a gene product is close to another gene product with similar cellular locations, molecular functions and biological processes. However, the colors and the properties of gene products are ontological terms which are hard to match using string metrics or the above approaches. We present our work on ontology-based object matching in Chapter 7. It is important to note that our ontology-based similarity measures can be used as domain-specific metrics in the above approaches, such as the framework in [34].

3.4.2 Similarity Measures on Ontologies

Recently, people have proposed a range of ontology-related approaches to compute the semantic similarity scores. The literature describes three distinct groups of similarity measures that can be applied to taxonomies (ontologies). The first group of measures, which we will refer to as *term-similarity* measures, can be used to compute the similarity of two individual terms. The other two groups of measures can be used when an object is labeled with multiple terms.

Wu and Palmer[114] defined a term-similarity measure based on the depth (in the taxonomy) of the least common ancestor of two terms relative to the depths of the terms individually. The "closer" the common ancestor is to the terms themselves, the greater the similarity. Wu and Palmer did not describe how to use their approach when the taxonomy is a DAG. Another problem with this approach is that some portions of the taxonomy may have been extensively developed and contain many terms, whereas other areas are sparse. Such variations in the "density" of terms make this and other measures (such as [91, 92]) that rely on edge counts a poor estimate of similarity.

To the best of our knowledge, the idea of using information content to measure similarity is due to Resnik[94]. Using the WordNet taxonomy and frequency estimates derived from a large body of English text, Resnik calculated the semantic similarity of word pairs by selecting the common ancestor with the greatest information content. For words with multiple senses, Resnik used the sense that produced the maximum similarity. Using judgments made by human subjects as

the standard, Resnik found that his measure worked better than earlier ones based on edge-counting. Although Resnik's measure can be used when the taxonomy is structured as a DAG, it cannot be used directly when objects are labeled with multiple terms, and it has a number of other disadvantages. Its range is not normalized to $[0, 1]$, but more importantly, by selecting the ancestor with the greatest information content it understates the similarity of objects by focusing on the single most significant aspect of their commonality, at the expense of all others.

Lin[66] provided an axiomatic definition of similarity, and showed how Resnik's approach could be adapted to fit this framework. Whereas Resnik's measure was based solely on the commonality between word meanings, Lin's also takes into account the differences in meaning to determine a normalized similarity score. Lin compared his measure to Resnik's and to Wu and Palmer's, and found that it produced scores that were better correlated with human judgments than those produced by the other two measures. However, Lin does not describe how to use his measure when the taxonomy is a DAG, or when multiple terms are used to describe an object.

The similarity measures in the second group described in the literature measure the similarity of objects based on the number or frequency of terms that are common to the descriptions of both objects. Measures in this group include Jaccard, Dice and Set Cosine, which are used frequently in information retrieval systems and differ in how the count of common terms is normalized, as well as the FMS measure of Keller at al[60]. These measures do not take the structure of the taxonomy into account. Any candidate object that does not share terms with the target will receive a score

of zero, even though it may be quite similar.

The third set of proposed similarity measures relies on an underlying term-similarity measure to determine the similarity between individual pairs of terms, and then combines these to yield an overall similarity score. Halkidi et al.[45] defined a similarity measure of this type for use in clustering web documents. Using the Wu and Palmer term similarity measure, they consider each term in the target and candidate sets individually, and find the most similar term from the other set. Then, over each term set (the target set and the candidate set), they average the similarity of these best matches. Finally, they combine the average similarity from the two sets with equal weight. Since they use Wu and Palmer as the underlying term similarity measure, it is not clear how to apply this measure when the taxonomy is a DAG.

Wang et al.[109] developed a similarity measure that avoids this problem by using a generalized form of Lin's measure to determine the similarity of each term-pair. They generalize Lin's formula for use in a DAG taxonomy by selecting the least common ancestor with the maximum information content. Note that they use a different function than Halkidi et al. for combining term-pair scores. Instead of averaging the scores of the best match from the other set for each term, they average the term similarity scores across all term pairs.

Keller et al.[60] present several ways of quantifying similarity using fuzzy measures based on the depth or information content of terms. However, their measures either require subjectively-specified weights, or the solution of a high-order polynomial equation for each pair of target and candidate objects. We believe this to be prohibitively expensive for large problems.

In Chapter 7, we present two taxonomy-based similarity measures which are applicable to practical classification systems, where taxonomies (ontologies) can be DAG-structured, objects can be labeled with multiple terms, and ambiguity can be introduced by an evolving ontology or classifiers with imperfect knowledge.

Chapter 4

Ontology-extended Relational Algebra

Heterogeneous relational databases being integrated vary widely in how the same data is represented. The discrepancy exists not only in the syntax, such as data structures and storage systems, but also in the semantics, such as the different expressions referring to the same concept. It is widely acknowledged that integration of data at the semantic level remains an open problem. As we mentioned in Chapter 1, semantic discrepancies degrade recall and affect the quality of query answers. In this chapter, we present our approach [19] on semantically integrating heterogeneous relational databases. An important notion in our approach is ontology extended relation.

Ontologies provide semantic information about some body of data (e.g. numbers, strings, etc.). An *ontology extended relation* is a relation that has an associated ontology. Intuitively, the ontology associated with a relation provides some semantical information about the relation and about the syntactic objects (e.g. attribute names, attribute values) associated with the relation. For example, given a relation about stock information, an ontology associated with this relation may contain an explanation of the terms in the relation and/or an explanation of some of the relationships between such terms. For example, it may tell us that database companies are software companies, and that a carburettor is part of a car and that a car is

Name	*Cost*	*Shipping*
Tire	54.19	20.05
Gasket	3.05	1.55
Valve	3.35	1.55
Brake pads	78.50	8.50
Evaporator	305.00	11.50
...	...	...

Table 4.1: *Parts1* relation

equivalent to an automobile. We use a running industrial parts example and an insurance example to illustrate the basic concepts of our approach. In addition, when performing queries over multiple ontology extended relations, we may need to merge multiple ontologies. We show how multiple ontologies may be merged.

4.1 Motivating Example

In this section, we present a simple parts database example and an insurance claims database example. Both these examples are toy examples used primarily to illustrate the basic definitions and results of this paper.

4.1.1 Simple Industrial Parts Example

Consider a very simple example consisting of industrial parts data. There are numerous companies in the US and elsewhere that maintain stocks of various

Item	*Price*	*ShipCost*
Wheel	50.05	18.00
Air Gasket	3.00	1.70
Valve	3.35	1.55
Hubcap	11.50	6.00
Spark Plug	20.00	8.50
...	...	...

Table 4.2: *Parts2* relation

industrial parts (e.g. Grainger in the USA). Two different companies of this kind may maintain part information such as those shown in the *Parts1* (Table 4.1) and *Parts2* (Table 4.2) relations below.

When we examine these two databases, we notice several things. First, the fields *Cost* and *Price* probably refer to the same things. Second, the fields *Shipping* and *ShipCost* probably refer to the same concept. However, by looking at the fields, we really are at a loss to determine whether these fields use the same units or not. For example, *Cost* in the *Parts1* relation may be in US dollars, while *Prices* in the *Parts2* table may be in Euros. In addition, wheel may be a part of tire, and likewise, a hubcap is also a part of a tire. Evaporator and AC Evaporator probably mean the same thing. And an air gasket probably is a gasket.

A user asking queries spanning these two databases would probably like answers that resolve the above types of semantic ambiguities when answering his queries. For example:

- If a user wants to find all tuples dealing with Gaskets for either the *parts1* or *parts2* database, then the system should be smart enough to know that the attribute names *Name* and *Item* are the same, and that an "Air Gasket" is a kind of gasket.

- Likewise, if the user wants to find prices of all tire parts, the system should be smart enough to know that a wheel is part of a tire and hence constitutes a tire part. Similarly for hubcaps. It should therefore return the tuples associated with wheels, hubcaps and tires.

- A more mundane retrieval may arise because the user may want to find valves that cost less than $3.50. In this case, if the *parts1* relation uses Euro to express units, then the system should automatically convert the value in Euros to USD.

In this paper, we develop a model of ontologies and a concept of ontology extended relations that address these issues.

4.1.2 Insurance Example

Consider the case of an insurance company that maintains records about insurance claims. Such a company may keep track of who made a claim, what the claim was about, etc. This information may be entered by diverse claim adjusters who work for the company and may have been entered in different ways over the years. When the insurance company buys another company, they may need to execute queries spanning both databases. We show below two such relations - *Claims1*

ClaimId	*Type*	*Cost*
1	burglary	2000
2	theft	150
3	mugging	860
4	arson	1800

Table 4.3: *Claims1* relation

ClaimNumber	*Type*	*Value*
1	robbery	400
2	fire	550
3	auto accident	500
4	burglary	250

Table 4.4: *Claims2* relation

(Table 4.3) and *Claims2* (Table 4.4) - that may contain the data from two such databases.

Again, in the above example, we see that burglaries and muggings are types of theft. Likewise, arson is a kind of fire. Burglaries are a kind of robbery (which is synonymous with theft). As a consequence, queries about thefts should include all tuples involving such terms. For example, consider the two situations listed below.

- If a user wanted to find all thefts that involved a cost of over $ 1000 dollars, the system should automatically recognize that burglaries, muggings and robberies count as thefts. If costs are represented in different currencies, conversions

should be automatically performed.

- Likewise, if a user wanted to automatically find all fires, it should recognize that arsons are fires and hence should return the appropriate tuples.

4.2 Ontologies and Ontology Integration

In this section, we define ontologies, and describe how to integrate a set of ontologies. We only consider ontologies involving simple concepts - ontologies involving boolean concepts of ontologies are not considered.

4.2.1 Ontologies

In order to define an ontology, we first need the notion of a hierarchy. A couple of definitions are needed in order to get a formal definition of a hierarchy. If S is a nonempty set and $\leq \subseteq S \times S$, then $(S, \leq)$ is an *ordering*. As usual, $(S, \leq)$ is a *partial ordering* if $\leq$ is a reflexive, transitive, and anti-symmetric binary relation on S.

Definition 4 (better than) *Suppose* $(S, \preceq_1)$, $(S, \preceq_2)$ *are two orderings. We say* $(S, \preceq_1)$ is better than $(S, \preceq_2)$ *iff* $(\forall x, y \in S) x \preceq_1 y \rightarrow x \preceq_2 y$.

We will say that $(S, \preceq_1)$ *is strictly better than* $(S, \preceq_2)$ iff $(S, \preceq_1)$ is better than $(S, \preceq_2)$ and $(S, \preceq_2)$ is not better than $(S, \preceq_1)$.

Definition 5 (hierarchy) *Suppose* S *is a partially ordered set under ordering* $\leq$. *A* hierarchy for S *is an ordering* $(S, \preceq)$ *such that*

1. $(S, \preceq)$ *is better than* $(S, \leq)$ *and*

2. $(S, \leq)$ *is the reflexive, transitive closure of* $(S, \preceq)$.

3. *there is no other ordering* $(S, \sqsubseteq)$ *satisfying the preceding two conditions such that* $(S, \sqsubseteq)$ *is strictly better than* $(S, \preceq)$.

A simple example is shown below.

Example 2 *Consider the set* S=$\{wheel, car, hubcap\}$. *A partial ordering reflecting the "part of" relation may say that a wheel is part of a car, a hubcap is part of a car, and a hubcap is part of a wheel*[1]. *In addition, everything is a part of itself. In this case, the natural definition of the part of relation, denoted* $\leq$ *is given by the set* $\{(wheel, wheel),\ (car, car),\ (hubcap, hubcap),\ (wheel, car),\ (hubcap, car),\ (hubcap, wheel)\}$.

There is only one *hierarchy associated with this partial ordering, viz. the set* $\{$ $(wheel, car),\ ((hubcap, wheel)\}$.

The astute reader will note that a hierarchy as defined above is nothing but the Hasse diagram associated with a partial ordering. Taking advantage of this fact, we will often talk about hierarchies as though they are graphs (hence, we will talk about paths in hierarchies, etc.).

Definition 6 *Suppose* Σ *is some finite set of strings and* S *is some set. An* ontology w.r.t. Σ *is a partial mapping* Θ *from* Σ *to hierarchies for* S.

Throughout the rest of this chapter, we will assume that Σ *and* S *are arbitrarily fixed. We will further assume that* Σ *contains a distinguished string called* isa *(representing the usual* isa *relationship) and that* $\Theta(\mathsf{isa})$ *is always defined.*

[1]The "part of" relation in this example has nothing to do with philosophical argument.

Example 3 *Suppose* $\Sigma = \{\mathsf{isa}\}$. *Figures 4.1(a) and 4.1(b) show ontologies associated with the relations Claims1 and Claims2 respectively.*

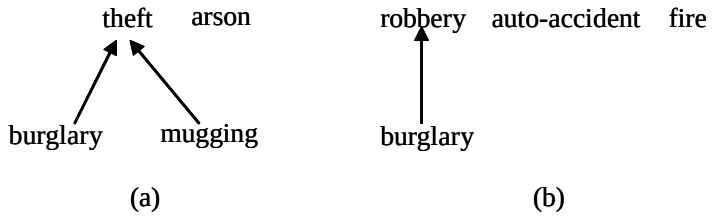


Figure 4.1: (a) Ontology associated with *Claims1* relation, (b) Ontology associated with *Claims2* relation

4.2.2 Ontology Integration

Recall that we intend to associate ontologies with relations (though we have not done so yet) and then to extend the relational algebra to handle such ontology-extended relations. When performing binary operations between two such ontology-extended relations, we need to know the relationship between the terms in each ontology. We start by describing the relationship between terms in two hierarchies.

Definition 7 (interoperation constraints) *Suppose* $(H_i, \leq_i), 1 \leq i \leq n$ *are* n *different hierarchies and suppose* $i \neq j$. *Then* $(x : i = y : j), (x : i \leq y : j), (x : i \not\leq y : j), (x : i \neq y : j)$ *are called* interoperation constraints.

NOTE: As the constraint $x : i = y : j$ can be written as two constraints $x : i \leq y : j$ and $y : j \leq x : i$, we will henceforth assume (without loss of generality) that equality constraints are not present.

For example, an interoperation constraint involving an English and a French veterinary practice may say that $dog : 1 = chien : 2$ explaining that Dog (in English) in the first hierarchy and Chien (in French) in the second hierarchy mean the same thing. In the case of the two claims relations for the insurance application, our interoperation constraints may say that: $theft : 1 = robbery : 2$ and that $arson : 1$ isa $fire : 2$.

Definition 8 (integration) *Suppose* $(H_i, \leq_i), 1 \leq i \leq n$ *are* n *different hierarchies and suppose* **IC** *is a finite set of interoperation constraints. A hierarchy* $(H, \leq)$ *is said to be an* integration *of* $(H_i, \leq_i), 1 \leq i \leq n$ *iff there are* n *injective mappings* $\psi_1, \ldots, \psi_n$ *from* $H_1, \ldots, H_n$ *respectively to* H *such that:*

1. $(\forall i \in \{1, \ldots, n\}) x \preceq_i y \rightarrow \psi_i(x) \leq \psi_i(y)$.
2. $(\forall x \in H_i)(\forall y \in H_j)(x : i \underline{\ op\ } y : j) \in \mathbf{IC} \rightarrow \psi_i(x) \underline{\ op\ } \psi_j(y)$.

In this case, $(H, \leq), \psi_1, \psi_2, \ldots, \psi_n$ *is a* witness to the integrability *of* $(H_i, \leq_i), 1 \leq i \leq n$ *in the presence of interoperation constraints* **IC**.

Intuitively, an integration of two hierarchies is a new hierarchy. Each member of the hierarchies being merged must be associated with a member of the integrated hierarchy. Axiom (1) above says that the integrated hierarchy must preserve the ordering associated with each of the input hierarchies. Axiom (2) above says that they must preserve the interoperation constraints.

It is important to note that there could be many different ways of integrating hierarchies. Some will be better than others.

Example 4 *Consider integrating the two hierarchies of Figures 4.1(a) and 4.1(b) w.r.t. the interoperation constraints* $theft : 1 = robbery : 2$ *and that* $arson : 1$ isa $fire : 2$. *Figure 4.2(a) shows the integrated hierarchy. The mappings that witness this integration are:*

ψ_1	ψ_2
$burglary \mapsto burglary$	$robbery \mapsto theft$
$theft \mapsto theft$	$fire \mapsto fire$
$mugging \mapsto mugging$	$autoaccident \mapsto autoaccident$
$arson \mapsto arson$	$burglary \mapsto burglary$

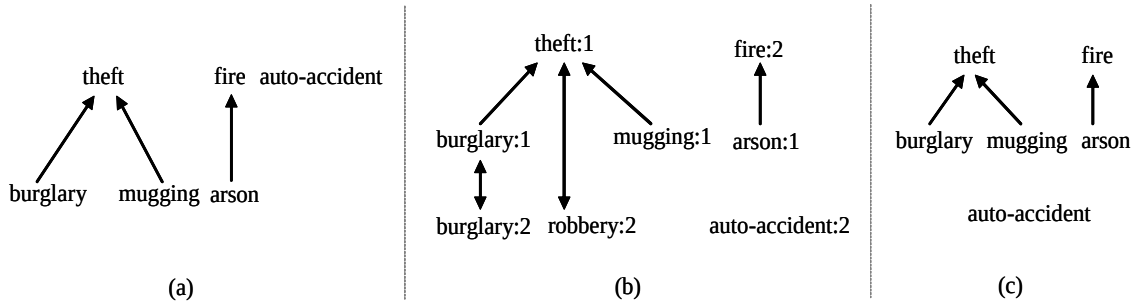


Figure 4.2: (a)Integrated Ontology integrating the *Claims1, Claims2* relations, (b)Hierarchy Graph associated with the *Claims1, Claims2* relations, (c)Canonical Hierarchy associated with the *Claims1, Claims2* relations

Definition 9 (graph associated with a set of hierarchies) *Suppose* $(H_i, \leq_i), 1 \leq i \leq n$ *are* n *different hierarchies and suppose* **IC** *is a finite set of interoperation constraints. The* hierarchy graph $\mathcal{G}$ *associated with* $(H_i, \leq_i), 1 \leq i \leq n$ *is defined as follows:*

- *The set of vertices is the set* $\{x : i \mid x \in H_i\}$*;*

- *The set of edges is the union of:*
  - $\{(x : i, y : i) \mid x, y \in H_i \wedge x \leq_i y\}$
  - $\{(x : i, y : j) \mid x : i \leq y : j \in \mathbf{IC}\}$.

Example 5 *Returning to our running claims example, Figure 4.2(b) shows the hierarchy graph associated with the two claims ontologies.*

Definition 10 (canonical hierarchy) *Suppose* $(H_i, \leq_i), 1 \leq i \leq n$ *are* n *different hierarchies and suppose* **IC** *is a finite set of interoperation constraints. The* canonical hierarchy $(H^\star, \leq^\star)$ *of* $(H_i, \leq_i), 1 \leq i \leq n$ *is defined as follows.*

- $H^\star$ *is the set of all strongly connected components*[2] *of the graph associated with* $(H_i, \leq_i), 1 \leq i \leq n$.
- *If* $x^\star, y^\star \in H^\star$, *then* $x^\star \leq^\star y^\star$ *iff either* $x^\star = y^\star$ *or there exists a directed path from* $x : i$ *to* $y : j$ *(for some* $x : i \in x^\star$ *and* $y : j \in y^\star$*) in the hierarchy graph associated with* $(H_i, \leq_i), 1 \leq i \leq n$.

The following example illustrates the concept of a canonical hierarchy.

Example 6 *Figure 4.2(c) shows the canonical hierarchy associated with our running claims ontologies.*

Definition 11 (canonical witness) *Suppose* $(H_i, \leq_i), 1 \leq i \leq n$ *are* n *different hierarchies and suppose* **IC** *is a finite set of interoperation constraints and* $(H^\star, \leq^\star)$

[2]Recall that the strongly connected components of a graph are the maximal sets of vertices C such that there is a directed path between any two vertices of C.

is the canonical hierarchy. The canonical witness *for* $(H_i, \leq_i), 1 \leq i \leq n$ *is* $(H^\star, \leq^\star), \psi_1^\star, \ldots, \psi_n^\star$ *where* $\psi_i^\star(x) = [x : i]$*, i.e.* $\psi_i^\star$ *maps* x *into the strongly connected component of the graph containing* x*.*

The canonical witness associated with our example is the obvious one - each element in either of the two hierarchies is mapped onto the node in the canonical hierarchy that contains that element.

Suppose $(H_i, \leq_i), 1 \leq i \leq n$ are n different hierarchies and suppose **IC** is a finite set of interoperation constraints.

The following lemma is an important result stating that the dependencies that hold in the canonical hierarchy $(H^\star, \leq^\star)$ of $(H_i, \leq_i), 1 \leq i \leq n$ must also hold in *any arbitrary witness* to the integration of the hierarchies $(H_i, \leq_i), 1 \leq i \leq n$.

Lemma 1 *Let* $(H_i, \leq_i), 1 \leq i \leq n$ *be a family of hierarchies and suppose* $(H^\star, \leq^\star)$ *is its canonical hierarchy. Suppose* $(H, \leq), \psi_1, \ldots, \psi_n$ *is any arbitrary witness to the integration of* $(H_i, \leq_i), 1 \leq i \leq n$*. Then:*

$$[x : i] \leq^\star [y : j] \rightarrow \psi_i(x) \leq \psi_j(y).$$

In the above, $[x : i]$ *(resp.* $[y : j]$*) denotes the strongly connected component containing* $x : i$ *(resp.* $y : j$*).*

The reader should note that the above theorem does *not* say that a witness to the integration of the hierarchies $(H_i, \leq_i), 1 \leq i \leq n$ exists ! It merely says that if at least one such witness exists, then that witness must include all the dependencies in the canonical hierarchy. The following theorem now says that a set of hierarchies can

be integrated *if and only if* the canonical hierarchy and the canonical translation constitutes a witness to the integrability of $(H_i, \leq_i), 1 \leq i \leq n$.

Theorem 1 *A set $(H_i, \leq_i), 1 \leq i \leq n$ of hierarchies is integrable if and only if the canonical witness of $(H_i, \leq_i)$, $1 \leq i \leq n$ is a witness to the integrability of $(H_i, \leq_i)$, $1 \leq i \leq n$.*

The above theorem is very important. It says that to check whether a set of hierarchies is integrable, all we need to do is to compute the canonical hierarchy associated with the hierarchies involved, and then explicitly check if the canonical hierarchy is a witness to their integrability. This can be easily done in polynomial time.

We are finally ready to define what it means to integrate two different ontologies.

Definition 12 (integrability of a set of ontologies) *Suppose Σ is some finite set of strings, S is some set, and $\Theta_1, \ldots, \Theta_n$ are ontologies w.r.t. Σ, S. Suppose* **IC** *is a finite set of interoperation constraints. The ontologies $\Theta_1, \ldots, \Theta_n$ are* integrable *iff for every $x \in \Sigma$, the hierarchies $\Theta_1(x), \ldots, \Theta_n(x)$ are integrable.*

The preceding results suggest a simple algorithm to integrate a collection $\Theta_1, \ldots, \Theta_n$ of ontologies w.r.t. Σ, S. First, for each $x \in \Sigma$, compute the canonical hierarchy $\mathbf{H}_x$ of $\Theta_1(x), \ldots, \Theta_n(x)$. Check if this canonical hierarchy satisfies the interoperation constraints. If this check evaluates to true for all $x \in \Sigma$, then we know that the ontologies $\Theta_1, \ldots, \Theta_n$ are integrable and in fact, the integration of these ontologies

is the new integrated ontology Θ which maps x to $\mathbf{H}_x$. We call Θ the *canonical fusion* (or just fusion) of the ontologies $\Theta_1, \ldots, \Theta_n$.

Once again, we emphasize that in this paper, we are only considering simple ontologies composed of atomic concepts. Some authors [24] have proposed that expressive logics much richer than those above be used for expressing ontologies, but have not provided extensions of the relational algebra incorporating such ontologies. Extension to richer ontology structures is left for future work.

4.3 Ontology Extended Relational Algebra

In this section, we extend the relational model of data and the relational algebra to utilize the information contained in ontologies associated with relations. Without this capability (as in the standard relational algebra) the answer returned by queries may be semantically incorrect and/or incomplete.

Due to space restrictions, in the rest of this paper, we will assume that the set Σ *equals* $\{\mathsf{isa}\}$*, i.e. only one hierarchy is associated with each ontology. The results of the paper can be easily extended to incorporate other kinds of hierarchies such as* $\mathsf{part\_of}$*.*

Types, Domain Values, and Hierarchies

We assume there exists a set $\mathcal{T}$ of strings called *types*. Each type $\tau \in \mathcal{T}$ has an associated set $\mathsf{dom}(\tau)$ called its *domain*. The members of $\mathsf{dom}(\tau)$ are called *values* of type τ. When $T \subseteq \mathcal{T}$, we will often use $\mathsf{dom}(T)$ to denote $\bigcup_{\tau \in T} \mathsf{dom}(\tau)$.

For example, we may have $\mathcal{T} = \{\mathsf{int}, \mathsf{string}\}$. We may have $\mathsf{dom}(\mathsf{int})$ be the set of all integers and $\mathsf{dom}(\mathsf{string})$ be the set of all strings over $\{a, \ldots, z\}$. However we may also have in $\mathcal{T}$ a type like mm whose domain is the set of all non-negative integers (representing millimeters).

A hierarchy of the form $H = (\mathcal{T}_H, \leq_H)$ where $\mathcal{T}_H$ is a set of types is called a *type hierarchy*.

Suppose $\tau \in \mathcal{T}_H$ is a type and $a \in \mathsf{dom}(\tau)$ is an arbitrary value of type τ. Clearly, we could also have a type called a with $\mathsf{dom}(a) = \{a\}$. Thus, each value of a type may also be viewed as a type. Given any type hierarchy $H = (\mathcal{T}_H, \leq_H)$ and a $\tau \in \mathcal{T}_H$, let

$$\mathsf{below}_H(\tau) := \{\tau' \mid \tau' \leq \tau\} \cup \mathsf{dom}(\tau)\,.$$

Intuitively, below_H extends the partial ordering of the hierarchy H to include the values of the types.

Extended Schemas and Relations

In this section, we show how to extend schemas and relations to include type information. Let $\mathcal{A}$ be a set of *attribute names*. If $A_1, \ldots, A_n \in \mathcal{A}$, and $\tau_1, \ldots, \tau_n \in \mathcal{T}$, then $(A_1 : \tau_1, \ldots, A_n : \tau_n)$ is called a *schema*.

Intuitively, an extended schema associates *types* with each attribute name. This is important because a type could be something like USD representing US Dollars or cm representing (intuitively) centimeters.

An *extended relation* is a triple (R,S,H_{isa}), where S is a schema (A_1:τ_1,...,

$A_n{:}\tau_n$), H_{isa} is an isa hierarchy[3] and the following constraints are satisfied:

$$\tau_1, \ldots, \tau_n \in \mathcal{T}_{\mathsf{isa}} \tag{4.1}$$

$$R \subseteq \mathsf{below}_{H_{\mathsf{isa}}}(\tau_1) \times \cdots \times \mathsf{below}_{H_{\mathsf{isa}}}(\tau_n) \tag{4.2}$$

where H_{isa} is the hierarchy associated with isa. If $t \in R$ and $t = \langle\, x_1, \ldots, x_n \,\rangle$, then we use $t(A_i)$ to denote x_i.

Conversion Functions

As the reader has already seen, as different units might be used in relations, there is often a need to *convert* from one unit to another. In this section, we introduce the concept of a conversion function.

For each pair of types τ_i and τ_j, we assume there exists at most one *conversion function* $\tau_i 2 \tau_j : \mathsf{dom}(\tau_i) \rightarrow \mathsf{dom}(\tau_j)$. We assume that conversion functions are total. Moreover, we assume that the set of conversion functions enjoys the following closure conditions and constraints:

- for each $\tau \in \mathcal{T}$, $\tau 2 \tau$ exists and equals the identity function,
- if both $\tau_1 2 \tau_2$ and $\tau_2 2 \tau_3$ exist, then $\tau_1 2 \tau_3$ exists and equals the composition $\tau_2 2 \tau_3 \circ \tau_1 2 \tau_2$. (Note that as $\tau_1 2 \tau_3$ is assumed to be unique, all the possible conversion function compositions that convert τ_1's values into τ_3's values must

[3]When we allow Σ to contain not just isa but other strings as well, the H_{isa} term needs to be replaced with Θ with obvious corresponding changes to the rest of the paper. For the sake of simplicity and due to space restrictions, in this section, we only consider the case where one hierarchy is used.

be equivalent.) Moreover, it is not necessary to specify $\tau_1 2 \tau_3$ explicitly. The system may automatically compose it from the other functions.

- for all hierarchies H, if $\tau_1 \leq_H \tau_2$ then there exists a conversion function $\tau_1 2 \tau_2$.

The third requirement may cause a significant burden during system specification and implementation. To reduce this burden we adopt the following intuitive strategy. Units of measure always require an explicit conversion function, because they always adopt different scales. Therefore, given a type 'Currency' with subtypes 'Euro' and 'Pound', we need a unified representation for Currency together with conversion functions Euro2Currency and Pound2Currency. On the contrary, when types represent classes of objects (as in the parts and insurance examples), it seems reasonable to assume by default that the conversion functions from subtypes to supertypes are the identity function. These two cases (types as units and types as classes) are explicitly distinguished through suitable declarations of the form: `unit` $\tau_1, \ldots, \tau_n$ and `class` $\tau_1, \ldots, \tau_n$.

There are cases in which types represent classes, but heterogeneous notation has been adopted in the subclasses. As a result, some objects may have multiple names or identifiers. In these cases, a conversion function is needed to produce a non-ambiguous naming, otherwise equalities and other comparisons involving these objects are potentially erroneous. Such types should not be declared to be units or classes, as that would force the system to check that all the necessary conversions have been specified.

4.3.1 Selection Conditions

Selection conditions are used both in defining the selection operation as well as the join operation. In this section, we define selection conditions.

Simple (or *atomic*) *conditions* have the form $X\ op\ Y$, where $op \in \{ =, \neq, <, \leq, >, \geq,$ `instance_of` , isa, `subtype_of`, `above`, `below` $\}$, and X, Y are *terms*, that is, attributes, types, or *typed values* $v{:}\tau$, with $v \in \mathsf{dom}(\tau)$. Types will sometimes be omitted in typed values when they can be unambiguously derived from the context. *Selection conditions* can now be defined as follows:

- all simple conditions are selection conditions,
- if c_1, c_2 are selection conditions then $c_1 \wedge c_2$, $c_1 \vee c_2$, $\neg c_1$ are selection conditions,
- nothing else is a selection condition.

Example 7 *For instance, in our claims example,* $Type$ isa $theft$ *and* $Type$ isa $theft \wedge Cost > 140 : pounds$ *are selection conditions. Note that if we apply the second selection condition to the claims1 relation and we assume that the type of Cost is USD (US dollars), we should only get the first tuple as the answer (the second tuple would have been in the answer if the selection condition had been* $Type$ isa $theft \wedge Cost > 140 : usd$.

In the rest of this section, let $\mathcal{R} = (R, (A_1{:}\tau_1, \ldots, A_n{:}\tau_n), H_{\mathsf{isa}})$ be an arbitrary but fixed extended relation. In the context of $\mathcal{R}$, the type of a term X is defined as follows:

$$\mathsf{type}(X) = \begin{cases} \tau_i & \text{if } X = A_i \\ \tau & \text{if } X = \tau \text{ or } X = v{:}\tau. \end{cases}$$

When $\mathsf{type}(X), \mathsf{type}(Y)$ are both in a type hierarchy T and the least upper bound of $\mathsf{type}(X)$ and $\mathsf{type}(Y)$ in T exists, then it is called the *least common supertype* of X and Y w.r.t. a.

We say that a simple condition $X\ op\ Y$ with $op \in \{=, \neq, <, \leq, >, \geq\}$ is *well-typed* if X and Y have a least common supertype τ and the conversion functions $\mathsf{type}(X)2\tau$ and $\mathsf{type}(Y)2\tau$ exist. Simple conditions with other operators are always well-typed. A selection condition is *well-typed* if all its simple subconditions are.

The *value* of a term X w.r.t. a tuple $t \in R$ is

$$X^t = \begin{cases} t.A_i & \text{if } X = A_i \\ \tau & \text{if } X = \tau \\ v & \text{if } X = v{:}\tau. \end{cases}$$

A tuple $t \in R$ *satisfies* a well-typed condition c in the context of $\mathcal{R}$ (denoted $\mathcal{R}, t \models c$) in the following cases:

- $c = X\ op\ Y$, $op \in \{=, \neq, <, \leq, >, \geq\}$, τ is the least common supertype of X and Y, and $(\mathsf{type}(X)2\tau)(X^t)\ op\ (\mathsf{type}(Y)2\tau)(Y^t)$ is true.
- $c = X\ \texttt{instance\_of}\ Y$, $Y^t \in \mathcal{T}$, $\mathsf{type}(X) \leq_H Y^t$, and $X^t \in \mathsf{dom}(Y^t)$.
- $c = X\ \texttt{subtype\_of}\ Y$, $X^t \in \mathcal{T}$, $Y^t \in \mathcal{T}$,$X^t \leq_H Y^t$.
- $c = c_1 \wedge c_2$, $R, t \models c_1$ and $R, t \models c_2$.

- $c = c_1 \vee c_2$, and either $R, t \models c_1$ or $R, t \models c_2$.
- $c = \neg c_1$, and $R, t \not\models c_1$.
- $c = X \,\texttt{below}\, Y$, and $R, t \models X \,\texttt{instance\_of}\, Y \vee X \,\texttt{subtype\_of}\, Y$.
- $c = X \,\texttt{above}\, Y$, and $R, t \models Y \,\texttt{below}\, X$.

4.3.2 Ontological Relational Algebra

Suppose $(R_1, S_1, H_1), \ldots, (R_z, S_z, H_z)$ are ontology extended relations , and suppose $\mathcal{F}$ is a fusion of $H_1, \ldots, H_z$ via witness $tr_{\mathcal{F}}$.

For all algebraic expressions E, $[E]_{\mathcal{F}}$ is inductively defined as follows. Let *op* range over the operators of the relational algebra.

- If E is a relation R_i with schema $S_i = (A_1 : \tau_1, \ldots, A_n : \tau_n)$, then $[E]_{\mathcal{F}} = (tr_{\mathcal{F}}(R), S, \Theta_{\mathcal{F}})$, where $S = (A_1 : tr_{\mathcal{F}}(\tau_1), \ldots, A_n : tr_{\mathcal{F}}(\tau_n))$. In this case, E is always well-typed.
- (Projection) If E is $\Pi_{A_{i_1},\ldots,A_{i_k}}(E')$ $(1 \leq i_j \leq n,\ 1 \leq j \leq k)$ and if $[E']_{\mathcal{F}} = (R', (A_1{:}\tau_1, \ldots, A_n{:}\tau_n),\ \Theta_{\mathcal{F}})$, then $[E]_{\mathcal{F}} = (R, S, \Theta_{\mathcal{F}})$, where R is the standard projection of R' onto $A_{i_1}, \ldots, A_{i_k}$ and $S = (A_{i_1}{:}\tau_{i_1}, \ldots, A_{i_k}{:}\tau_{i_k})$. If E' is well-typed then E is.
- (Cross product) If E is $E_1 \times E_2$ and $[E_i]_{\mathcal{F}} = (R_i, S_i, \Theta_{\mathcal{F}})$, $(i = 1, 2)$, then $[E]_{\mathcal{F}} = (R, S, \Theta_{\mathcal{F}})$, where R is the standard cross product of R_1 and R_2 and S is the concatenation of S_1 and S_2. If E_1, E_2 are well-typed and S_1, S_2 have no common attribute, then E is well-typed.

- (Selection) If E is $\sigma_c(E')$, $[E']_{\mathcal{F}} = (R', S, \Theta_{\mathcal{F}})$, and c is a well-typed selection condition in the context of $(R', S, \Theta_{\mathcal{F}})$, then $[E]_{\mathcal{F}} = (R, S, \Theta_{\mathcal{F}})$, where $R = \{t \in R' \mid (R', S, \Theta_{\mathcal{F}}), t \models c\}$. E is well-typed if E' and c are.

The standard relational condition join operator can be expressed as usual as a composition of σ_c and $\times$. For the remaining set theoretic algebraic operators we need a notion of schema compatibility.

Two schemas $S' = (A_1{:}\tau'_1, \ldots, A_n{:}\tau'_n)$ and $S' = (A_1{:}\tau''_1, \ldots, A_n{:}\tau''_n)$ are *compatible* if for all $i = 1, \ldots, n$, τ'_i and τ''_i have a least common supertype τ_i and the conversion functions $\tau'_i 2 \tau$ and $\tau''_i 2 \tau$ exist. When this is the case, $(A_1{:}\tau_1, \ldots, A_n{:}\tau_n)$ is called the *least common superschema* of S' and S''. Moreover, the conversion function $S'2S$ is defined by $S'2S(R) = \{\langle\, \tau'_1 2 \tau_1(x_1), \ldots, \tau'_n 2 \tau_n(x_n) \,\rangle \mid \langle\, x_1, \ldots, x_n \,\rangle \in R\}$.

- (Union, Intersection and Difference) If $E = E_1$ *op* E_2 where *op* $\in \{\cup, \cap, -\}$, and $[E_i]_{\mathcal{F}} = (R_i, S_i, \Theta_{\mathcal{F}})$, ($i$=1,2), and S_1, S_2 have a least common superschema S, then $[E]_{\mathcal{F}} = (R, S, \Theta_{\mathcal{F}})$, where R is the standard result of $S_1 2 S(R_1)$ *op* $S_2 2 S(R_2)$. E is well-typed if E_1, E_2 are, and S_1, S_2 have a least common superschema.

- (Cast) If $E = (S)E'$ where S is a schema, and $[E']_{\mathcal{F}} = (R, S', \Theta_{\mathcal{F}})$, then $[E]_{\mathcal{F}} = (S'2S(R), S, \Theta_{\mathcal{F}})$. If S and S' have the same number of attributes and the conversion functions needed for $S'2S$ exist, and E' is well-typed, then E is well-typed.

Proposition 1 *For all relational expressions E over (R_1, S_1, H_1),...,(R_z, S_z, H_z) and all fusions $\mathcal{F}$ of $H_1, \ldots, H_z$, $[E]_{\mathcal{F}}$ is an extended relation (i.e., it satisfies conditions (4.1) and (4.2)).*

4.4 Implementation and Experiments

In this section, we briefly describe our **HOME** (Heterogeneous Ontology Management Engine) system.

4.4.1 HOME System

The **HOME** system is implemented in Java and currently consists of 23,562 lines of code. **HOME** accesses an Oracle server across the network. **HOME** consists of the following components:

1. **HOME**'s graphical user interface allows users to express queries as well as view answers. In addition, users can create ontologies and edit them.

2. **HOME** has an *Ontology Maker* that automatically takes a relational database as input and uses WordNet [77] and a commercial thesaurus (ThesDB) to automatically generate ontologies. In addition, the *Ontology Maker* contains a rule maker that allows a user to specify rules that can be used to further derive isa and part_of relations over and above those derived automatically. In addition, the Ontology Maker has a zooming graphical user interface that allows users to zoom in on part of the ontology and either browse it or modify it.

3. **HOME**'s *Query Executor* parses queries expressed via the **HOME** GUI. The Query Executor contains code that transforms a user query into a query that takes ontological information into account. It implements the ontology extended algebra described in this paper. The query executor also contains code

to access any JDBC source (the current backend used is an Oracle server). The query executor contains the necessary socket code needed to access remote servers.

In addition to the above components, we have also developed code to query XML and MPEG-7 data (on top of the XINDICE database system) using ontologies as well as to infer ontologies from XML and MPEG-7 sources.

4.4.2 Experiments

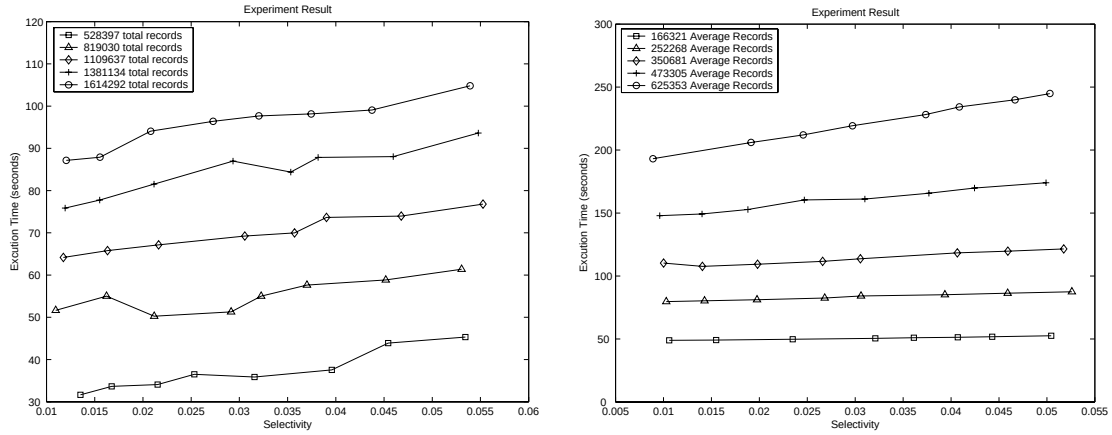
In all our scalability experiments, we used the GNIS data sets from the US Geological Survey which contains data about social and cultural locations in the US on a state by state basis. We populated Oracle relations with this data on a state by state basis. The data was stored in a Windows NT server with 300 MHz Pentium II processor and 64 MB memory. In addition, we used the *Ontology Maker* to generate ontologies based on the GNIS data sets. The experiments were run on a workstation with 2GHz Pentium 4 processor, 1 GB memory and RedHat Linux enterprise platform. Notice that the workstation was not stand-alone during the experiments.

The times taken in all experiments include the following: (i) time to ship the query to a networked Oracle server running multiple concurrent workloads (not just our experiments), (ii) the time to execute the query using JDBC, (iii) the time to ship the answer back to the client from the remote networked Oracle server, and (iv) the time required to convert the JDBC format to the the ***HOME*** *GUI's presentation*

format.

Scalability of selection

Figure 4.3(a) shows the results of evaluating conjunctive selection queries where each atomic selection condition in the conjunct has the form A `subtype_of` v or $A = v$. We vary selectivity (ratio of the number of tuples selected and the number of tuples in the relation) for data sets of successively larger sizes.



(a) (b)

Figure 4.3: (a)Performance of **HOME** for conjunctive selection queries, (b)Join queries with varying join selectivity and varying number of tuples per relation

The reader can easily see that our algorithms handle approximately 1,600,000 tuples in 88-104 seconds (the variation depends on selectivity). The slope of the graph is more or less linear - approximately 16,000-20,000 records can be processed in one second.

Scalability of join

We also ran scalability experiments on join. Figure 4.3(b) shows how performance of join changes when we vary the join selectivity (ratio of number of tuples in the answer and the product of the number of tuples in the relations being joined). We plotted different charts based on the number of tuples (from the GNIS data sets) that were considered. The reader can see that on the average, selectivity did not make a big difference but the number of tuples in the relations being queried did. As the average size of the relations being joined increased, we had a linear increase in the time taken to do the joins. Unlike relational databases, varying selectivity did not make a big difference. Part of the reason for this is that the computation time was dominated by network costs and by ontology management costs.

Figure 4.4 shows the scalability of join as the size of the two ontology extended relations varies up to a little over 1.6 million tuples.

Scalability of ontology integration algorithm

We tested the scalability of our ontology integration algorithm by varying the size of the ontologies being integrated as well as the number of interoperation constraints. Figure 4.5 shows that as the average size of the two ontologies being merged increased from 5000 to 32,000, the time taken to merge that ontologies was under 2 seconds, exhibiting only a linear increase. However, when the average size of the ontologies went over 32,000, there was a sharp increase in the time taken. We believe that was because we almost ran out of memory.

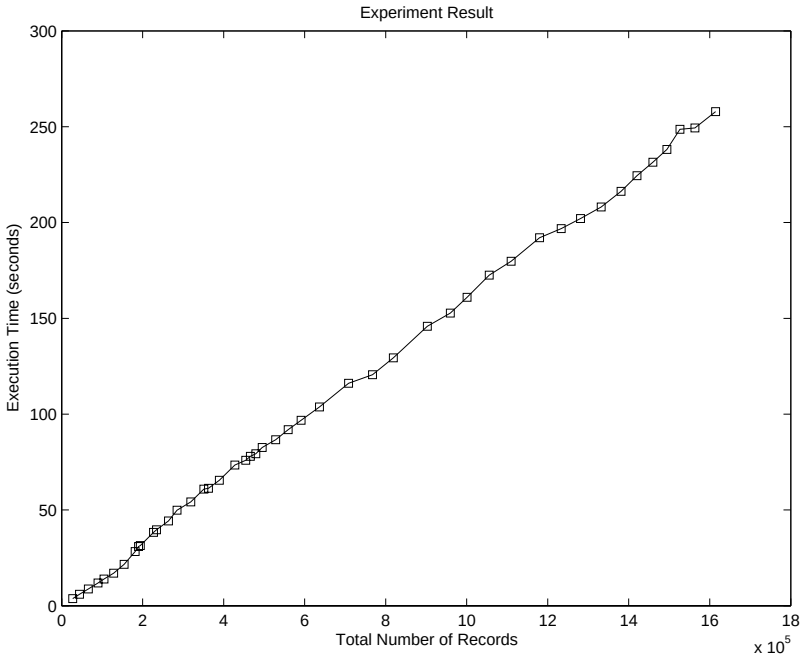


Figure 4.4: Performance of **HOME** for join queries as size of relations being joined is varied

4.5 Summary

In this chapter, we provide a graph-based simple definition of an ontology and propose the notion of an ontology extended relation (OER) to semantically integrate heterogeneous relational databases. For the problem of ontology integration, we formally define the notion of canonical witness to the integrability of a set of ontologies under a set of interoperation constraints. We have established a theory that a set S of ontologies is integrable if and only if the canonical witness is a witness to the integrability of the ontologies in S. Furthermore, we provide an efficient algorithm to compute the canonical witness. We then extend the relational algebra to query OERs. Finally we describe an implementation of the OER model and show (via

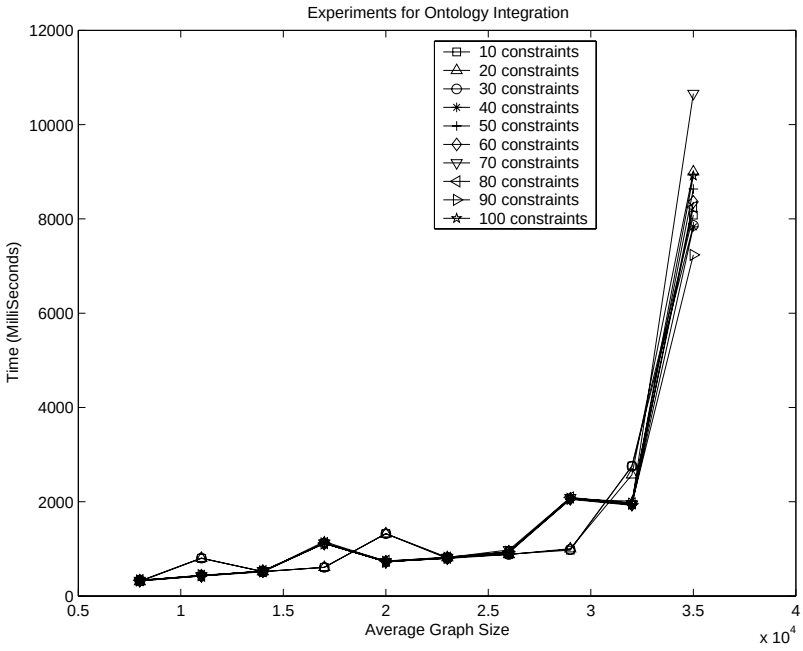


Figure 4.5: Performance of ontology integration algorithms

experiments) that the system scales well.

Chapter 5

TOSS: An Extension of TAX with Ontologies and Similarity Queries

TAX is perhaps the best known extension of the relational algebra to handle queries to XML databases. One problem with TAX (as with many existing relational DBMSs) is that the semantics of terms in a TAX DB are not taken into account when answering queries. In this chapter, we introduce our TOSS algebra [55], which extends the TAX algebra and improves the recall of TAX via the concept of a *similarity enhanced ontology* (SEO). We introduce the following mechanisms to alleviate the recall problem:

1. We define an *ontology extended semistructured instance* (OES instance), which consists of a semistructured instance (e.g. an XML instance) and an associated ontology. Intuitively, an ontology is a set of graphs describing relationships (such as isa, partof, etc.) between terms in a DB (see the definition in Chapter 4). With an ontology, we can express that "US Census Bureau" is part_of "US government" or "Google" isa "web search company" isa "Computer company" isa Company (in case people want to ask the DBLP database for a list of all authors of papers written by someone in a web search company).
2. We consider the delicate issue of how to take similarities into account when merging ontologies together. We show that once ontologies are merged together *without* taking similarity concepts into account, we can apply a *simi-*

larity enhancement operation to the merged ontology to capture such notions of similarity. The resulting similarity enhanced ontology (SEO for short) can be generated using *any* notion whatsoever of similarity between strings with a specified threshold. *We do not propose to reinvent notions of string/lexical similarity in this paper.* Rather, we can adapt many notions of similarity in the text processing literature, such as Levenstein distance, Monge-Elkan distance [82], Jaro metric[57], Jaccard similarity[28] and rule-based similarity where a set of domain-specific rules are used to define what is similar to what. Thus, we can identify similarities between terms (even if the term are not necessarily semantically identical). Note that not all of the similarities can be captured by a lexical semantical system such as WordNet [77]. For example, the strings "J. Ullman", "J.D. Ullman" and "Jeffrey D. Ullman" may all refer to the same DB researcher.

3. We show how to extend the TAX algebra to answer queries w.r.t. such a similarity enhanced ontology. Our algebra is called TOSS.

We develop a prototype implementation of TOSS on top of the Apache Xindice XML database system[115]. We ran experiments to measure the following properties: (i) how does the quality[1] of answers returned by TOSS (with different similarity threshold) compare against the quality of answers returned by TAX? (ii) how is the performance of TOSS queries compared with that of TAX when we vary the size of the semistructured instance being queried and/or the size of the ontology used

[1]The quality of an answer is the square root of the product of the precision and recall of the answer[107].

in executing the query? (iii) how is the performance of TOSS queries affected by the similarity threshold? Our experiments were run on the entire SIGMOD XML proceedings data set[96] and on a part of the DBLP data set[31].

In this chapter, we first provide a quick overview of TAX. We start with the notion of a semistructured instance, followed by the concept of a pattern tree used to express queries in TAX. We will use a small subset of DBLP[31] and SIGMOD[96] bibliographies (in XML format) to illustrate why we may get (intuitively) inadequate answers from TAX and how the ontology and similarity mechanisms we propose can overcome these problems.

5.1 Semistructured Instance and TAX

TAX assumes XML documents to be in ordered tree structures. Figure 5.1 (a) and (b) show the sample DBLP and SIGMOD bibliographic data respectively.

Definition 13 *A* **semistructured instance** I *over a set* $\mathcal{O}$ *of objects, a set* $\mathcal{L}$ *of strings called* labels*, a set* $\mathcal{T}$ *of types, and a domain* $dom(\tau)$ *for each type* $\tau \in \mathcal{T}$*, is a triple* $I = (V, E, t)$ *where:*

1. $G = (V, E)$ *is a set of rooted, directed trees where* $V \subseteq \mathcal{O}$ *and* $E \subseteq V \times V$*.*
2. t *is a mapping such that for each object* $o \in V$*,* $t(o, \mathbf{string})$ *assigns a type in* $\mathcal{T}$ *with:*
   - *the tag of o, i.e. o.tag, if* $\mathbf{string} = tag$*,*
   - *the content of o, i.e. o.content, if* $\mathbf{string} = content$

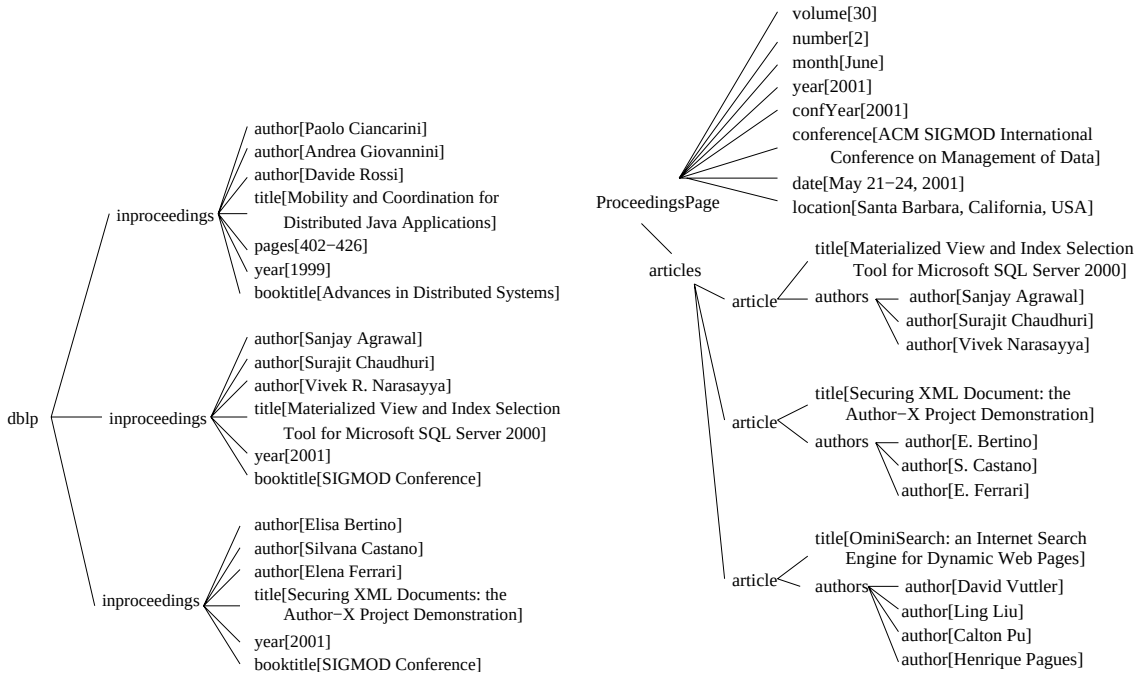


Figure 5.1: (a)A DBLP example, (b)A SIGMOD example

Intuitively, *o.tag* is the label of the edge between o and its parent. We call an object's tag and content its *attributes*. A *semistructured database* (SDB) is a finite set of semistructured instances.

Example 8 *Consider the DBLP example in Figure 5.1(a). V consists of all nodes in the tree and E consists of all edges. Let us use o to denote the first node "author" at the top of the figure. Then, $o.tag = author, t(o, tag) =$* **string**, *$o.content =$ Paolo Ciancarini and $t(o, content) =$* **string**.

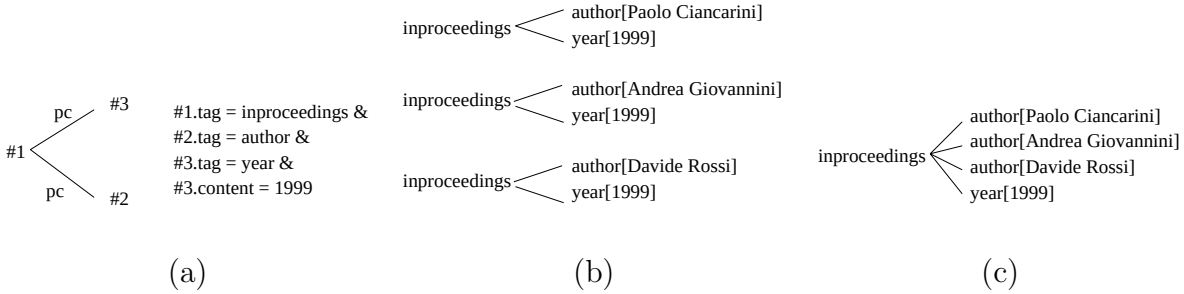
5.1.1 Embeddings and Witness Trees

TAX uses the concept of a *pattern tree* to express queries. We recapitulate the concepts of a pattern tree, an embedding and a witness tree from [56].

Definition 14 *A pattern tree is a pair $P = (T, F)$, where $T = (V, E)$ is an object-labeled and edge-labeled tree such that:*

- *each object in V has a distinct integer as its label;*
- *each edge is either labeled pc (for parent-child) or ad (for ancestor-descendant);*
- *F is a selection condition applicable to objects.*

Example 9 *Figure 5.2(a) shows an example pattern tree, where T is the tree on the left and F is the condition on the right.*



(a) (b) (c)

Figure 5.2: (a)A pattern tree P1, (b)A selection result, (c)A projection result

Suppose SDB is a semistructured database and $P = (T, F)$ a pattern tree. An *embedding* of a pattern tree P into SDB is a total mapping $h : P \rightarrow \bigcup_{(V,E)\in \mathsf{SDB}} V$ from the nodes of T to those in SDB such that:

- h preserves the structure of T, i.e., whenever (u, v) is a *pc* (resp., *ad*) edge in T, $h(v)$ is a child (resp., descendant) of $h(u)$ in SDB.
- The image under the mapping h satisfies the selection condition F. (The reader may consult [56] for a formal definition of satisfaction).

Each embedding h of a pattern tree P into SDB induces a *witness tree* to the embedding, denoted $h^{\mathsf{SDB}}(P)$, which is defined as follows:

- a node n of SDB is in the witness tree if $n = h(u)$ for some node u in the pattern tree P.
- for any pair of nodes n, m in the witness tree, whenever m is the closest ancestor (of the nodes in the witness tree) of n in SDB, the witness tree contains the edge (m, n).
- the witness tree preserves order between nodes in SDB, i.e., for any two nodes in $h^{\mathsf{SDB}}(P)$, whenever m precedes n in the preorder enumeration of SDB, m precedes n in the preorder node enumeration of $h^{\mathsf{SDB}}(P)$ as well.

Example 10 *Figure 5.2(b) shows all the possible witness trees from the pattern tree* $P1$ *in Figure 5.2(a) to the DBLP example in Figure 5.1(a).*

5.1.2 TAX

Selection in TAX. Consider the pattern tree $P1$ in Figure 5.2(a) and suppose SL is any set of nodes. In TAX, the selection query $\sigma_{P,SL}(\mathsf{SDB})$ will return all witness trees w.r.t. pattern query P and SDB. In addition, if a node n in SL appears in a witness tree above, then all descendants of n will also be added to the witness tree.

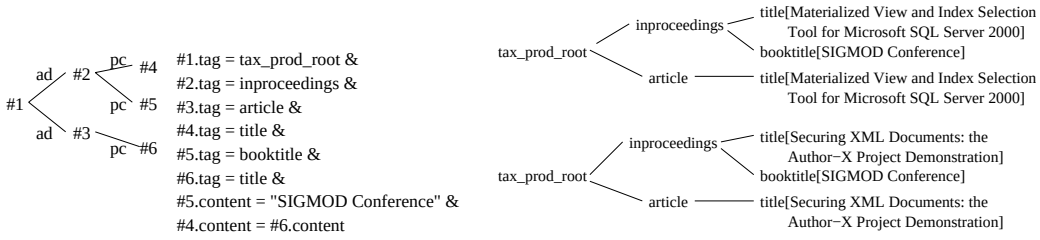
Example 11 *Figure 5.2(b) shows the result of* $\sigma_{P1,\{\}}(dblp)$.

Projection in TAX. Projection takes a semistructured DB SDB, a pattern tree P and a projection list PL (a list of node labels appearing in P) as inputs. The projection operation $\pi_{P,PL}(\mathsf{SDB})$ returns tree(s) consisting of all nodes n selected from SDB such that for every node n in the result, there exists some witness tree $h^{\mathsf{SDB}}(P)$ and $n' \in PL$ where $h^{\mathsf{SDB}}(n') = n$.

Example 12 *To find the authors of papers published in 1999, we can use the pattern tree P*1 *shown in Figure 5.2(a) and apply projection w.r.t. the semistructured DB shown in Figure 5.1(a). This query,* $\pi_{P1,\{\$1,\$2,\$3\}}(dblp)$*, returns the collection of trees in Figure 5.2(c).*

Product in TAX. The product $\mathsf{SDB}_1 \times \mathsf{SDB}_2$ of two semistructured DBs $\mathsf{SDB}_1, \mathsf{SDB}_2$ contains for each pair of trees $T_1 \in \mathsf{SDB}_1, T_2 \in \mathsf{SDB}_2$, a tree, whose root is a new node (called *tax_prod_root*), left child is the root of T_1 and right child is the root of T_2. Condition join consists of a product followed by selection.

Example 13 *The join of the two XML instances in Figure 5.1(a) and Figure 5.1(b), by taking the product and applying a selection with the pattern tree shown in Figure 5.3(a), yields the result shown in Figure 5.3(b).*



(a) (b)

Figure 5.3: (a)A pattern tree P2, (b)A join result

5.1.3 Problems with TAX

When we examine the two databases in Figure 5.1 (a) and (b), we notice several things. First the object tags *booktitle* and *conference* refer to the same thing. Also, the name of the SIGMOD conference is stored differently in the DBLP and SIGMOD

databases ("SIGMOD Conference" in DBLP DB but the full name in SIGMOD DB), but they both refer to the same thing. Likewise, names of the three authors of the paper titled "Securing XML Documents..." are stored differently: their first names are stored in full in DBLP but only initials are stored in SIGMOD. In general, there may be many people with the same last name and first initial of the first name, e.g., Marco Ferrari and Mauro Ferrari. Furthermore, there may be errors in data collection and input, e.g., Gian Luigi Ferrari and GianLuigi Ferrari are probably the same person.[2] Thus, it is reasonable to use some similarity measure d_s to indicate the degree of similarity between two strings. For example, d_s(Gian Luigi Ferrari, GianLuigi Ferrari) = 0.1 (i.e., very similar), d_s(Marco Ferrari, Mauro Ferrari) = 2.2 (quite similar) and d_s(Marco Ferrari, GianLuigi Ferrari) = 6.5 (much less similar).

Second, *cost* and *book-price* probably refer to the same concept. However, by looking at the objects, we really are at a loss to determine whether these fields use the same units or not. For example, *cost* may be in US dollars, while *book-price* may be in Euros. Our TOSS system can handle this as well via a clean extension of the TAX algebra.

A user asking queries spanning these two databases would probably like these semantic ambiguities resolved. For example if we wish to find all conference papers published by Mauro Ferrari by looking at both DBLP and SIGMOD DBs, the system must know that: (1) the tag names *booktitle* and *conference* are identical, (2) the short names and full names of SIGMOD conference refer to the same thing. (3) It should be also smart enough to consider sufficiently similar names to be the same.

[2]All the four names above can be found in DBLP.

5.2 Similarity Enhanced Ontologies

In this section, we define ontologies and then recapitulate how [19] integrates ontologies based on graph merging algorithms in [20]. We then show how to enhance an ontology with similarity measures.

5.2.1 Ontologies

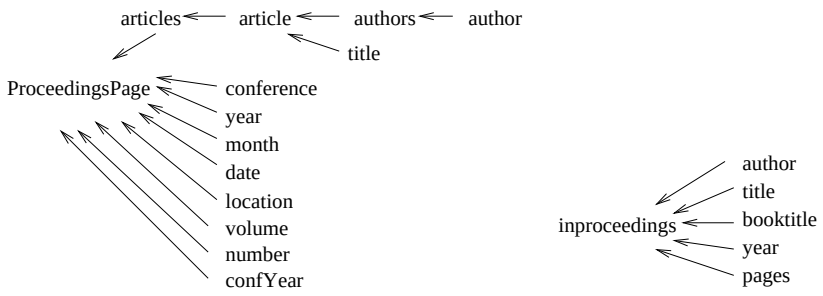
Suppose $(S, \leq)$ is a partially ordered set. A *hierarchy for* $(S, \leq)$ is the Hasse diagram for $(S, \leq)$. Note that the Hasse diagram for a partial order $(S, \leq)$ is a directed acyclic graph whose set of nodes is S. The Hasse diagram of $(S, \leq)$ has a *minimal* set of edges such that there is a path from u to v in the Hasse diagram iff $u \leq v$.

Example 14 *Consider the set* $S = \{article, author, title\}$. *A partial ordering reflecting the* part_of *relation may say that author is part of article and title is part of article. In addition, everything is part of itself. In this case, the natural definition of the* part_of *relation, denoted* $\leq$ *is given by the set* $\{ (author, article), (title, article), (article, article), (author, author), (title, title)\}$. *There is only* one *hierarchy associated with this partial ordering, viz. the set* $\{(author, article), (title, article)\}$.

Definition 15 *Suppose* Σ *is some finite set of strings and* S *is some set. An* ontology w.r.t. Σ *is a partial mapping* Θ *from* Σ *to hierarchies for* S.

Throughout the rest of this paper, we will assume without loss of generality that Σ *and* S *are arbitrary but fixed. We will further assume that* Σ *contains distinguished*

strings called isa *and* part_of *(representing the usual* isa *and* part_of *relationships) and that* $\Theta(\mathsf{isa})$ *and* $\Theta(\mathsf{part\_of})$ *are always defined.*



(a) (b)

Figure 5.4: (a)An example of ontology associated with SIGMOD, (b) An example of ontology associated with DBLP

Example 15 *Suppose* $\Sigma = \{\mathsf{part\_of}\}$. *Figures 5.4 (a) and (b) show the simplified examples of ontologies associated with SIGMOD and DBLP respectively.*

5.2.2 Integrating Ontologies

Suppose our $\mathsf{SDB} = \{I_1, \ldots, I_n\}$. Suppose, for the sake of simplicity, that $\Sigma = \{\mathsf{part\_of}\}$. Suppose $(H_j, \leq_j)$ is the part_of hierarchy associated with I_j. To answer queries spanning these different semistructured instances, we need to take into account the relationships between terms in these hierarchies. These are captured by interoperation constraints (specified by the database administrators) defined in Chapter 4. The formal definitions and algorithm for integrating ontologies are given in Chapter 4. In this section, we present examples of interoperation constraints, hierarchy graph and canonical fusion of ontologies to illustrate the concepts.

Example 16 *We may write an interoperation constraint* booktitle:1 = conference:2 *to indicate that* booktitle *(in DBLP) is the same as* conference *in the SIGMOD collection.*

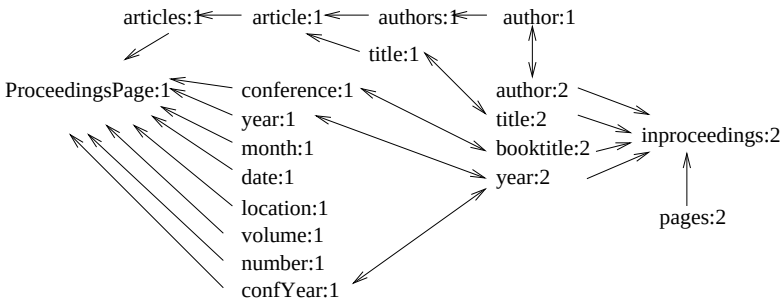


Figure 5.5: Hierarchy Graph associated with SIGMOD and DBLP

Example 17 *Consider integrating the two hierarchies of Figures 5.4 (a) and (b) w.r.t. the following five interoperation constraints conference:1 = booktitle:2, title:1 = title:2, author:1 = author:2, year:1 = year:2, confYear:1 = year:2, which means that conference in the first hierarchy (SIGMOD) is equal to booktitle in the second hierarchy (DBLP), etc. Figure 5.5 shows the hierarchy graph associated with the two ontologies. Note that there may be many different ways of integrating hierarchies. [20, 19] describe how to find an integration, which is called the canonical hierarchy. We call this integration the canonical fusion (or just fusion)* Θ *of ontologies* $\Theta_1, \ldots, \Theta_n$ *via a witness (which maps a node of a hierarchy onto a node in the canonical hierarchy). Figure 5.6 shows the fused ontology.*

We would like to mention that there are various works in the literature on integrating ontologies that have a very rich structure (e.g. definitions where an

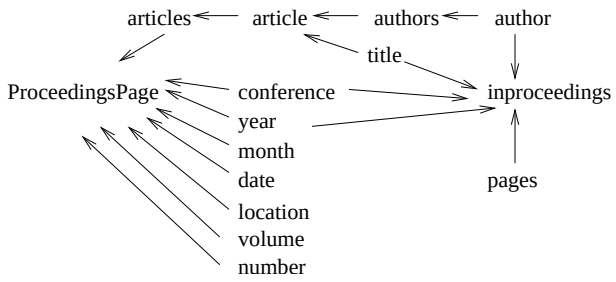


Figure 5.6: Canonical fusion of the ontologies of SIGMOD and DBLP

ontology is any arbitrary first order logic theory, and so on) [24]. However, to date, we have seen no extension that uses these very rich ontology definitions to answer queries over even relational sources. In this paper, we have only considered simple ontologies expressible as graphs.

5.2.3 Similarity Enhanced Ontologies

In the rest of this section, we assume that given a semistructured database SDB, an ontology associated with each $I \in \mathsf{SDB}$ and a set of interoperation constraints IC, we have computed the fusion of all the ontologies w.r.t. IC. We will now show how to enhance such a fused ontology using similarity notions. As is common in many aspects of information retrieval, similarity between terms is measured using a distance function[28].

Definition 16 *A* **string similarity measure** d_s *is any function which takes two strings* X, Y *and returns a non-negative real number such that:*

- $\forall X,\ d_s(X, X) = 0.$
- $\forall X, Y,\ d_s(X, Y) = d_s(Y, X).$

A string similarity measure d_s *is* **strong** *iff for all strings* X, Y, Z, $d_s(X, Y) + d_s(Y, Z) \geq d_s(X, Z)$.

A **similarity measure** d *is any function which takes nodes* A, B *as input and returns a non-negative real number such that* $d(A, B) = min_{X \in S_A, Y \in S_B} d_s(X, Y)$, *where* d_s *is a string similarity measure,* S_A, S_B *are the sets of strings contained in nodes* A, B *respectively, and for any node* A, *if* $X, Y \in S_A$, *then* $d_s(X, Y) = 0$. d *is strong iff* d_s *is strong and satisfies the previous condition.*

Intuitively, two nodes get more and more similar as the distance between them gets smaller.

The following result says that for strong similarity measures, we can avoid trying all combinations of strings between two nodes.

Lemma 2 *Suppose* S_A, S_B *are the sets of strings contained in nodes* A, B *respectively. If a similarity measure* d *is strong, then* $\forall X, P \in S_A, Y, Q \in S_B$, $d_s(X, Y) = d_s(P, Q)$. *Therefore,* $\forall X \in S_A, Y \in S_B$, $d(A, B) \equiv d_s(X, Y)$.

Proof sketch: $\forall Y, Q \in S_B$, $d_s(Y, Q) = d_s(Q, Y) = 0$. Because d is strong, so d_s is also strong. Therefore, $d_s(X, Y) + d_s(Y, Q) \geq d_s(X, Q)$, i.e., $d_s(X, Y) \geq d_s(X, Q)$. Similarly, $d_s(X, Q) + d_s(Q, Y) \geq d_s(X, Y)$, i.e., $d_s(X, Q) \geq d_s(X, Y)$. Thus, $d_s(X, Q) = d_s(X, Y)$. Similarly, we can prove $\forall X, P \in S_A, d_s(X, Q) = d_s(P, Q)$. Thus, $\forall X, P \in S_A, Y, Q \in S_B$, $d_s(X, Y) = d_s(P, Q)$. □

Many different similarity measures can be used. The information retrieval community has proposed various string distances such as Levenstein distance[3], Monge-

[3]The Levenstein distance is strong. It assigns a unit cost to every edit operation.

Elkan distance[82], Jaro metric[57], and token-based distance like Jaccard similarity[4] and cosine similarity, etc[28]. In addition, products such as WordNet[77] define similarity measures between terms in English and various foreign languages (but not between proper nouns). In certain domains, rule based methods can also be used to specify similarity between proper nouns (in our SIGMOD/DBLP application for example, we could write a set of rules describing when two names are considered similar). There are also *statistical* approaches to checking similarity between strings used in methods such as latent semantic indexing. *Our goal in this paper is not to invent a new notion of similarity between terms - this has already been well done by the IR community. Instead, the TOSS framework can plug in any such similarity implementation and use it to answer queries in a manner that increases the quality of an answer.*

Definition 17 (similarity enhancement) *Suppose H is an integrated hierarchy, d is a similarity measure and $\epsilon \geq 0$. The pair (H', μ) is called a* similarity enhancement *of H w.r.t. d, ϵ iff H' is a hierarchy and μ is a function from H to $2^{H'}$ ($2^{H'}$ is the power set of H') such that*

1. *$\forall A, B \in H, A_\mu \in \mu(A), B_\mu \in \mu(B)$, if there is a path from A to B in H, then there is a path of length zero or more from A_μ to B_μ in H'. Let us define $\mu^{-1} : H' \rightarrow 2^H$ such that $\mu^{-1}(A') = \{A \mid A \in H \wedge A' \in \mu(A)\}$. Similarly, $\forall A', B' \in H'$, if there is a path for A' to B' in H', then $\forall A \in \mu^{-1}(A'),\ B \in \mu^{-1}(B)$, there is a path from A to B in H.*

[4]Jaccard similarity between two word sets S, T is $|\frac{S \cap T}{S \cup T}|$.

2. $\forall A, B \in H, A_\mu \in \mu(A), B_\mu \in \mu(B)$, *if* $A_\mu = B_\mu$, *then* $d(A, B) \leq \epsilon$.

3. $\forall A, B \in H$, *if* $d(A, B) \leq \epsilon$, *then* $\exists A_\mu \in \mu(A), B_\mu \in \mu(B)$ *such that* $A_\mu = B_\mu$.

4. $\forall A', B' \in H'$, $\mu^{-1}(A') \cap \mu^{-1}(B') \neq \mu^{-1}(A')$.

The first condition ensures that the similarity enhanced hierarchy preserves the original partial orderings in H and does not include unwarranted orderings in the result. The second condition ensures that all nodes mapped into the same node are sufficiently similar to each other - the threshold ϵ, specified by the database administrator, says two terms are sufficiently similar if the distance between them is ϵ or less. The fourth condition removes redundant nodes which are a subset of some other node in the similarity enhancement. Furthermore, our similarity enhancement says that two strings are similar if and only if the two strings are jointly present in some node in the similarity enhancement. That is why we have the third condition.

For example, let us consider a hierarchy H consisting of nodes A, B, C. If $d(A, B) \leq \epsilon$, $d(A, C) \leq \epsilon$, but $d(B, C) > \epsilon$, then we must have one and only one enhancement: we should have a node containing $\{A, B\}$ AND a node containing $\{A, C\}$. We define the similarity enhancement in this way such that if a query is on a string in A, then we can find the result of all similar strings in the nodes containing $\{A, B\}$ and $\{A, C\}$ easily. If we just have two nodes containing $\{A, B\}$ and $\{C\}$ then we do not know that strings in C are similar to those in A from the resulting ontology. In this case, we will need to either (i) compare all nodes with A to find those similar; or (ii) use an index to find those nodes similar to A quickly. But then, we can just use (i) or (ii) directly and do not need to use the similarity

enhancement.

By definition, an ontology is not allowed to have cycles. This may mean, on occasion, that no similarity enhancement exists *for some similarity measures and some thresholds* ϵ.

Definition 18 (similarity inconsistency) *Suppose H is a hierarchy, d is a similarity measure and $\epsilon \geq 0$. The triple (H, d, ϵ) is* similarity consistent *(resp. similarity inconsistent) iff there exists (resp. does not exist) a similarity enhancement of H w.r.t. d, ϵ.*

The following theorem states that all similarity enhancements of a particular hierarchy (w.r.t a similarity measure d and a threshold ϵ) are equivalent.

Theorem 2 *(Equivalence of Similarity Enhancements) Suppose $H = (S, \preceq)$ is a hierarchy, d is a similarity measure and $\epsilon \geq 0$. If (H'_1, μ_1) and (H'_2, μ_2) are similarity enhancements of H, then there exists an one-to-one mapping $M : S'_1 \rightarrow S'_2$ such that $M \circ \mu_1 = \mu_2$ and $\forall A, B \in S'_1, (A, B) \in \preceq_1$ iff $(M(A), M(B)) \in \preceq_2$, where $H'_1 = (S'_1, \preceq_1)$ and $H'_2 = (S'_2, \preceq_2)$.*

Proof sketch: Conditions (2) – (4) of Definition 17 equivalently define the set of nodes and the mapping in the similarity enhancement as follows. First, we define $S'' = \{V \mid \forall A, B \in V, A \in S \wedge B \in S \wedge d(A, B) \leq \epsilon\}$. Then, define $S'' \leftarrow S'' - \{V \mid \exists V' \in S'' \wedge V \subset V'\}$. Thus, S'' is unique. Next, we can define a mapping $M_1 : S \rightarrow 2^{S''}$ s.t. $M_1(A) = \{V \mid A \in V\}$.

For any similarity enhancement, say, (H'_1, μ_1), we can define a one-to-one mapping $M_2 : S'' \rightarrow S'_1$ s.t. $M_2(B) = C$ iff $\forall A \in B$, $C \in \mu_1(A)$.

Similarly, for (H_2', μ_2), we can define a one-to-one mapping $M_2' : S'' \rightarrow S_2'$ s.t. $M_2'(B) = C$ iff $\forall A \in B$, $C \in \mu_2(A)$.

Now we can define $M : S_1' \rightarrow S_2'$ s.t. $M(C_1) = C_2$ iff $\forall A \in \{A \mid C_1 \in \mu_1(A)\}$, $C_2 \in \mu_2(A)$. Thus, $M \circ \mu_1 = \mu_2$.

Because $M \circ \mu_1 = \mu_2$, $\mu_1^{-1} \circ M^{-1} = \mu_2^{-1}$.

From condition (1) in Definition 17, $\forall A, B \in S_1'$, $(A, B) \in \preceq_1$ iff $\forall a \in \mu_1^{-1}(A)$, $b \in \mu_1^{-1}(B)$, $(a, b) \in \preceq$. Similarly, $\forall A, B \in S_2'$, $(A, B) \in \preceq_2$ iff $\forall a \in \mu_2^{-1}(A)$, $b \in \mu_2^{-1}(B)$, $(a, b) \in \preceq$. Thus, $\forall A, B \in S_1'$, $(M(A), M(B)) \in \preceq_2$ iff $\forall a \in \mu_2^{-1}(M(A))$, $b \in \mu_2^{-1}(M(B))$, $(a, b) \in \preceq$. But $\forall X \in S_1'$, $\mu_2^{-1}(M(X)) = \mu_1^{-1} \circ M^{-1}(M(X)) = \mu_1^{-1}(X)$. Thus, $\forall a \in \mu_2^{-1}(M(A))$, $b \in \mu_2^{-1}(M(B))$, $(a, b) \in \preceq$ is equivalent to $\forall a \in \mu_1^{-1}(A)$, $b \in \mu_1^{-1}(B)$, $(a, b) \in \preceq$. Therefore, $\forall A, B \in S_1'$, $(M(A), M(B)) \in \preceq_2$ iff $(A, B) \in \preceq_1$. □

Table 5.1 presents the SEA algorithm (short for **Similarity Enhancement Algorithm**) to find a similarity enhancement of a hierarchy w.r.t. d, ϵ.

In Line 14, the function check-acyclic(H') checks whether H' is acyclic or not. We can modify a depth-first search to detect cycles in $O(\mid \preceq' \mid)$ time where $\mid \preceq' \mid$ indicates the number of edges in the hierarchy of the similarity enhancement. The time complexity of the **SEA** algorithm is $O(|S| \cdot |S'|(|S| + |S'|) + \mid \preceq \mid\mid S'|^2)$.

Theorem 3 *(Correctness of Algorithm) Given a hierarchy H, a similarity measure d and a real number $\epsilon \geq 0$, the SEA algorithm in Table 5.1 returns a similarity enhancement (H', μ) of H if one exists.*

algorithm SEA(H, d, ϵ)

Input: hierarchy $H = (S, \preceq)$, similarity measure d, real number $\epsilon \geq 0$

Output: $H' = (S', \preceq'), \mu$

1. $S' \leftarrow \emptyset$

2. **for** each $s \in S$,

3. $\quad S' \leftarrow S' \cup \{s\}$

4. $\quad$ **for** each $s' \in S'$,

5. $\quad\quad$ **if** $\forall a \in s'$, $d(s, a) \leq \epsilon$, **then** $s' \leftarrow s' \cup \{s\}$

6. $\quad\quad$ **else if** $\exists a \in s'$, $d(s, a) \leq \epsilon$, **then**

7. $\quad\quad\quad S' \leftarrow S' \cup \{\{s\} \cup \{a \mid a \in s' \wedge d(s, a) \leq \epsilon\}\}$

8. $\quad S' \leftarrow S' - \{s' \mid \exists s'' \in S' \wedge s' \subset s''\}$

9. **for** each $s \in S$,

10. $\quad$ define $\mu(s) = \{s' | s' \in S' \wedge s \in s'\}$

11. $\preceq' \leftarrow \emptyset$

12. **for** each $(a, b) \in \preceq$,

13. $\quad \preceq' \leftarrow \preceq' \cup \{(c, d) \mid c \in \mu(a) \wedge d \in \mu(b) \wedge c \neq d\}$

14. **if** check-acyclic(H') == TRUE, **then**

15. $\quad$ **return** $H' = (S', \preceq'), \mu$

16. **else return** "Similarity inconsistent"

Table 5.1: Similarity Enhancement Algorithm **SEA**

Proof sketch: Lines 5 – 7 produce nodes such that the second condition of a similarity enhanced hierarchy is satisfied. Lines 3, 5 – 7 ensure that the third condition is satisfied because line 3 ensures that, for every $s \in S$, there exists some node in S' that contain s, and lines 5 – 7 ensures that $\forall s_1, s_2 \in S$, if $d(s_1, s_2) \leq \epsilon$, then there must exist some node in S' to contain both of them. Line 8 removes redundant nodes to ensure that the fourth condition is satisfied. Lines 11 – 13 create $\preceq$ to satisfy the first condition. Lines 9 – 10 define μ according to the definition. □

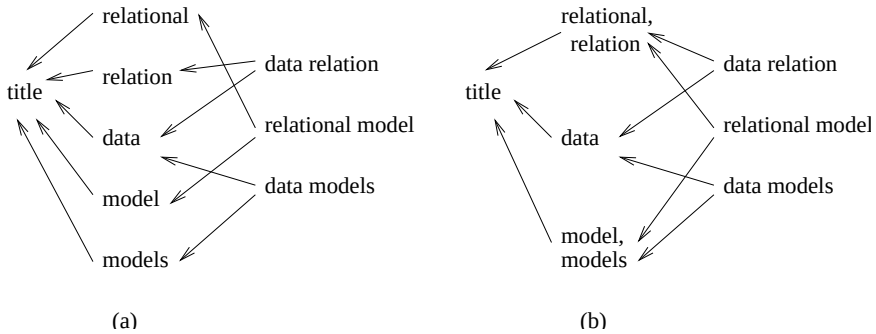


Figure 5.7: (a) An example ontology and (b) its similarity enhancement

Example 18 *Figure 5.7(a) shows a toy ontology consisting of a* isa *hierarchy. Suppose we use Levenstein distance as the similarity measure d and a threshold $\epsilon = 2$. Because $d(relation, relational) = 2$, $d(model, models) = 1$, the algorithm SEA will form two new nodes in the similarity enhancement, containing $\{relation, relational\}$ and $\{model, models\}$ respectively. SEA will also remove the four redundant nodes relation, relational, model and models. Then it defines μ, which maps each node in the original hierarchy to itself in the result, except for the four nodes removed, which are mapped to the two new nodes respectively. $\preceq'$ is defined as shown in Figure 5.7(b).*

5.3 Ontology Extended Semistructured Data Model

In this section, we extend the semistructured data model and the TAX algebra to handle ontology and similarity concepts. *For the sake of simplicity, in the rest of this section, we will assume that the set* Σ *equals* {isa}*, i.e. only one hierarchy is associated with each ontology. The results of our work can be easily extended to incorporate arbitrary hierarchies such as* part_of*.*

SEO Semistructured Instance. In this section, we show how to extend a semistructured instance to include type[5], ontology, and similarity information. Recall that a semistructured instance is defined as $I = (V, E, t)$ where t associates a type in $\mathcal{T}$ with each attribute (*tag* and/or *content*) of each object $o \in V$.

Intuitively, an extended instance associates *types* with each object attribute. This is important because a type could be something like USD representing US Dollars or cm representing (intuitively) centimetres. It is also associated with an ontology (which for now is just a hierarchy H_{isa}, a distance function d and a real number $\epsilon > 0$).

An *ontology extended semistructured instance* is a quadruple $(V, E,\ t,\ H_{\mathsf{isa}})$, where H_{isa} is an isa hierarchy[6] and the following constraints are satisfied:

[5]The formal definitions of types, domain values and hierarchies (as well as conversion functions) are given in Chapter 4.

[6]When we allow Σ to contain not just isa but other strings as well, the H_{isa} term needs to be replaced with Θ with obvious corresponding changes to the rest of the paper. For the sake of simplicity and due to space restrictions, in this section, we only consider the case where one hierarchy is used.

- $\forall$ object $o, t(o, tag) \in \mathcal{T}_{\mathsf{isa}}$ and $o.tag \in \mathsf{below}_{H_{\mathsf{isa}}}(t(o, tag))$
- $\forall$ leaf object $o, t(o, content) \in \mathcal{T}_{\mathsf{isa}}$ and $o.content \in \mathsf{below}_{H_{\mathsf{isa}}}(t(o, content))$

An SEO*(similarity enhanced ontology extended) semistructured instance* is a quadruple $(V, E, t, H_{\mathsf{isa}}{}')$ where $(H_{\mathsf{isa}}{}', \mu)$ is the similarity enhancement of H_{isa} for some μ w.r.t. similarity measure d and threshold ϵ.

5.3.1 TOSS Algebra

Throughout this section, we assume without loss of generality that we are querying some semistructured database SDB and that the ontologies of semistructured instances in SDB have been fused together as described in Section 5.2.2 and that the fused ontology was then enhanced with similarity using the SEA algorithm shown in Table 5.1. All algebraic operators therefore assume the existence of this similarity enhanced (fused) ontology.

SEO Embeddings and Witness Trees

Selection conditions are used in pattern trees which are used in both selection and projection. In this section, we define selection conditions and show how to extend the definitions of embedding and witness trees to support ontologies and similarity.

Simple (or *atomic*) *conditions* have the form $X\ op\ Y$, where $op \in \{ =, \neq, <, \leq, >, \geq, \sim,$ `instance_of` , isa, `subtype_of`, `above`, `below` $\}$, and X, Y are *terms*, that is, attributes, types, or *typed values* $v{:}\tau$, with $v \in \mathsf{dom}(\tau)$. Types will sometimes be

omitted in typed values when they can be unambiguously derived from the context. Here $\sim$ represents the similarTo operation which will return true if both operands are sufficiently similar (i.e., $\leq \epsilon$ for a given similarity measure d).

Selection conditions can now be defined as follows:

- all simple conditions are selection conditions,
- if c_1, c_2 are selection conditions then $c_1 \wedge c_2$, $c_1 \vee c_2$, $\neg c_1$ are selection conditions,
- nothing else is a selection condition.

Example 19 *Suppose we want to find the titles of all papers in DBLP related to Microsoft (independently of the field in which Microsoft appears). We may change the selection condition in the pattern tree in Figure 5.2 (a) to the following and apply it to* dblp*:* $\#1.tag = inproceedings \wedge \#2.tag = title \wedge \#3.tag$ part_of $inproceedings \wedge \#3.content =$ "$\star\, Microsoft\, \star$" *(where* $\star$ *is a wild card).*

In order to ensure an embedding to be correct w.r.t an SDB with an associated similarity enhanced ontology, we need to redefine the satisfaction of selection conditions.

In the rest of this section, let $EI = (V, E, t, H_{\mathsf{isa}})$ be an arbitrary but fixed ontology extended or SEO instance.[7] In the context of EI, the type of a term X

[7]The semantics of both ontology extended and SEO instances are similar except that an ontology extended instance uses an ontology but a SEO instance uses a SEO and it allows similarity operations.

w.r.t. a mapping h[8] from a pattern to a tree collection is defined as follows:

$$\mathsf{type}(X)^h = \begin{cases} t(h(X), tag) & \text{if } X \text{ is the tag of an object} \\ t(h(X), content) & \text{if } X \text{ is the content of an object} \\ \tau & \text{if } X = \tau \text{ or } X = v{:}\tau. \end{cases}$$

When $\mathsf{type}(X), \mathsf{type}(Y)$ are both in a type hierarchy T and the least upper bound of $\mathsf{type}(X)$ and $\mathsf{type}(Y)$ in T exists, then it is called the *least common supertype* of X and Y.

We say that a simple condition $X\ op\ Y$ with $op \in \{=, \neq, <, \leq, >, \geq\}$ is *well-typed* if X and Y have a least common supertype τ and the conversion functions $\mathsf{type}(X)^h 2\tau$ and $\mathsf{type}(Y)^h 2\tau$ exist. Simple conditions with other operators are always well-typed. A selection condition is *well-typed* if all its simple subconditions are.

The *value* of a term X w.r.t. a mapping h from a pattern to a tree collection is

$$X^h = \begin{cases} h(X).tag & \text{if } X \text{ is the tag of an object} \\ h(X).content & \text{if } X \text{ is the content of an object} \\ \tau & \text{if } X = \tau \\ v & \text{if } X = v{:}\tau. \end{cases}$$

The image (witness tree WI) under the mapping h (from a pattern P to a collection C), i.e., $h^C(P)$, *satisfies* a well-typed condition c in the context of the ontology extended instance EI (denoted $EI, WI \models c$) in the following cases:

[8]Strictly speaking, h maps a node in the pattern tree to a node in a tree collection. However, X in $h(X)$ is not a node but a node's attribute. For convenience, here we abuse the notation such that, suppose $X = y.b$ where y is a node and b is an attribute, we use $h(X)$ to denote $h(y)$.

- $c = X\ op\ Y$, $op \in \{=, \neq, <, \leq, >, \geq, \sim\}$, τ is the least common supertype of X and Y, and $(\mathsf{type}(X)^h 2\tau)(X^h)\ \ op\ (\mathsf{type}(Y)^h 2\tau)(Y^h)$ is true. For $A \sim B$, the condition is true iff $\exists$ a node containing both of them in the similarity enhancement.
- $c = X\ \texttt{instance\_of}\ Y$, $Y^h \in \mathcal{T}$, $\mathsf{type}(X)^h \leq_H Y^h$, and $X^h \in \mathsf{dom}(Y^h)$.
- $c = X\ \texttt{subtype\_of}\ Y$, $X^h \in \mathcal{T}$, $Y^h \in \mathcal{T}$,$X^h \leq_H Y^h$.
- $c = c_1 \wedge c_2$, $EI, WI \models c_1$ and $EI, WI \models c_2$.
- $c = c_1 \vee c_2$, either $EI, WI \models c_1$ or $EI, WI \models c_2$.
- $c = \neg c_1$, $EI, WI \not\models c_1$.
- $c = X\ \texttt{below}\ Y$, $EI, WI \models X\ \texttt{instance\_of}\ Y \vee X\ \texttt{subtype\_of}\ Y$.
- $c = X\ \texttt{above}\ Y$, $EI, WI \models Y\ \texttt{below}\ X$.

TOSS Algebra

In this section, we are able to give a succinct definition of the TOSS algebra. Note that this algebra is easy to implement on top of any semistructured DBMS system.

Suppose $EI_1 = (V_1, E_1, t_1, H_1), \ldots, EI_z = (V_z, E_z, t_z, H_z)$ are ontology extended instances. Suppose $\mathcal{F}'$ is a fusion of $H_1, \ldots, H_z$ via witness $tr_{\mathcal{F}'}$, $(\mathcal{F}, \mu)$ is the similarity enhancement of $\mathcal{F}'$, $tr_{\mathcal{F}} = \mu \circ tr_{\mathcal{F}'}$. $\Theta_{\mathcal{F}}$ is the ontology that associates the hierarchy $\mathcal{F}$ with isa.

For all algebraic expressions Exp, $[Exp]_{\mathcal{F}}$ is inductively defined as follows.

- If Exp is an instance EI_i, then $[Exp]_{\mathcal{F}} = (tr_{\mathcal{F}}(V_i), E_i, tr_{\mathcal{F}}(t_i), \Theta_{\mathcal{F}})$, which is

an SEO instance. Here $tr_{\mathcal{F}}(V_i)$ (resp. $tr_{\mathcal{F}}(t_i)$) maps the tags and contents of objects $o \in V_i$ (resp. their types) to terms in $\Theta_{\mathcal{F}}$. In this case, Exp is always well-typed.

- (Selection) If Exp is $\sigma_{P,SL}(Exp')$ and the selection condition F in P is well-typed in the context of $[Exp']_{\mathcal{F}}$, then $[Exp]_{\mathcal{F}}$ returns the set of witness trees WI such that $[Exp']_{\mathcal{F}}, WI \models F$. SL specifies the nodes whose descendants should be included in the final output. Exp is well-typed if Exp' and F are.
- (Projection) If Exp is $\pi_{P,PL}(Exp')$ and the condition F in P is well-typed in the context of $[Exp']_{\mathcal{F}}$, then $[Exp]_{\mathcal{F}}$ keeps the nodes in $[Exp']_{\mathcal{F}}$ which match some pattern node in P (and also in PL) for some embedding $h : P \rightarrow C$ (such that the witness tree under h satisfies F in the context of $[Exp']_{\mathcal{F}}$) and their hierarchical relationships but eliminates all other nodes. Exp is well-typed if Exp' and F are.
- (Cross product) If Exp is $Exp_1 \times Exp_2$ then $[Exp]_{\mathcal{F}}$ contains for each pair of trees $T_i \in [Exp_1]_{\mathcal{F}}, T_j \in [Exp_2]_{\mathcal{F}}$, a tree, whose root is a new node, left child is the root of T_i and right child is the root of T_j. If Exp_1, Exp_2 are well-typed, then Exp is.

The join operator can be expressed in the standard way as a composition of $\sigma_{P,SL}$ and $\times$. For the remaining set theoretic algebraic operators we need to specify when two data trees should be considered identical as described in [56]: two data trees T_1, T_2 are *equal* iff there exists an isomorphism $\iota : T_1 \rightarrow T_2$ between the two sets of nodes that preserves edges and order, and furthermore, for every value-based

atom of the form "attribute θ value" (θ is one of $=, \neq, >$, etc), the atom is true at node u in T_1 iff it is true at node $\iota(u)$ in T_2.

- (Union, Intersection and Difference) If $Exp = Exp_1\ op\ Exp_2$ where $op \in \{\cup, \cap, -\}$, then $[Exp]_{\mathcal{F}}$ is the standard result of $[Exp_1]_{\mathcal{F}}\ op\ [Exp_2]_{\mathcal{F}}$. Exp is well-typed if Exp_1, Exp_2 are.

Proposition 2 *For all algebraic expressions Exp over $EI_1 = (V_1, E_1, t_1, H_1), \ldots, EI_z = (V_z, E_z, t_z, H_z)$ and the similarity enhanced fusion $\mathcal{F}$ of $H_1, \ldots, H_z$, $[Exp]_{\mathcal{F}}$ is an* SEO *instance.*

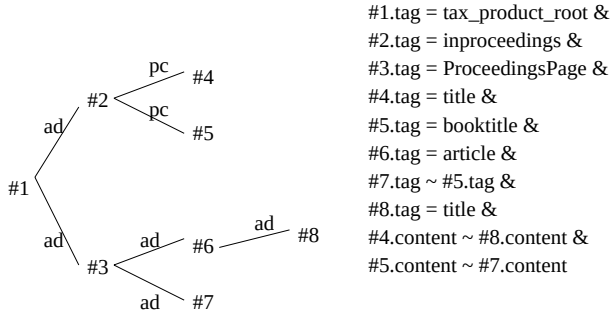


Figure 5.8: A pattern tree P3

Example 20 *Suppose we want to find the papers in SIGMOD DB such that the title of that paper is similar to the title of some SIGMOD conference paper recorded in DBLP. A join of both databases by taking product of them and a selection with the pattern tree shown in Figure 5.8 can solve this task, i.e., $\sigma_{P3,\{\}}(dblp \times ProceedingsPage)$. The result will contain two trees corresponding to the papers titled "Materialized View ..." and "Securing XML ..." respectively.*

5.4 Implementation and Experiments

Figure 5.9 shows the architecture of our TOSS system, which is implemented in Java and currently consists of 24402 lines of code.

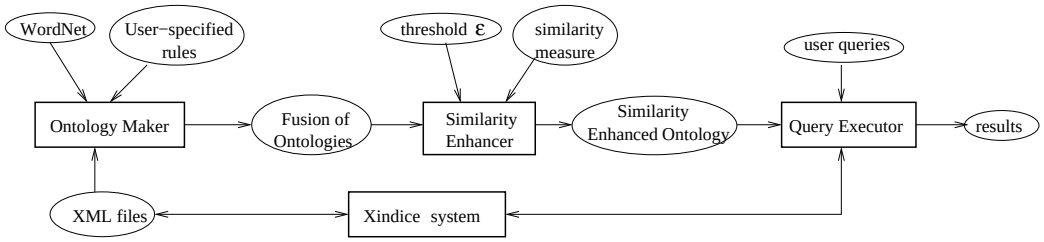


Figure 5.9: Architecture of TOSS system

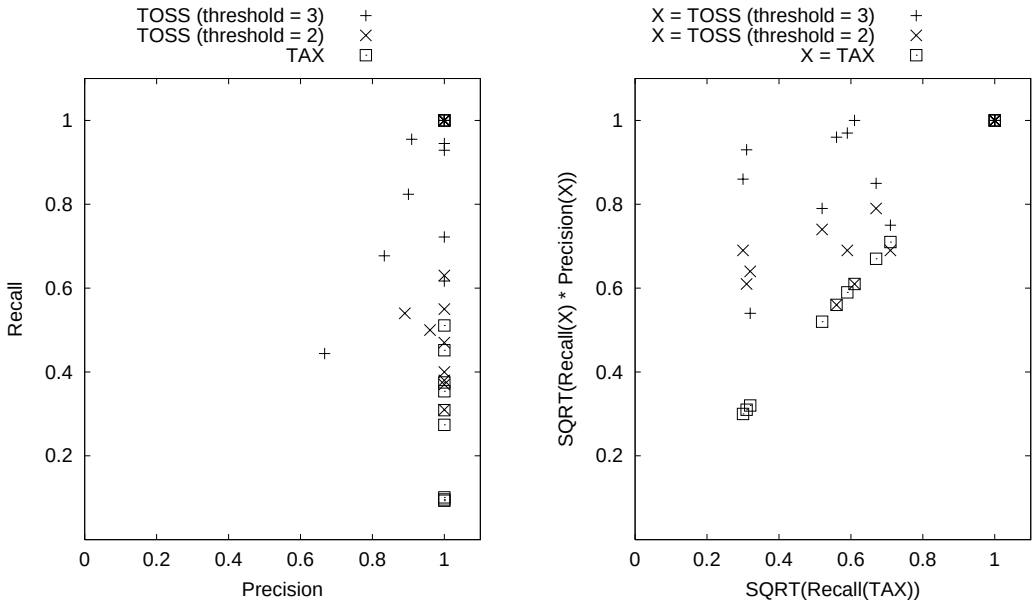
TOSS is built on top of the Xindice[115] database system and consists of the following components: (1) the *Ontology Maker* automatically takes XML files as input and uses WordNet[77] and user-specified rules to automatically generate ontologies, and then integrate them to obtain their fusion; (2) the *Similarity Enhancer* automatically finds the similarity enhancement of the ontology according to the threshold ϵ and the similarity measure specified by the database administrators among a variety of possible choices (supported by a JAVA toolkit available in [28]); (3) the *Query Executor* implements the ontology extended algebra, transforms a user query into a query that takes ontological information into account, and accesses the remote or local Xindice system.

Recall and precision. We obtained the recall and precision of results returned by TOSS and TAX by evaluating 12 selection queries on 3 data sets (each containing 100 random papers from DBLP). Each query contains 1 isa, 1 similarTo and 3 tag matching conditions. For isa and similarTo conditions, "contains" and exact match

are used for TAX respectively. The precision and recall of TOSS and TAX results for each query are calculated by checking against semantically correct results generated manually. A query result contains 1 to 38 papers.

Figure 5.10 (a) shows that TAX always get 100% precision but low recall (lower than 0.5 for 75% of queries) because the exact match in TAX guarantees that the results it returns are correct but it misses most of the correct results. Its recall is 1 for 3 queries whose semantically correct results contain 3 or fewer papers. In contrast, the average precision and recall of TOSS (with $\epsilon = 3$) results are 0.942 and 0.843. For most of queries, it maintains its precision close to 1 with much higher recall. For the query with lowest TOSS precision, by comparing with TAX recall, it shows that it takes a tradeoff of 1/3 of precision degradation for 3 times of recall increase. For TOSS (with $\epsilon = 2$), the average precision and recall are 0.987 and 0.596. The higher precision and lower recall compared with the one of $\epsilon = 3$ are expected because the SEO obtained using a lower ϵ can only merge very similar terms together, missing some correct answers.

Figure 5.10(b) shows the square root of product of recall and precision as a quality measure[107] of TOSS and TAX results against the square root of TAX recall for the corresponding query. It clearly shows that TOSS (ϵ=3) outperforms TAX for all queries (except the 3 queries mentioned above). Figure 5.11(a) shows how much the recall is improved by TOSS compared with TAX normalized by the precision. In TOSS (ϵ=3), most of the queries get their normalized recall more than doubled. Again, the performance of TOSS (ϵ=2) lies between TOSS (ϵ=3) and the



(a) (b)

Figure 5.10: (a) Recall against precision for TOSS and TAX results, (b) Square root of product of recall and precision of TOSS and TAX results against the square root of corresponding TAX recall for each query.

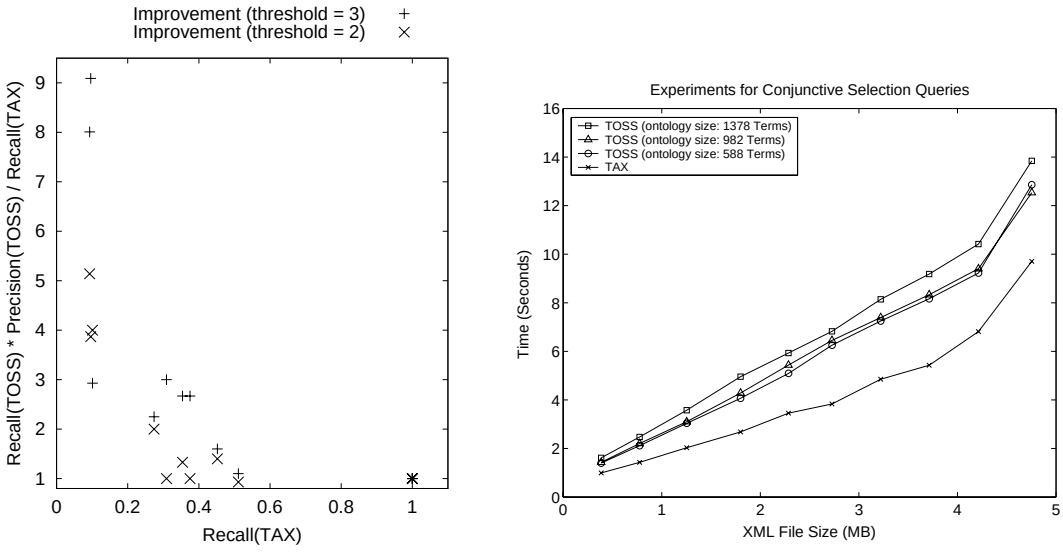
original TAX with the similar reason.

We also compared TAX and TOSS with *F-measure*, defined as

$$F - measure = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

Table 5.2 shows the performance of TAX and TOSS with regard to *F-measure*. Similar to the result in Figure 5.10 (b), TOSS (ϵ=3) still outperforms TAX for all queries (except the 3 queries mentioned above). Except query 9, TOSS (ϵ=2) has equal or better *F-measure* quality than TAX. It is even more than three times better for query 2 and 6.

Scalability experiments. In all our scalability experiments, we used the DBLP and SIGMOD proceeding data. The total DBLP file size is 188,566,002 bytes. We



(a) (b)

Figure 5.11: (a) Improvement factor of TOSS recall compared with TAX recall normalized by precision, (b) Performance of TOSS and TAX for selection queries.

selected all proceeding records and removed unnecessary spaces. Then we truncated the file such that it has 4,753,774 bytes containing 3712 papers due to the 5MB maximum data size limitation of Xindice. The SIGMOD data contains 16 proceedings pages having a total size of 712KB. The experiments were run on a PC with 1.4GHz CPU, 524MB memory and Windows 2000 Professional platform.

The times taken in all experiments include the following: (i) time to parse a pattern tree and rewrite the pattern tree into XPath queries, (ii) time to execute the XPath queries in the Xindice system, (iii) time to parse the result returned from the Xindice system and convert the result to the form defined by TAX.

Scalability of selection. Figure 5.11 (b) shows the results of evaluating TOSS and TAX conjunctive selection queries, each of which contains 2 isa and 4 tag matching conditions, on the DBLP data. For isa conditions, exact match is used for TAX. We

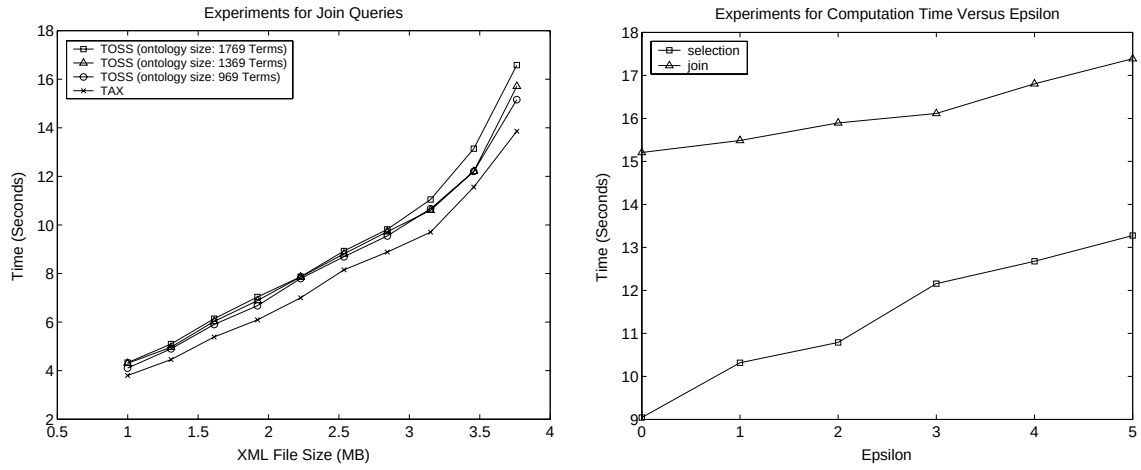
Query	TAX	TOSS (threshold=2)	TOSS (threshold=3)
1	0.62	0.77	0.84
2	0.18	0.57	0.53
3	0.52	0.64	0.97
4	0.47	0.47	0.96
5	0.43	0.71	0.76
6	0.17	0.67	0.86
7	0.55	0.55	1
8	1	1	1
9	0.68	0.66	0.75
10	1	1	1
11	0.18	0.54	0.93
12	1	1	1

Table 5.2: F-measure for TOSS and TAX results

tested the scalability of selection queries by varying the size of ontologies (only for TOSS) as well as the size of XML files.

The reader can easily see that our algorithms handle data size of approximately 4.75 MB in 12.5-14 seconds, almost independent of the ontology size. The slope of the curves are more or less linear - approximately 500 KB can be processed in one second. As the last XML data file has 4,753,774 bytes, very close to the 5MB maximum data size of Xindice, there was a sudden slope up at the last point.

The difference between the lowest TOSS curve in the figure and TAX is in 0.4064-3.1544 seconds. The difference between the highest TOSS curve and TAX is in 0.6228-4.1402 seconds. When data size grows, the difference increases because



(a) (b)

Figure 5.12: (a) Performance of TOSS and TAX for join queries, (b) TOSS computation time of selection and join against ϵ.

there are more accesses to the ontology. The bigger difference is also due to the empty result returned by TAX.

Scalability of join. We also ran scalability experiments on join. Each query contains 5 tag matching and 1 similarTo conditions. For similarTo conditions, exact match is used for TAX. Figure 5.12 (a) shows the scalability of join of the DBLP and SIGMOD data as the total size of the two XML files varies up to a little over 3.7 MB.

As the total size of the data being joined increased, we had a linear increase in the time taken to do the joins except the two last data points. These two data points show a faster increase because Xindice system returns intermediate results in a super-linear time and dominates the total execution time when the size of intermediate result is large. For example, in the experiment of the last data point,

the total execution time was 17 seconds, where 12.1 seconds is from Xindice and 4.9 seconds from our code.

The difference between the lowest TOSS curve in the figure and TAX is in 0.3078-1.3 seconds. The difference between the highest TOSS curve and TAX is in 0.532-2.7238 seconds. The difference increases with the data file size because the number of accesses to the ontology increases.

TOSS computation time vs ϵ. Finally, we conducted experiments to show how the evaluation time of conjunctive selection queries and join queries are affected by the different values of ϵ (in generating SEO). The selection experiment was on a dblp file with 4,753,774 bytes with ontology size of 1003 terms. The join experiment was on a dblp file of 3,071,430 bytes and the sigmod data of 692,188 bytes with ontology size of 1769 terms. Figure 5.12 (b) shows that both execution time increase approximately linearly to the increase of ϵ because when ϵ increases, each node will contain more similar terms in average and thus more time is needed to output a larger result.

5.5 Summary

Our work builds on two important recent contributions in databases, viz. the TAX algebra for semistructured databases by Jagadish et al.[56] and the need to use ontologies for query processing by Wiederhold's group[110, 111].

As mentioned before, TAX is the starting point for our work. The landmark TAX paper proposes a formal theoretical basis for semistructured data, together

with an algebra to query semistructured data sources using the concept of a pattern tree. Wiederhold's group was the first to notice that ontologies can be used to improve the quality of answers to queries. They proposed an *ontology algebra* [112, 73, 81].

TOSS is fundamentally different from both these works and other related works in many ways. First, in TOSS, ontologies are merged under some *interoperation constraints* – no such constraints were considered in [112, 73]. However, this is important because terms in one ontology may have a relationship with terms in another. Second, TOSS does not just merge ontologies together - it extends the semistructured DB model and algebra so that semistructured instances with associated ontologies can be queried. Third, our framework introduces for the first time, the concept of similarity search in semistructured databases - something that neither the TAX work nor Wiederhold's prior work did. Fourth, we have developed experimental results showing that TOSS greatly improves the quality of answers compared to TAX.

Chapter 6

A Theoretical Foundation for Integrating RDF Ontologies

Resource Description Framework (RDF) is a W3C recommendation for processing metadata and exchanging machine-understandable information on the Web. It defines a model for describing relationships between resources. Triples with the form (subject, predicate, object) are the basic elements of RDF.

RDF has become the de facto standard for the description and exchange of metadata on the Web. Many RDF ontologies have been constructed in domains such as food, geography and transportation. For example, both SchemaWeb (www.schemaweb.info) and DAML web site (www.daml.org) have hosted a number of RDF ontologies. Since many of these ontologies are about the same topics, there is a growing need to integrate them in order to facilitate reasoning and query answering across distributed data sources.

In this chapter, we present our work on integrating RDF ontologies [103] which includes a theoretical framework and a set of experiments on real and synthetic ontologies. We focus on the integration of RDF ontologies at the instance level[1], each of which is a finite set of triples.

[1]In some literature, such an RDF ontology at the instance level may be called an RDF graph.

6.1 Preliminaries

In this section, we provide a brief overview of the most important constructs in RDF and show how RDF documents may be viewed as graphs.

An *RDF-ontology* is a finite set of triples (r, p, v) where r is a *resource* name, p is a *property* name, and v is a value (which could also be a resource name). RDF-ontologies assume the existence of some set $\mathcal{R}$ of resource names, some set $\mathcal{P}$ of property names, and a set $dom(p)$ of values associated with any property name p. We do not address reification and containers in RDF due to space constraints. *Throughout the rest of this chapter, we will assume that $\mathcal{R}, \mathcal{P}, dom$ are all arbitrary, but fixed.*

Definition 19 *(RDF Ontology graph). Suppose $\mathcal{O}$ is an RDF-ontology. An* RDF ontology graph *for O is a labeled graph (V, E, λ) where*

(1) $V = \mathcal{R} \cup \bigcup_{p \in \mathcal{P}} dom(p)$ *is the set of nodes.*

(2) $E = \{(r, r') \mid$ *there exists a property p such that* $(r, p, r') \in O\}$ *is the set of edges.*

(3) $\lambda(r, r') = \{p \mid (r, p, r') \in O\}$ *is the edge labeling function.*

It is easy to see that there is a one-one correspondence between RDF-ontologies and RDF-ontology graphs. Given one of them, we can uniquely determine the other. As a consequence, we will often abuse notation and interchangeably talk about both RDF-ontologies and RDF-ontology graphs. Figure 6.1 shows parts of RDF ontologies 64 and 322 from the DAML web site (www.daml.org) — their graphs are

```
<Property ID ="affiliateOf ">
  <domain resource ="#Organization " />
  <range resource ="#Person " />
</Property >
<Class ID="Student "> <subClassOf resource ="#Person" /> </Class>
<Class ID="GraduateStudent  "> <subClassOf resource ="#Student " /> </Class>
<Class ID="Organization "> <subClassOf resource ="#SHOEEntity " /> </Class>
<Class ID="Faculty "> <subClassOf resource ="#Worker" /> </Class>
<Class ID="Person"> <subClassOf resource ="#SHOEEntity " /> </Class>
<Class ID="Worker"> <subClassOf resource ="#Person" /> </Class>
```

```
<owl:Class rdf:ID="Person "/>
<owl:Class rdf:ID="Organization "/>
<owl:Class rdf:ID="Working-Person "><rdfs:subClassOf rdf :resource="#Person"/></owl:Class>
<owl:Class rdf:ID="Affiliated -Person"><rdfs :subClassOf rdf :resource="#Person"/></owl:Class>
<owl:Class rdf:ID="Organization -Unit"/>
<owl:Class rdf:ID="Researcher "><rdfs:subClassOf rdf :resource="#Working-Person"/></owl:Class>
<owl:Class rdf:ID="Researcher -In-Academia">
  <rdfs :subClassOf rdf :resource="#Academic "/>
  <rdfs :subClassOf rdf :resource="#Researcher "/>
</owl:Class>
<owl:Class rdf:ID="Employee ">
   <rdfs :subClassOf rdf :resource="#Working-Person "/>
   <rdfs :subClassOf rdf :resource="#Affiliated -Person"/>
</owl:Class>
<owl:Class rdf:ID="Educational -Employee"><rdfs:subClassOf rdf :resource="#Employee "/></owl :Class>
<owl:Class rdf:ID="Academic "><rdfs:subClassOf rdf :resource="#Educational -Employee"/></owl:Class>
<owl:Class rdf:ID="Student "><rdfs: subClassOf rdf :resource="#Affiliated -Person "/></owl:Class>
<owl:Class rdf:ID="PhD -Student "><rdfs :subClassOf rdf :resource="#Student "/></owl:Class>
<owl:ObjectProperty rdf :ID="has-sub-unit">
  <rdfs :range rdf :resource="#Organization -Unit"/>
  <rdfs :domain rdf :resource="#Organization "/>
</owl:ObjectProperty >
```

Figure 6.1: RDF (respectively OWL) for the two example ontologies

shown in Figure 6.2. Note that both ontologies have had terms renamed (through the attachment of strings ":1" and ":2" respectively). Thus, STUDENT:2 refers to the student concept in ontology 2.

Given two nodes r, r' in an RDF-ontology graph, and a property name p, we say that there exists a *p-path* from r to r' if there is a path from node r to node r' such that every edge along the path contains p in its label. For example, in the second graph of figure 6.2, there is an S-path from RESEARCHER-IN-ACADEMIA to EMPLOYEE. Here S stands for "subClassOf". However, there is no A-path from STUDENT:2 to ORGANIZATION-UNIT:2 where A stands for "Affiliate-Of".

Our techniques differentiate between *transitive* and *non-transitive* properties. For instance, SUBCLASSOF is a transitive relationship, while AFFILIATEOF as depicted in Figure 6.2 is not. A cycle involving a transitive relationship could indicate a semantic problem (e.g. a BOSSOF b, b BOSSOF c, c BOSSOF a seems to indicate a problem). However, a cycle involving a non-transitive properties may not be a problem (e.g. a FRIENDOF b, b FRIENDOF c, c FRIENDOF a is not unusual).

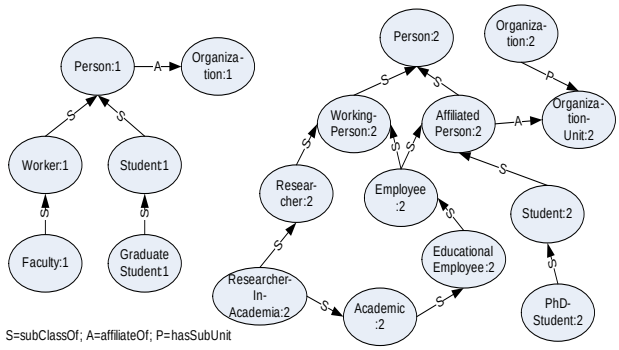


Figure 6.2: Two simple ontologies

Definition 20 (Graph embedding) *Let* $G_1 = (V_1, E_1, \lambda_1)$ *and* $G_2 = (V_2, E_2, \lambda_2)$ *be two RDF-ontology graphs.* G_1 *can be* embedded *into* G_2 *(denoted by* $G_1 \sqsubseteq G_2$*) iff there exists a mapping* $\omega : V_1 \rightarrow V_2$ *such that:*

(1) For p *transitive, if there is a* p*-path from* r *to* r' *in* G_1*, then there is a* p*-path from* $\omega(r)$ *to* $\omega(r')$ *in* G_2*.*

(2) For q *non-transitive, if there is an edge labeled with* q *from* r *to* r' *in* G_1*, then there is an edge labeled with* q *from* $\omega(r)$ *to* $\omega(r')$ *in* G_2*.*

Graphs G_1 *and* G_2 *are* equivalent *iff* $G_1 \sqsubseteq G_2$ *and* $G_2 \sqsubseteq G_1$*.*

Note that for any RDF ontology O, we can uniquely determine its RDF-ontology graph — however, there may be many other graphs that are equivalent to it.

6.2 Horn constraints

When integrating two RDF ontologies, there may be various kinds of constraints linking the terms in the two ontologies. For instance, we may say that the

terms PERSON:1 and PERSON:2 are equivalent. Likewise, we may say that if x is a RESEARCHER-IN-ACADEMIA:2 and also a STUDENT:2, then x is also a GRADUATE-STUDENT:1. In general, for any property p, we may have a Horn clause saying that as far as property p is concerned, if an individual is in all classes in a given set, then he is also in another given class.

Definition 21 (Horn constraint) *If $r_1, \ldots, r_n, r$ are all resource names, then*

$$r_1 \wedge \cdots \wedge r_n \rightarrow r$$

is a Horn[2] constraint*.*

Note that Horn constraints only specify class subclass relationships, and nothing else.

Definition 22 (Negative constraints) *Suppose a, b are nodes and q is a property. Then:*

(1) $a \neq b$ is a negative constraint.

(2) $a \not\rightarrow_q b$ is a negative constraint (which states that there is no q-path from a to b in the graph in question).

Two examples of negative constraints associated with Figure 6.2 are:

- FACULTY:1$\not\rightarrow_S$ STUDENT:2 which states that Faculty is not a subclass of Student.

[2]Strictly speaking, we should say "definite clause"[68] constraint here rather than Horn constraint, but we will abuse notation and call these Horn clauses.

- STUDENT:2$\not\rightarrow_S$ FACULTY:1 which states that STUDENT is not a subclass of FACULTY.

Satisfaction of a Horn clause or a negative constraint by an ontology is straightforward to define.

6.3 The RDF Ontology Integration Problem

In this section, we will declaratively define the problem of integration of two RDF ontologies under a given set of Horn and negative constraints. To do this, we first define the notion of a "Horn Ontology Graph" (HOG for short) which merges Horn constraints with an ontology.

Definition 23 (Horn Ontology Graph (HOG)) *Suppose O is an ontology and H is a finite set of Horn constraints over $\mathcal{R}, \mathcal{P}, dom$. The* Horn Ontology Graph*, $\mathsf{HOG}(O, H)$ is defined as the labeled graph $G = (V, E, \lambda)$ where:*

(1) If a is either a resource name or a domain value in O, then $\{a\}$ is a node in V.

(2) If $r_1 \wedge \ldots \wedge r_n \rightarrow r$ is in H, then there is a node in V called "$\{r_1, \ldots, r_n\}$" — sometimes, we may abuse notation and write this node's label as "$r_1 \wedge \ldots \wedge r_n$"

(3) Whenever there are two nodes A, B in V such that $A \subseteq B$, then there is an edge labeled S from B to A.

(4) If $r_1 \wedge \ldots \wedge r_n \rightarrow r$ is in H, then there is an edge labeled S in E from "$\{r_1, \ldots, r_n\}$" to $\{r\}$.

(5) If there is an edge in O's graph from a to b labeled p, then there is an edge in E from $\{a\}$ to $\{b\}$ labeled p.

Intuitively, the nodes in the Horn ontology graph are obtained by taking nodes in O's graph and making singleton sets out of them. In addition, we take the body of each Horn constraint and make it a node labeled by the entire body of the Horn constraint.

As far as edges are concerned, condition (5) says that all edges in O are present in the HOG (the only difference is that an edge in the original RDF graph from a to b ends up being an edge from $\{a\}$ to $\{b\}$). In addition, for every Horn clause in H, we add an edge from the set associated with its body to the singleton set associated with its head.

Strictly speaking, we may be able to eliminate some edges from $HOG(O, H)$. For instance, if we have one Horn clause with the body $\{a, b, c\}$, another with the body $\{a, b\}$ and yet another with the body $\{a\}$, then condition (3) above states that there should be an edge from both $\{a, b\}$ and $\{a, b, c\}$ to $\{a\}$ as well as an edge from $\{a, b, c\}$ to $\{a, b\}$. We henceforth assume all such redundant edges are eliminated. It is easy to see that the Horn Ontology Graph thus defined is unique and can be constructed from H and O in time $\mathcal{O}(|H| \cdot |\mathcal{R} \cup \bigcup_{p \in \mathcal{P}} dom(p)|)$.

Figure 6.3 shows the HOG associated with the union of the two RDF graphs shown in Figure 6.2 under a given set of Horn constraints.

Definition 24 (Integrability witness) *Suppose O_1, O_2 are two ontologies, H is a finite set of Horn clauses and NC is a finite set of negative clauses. An ontology*

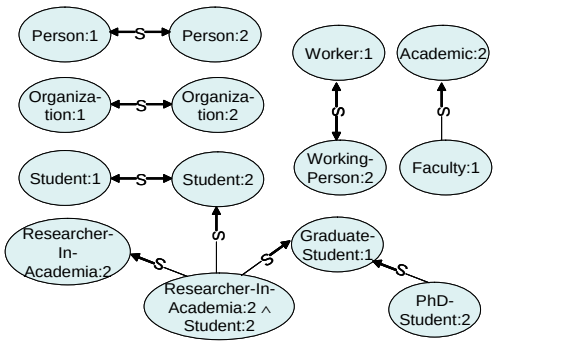


Figure 6.3: HOG example (partial)

graph $G = (V, E)$ *is said to be a* witness to the integrability *of* O_1, O_2 *w.r.t.* H *and* NC *iff:*

(1) G *contains no* p*-cycles*[3] *for any transitive property* p*,*

(2) $HOG(O_1 \cup O_2, H) \sqsubseteq G$*.*

We define the *distance* between a witness G to the integrability of ontologies O_1, O_2 subject to Horn constraints H and negative constraints NC as: $d(G, \langle O_1, O_2, H \rangle) = |G_1| + |G_2| + |H| - |G|$ where G_i is the graph associated with ontology O_i.

Definition 25 (Minimal integrability witness) *Suppose* O_1, O_2 *are two ontologies with associated graphs* G_1, G_2*,* H *is a finite set of Horn constraints and* NC *is a finite set of negative constraints. A witness* G *to the integrability of* G_1, G_2 *w.r.t.* H *and* NC *is* minimal *if and only if there is no strict subgraph*[4] G' *of*

[3] A p-cycle is a p-path of length 1 or more from a node to itself.

[4] We use the standard definition of subgraph.

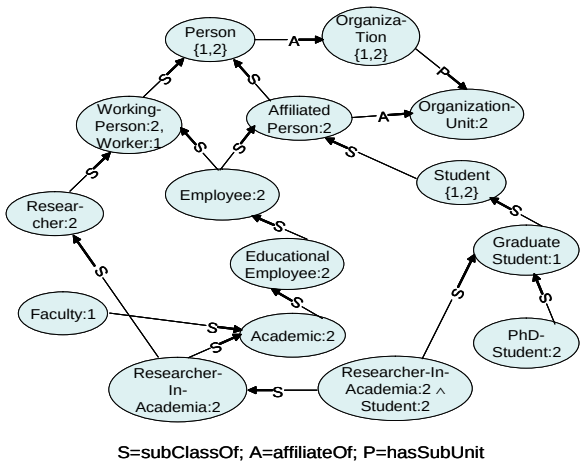


Figure 6.4: Minimal integrability witness

G which is a witness to the integrability of O_1, O_2 *w.r.t.* H *and* NC *such that* $0 \leq d(G', \langle O_1, O_2, H\rangle) \leq d(G, \langle O_1, O_2, H\rangle)$.

Note that the minimal integrability witness is minimal in two respects: first, its graph structure is minimal (i.e. it does not contain unnecessary nodes and edges) and in addition, the distance according to the d-metric is minimized. Figure 6.4 shows a minimal witness to the integrability of the ontologies in Figure 6.2 using the Horn Constraints shown in Figure 6.3 and the negative constraints $\textsc{Faculty}{:}1 \not\rightarrow_S \textsc{Student}{:}2$ and $\textsc{Student}{:}2 \not\rightarrow_S \textsc{Faculty}{:}1$ discussed earlier.

RDF Ontology Integration Problem. The RDF ontology integration problem is specified as follows:

Input: RDF ontology graphs O_1, O_2, a finite set H or Horn constraints, and a finite set NC of negative constraints.

Output: Return a minimal witness to the integrability of O_1, O_2 w.r.t. H and NC if a witness exists — otherwise return NULL.

6.4 The CROW Algorithm to integrate Ontologies

In this section, we present the CROW (Computing RDF Ontology Witness) algorithm to solve the RDF ontology integration problem. The CROW algorithm has four distinct phases:

(1) In the *pre-processing phase*, we rename the ontologies being merged so that terms in different ontologies are different (this is done by merely appending the ontology id to the end of each term).

(2) In the *graph construction phase*, CROW constructs $\mathsf{HOG}(O, H)$.

(3) In the *graph transformation phase*, the above graph is simplified using various kinds of strongly connected computations which we will describe shortly. Redundant edges are also eliminated during this phase.

(4) In the *negative constraint check phase*, we check whether the graph produced above satisfies the negative constraints. If not, we return "NULL" otherwise we return the graph produced in (3) above.

The heart of the algorithm is in step (3) above — we have already explained steps (1) and (2) earlier on in the paper. We apply two graph transformations.

Definition 26 *Suppose G is an RDF ontology graph, and p is any relation in $\mathcal{P}$ that is known to be transitive.*

- *A* p-strongly connected component *(or* p-SCC *for short) is any set S of nodes in G such that for all $a, b \in S$, there is a p-path from a to b.*

- *The operator $\varsigma_p(G)$ returns that graph $G' = (V', E')$ where:*
  - *V' is the set of all p-SCCs in G and*
  - *E' contains an edge from an SCC S_1 to an SCC S_2 labeled p iff there is an edge in G from some node in S_1 to some node from S_2 labeled p.*

$\varsigma_p(G)$ reduces all p-SCCs in G to a single node and then draws edges between two such reduced nodes if there was some node in one of the SCCs that was connected in the original graph G to some node in the other SCC. This is an intuitive method for reducing cycles for transitive relationships (e.g.: if WORKER:1 SUBCLASSOF WORKINGPERSON:2 and WORKINGPERSON:2 SUBCLASSOF WORKER:1, it would be safe to conclude that WORKER:1=WORKINGPERSON:2). Note that $\varsigma_p(G)$ is a lossy transformation as edges not labeled with p between nodes in the p-SCCs are lost. The second transformation - $\upsilon_p(O)$ - is used to eliminate redundant edges and thus obtain a minimal integration witness.

Definition 27 *An edge from a to b labeled p is said to be* redundant *w.r.t. graph G iff there is a p-path from a to b in the graph obtained by deleting this edge from G.*

The operator $\upsilon_p(G)$ returns a graph G' by eliminating as many redundant edges on transitive properties from G as possible.

There may be many ways in which redundant edges are removed from a graph G — all we require here is that $\upsilon_p(G)$ return any subgraph of G with no redundant

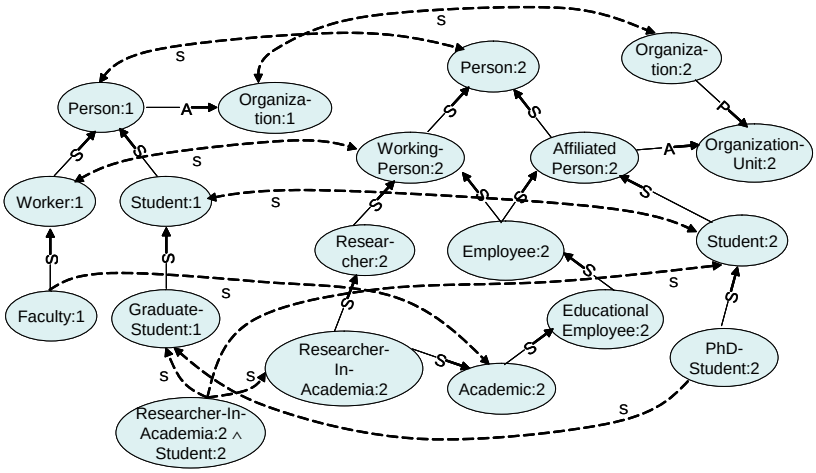


Figure 6.5: Intermediate result *CROW* phase (1)

edges.[5]

Theorem 4 *For any graph G:*

(1) $G \sqsubseteq \varsigma_p(G)$

(2) $G \sqsubseteq \upsilon_p(G)$.

The proof is based on the analysis of a single p-SCC or redundant edge elimination operation; each such operation preserves the $\sqsubseteq$ relation.

Proof sketch: (1) We represent $\varsigma_p(G)$ as a sequence of operations $\langle \varsigma_p^1(G), \ldots, \varsigma_p^i(G), \ldots \rangle$, where each $\varsigma_p^i(G)$ collapses exactly one p-SCC into one node. Let us denote by G_i the graph resulting from the application of the first i transformations and let V_i

[5]As such there can be many different implementations of $\upsilon_p(G)$. One such algorithm works as follows. First, it computes an adjacency matrix A for G. It then computes a path matrix B such that $B[i,j] = 1$ iff there is a path of length two or more from node i to j. This can be computed by a straightforward adaptation of Dijkstra's algorithm. Now remove all edges (i,j) for which $A[i,j] = 1$ and $B[i,j] = 1$.

be its set of nodes. Let S_i be the set of nodes that form a p-SCC and should be collapsed at step $i+1$. Let n_{i+1}^S be the node resulting by collapsing S_i and let m_i be an arbitrary node in S_i. Then we define a mapping $\omega : V_i \rightarrow V_{i+1}$ such that $\forall\ x \in S_i, \omega(x) = n_{i+1}^S$ and $\forall\ x \in V_i - S_i, \omega(x) = x$. To prove that ς_p is order preserving, we need to show that ω satisfies the conditions in Definition 20. Let $x, y \in V_i$, $x \neq y$ be such that there exists a q-path between x and y represented by $\langle n_1, \ldots, n_k \rangle$. Let us consider the sequence $\langle \omega(n_1), \ldots, \omega(n_{k'}) \rangle$, where we have omitted consecutive duplicate nodes. As ς_p only removes edges not labeled with p in S_i (collapsed into one node in G_{i+1}), this is a q-path in G_{i+1} (maybe containing fewer nodes than the original). Hence, every step in ς_p is order preserving. So by induction we conclude that ς_p is order preserving.

(2) Let $path_q(x, y)$ be true iff there is a q-path between x and y. By definition, $\upsilon_p(G)$ does not modify the value of this predicate for any property q. Hence, the conditions in Definition 20 are preserved. □

We are now ready to present the CROW algorithm (after the pre-processing step) in Table 6.1. Although the CROW algorithm is designed for integrating two ontologies, a simple extension can be used to integrate sets of ontologies [6].

In lines 1-9, CROW first constructs the Horn Ontology Graph. The result of this first phase is depicted in Figure 6.5. The reader may notice that this graph contains both strongly connected components for the SUBCLASSOF hierarchy (STUDENT:1 $\leftrightarrow_S$ STUDENT:2) as well as redundant edges (PHD-STUDENT:2

[6] For instance, by using CROW repeatedly or by merging all the ontology graphs at the same time.

Algorithm: CROW(O_1, O_2, H, NC)

Inputs: Ontologies O_1, O_2, set of Horn constraints H and set of negative constraints NC. Let $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ be the associated graphs.

Output: $G = (V, E, \lambda)$, minimal witness to the integrability of O_1, O_2 w.r.t. H and NC or $NULL$ if no minimal witness exists.

1. $V \leftarrow V_1 \cup V_2$;
2. $E \leftarrow E_1 \cup E_2$;
3. **for** $r_1 \wedge \ldots \wedge r_k \rightarrow r \in H$ in ascending order of k **do**
4. $\quad V \leftarrow V \cup \{r_1, \ldots, r_k\}$;
5. $\quad E \leftarrow E \cup \{(\{r_1, \ldots, r_k\}, r)\}$;
6. $\quad$ **for** all $n \in V$, $n \subset \{r_1, \ldots, r_k\}$ **do**
7. $\quad\quad E \leftarrow E \cup \{(\{r_1, \ldots, r_k\}, n)\}$;
8. $\quad$ **end**
9. **end**
10. **for** all properties p
11. $\quad$ **if** p transitive **then**
    (* the transitivity of p is specified a priori *)
12. $\quad\quad$ **while** $\exists$ S, p-SCC in G **do**
13. $\quad\quad\quad$ **for** $a \rightharpoonup b \in NC$ **do**
    (* by $\rightharpoonup$ we denote any type of negative constraint *)
14. $\quad\quad\quad\quad$ **if** $a \in S$ and $b \in S$ **return NULL**;
15. $\quad\quad\quad$ **end**
16. $\quad\quad\quad$ $G = \varsigma_p^S(G)$;
    (* it collapses only S into one node *)
17. $\quad\quad$ **end**
18. $\quad\quad$ $G = \upsilon_p(G)$;
19. $\quad$ **end**
20. $\quad$ **for** $cons \in NC$ labeled with p **do**
21. $\quad\quad$ **if** G does not satisfy $cons$ **then return NULL;**
22. $\quad$ **end**
23. **end**
24. **return** O;

Table 6.1: Computing RDF Ontology Witness Algorithm **CROW**

subClassOf Student:2). The reduction of strongly connected components and redundant edges is suitable only for transitive properties, as discussed in the Preliminaries Section. After these transformations (lines 16-18) the algorithm verifies the satisfiability of the negative constraints against the integrated ontology (lines 20-22). Note that some negative constraints can be checked directly while detecting SCC, which optimizes the response time for cases when there is no minimal integrability witness. Figure 6.4 shows the result of integrating the ontologies in Figure 6.2 using the Horn constraints in Figure 6.3 and using the negative constraints: $NC = \{Faculty{:}1 \not\rightarrow_S Student{:}2, Student{:}2 \not\rightarrow_S Faculty{:}1\}$.

The following result shows that CROW is correct.

Theorem 5 (Correctness of CROW) *CROW is correct, i.e.*

(i) If CROW(O_1, O_2, H, NC) does not return NULL, then it returns a minimal witness to the integrability of G_1, G_2 w.r.t. H and

(ii) If CROW(G_1, G_2, H, NC) returns NULL, then there is no witness to the integrability of G_1, G_2 w.r.t. H.

Proof sketch. (i) Since all sets are finite and operations on lines 10-19 reduce the number of edges and/or nodes in the ontology, the algorithm will terminate. By the construction of the HOG in lines 1-9 we can easily see that $G_1 \sqsubseteq HOG(G_1 \cup G_2, H)$ and $G_2 \sqsubseteq HOG(G_1 \cup G_2, H)$. Furthermore, by Theorem 4 the graph transformations applied to $G = HOG(G_1 \cup G_2, H)$ preserve order. Also, G will satisfy NC according to lines 14, 20-22. Hence, the returned ontology G is an integration witness for G_1, G_2 w.r.t H and NC. Let us assume that G is not a minimal integrability witness

and $\exists$ G' an integrability witness which is a strict subgraph of G. If G' has fewer nodes than G, it would violate the distance condition in Definition 25. Otherwise, if G' has fewer edges than G, since all redundant edges for transitive properties have been eliminated in G, that would mean G' violates condition (2) of Definition 24. Our hypothesis was false, hence G is a minimal integrability witness. (ii) Let us assume that there is a minimal integrability witness G that the CROW algorithm does not find. This means that CROW has returned NULL on either line 14 or 21. However, these directly correspond to checks against the negative constraints in NC. Therefore, G does not satisfy NC and cannot be a minimal integrability witness. □

The following theorem states that CROW runs quite fast.

Theorem 6 (CROW complexity) *Let n_i, e_i be the number of nodes and respectively edges in G_i, $i \in \{1, 2\}$. Let n_h be the number of constraints in H and t the number of transitive properties. We denote by n, e the number of nodes and respectively edges in G. Then according to the CROW algorithm:*

(1) $n \leq n_1 + n_2 + n_h$ and $e \leq e_1 + e_2 + 2 \cdot n_h$.

(2) CROW is $\mathcal{O}(t \cdot n \cdot e \cdot (n + e))$.

The proof follows directly from the analysis of the CROW algorithm. Proof: **Proof sketch**. (i) The CROW algorithm adds all nodes from the source ontologies and at most one node for each Horn constraint to G. Also, aside from the edges in the source ontologies, each Horn constraint can add at most two edges. One is the

edge specified by the Horn constraint itself. The other may be a subset-induced edge according to condition (3) in Definition 23. There is at most one such edge in G since S is a transitive relationship and redundant edges for transitive relationships are eliminated.

(ii) The initial phase of the algorithm (lines 1-9) is performed in $\mathcal{O}(n_h \cdot n)$. SCC detection on the constructed graph can be performed in $\mathcal{O}(n + e)$, while ς_p^S elimination for a p-SCC S can be performed in $\mathcal{O}(e \cdot (n + e))$ (due to the edge rebuilding phase in Definition 26). Therefore, ς_p for all p-SCCs can be performed in $\mathcal{O}(n \cdot e \cdot (n + e))$. Meanwhile, by adapting Dijkstra's algorithm, in $\mathcal{O}(n^2)$ we can detect whether there is a p-path of length more than 1 from a fixed source to a target. Therefore, $\upsilon_p(G)$ can be performed in $\mathcal{O}(e \cdot n^2)$ to eliminate all the redundant edges. Since these two transformations are performed for all transitive relationships, the G ontology is obtained in $\mathcal{O}(t \cdot n \cdot e \cdot (n + e))$. Negative constraint satisfiability is linear in the number of constraints, which cannot be greater than n^2 (which would mean there are negative constraints for every pair of nodes in the graph). Therefore, the complexity of the CROW algorithm is bounded by $\mathcal{O}(t \cdot n \cdot e \cdot (n + e))$. □

6.5 Implementation and Experiments

Our prototype is implemented in Java and currently consists of 3041 lines of code. It consists of the following components:

1. The *RDF Graph Generator*. We use Jena 2.2, a Semantic Web toolkit from Hewlett-Packard Labs, to digest an ontology data file. Then the *RDF Graph*

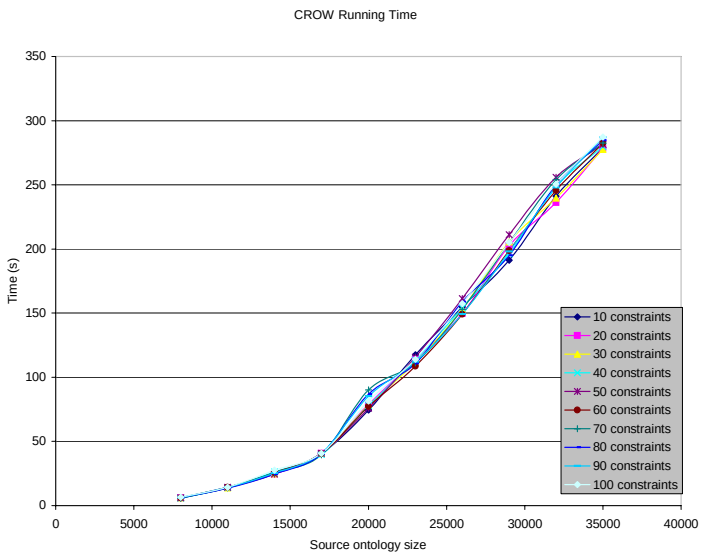


Figure 6.6: CROW running time

Generator constructs an ontology graph by reading a set of RDF triples stored in Jena.

2. The *HOG Generator* generates a HOG for a set of Horn clauses.

3. The *Synthetic Data Generator* randomly generates ontology graphs, Horn constraints and negative constraints.

4. The *Ontology Integrator* integrates ontologies by taking a set of Horn constraints and a set of negative constraints into account.

Scalability experiments. We report here the performance of CROW algorithm on both real-life and synthetically generated ontologies. The experiments were run on a PC with 1.4GHz CPU, 524MB memory and Windows 2000 Professional platform. The times taken in the experiments include *the time to construct an HOG* and *the time to integrate ontologies*.

Ont 1	Ont 2	# conts	$size_1$	$size_2$	$size_3$	$size_4$ (KB)	# trans	Time (ms)
D - 64	D - 189	17	485	820	826	17.944	3	40.2
D - 4	D - 62	25	1171	2142	2147	52.564	1	32
D - 62	ka2	20	1019	1864	1868	51.972	1	22.2
S - 236	S - 197	8	959	1829	1841	51.152	1	24
D - 276	S - 162	32	1706	2938	2946	81.667	4	58.4

Table 6.2: Experimental results

Scalability on real-life ontologies. We have applied our CROW algorithm to five pairs of similar ontologies[7] selected from DAML, SchemaWeb and OntoBroker ontology libraries. We list the detailed experimental results in Table 6.2. In the table, due to space constraints, we use D to stand for DAML, S for SchemaWeb and ka2 for the ontology ka2.daml in OntoBroker. In addition, "# conts" represents the number of constraints and "# trans" is the number of transitive properties. We have used various metrics for the size of ontologies (using the notations from Theorem 6): (i) $size_1 = n_1 + n_2 + n_h$,(ii) $size_2 = n_1 + n_2 + e_1 + e_2 + n_h$, (iii) $size_3$ is the number of nodes and edges in the HOG and (iv) $size_4$ is the total file size of the source ontologies. The running time of CROW algorithm is related to not only the number of terms and edges, but also to the number of transitive properties. We can see the effect of this factor in Table 6.2 by comparing the first and third pair of ontologies.

Scalability on synthetic data. We randomly generated ten pairs of ontology graphs (each with only one transitive property). The total number of terms in each

[7]Two on academic departments, one each on publications, travel and music.

pair of ontologies varies from 8000 to 35000 with a step of 3000. For each pair, we generate ten sets of constraints (including Horn and negative constraints) whose sizes vary from 10 to 100 with a step of 10.

Figure 6.6 shows the scalability of CROW algorithm on the ten pairs of ontologies. As the total number of terms increased, we had a slightly faster than linear increase in the time taken to do the integration. The increase in running time is mainly due to the increase in number of terms (and the size of the HOG) more than to the increase in number of constraints, as shown in Figure 6.6.

6.6 Summary

In this chapter, we have developed a formal model to integrate RDF ontologies in the presence of both Horn constraints and negative constraints. We have explained the concept of a witness to the integrability of two ontologies and we have developed the CROW algorithm to integrate two ontologies. We have tested CROW and found that it is very fast, both on ontologies at the DAML, SchemaWeb and OntoBroker sites, and on synthetically generated ontologies.

Chapter 7

Finding Similar Objects Using a Taxonomy

Finding interoperation constraints between ontologies is fundamentally important for integrating them. In this chapter, we present our work on ontology-based similarity measures [95] applicable for identifying interoperation constraints and searching similar objects.

Taxonomies provide a precise way to name classes of individuals that share certain properties or behavior. They also provide a means of determining how similar one such individual is to another. Two individuals share the properties of the most specific grouping that includes both of them, and the degree to which the two individuals are similar depends on the location of this class in the hierarchy. As we have introduced in Chapter 3, several authors have defined ways of turning this intuitive idea of similarity into a numeric value that can be used to rank the similarity of objects.

The ability to find similar objects given a description of a target is useful in many domains. For example, one may wish to find patents similar to a given patent, or gene products (e.g. proteins) similar to a given gene product. In these domains, and many others, comprehensive taxonomies have been defined and used by various organizations to classify sets of objects. Examples include the Gene Ontology (GO)[7], Medical Subject Headings (MESH)[83] and the patent classification tax-

onomies of the United States Patent and Trademark Office (USPTO)[105] and the World Intellectual Property Organization (WIPO)[113].

In this chapter, we address several issues that arise when putting the idea of searching similar objects with taxonomies into practice. In particular, we provide a definition of information-theoretic similarity for taxonomies that are structured as directed acyclic graphs from which multiple terms may be used to describe an object. We discuss how our definition should be adapted in the presence of ambiguity, as introduced by an evolving taxonomy or classifiers with imperfect knowledge. We also present a pragmatic implementation of our similarity measures, and a variety of others from the literature, that is tightly integrated with an object-relational database and scales to large taxonomies and large datasets. Our experiments — on a bioinformatics application that matches the annotations for gene products in the Gene Ontology supplied by Proteome Inc. and UniProt — show that our two measures outperformed previous approaches.

The remainder of this chapter is organized as follows. At the beginning of Section 7.1, we review the information-theoretic definition of similarity. Next, in Section 7.1.1, we define our holistic similarity measure. In Section 7.1.2, we consider how classifiers that have incomplete knowledge of the objects being classified introduce ambiguity, and define the generic holistic similarity measure for such situations. Section 7.2 briefly summarizes related work. In Section 7.3 we describe our implementation, and Section 7.4 compares the efficacy of our new similarity measures to a variety of others. Section 7.5 summarizes our contributions.

7.1 Information-Theoretic Similarity

The idea of using an information-theoretic definition of similarity to compare objects labeled using a taxonomy was introduced by Resnik[94]. Given a taxonomy defined over a set of terms T, Resnik defined the similarity of two terms $t_1 \in T$ and $t_2 \in T$ with respect to a corpus of objects $\mathcal{O}$ as

$$sim_{Resnik} = \max_{\hat{t} \in S(t_1, t_2)} [-\log p(\hat{t})]$$

where $S(t_1, t_2)$ is the set of terms in T that subsume[1] t_1 and t_2 and $p(t)$ is the probability that an object randomly chosen from $\mathcal{O}$ represents an occurrence (instance) of term t [94]. An object represents an occurrence of a term t if it is labeled with t or a descendant of t.

The quantity

$$I(t) = -\log p(t)$$

is known as the *information content* of term t. Resnik's definition of similarity captures the idea that the similarity between two objects depends not only on what they have in common, but also on the context in which they are compared. It has many intuitively appealing properties. If the taxonomy is a hierarchy (i.e a tree), the term $\hat{t}$ satisfying Resnik's definition will be the least common ancestor of t_1 and t_2. As the commonality of the terms decreases, the position of $\hat{t}$ moves higher in the hierarchy, becoming the root r if the terms have nothing in common. Since every object in $\mathcal{O}$ represents an occurrence of the root, $p(r) = 1$ and $I(r) = 0$. Conversely, if $t_1 = t_2$, $I(\hat{t}) = I(t_1) = I(t_2)$.

[1] Concept C_1 subsumes C_2 if C_1 is a superclass of C_2 in the taxonomy.

Lin[66] derives an axiomatic definition for similarity based on a limited set of assumptions that can be summarized as follows:

1. The maximum similarity between two (identical) objects is 1 and the minimum similarity is 0.

2. The similarity between two objects is a function of their commonality and their differences.

3. If each object can be described from several perspectives, the overall similarity of two objects is a weighted average of their similarities as seen from the individual perspectives.

Under these assumptions, and additionally assuming that the target and candidate objects are selected independently, the similarity between a target object T and a candidate object C is[66]:

$$sim(T, C) = \frac{I(common(T, C))}{I(descr(T)) + I(descr(C))} \tag{7.1}$$

where $common(T, C)$ represents a description of what the two objects have in common, and $descr(T)$ and $descr(C)$ represent descriptions of the two objects individually.

If T and C are each labeled by a single term from a hierarchical taxonomy, their commonality is represented by the term that is the least common ancestor a of the terms t and c that were used to describe T and C, respectively. With respect to a particular corpus of objects $\mathcal{O}$, the similarity becomes[66]:

$$sim_{Lin}(T, C) = \frac{2I(a)}{I(t) + I(c)} = \frac{-2 \log p(a)}{-\log p(t) - \log p(c)} \tag{7.2}$$

Suppose, for example, that $\mathcal{O}$ is a set of edible things and that t, the label of T, is "apple" and c, the label of C, is "pear". If a, the least common ancestor of t and c, is the term "fruit", then the numerator of the similarity measure represents the probability that two objects randomly chosen from $\mathcal{O}$ are both labeled with terms that denote kinds of fruit, and the denominator represents the probability that one is labeled an apple and one is labeled a pear.

7.1.1 Holistic Similarity

Equation 7.2 intuitively captures the idea of similarity in the most straightforward case, in which the taxonomy is structured as a tree and each object is labeled by a single term. However, in practice many taxonomies are not trees, but allow a new term to be derived from multiple parents. Furthermore, classification systems often allow an object to be labeled using multiple terms. In this section, we examine how to define similarity under these more general conditions. We begin by defining a taxonomy as a directed acyclic graph.

Definition 1 (Taxonomy) *A taxonomy $\mathcal{T}$ is a directed acyclic graph (N, E, r), where N is a set of nodes, $E \subseteq \{N \times N\}$ is a set of directed edges, $r \in N$ is a unique root node of $\mathcal{T}$, and for every other node $n \in N$, there is at least one path from n to the root r.*

Each node $n \in N$ represents a term of $\mathcal{T}$, and we will use the words "node" and "term" interchangeably. Each directed edge $e \in E$ connects a more specific child term to a more general parent term. Typically, the relationship between parent and

child terms can be described as an *is-a* or *kind-of* relationship, but our measures can also be used with other transitive relationships, such as *part-of*, *works-for*, *belongs-to*, etc., as long as one wishes to treat objects closely related in these ways as similar.

For convenience, we will refer to nodes in the out-neighbor region of a given node (i.e all nodes on the paths from the given node to the root) as "ancestors" of the given node, and similarly use the term "descendants" to refer to nodes in the given node's in-neighbor region. We will also find it useful to describe portions of the taxonomy as subgraphs of $\mathcal{T}$. We will use the notation $Terms(g)$ to refer to the set of nodes in a subgraph g.

A *label* is a subset of the terms in the taxonomy. A label can be used to represent the classification of a specific object, but it can also be used to represent more general concepts, like what two objects have in common.

Definition 2 (Label) *Given a taxonomy $\mathcal{T} = (N, E, r)$, a label is a nonempty set of terms $L \subseteq N$.*

A cheeseburger labeled with terms from a food taxonomy, for example, might have the label $\{Beef, Cheese\}$.

If a term in a taxonomy applies to an object, so do the terms that correspond to each of its ancestors. Therefore, some of the terms in a label may be redundant. Although, as will be seen, such redundant terms do not affect similarity as calculated by our measures, their presence complicates the descriptions of objects, and obscures the equivalence of labels. We therefore define the concept of a *minimal label* to eliminate such terms.

Definition 3 (Minimal Label) *A label L is a minimal label if for every term $l \in L$, no ancestor of l is also in L.*

Given a label L, one can derive a unique minimal label L' by removing from L every term that is an ancestor of another term in L. Let $Lmin(L)$ denote the minimal label derived from L.

A *labeling* assigns a label to each object in some corpus.

Definition 4 (Labeling) *Given a taxonomy $\mathcal{T}$ and a corpus $\mathcal{O}$, a labeling is a total function $L : \mathcal{O} \mapsto 2^N$. If o is an object in $\mathcal{O}$, let $L(o)$ denote the label for object o.*

A label can also be associated with any subgraph of the taxonomy. The label for a subgraph g is the set of nodes contained in the subgraph, i.e $Terms(g)$. For convenience, however, we will generally refer to labels with the notation L_X, where X may be either an object or a graph.

A labeling is a *proper labeling* if for all $o \in \mathcal{O}$, $L(o) \neq \emptyset$, and a *minimal labeling* if for all $o \in \mathcal{O}$, $L(o)$ is a minimal label. For the remainder of this chapter we will assume that all labelings are minimal proper labelings unless otherwise noted.

We interpret the labeling of an object with one or more terms to imply only that *at least* those terms (and their ancestors) apply to the object. This open-world interpretation of labels is an important assumption of our work, and contrasts with the closed-world model of traditional databases, under which all terms not included in a label do not apply to the labeled object. We adopt the open-world model in this paper for a number of reasons. In the first place, we believe it more accurately

reflects how taxonomies are used in many domains. Frequently, both the label of an object and the structure of the taxonomy itself change as knowledge accumulates concerning the domain and the objects of interest. At any point in time, the label of an object only reflects what has been discovered about it so far. Secondly, when the taxonomy is structured as a DAG, the closed-world assumption is incompatible with the use of interior terms in labels. We defer a discussion of this latter point to Section 7.1.2.

It is also useful to be able to enumerate all the terms associated with a label, either directly or indirectly. We therefore formalize the concept of an *ancestor graph*.

Definition 5 (Ancestor Graph) *Let L be a label. For each term $l \in L$, let $out(l)$ be the out-neighbor region of l in $\mathcal{T}$. The ancestor graph of L is the union of the out-neighbor regions of the terms contained in L. That is:*

$$Anc(L) = \cup_{l \in L} out(l)$$

Given an object o with label L_o, we will use the notation $Anc(o)$ to refer to the ancestor graph of L_o. The set of nodes $Terms(Anc(o))$ represent an exhaustive list of the terms associated with o.

We now associate a probability with an arbitrary label. Its value is the probability of finding an object to which *at least* those terms in the label L apply. We refer to this as the *inclusion probability*, $p_i(L)$.

Definition 6 (Inclusion Probability) *Let L be a label. Then $p_i(L)$ is the probability that the ancestor graph of the label of an object chosen at random from $\mathcal{O}$*

contains L. That is, given a randomly chosen object o:

$$p_i(L) = p(L \subseteq Terms(Anc(o)))$$

If the ancestor graph of an object's label contains L, it also contains $Lmin(L)$ and vice versa. Hence, if L is not minimal, the extra terms do not affect its inclusion probability. That is, $p_i(L) = p_i(Lmin(L))$.

Inclusion probability gives us the tool we need to apply Lin's general definition of similarity from Equation 7.1 to a taxonomy. To quantify the individual information content of the objects being compared (the denominator of Equation 7.1), we use the inclusion probability of their labels. To quantify the information content of the commonality between the two objects (the numerator of Equation 7.1), we find a label L_A to represent that commonality, and use the corresponding inclusion probability. The label L_A is constructed by intersecting the ancestor graphs of the labels of the objects being compared. We refer to the resulting measure as *holistic similarity* because it treats all the terms in a label as a group. As we will see in Section 7.2, other measures that have been suggested for use when objects are labeled with multiple terms treat each term individually.

Definition 7 (Holistic Similarity) *Let L_T and L_C be the labels of objects T and C, respectively, and let $L_A = Lmin(\ Terms(Anc(L_T) \cap Anc(L_C)))$. Then:*

$$sim_H(T, C) = \frac{-2 \log p_i(L_A)}{-\log p_i(L_T) - \log p_i(L_C)}$$

Alternatively, expressed in terms of the information content corresponding to each

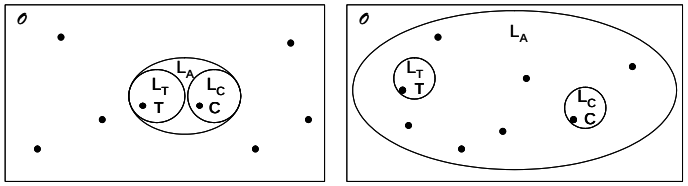


Figure 7.1: Similarity in Context

probability:

$$sim_H(T, C) = \frac{2I(L_A)}{I(L_T) + I(L_C)}$$

Figure 7.1 illustrates the idea behind our measure graphically, by depicting two objects being compared, T and C, as members of populations to which their particular label applies. All such objects are also members of the larger set of objects for which label L_A applies, and are of course also members of $\mathcal{O}$, the corpus. The sizes of these sets relative to each other and the size of the corpus determine the similarity of the objects being compared. The larger the region associated with L_A relative to the size of $\mathcal{O}$, the less significant it is that the objects being compared are included in this set, because the same also applies to many other objects in $\mathcal{O}$. Similarly, the smaller the size of regions L_T and L_C relative to L_A, the less significant it is that both objects are in L_A, because many objects share this trait. Thus, as illustrated in Figure 7.1(a), maximum similarity (short of identity) occurs when L_A is a small fraction of $\mathcal{O}$ and only objects labeled with L_T or L_C are contained within it. Conversely, similarity is at a minimum when L_A contains most of $\mathcal{O}$, including many objects other than those labeled with L_T or L_C, as in Figure 7.1(b).

When the taxonomy is a tree and each object is labeled with a single term, Definition 7 reduces to Equation 7.2. Consider objects T and C, labeled with terms t and c, respectively. When $\mathcal{T}$ is a tree, L_A, the minimal label for the intersection of the ancestor graphs of t and c, is a, the least common ancestor of t and c in $\mathcal{T}$.

As defined by Equation 7.2, Lin's formulation cannot be used when the taxonomy is a DAG, because the terms describing the objects do not necessarily have a unique least common ancestor. Nor is it applicable when objects are labeled with multiple terms. Resnik's similarity measure, which is defined in terms of the ancestor with the maximal information content, could be used with a DAG taxonomy, but not when multiple terms are used as labels. However, because it is specified over labels, not terms, holistic similarity is well-defined in both of these cases.

Definition 7 can also be viewed in terms of conditional probability. Suppose one has randomly picked two objects, o_1 and o_2 from $\mathcal{O}$. Let $p(X)$ represent the probability that label L_1 applies to one of the objects and label L_2 applies to the other, given that L_A applies to both of them. Then:

$$\begin{aligned} p(X) &= p_i(L_1|L_A)p_i(L_2|L_A) \\ &= \frac{p_i(L_1 \cap L_A)p_i(L_2 \cap L_A)}{p_i(L_A)^2} \end{aligned}$$

Since L_A applies to any object to which L_1 applies, $p(L_1 \cap L_A) = p(L_1)$, and likewise $p(L_2 \cap L_A) = p(L_2)$. So:

$$\begin{aligned} p(X) &= \frac{p_i(L_1)p_i(L_2)}{p_i(L_A)^2} \\ &= \log^{-1}[\log p_i(L_1) + \log p_i(L_2) - 2\log p_i(L_A)] \\ I(X) &= -\log p(X) \end{aligned}$$

$$= \quad I(L_1) + I(L_2) - 2I(L_A)$$

When $I(X)$ is large, the situation is analogous to that of Figure 7.1(b): knowing that L_A applies to both objects tells one relatively little about whether the objects come from the populations labeled L_1 and L_2. As $I(X)$ approaches its minimum value, the situation becomes analogous to that of Figure 7.1(a): knowing that L_A applies to both objects is tantamount to knowing that they are drawn from the desired populations.

$I(X)$ ranges between 0 and $I(L_1) + I(L_2)$. If we normalize $I(X)$ to the range [0..1], we find that:

$$\begin{aligned}
\frac{I(X)}{I(L_1) + I(L_2)} &= \frac{I(L_1) + I(L_2) - 2I(L_A)}{I(L_1) + I(L_2)} \\
&= 1 - \frac{2I(L_A)}{I(L_1) + I(L_2)} \\
&= 1 - sim_H(o_1, o_2) \\
sim_H(o_1, o_2) &= 1 - \frac{I(X)}{I(L_1) + I(L_2)}
\end{aligned}$$

In other words, $I(X)$ represents the un-normalized distance between the labels of objects o_1 and o_2, and $sim_H(o_1, o_2)$ normalizes this value and converts it from a distance to a similarity.

7.1.2 Generic Similarity

In this section, we consider more carefully the use of the interior terms of a taxonomy in labels. We begin by examining the meaning of using such a term in a label.

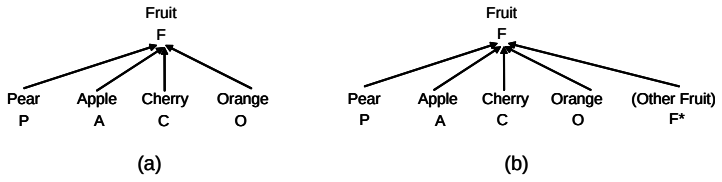


Figure 7.2: Use of Anonymous Terms

When an interior term is used in a label, there are two possible interpretations. The term may have been selected because no more specific term in the taxonomy applies to the object in question. For example, consider the taxonomy of Figure 7.2(a). If the object to be labeled is a mango, the interior term "fruit" is the most appropriate label, because none of the more-specific terms apply. On the other hand, the individual doing the labeling may choose an interior term because he or she does not know which (if any) more specific term applies. We refer to the first type of labeling as *careful* and to the second as *generic*. Both often occur in practice, in particular as a taxonomy evolves. Initially, a single term may be applied to what later turns out to be a whole group of distinct subclasses of objects. Over time, as these subclasses are recognized, new more-specific terms are created. However, objects classified under one version of the taxonomy are not necessarily reclassified whenever the taxonomy evolves.

To accommodate the careful use of interior terms as labels, we augment the taxonomy by adding a new descendant term, X^*, for every interior term X used as a label. We refer to X^* as an *anonymous term*, because it describes an unnamed subset of the objects to which the term X applies. In the taxonomy of Figure 7.2(b), an anonymous term F^* has been added to the taxonomy of Figure 7.2(a). An object

carefully labeled "fruit" will be considered an instance of F^* and all of its ancestors (including F). If an object can be labeled with multiple terms, it may also be necessary to introduce an anonymous term for combinations of terms that are used carefully. As far as our definition of similarity is concerned, anonymous terms behave exactly like real terms in T and we will not consider them further in this chapter.

Unlike careful labeling with interior terms, generic labeling forces us to rethink our basic understanding of similarity by introducing uncertainty into the labeling of objects. If a target object is labeled "fruit", and we are uncertain as to which specific kind of fruit it is, candidate objects labeled "apple", "pear", or 'fruit" all fulfill the only specific requirement posed by the labeling of the target object, that of being a fruit. However, if we apply Definition 7, only the candidate labeled "fruit" will receive a similarity score of 1 with respect to the target. In effect, sim_H penalizes objects labeled "apple" or "pear" for being "too specific" when the target object is generic. Note that the situation changes when the roles of target and candidate are reversed. If the target object is labeled "apple", a candidate object labeled "apple" is a better match than one labeled "pear" or "fruit".

To reflect the asymmetry introduced by generic labeling, we define a revised similarity measure, sim_G, such that $sim_G(T, C) = 1$ if and only if object C is *substitutable* for object T. This idea is familiar from the use of type hierarchies in programming languages, where an instance of a subtype is substitutable for an instance of a supertype, but not the other way around.

Definition 8 (Generic Holistic Similarity) *Let L_T and L_C be the labels of ob-*

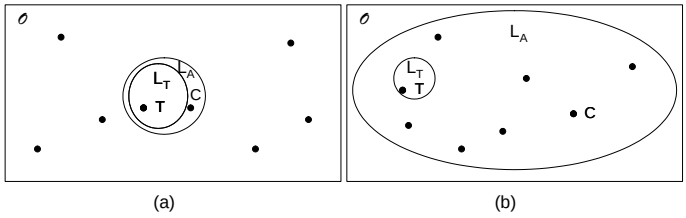


Figure 7.3: Generic Similarity in Context

jects T and C, respectively, and let $L_A = Lmin(Terms(Anc(L_T) \cap Anc(L_C)))$. Then:

$$sim_G(T, C) = \frac{-2 \log p_i(L_A)}{-\log p_i(L_T) - \log p_i(L_A)}$$

Alternatively, expressed in terms of the information content corresponding to each probability:

$$sim_G(T, C) = \frac{2I(L_A)}{I(L_T) + I(L_A)}$$

The generic similarity measure views the candidate object, C, as an instance of the most specific class of objects that includes both the target and the candidate. Figure 7.3 depicts a target object T compared to a candidate object C using sim_G. As in the case of sim_H, the larger the region associated with L_A relative to the size of $\mathcal{O}$, the less significant it is that the objects being compared are included in this set. Similarly, the smaller the size of region L_T relative to region L_A, the less significant it is that both objects are in L_A. As illustrated in Figure 7.3(a), maximum similarity (short of identity) occurs when L_A is a small fraction of $\mathcal{O}$ and most of the objects within it are labeled with L_T. Conversely, similarity is at a minimum when L_A contains most of $\mathcal{O}$, including many objects other than those

labeled with L_T, as in Figure 7.3(b).

Like sim_H, sim_G can also be viewed in terms of conditional probability. Suppose one has randomly picked two objects, o_1 and o_2 from $\mathcal{O}$. Let $p(X)$ represent the probability that label L_1 applies to o_1, given that L_A applies to both objects. Then:

$$\begin{aligned}
p(X) &= p_i(L_1|L_A) \\
&= \frac{p_i(L_1 \cap L_A)}{p_i(L_A)} \\
&= \frac{p_i(L_1)}{p_i(L_A)} \\
&= \log^{-1}[\log p_i(L_1) - \log p_i(L_A)] \\
&= \log^{-1}[\log p_i(L_1) + \log p_i(L_A) - 2\log p_i(L_A)] \\
I(X) &= -\log p(X) \\
&= I(L_1) + I(L_A) - 2I(L_A)
\end{aligned}$$

Normalizing $I(X)$ to the range [0..1], gives:

$$\begin{aligned}
\frac{I(X)}{I(L_1) + I(L_A)} &= \frac{I(L_1) + I(L_A) - 2I(L_A)}{I(L_1) + I(L_A)} \\
&= 1 - \frac{2I(L_A)}{I(L_1) + I(L_A)} \\
&= 1 - sim_G(o_1, o_2) \\
sim_G(o_1, o_2) &= 1 - \frac{I(X)}{I(L_1) + I(L_A)}
\end{aligned}$$

So, as in the case of sim_H, sim_G represents a normalized distance viewed as a similarity. In this case, the distance is between L_1, the label of o_1, and L_A, the label representing what o_1 and o_2 have in common.

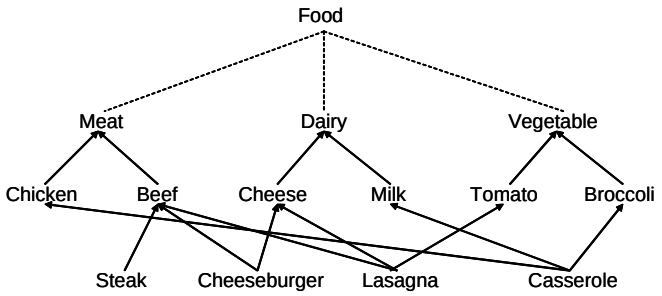


Figure 7.4: Labeling With Generic Terms

The use of generic labeling also reinforces our choice of an open-world model, because when the taxonomy is structured as a DAG, the closed-world model is incompatible with the use of generic terms as labels. To see why, consider the taxonomy of Figure 7.4. If an object is labeled with the generic term $\{Beef\}$, under generic labeling we interpret that to mean the labeler is uncertain as to which of the terms describing a more specific type of beef applies. The object could be a steak, which is "just" beef, or it could be a cheeseburger. Because a cheeseburger contains additional ingredients (i.e. cheese), the ancestor graph of the label $\{Cheeseburger\}$ contains terms like $\{Cheese\}$ that are neither descendants nor ancestors of the term $\{Beef\}$, and these terms may also apply to the object in question. Thus, labeling a cheeseburger $\{Beef\}$ would violate the closed-world assumption, which states that terms not in the ancestor graph of $\{Beef\}$ do not apply to the object.

We conclude this section with a set of examples that demonstrates how sim_G orders candidate objects with respect to a fixed target. The examples are based on the taxonomy of Figure 7.4, and are intended to be illustrative rather than to provide an exhaustive case analysis.

C	L_C	L_A	$sim_G(T, C)$
c_1	$\{Cheeseburger\}$	$\{Beef, Cheese\}$	$\frac{2I(Beef,Cheese)}{I(Beef,Cheese)+I(Beef,Cheese)} = 1$
c_2	$\{Beef, Cheese, Tomato\}$	$\{Beef, Cheese\}$	$\frac{2I(Beef,Cheese)}{I(Beef,Cheese)+I(Beef,Cheese)} = 1$
c_3	$\{Steak\}$	$\{Beef\}$	$\frac{2I(Beef)}{I(Beef,Cheese)+I(Beef)}$
c_4	$\{Steak, Milk\}$	$\{Beef, Dairy\}$	$\frac{2I(Beef,Dairy)}{I(Beef,Cheese)+I(Beef,Dairy)}$
c_5	$\{Casserole\}$	$\{Meat, Dairy\}$	$\frac{2I(Meat,Dairy)}{I(Beef,Cheese)+I(Meat,Dairy)}$

Table 7.1: Similarity to Target Labeled $\{Beef, Cheese\}$

Table 7.1 shows the similarity of five candidate objects to a common target object T labeled $\{Beef, Cheese\}$. The first candidate, labeled $\{Cheeseburger\}$, is a specialization of the target object and therefore substitutable for it. It receives a similarity score of 1 because its ancestor graph includes the complete ancestor graph of the target. The same is true of c_2, the candidate labeled $\{Beef, Cheese, Tomato\}$.

The ancestor graph of c_3, labeled $\{Steak\}$, includes the term "Beef" and all its ancestors, but only intersects with the ancestors of "Cheese" at the root of the taxonomy ("Food"). L_A is therefore $\{Beef\}$, the minimal label of the intersection of the target and candidate ancestor graphs. Since $I(Beef)$ is calculated using inclusion probability, it reflects not only the number of objects labeled $\{Beef\}$, but also the populations of objects whose labels are descendants of $\{Beef\}$ (e.g. $\{Steak\}$, $\{Lasagna\}$) and those that include the term "Beef" among others in their label (e.g. $\{Beef, Broccoli\}$). c_4, the candidate labeled $\{Steak, Milk\}$, is similar to c_3, but in this case $L_A = \{Beef, Dairy\}$. The size of the population associated with

this label is smaller than the one associated with $\{Beef\}$, since it includes only those objects whose label includes the term "Dairy" (or one of its descendants) as well as the term "Beef" (or one of its descendants). Hence the information content of this label is higher, and c_4 will receive a higher similarity score than c_3. For c_5, labeled $\{Casserole\}$, $L_A = \{Meat, Dairy\}$. Because inclusion probability associates more objects with this label than with the label $\{Beef, Dairy\}$, c_5 will receive a lower similarity score than c_4.

7.2 Related Work

We have introduced in Chapter 3 three groups of similarity measures which can be applied to taxonomies: the *term-similarity* measures, the *common-term* measures, the measures that *combine* the similarity scores between individual pairs of terms. In this section, we briefly review some of them, which are evaluated and compared with our measures in the experimental section.

The simplest measure we studied was **Common Term Count**, which ranks candidates based solely on the number of terms they have in common with the target. We also evaluated the **Jaccard** measure, a normalized form of **Common Term Count** that is commonly used to measure similarity in IR tasks. If L_T and L_C represent the labels of the target and candidate objects respectively:

$$sim_{Jaccard}(T, C) = \frac{|L_T \cap L_C|}{|L_T \cup L_C|}$$

Another two analogous measures that are also based on common terms are **Common Term IC** and its normalized form. For these two measures, we weight

each common term by its information content. For the measure of **Common Term IC**, we sum the information content of the common terms to determine similarity:

$$sim_{CTIC}(T, C) = \sum_{a \in L_T \cap L_C} I(a)$$

We then normalize this measure by comparing the common information content to the total information content of all the terms in the two labels:

$$sim_{Norm-CTIC}(T, C) = \frac{2 * \sum_{a \in L_T \cap L_C} I(a)}{\sum_{t \in L_T} I(t) + \sum_{c \in L_C} I(c)}$$

We also considered the measure proposed by Wang et al.[109] and the one proposed by Halkidi et al.[45]. These two measures are not based solely on common terms, but also count as similar pairs of terms that are close in the taxonomy. They calculate the pairwise similarity of individual terms from the target and candidate labels, and combine these to produce an overall score.

As mentioned in Chapter 3, Wang et al.'s measure uses a generalized form of Lin's similarity measure to determine the similarity of each term-pair. If t and c are the target and candidate terms:

$$sim_{Lin*}(t, c) = \frac{2 * \max_{t_a \in Terms(Anc(c) \cap Anc(t))}(I(t_a))}{I(t) + I(c)}$$

Note that this is not the same as applying our formulation in Definition 7 to a pair of single terms, and does not follow directly from Lin's axiomatic definition of similarity in Equation 7.1 because it considers only a portion of the commonality between the terms (selecting the least common ancestor with the maximum information content). Given this definition of term similarity, Wang et al.'s measure combines the pairwise

similarities by averaging them across all pairs of terms in the two labels:

$$sim_{Wang}(T, C) = \frac{1}{|L_T||L_C|} \sum_{t \in L_T, c \in L_C} sim_{Lin*}(t, c)$$

Halkidi et al. used the similarity measure defined by Wu and Palmer[114] to compute the similarity of term pairs, but it is unclear how to generalize this to a taxonomy structured as a DAG. Hence we instead used the same generalized form of Lin's measure as Wang et al. in order to evaluate Halkidi et al.'s measure:

$$sim_{Halkidi-IC}(T, C) = \frac{1}{2}\left[\frac{1}{|L_T|}\sum_{t \in L_T} \max_{c \in L_C}(sim_{Lin*}(t, c)) + \frac{1}{|L_C|}\sum_{c \in L_C} \max_{t \in L_T}(sim_{Lin*}(c, t))\right]$$

Note that Halkidi et al.'s measure considers each term individually, and finds the most similar term from the other set. Then, the similarity of these best matches is averaged over the terms in each set. Finally, the average similarity from the two sets is combined with equal weight.

It is important to note that our holistic measures sim_H and sim_G do not belong to any of these groups. Unlike the term similarity measures, they can be used when labels contain multiple terms. Unlike the common term measures, they take distinct but similar terms into account. Unlike the pairwise measures, they do not consider individual terms, but rather take all the terms in each label into account simultaneously.

In Section 7.4, we describe how we evaluated our similarity measures using the taxonomy and associations defined by the Gene Ontology[7]. Other authors

have also used taxonomic similarity measures over the Gene Ontology to address the problem of finding gene products similar to a specified target. For example, Lord et al.[69] evaluate the Resnik and Lin measures, and a distance measure due to Jiang and Conrath[58], by determining how well their similarity scores correlate with similarities discovered by comparing the genetic sequences of the target and candidate objects. Similarly, Wang et al.[109] evaluate their measure by determining how well it correlates with functional properties derived from gene expression data.

7.3 Implementation

To accommodate both large taxonomies and large corpora of objects, we built our implementation in the context of an object-relational database management system, specifically IBM DB2 Universal Database V8.2.[2] We believe this approach offers a number of advantages. In the first place, storing the corpus in a database allows the full power of SQL to be used to select those objects of interest in a particular situation. For example, a user searching for similar objects in a large database of gene products may wish to restrict the search to human gene products. Secondly, the ability to extend the database management system with user-defined types and functions allowed us to implement certain critical operations very efficiently within the database, without requiring large amounts of data to be retrieved for manipulation by an application.

In our implementation, each similarity measure is implemented as an SQL

[2]http://www.ibm.com/software/data/db2/udb/

query against a set of relations with a fixed schema. The primary purpose of these relations is to present, in a standard format, the taxonomy itself and the associations between terms from the taxonomy and objects in the corpus. In our database, the taxonomy is represented as a table of (parent term id, child term id) pairs; the associations are represented as (term id, object id) pairs. Since the associations for the target and candidate objects may in general come from different sources, we use separate relations to represent them. These relations may be base tables, but are more likely views created over the native representation of the associations.

Once the taxonomy and the associations have been specified, additional tables are defined to store information that can be precomputed once and used repeatedly in subsequent evaluations of the similarity measure. We will provide more information on this auxiliary data in Section 7.3.2.

7.3.1 User-Defined Types

Within each query, certain critical operations are implemented as User Defined Functions (UDF's) that operate on User Defined Types (UDTs) represented in the database as Binary Large Objects (BLOBs). Two types of operations warranted such special treatment.

Firstly, the taxonomy is naturally represented as a directed graph, and a key step in the evaluation of sim_H, sim_G, and various other measures involves the determination of common ancestors between the labels of target and candidate objects. Therefore, we have made extensive use of a general-purpose graph library for DB2.

The library allows graphs to be constructed efficiently from database data using a user-defined aggregate function. Once constructed, they can be stored in the database as BLOBs and manipulated by UDFs that implement a wide range of graph-theoretic operations, e.g. finding the in-neighbor region of a node, intersecting graphs, etc. Table functions are provided that return the edges or nodes in a graph, along with payload values and various properties, e.g. the incoming edge count for a node. A full discussion of the graph library, which scales to very large graphs, is beyond the scope of this chapter.

A second critical step in the evaluation of sim_H and sim_G is the determination of the inclusion probabilities of particular labels. To determine the inclusion probability of a label L, one must know its frequency, that is, the number of objects in the corpus to which all of the terms in L apply. Recall that a term t applies to an object o if and only if the ancestor graph of the object's label includes the term in question, i.e. if $t \in Terms(Anc(o))$.

In principle, one could precompute frequencies for each of the $2^{|N|}$ combinations of terms that can be used as a label. For taxonomies of realistic size, however, this approach is impractical. Instead, we build an inverted list for each term, identifying the objects to which the term applies, i.e. those objects whose labels contain the term or any of its descendants. Let $O(t)$ denote the list of objects for term t. The frequency of a label can then be determined by finding the size of the intersection of the inverted lists of its individual terms. The inclusion probability is therefore:

$$p_i(L) = \frac{|\cap_{t \in L} O(t)|}{|\mathcal{O}|}$$

Like the taxonomy graph, the inverted lists are implemented as a User Defined Type optimized to support the operations needed to compute label frequency: intersection and length. The inverted list UDT stores a list of object identifiers as a simple vector. Identifiers can be inserted in any order as the list is built (using a user-defined aggregate function), and the list is sorted once when insertion is complete. The intersection of two lists can be computed with a single pass through both lists, and a user-defined aggregate function is provided to find the intersection of a set of lists.

7.3.2 Precomputation

In principle, all the information needed to compute sim_H or sim_G can be derived dynamically from the relations that define the taxonomy and the association corpus. However, certain information used to find and rank candidate objects can be used repeatedly for different target objects, as long as neither the taxonomy nor the corpus changes. In this section we consider several situations in which we chose to precompute such values.

Whenever a target and candidate object are compared, we need to find the intersection of their respective ancestor graphs. Furthermore, at least in our experiments, the same candidates are evaluated for many different targets. We therefore precompute the ancestor graph for the label of each object in the corpus of candidate objects. The graph library can generate ancestor graphs quite quickly, so precomputation is practical even for corpora of large size. The space requirement is

modest, because the ancestor graph for an object is typically a small fraction of the entire taxonomy. Details for our experimental scenario can be found in Section 7.4. Updates to the candidate corpus can be handled incrementally, but updates to the taxonomy may require a complete recomputation of these graphs.

Note that we only precompute the ancestor graphs for candidate object labels, not target object labels, because each target object is generally only referenced once. However, we also precompute the ancestor graph for each individual term in the taxonomy. These graphs make the dynamic computation of ancestor graphs for target object labels more efficient, and are reused many times since many targets refer to the same terms. They consume much less space than the candidate label ancestor graphs, because there are far fewer terms than labels, and each graph is smaller. The term ancestor graphs are not affected by updates to the corpus, but may need recomputation when the taxonomy changes.

We also precompute the inverted lists. For each term in the taxonomy, we build a list containing the identifiers of all objects in the corpus that contain the term, or a descendant of the term, in their label. This is for the purpose of finding the inclusion probability of an arbitrary label by intersecting the lists corresponding to its terms. Note that the size of these lists is proportional to the size of the corpus. The lists can be updated incrementally as objects are added to the corpus, but may need to be recomputed if objects are deleted or the taxonomy changes.

7.4 Experimental Evaluation

Similarity measures are difficult to evaluate, since similarity of objects is normally subjective in nature and individuals may differ in how they rank candidate objects with respect to a particular target. Other authors have tried to work around this problem by comparing their rankings to those of a panel of human subjects (e.g., [94, 66]) or to an independent measure of similarity appropriate to their domain (e.g., [69, 109, 60]).

We took a different approach. In search of a result that could be more objectively quantified, we applied our similarity measures to the more difficult problem of matching. Suppose that two sources of associations are using the same taxonomy to assign labels to objects. Given the label of a target object as supplied by one source, the task is to find the same object in a corpus of labels supplied by another source. In our experiments, the two sources use a common identifier for objects, so it is easy to verify that a match has been found.

Specifically, we use the Gene Ontology (GO)[7] as the taxonomy in our experiments. This taxonomy contains about 17000 terms, and can be represented by a graph with as many nodes and about 22000 edges. The terms are divided into three independent facets, representing a gene product's cellular location(s), molecular function(s) and the biological process(es) in which it participates. Each facet is a directed acyclic graph, and in our experiments, we ignored the facets and considered all terms as a single label. The relationships between terms are characterized as either *is-a* or *part-of*. We also ignored this distinction, since both relationships

possess the transitive behavior that underlies our understanding of classification. Finally, a small number of terms in the taxonomy are considered obsolete, and have been gathered together as children of a special node. We removed these nodes from the taxonomy, since their location in the taxonomy does not reflect their semantics.

Various organizations have used the GO taxonomy to annotate a large number of gene products, and the taxonomy and annotations can be downloaded from the GO Consortium.[3] Over one million gene products have been annotated with over 4 million term associations, but we restrict our attention to human gene products registered in the SwissProt databank, which number about 28000. Almost all of these gene products have annotations supplied by the UniProt Knowledgebase[5], and we use these gene products as our candidate objects. There are approximately 110000 annotations for these objects, and these annotations constitute our corpus of term associations. We removed only those annotations that associate a gene product with an obsolete term.

Of the 28000 gene products, 5789 also have annotations supplied by Proteome, Inc.[4] These gene products constitute our target objects, and the corresponding set of about 20000 annotations becomes our target associations. Therefore our matching task is: given the annotations for a gene product supplied by Proteome Inc., find the same gene product using the annotations supplied by UniProt.

We note that this is inherently a very difficult problem. The two organizations that generated the annotations we use for matching are completely independent.

[3]http://www.geneontology.org

[4]http://www.proteome.com

They did not necessarily have the same goals or use the same guidelines in assigning terms to gene products, nor did they necessarily work from the same source information. Furthermore, the space to search is large (28000 objects), and often contains many objects similar to the desired target.

To apply a similarity measure to this problem, we rank the candidate objects using the similarity measure, and then determine whether the matching object is found in the top K objects as ranked by the measure. We also keep track of the depth in the candidate list at which the matching object was found.

In practice, it is too expensive to compute the similarity score of a target with respect to all 28000 objects in the corpus. Hence we applied the following heuristic to limit the candidate set: we considered as candidates only those objects whose labels have at least one term in common with the label of the target object. We chose this heuristic because it could be applied to all the measures we wished to evaluate, some of which are based entirely on common terms and their properties. However, it should be noted that for many target objects, the labels supplied by the two organizations have no terms in common. Thus, regardless of the measure used, this heuristic limits the number of targets that can be successfully matched to a maximum of 3660 of 5789, or 63.2%. We considered other heuristics for determining the candidate set; see Section 7.4.3 for discussion.

7.4.1 Results

We tested sim_H, sim_G and the six measures described in Section 7.2 on targets from the set of 5789 gene products annotated by both UniProt and Proteome, Inc. Using the candidate-selection heuristic described above, we calculated the similarity score for each candidate with respect to the target. If the measure ranked the object corresponding to the target among the top 100 candidates, we considered it to have found a match. While this may seem like a generous definition of success, we note again the inherent difficulty of the matching problem.

We calculated the success rate for each measure, along with the depth at which the matching object was found, averaged over the successful matches. The results are presented in Table 7.2.[5]

Recall that the maximum possible success rate using the common-term heuristic for candidate selection is 63.2%.

We do not compare the measures on the basis of speed, since we made a concerted effort to reduce costs only for sim_H and sim_G. However, we note that our strategy of precomputation enabled us to test a single candidate in about 4.4ms using sim_G and about 6.4ms using sim_H. These values are for a 2.4GHz Pentium 4 with 1Gb of memory, running Windows XP and DB2 UDB v.8.2. The space requirements for the precomputed object ancestor graphs, term ancestor graphs and term inverted lists were modest: 122Mb, 40Mb and .5Mb, respectively.

[5]For one measure where we did not test the entire set of targets, a confidence interval is provided for the success rate.

Method	Success Rate (%)	Average Depth
sim_H	39.3	23.9
Common Term IC	39.1	22.2
sim_G	37.5	25.2
Normalized CT IC	36.9	25.4
Halkidi IC	36.1	26.2
Common Term Cnt.	35.2	28.4
Jaccard	30.9	28.7
Wang	24.5 ($\pm$ 2.42@95%)	35.9

Table 7.2: Comparison of Similarity Measures

7.4.2 Discussion

Although the differences in the success rates are not striking, some observations can be made. Firstly, the measures based solely on common terms did not adequately take into account pairs of terms that were closely related, but not identical. Because such pairs do not contribute at all to the similarity score, common-term measures that penalize candidates for terms that are not shared with the target understate similarity to a greater extent than those which do not. Thus, both **Common Term Count** and **Common Term IC**, which ignore unmatched terms, outperformed their normalized counterparts, **Jaccard** and **Normalized Common Term IC**, which incur a penalty for unmatched terms. For the holistic measures, all terms contribute to the similarity score, and indeed sim_H outperforms the other measures. However, our candidate-selection heuristic requires each candidate to have at least one term in common with the target. While this heuristic provides a level playing

field for comparing the various measures, it limits the beneficial effect of taking related (but not common) terms into account. We explore the effects of relaxing this restriction in Section 7.4.3.

We also observe that measures that weight the importance of pairs of common terms based on their information content performed better than the analogous measures based on counting common terms. In particular, **Common Term IC** outperformed **Common Term Count**, and **Normalized Common Term IC** outperformed **Jaccard**.

The differences among the measures based on information content stem from differences in how similarity is derived from information content in each case. Both **Wang** and **Halkidi IC** normalize the common information content of each term-pair independently, and then combine these normalized values to reach an overall similarity value. As a result, term-pairs with relatively low common information content are given the same weight as pairs with much greater common information content. This is particularly so in the case of **Wang**, which pairs each term with every other, as opposed to **Halkidi IC**, which just pairs each term with its best match. However both measures suffer in comparison to those based on common term information content, which weight each common term in accordance with its information content. Another source of error in these measures is correlation. If a label contains two terms whose occurrence is correlated, these measures overestimate their information content when they occur together.

The holistic measures sim_H and sim_G avoid both these problems by calculating the combined information content of all the terms in each relevant set. Although we

believe that generic labeling with interior terms occurs pervasively in our corpus, we note that sim_H outperformed sim_G in our experiments. We believe this reflects our choice of a matching problem rather than one based strictly on similarity. The difference between the measures is that sim_G does not penalize a candidate for being labeled with terms that are more specific than those used to label the target. This may elevate the score of the matching candidate enough to make it competitive with others labeled with more general terms. However, it also elevates the scores of candidates that use more specific terms than the matching candidate and its competitors, which has the opposite effect. If one views the terms associated with the target as requirements, these additional candidates satisfy the requirements as well as the matching candidate does, and therefore they are equivalently similar. But in a matching problem, we are looking for a particular object, and the presence of the others makes finding it more difficult.

7.4.3 Candidate Selection

As we mentioned previously, all the measures we evaluated were limited by our candidate-selection heuristic. For the measures entirely based on common terms, this limitation does not hurt, since objects that have no terms in common with the target have a similarity score of zero. However, the other measures we evaluated have the potential to find additional matches if we consider additional candidates.

We tested this hypothesis with an alternative candidate-selection heuristic. For each association between a target object and a term, we augmented the set of

KNN	Patent ID	Score	Title
0	14019	1.0	'Apparatus and method for collecting flue gas particulate with high permeability filter bags'
0	47255	0.9673537	'Advanced hybrid particulate collector and method of operation'
0	195208	0.9673537	'Volatile materials treatment system'
0	265087	0.9673537	'Char for contaminant removal in resource recovery unit'
2	304641	0.9673537	'System and method for removing gas from a stream of a mixture of gas and particulate solids'
0	344644	0.9673537	'Electric dust collector and incinerator'
1	21467	0.9441908	'Thief process for the removal of mercury from flue gas'
0	25179	0.9441908	'Multi-stage particulate matter collector'
1	473644	0.9441908	'Method of regulating the flue gas temperature and voltage supply in an electrostatic precipitator...'
0	92698	0.92622423	'Electrically enhanced electrostatic precipitator with grounded stainless steel collector electrode...'
0	354751	0.92622423	'Device and method for filtering internal combustion engine exhaust gases and vehicle equipped'...'

Table 7.3: Patent Similarity Search

terms associated with the target by adding associations between the target object and the original term's immediate neighbors. We then used as candidates all objects that have at least one term in common with this expanded set. We refer to this heuristic as *KNN-1*, because the candidate set comprises those objects that are described by at least one of the original terms or a neighboring term one hop away. The original common term heuristic could be described as KNN-0; KNN-2 and other values could be considered as well.

When we tested sim_H with KNN-1 on the matching problem, we found that it raised the success rate from 39.3% to 40.5%. Although not a dramatic increase, this suggests that our measures can find similar objects even when they have no terms in common with the target. However, the time required to match a target is proportional to the number of candidates considered, and increasing the value

of N increased the number of candidates to test significantly. For KNN-0 on our dataset, the average number of candidates per target was 1429. For KNN-1 this rose to 3705, causing a greater than twofold increase in the average time required to match a target. For KNN-2, the average number of candidates rose to 6824.

We also tested the KNN candidate-selection heuristic on a second dataset. The United States Patent and Trademark Office (USPTO) maintains a classification scheme for patents based on a tree-structured taxonomy of about 160000 terms, referred to as classes and subclasses.[6] Individual patents are labeled with one or more terms from the taxonomy. We tested sim_H and sim_G on a corpus of approximately 500000 patents filed between 2001 and 2003, as classified by about 1.9 million associations between patents and terms.

While we could not perform an objective matching experiment with this dataset, it nevertheless provided a number of interesting opportunities for evaluating our measures and their implementation. Firstly, we were able to verify that our implementation could scale to a much larger corpus and a much larger taxonomy with satisfactory performance. The USPTO taxonomy also has a different structure than the GO taxonomy; it is broader and flatter, and does not have facets or multiple inheritance. Finally, our candidate-selection heuristics generated fewer candidates per target than in the matching experiment. We were thus able to more easily observe the value of the KNN heuristic, coupled with our measures' ability to detect similarity without the presence of common terms between the candidate and target objects.

[6] http://www.uspto.gov/web/patents/classification/

Table 7.3 shows the combined result of three similarity searches for a typical target, with KNN values of 0, 1 and 2. The first patent listed, ID 14019, is the target, and thus had a perfect similarity score of 1.0. Below, similar patents are listed with their similarity scores, as well as the KNN value of the search in which they were initially found. For example, patent 304641 was found by the search that used KNN-2 to select candidates, and scored as highly as any of the patents found with smaller KNN values. Similarly, two other highly-scored patents were found with KNN-1 that would not have been found with KNN-0, nor by any of the measures that rely solely on common terms.

7.5 Summary

Similarity ranking of objects labeled using a taxonomy is an interesting problem with a variety of useful applications. Our work has made several contributions to the state of the art. Starting from Lin's axiomatic definition of similarity, we developed new similarity measures that are applicable to real classification systems, in which the taxonomy can be structured as a DAG, objects can be labeled with multiple terms, internal terms can be used in labels, and different users may label the same object in different ways.

We implemented our measures using SQL and a pair of libraries for specialized data structures, realized as user-defined types. The result is a flexible, scalable implementation that is tightly integrated with a database management system and achieves good performance through strategic precomputation of key data structures.

We evaluated our measures on an object-matching task using the Gene Ontology, a taxonomy with all the properties noted above. Our experiments show that Our measures were more successful at matching objects than those reported in the literature. We also tested our measures on a search task, using the patent classification taxonomy of the USPTO to find patents similar to a specified target. We evaluated two heuristics for candidate selection, a critical issue for large data sets.

Chapter 8

Future Work

In this chapter, we present future directions for improving or extending our work on ontologies and ontology-based query answering.

8.1 Improving Performance of Ontology-Extended Systems

It is true that on average, existing ontologies in various domains have a moderate size (usually of the order of hundreds of terms) [102]. For example, most ontologies at www.daml.org have less than 500 terms. However, recently, in some application domains, we often encounter ontologies of huge size. For example, the Gene Ontology has 17000 nodes and the taxonomy of the United States Patent and Trademark Office has 160000 nodes.

On the other hand, in previous chapters, we have presented experimental results to demonstrate that our algorithms can scale well to handle certain amount of data. However, most of the operations are still memory-based, e.g., query processing and ontology integration. It is not efficient to frequently run graph-based operations on a large ontology in memory. Hence, it would be natural to have some index structures to take those operations on large ontologies into account. This ontology-based indexing mechanism will help our system respond to user's queries in much shorter time. Furthermore, with such index structures at hand, it would be

feasible to design algorithms in support of disk-based integration for large ontologies.

8.2 Learning Ontologies and Interoperation Constraints

In Chapter 4 and Chapter 6, we show how to integrate graph-based and RDF ontologies respectively. These efforts also present a challenge in finding interoperation constraints. In our HOME (Chapter 4) and TOSS (Chapter 5) systems, there is a component called *Ontology Maker*. It automatically takes relational or XML data as input and uses WordNet [77] and user-specified rules to generate ontologies. Inside *Ontology Maker*, there is a subcomponent called *Rule Maker*, which helps users to specify the interoperation constraints between ontologies. However, it is not reasonable to assume that end-users can always provide interoperation constraints. In our probabilistic ontologies project [102], we have proposed an algorithm to infer simple equality constraints. This part of our work is yet far from completion. Learning ontologies and interoperation constraints are still open problems [72, 33].

Notice that there has been a strong interest in recent years to learn ontologies from multimedia data, such as audio and video data. Such ontologies can be used to convey semantics in audio and video and to answer semantic queries over the multimedia data.

On the other hand, we often need to incrementally maintain the interoperation constraints when changes are made to the ontologies. In [32], we present methodologies for maintaining mappings (constraints) between evolving schemas by using the rich constructs in OWL. This area is still relatively new, however, and more

efforts are needed as well as collaboration with other areas (e.g., machine learning and natural language processing).

8.3 Semantic Similarity Join

As we mentioned in Chapter 3, it is often the case that the same entity may have different representations (identifications) in heterogeneous data sources. In previous approaches for integrating such data sources, people usually assume that objects from different data sources can be mapped into a globe domain such that matching those objects is easy. However, in practice, such a common domain is not easy to construct due to the subtle difference on the semantics of the objects as well as various understandings to the semantics of even a single object. Hence, several research groups have investigated various methods of approximate joins between string-valued or text-valued columns in relational databases.

Cohen [27] proposes a system called WHIRL to reason about the similarity of textual values in databases. The similarity is measured using the vector-space model based on the intuition that two documents are more similar if they share more pairs of terms with high weights. The system relies on an A* search to return the highest-scoring answers efficiently. Gravano et al. [39] present a filtering technique for computing approximate string joins based on the notion of edit distance. Their core idea is to match the q-grams (substrings of length q) of the two strings to be compared. They have identified three important q-gram properties to filter the candidate set of strings for join. Later, Gravano et al. [38] propose a sampling-based

technique for approximate text join by adopting the cosine similarity metric over the vector-space model. To join relation R_1 and R_2, they extract a sample set of tuples from relation R_2 for each tuple t in R_1. This sample set contains tuples highly similar to t. For each tuple t' in R_2, whether t' should be put into the sample set for tuple t in R_1 depends on a probability function approximating the similarity between t and t'.

However, none of these decent methods considers the semantic similarity functions based on taxonomies (ontologies). On the other hand, these approaches are not applicable for joining two sets of ontological terms since they are based on edit distance or the cosine similarity metric. Hence, it would be interesting to investigate new join methods that take ontology-based similarity functions into account.

8.4 Semantic Web Databases

Recently, there has been a large amount of web data and services available in RDF. Several research teams have been working on the formal aspects of the RDF data model, the RDF model's extension to represent additional information (e.g., temporal and context-dependent data), and query processing on RDF data.

Many query languages for RDF data have been proposed, e.g., RQL [59], RDQL [93], and SPARQL [89]. Considering that RDF data on the web is subject to frequent change, several research groups have been working on RDF views and the maintenance. For example, Voltz et al [108] introduced an RDF view mechanism which requires that (i) the results contain class instances (i.e., a subject or

object variable), or (ii) the result itself has the pattern of RDF statement (i.e., a triple containing subject, predicate and object). Later, Hung et al [54] developed sophisticated view maintenance algorithms that take the graph structure of RDF data into account in order to answer queries very efficiently. They also proposed an efficient algorithm to answer aggregate queries over RDF databases and a set of algorithms to maintain views involving those aggregates.

The research efforts on RDF databases have also motivated the study on formal aspects of RDF data model and query processing. Gutiérrez et al[44] have studied the features of RDF data model (such as reification, typing and inheritance) as well as the foundational aspects of RDF query languages (such as complexity of query processing and query containment). They defined a formal query language for RDF and investigated the theoretical aspects with regard to this language.

More recently, there has been a surge of interest on extending RDF data model to represent temporal and context-dependent information. For example, Gutierrez et al [43] present a framework to incorporate temporal reasoning into RDF. They present both a syntax and a semantics for temporal RDF graphs. They also show the complexity analysis for the entailment in temporal RDF graphs. Udrea et al [104] presented a unified framework, annotated RDF, to support all forms of reasoning in RDF, such as temporal reasoning and uncertainty reasoning. In this framework, RDF triples are annotated by members of a partially ordered set. They also presented a formal semantics for annotated RDF and proposed algorithms for consistency checking and query answering in this extend RDF data model.

Nevertheless, there is still a large set of open issues in this area to explore, for

instance, how to represent temporal, spatial and uncertainty information in OWL data model; how to query on these extended OWL data models; how to learn such models from data etc. OWL has more complicated features and constructs than RDF which has made the extension of OWL data model very interesting and challenging.

8.5 OWL Integration

In Chapter 4 and Chapter 6, we show how to integrate graph ontologies and RDF ontologies respectively. It would be interesting to continue this journey on OWL ontologies as well. Yet, integrating OWL ontologies is a more challenging task due to the more complex features and constructs in the model. There are a range of problems to be considered in this task, e.g., consistency checking, how to eliminate inconsistency from an integrated ontology, how to incorporate open-world assumption in the integration, etc.

BIBLIOGRAPHY

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] S. Agrawal, S. Chaudhuri, and G. Das. DBXplorer: A system for keyword-based search over relational databases. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, Washington, DC, USA, 2002.

[3] S. Al-Khalifa, C. Yu, and H. V. Jagadish. Querying structured text in an XML database. In *Proc. ACM SIGMOD Conf. on Management of Data*, San Diego, CA, 2003.

[4] S. Amer-Yahia, C. Botev, and J. Shanmugasundaram. TeXQuery: A full-text search extension to XQuery. In *WWW2004*, pages 583–594, New York, USA, 2004.

[5] R. Apweiler, A. Bairoch, C. Wu, W. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, and M. Magrane. UniProt: the universal protein knowledgebase. *Nucleic Acids Res.*, 32(1):D115–119, Jan. 2004.

[6] Y. Arens, C. Y. Chee, C. N. Hsu, and C. A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal of Cooperative Information Systems*, 2(2):127–158, 1993.

[7] M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig, M. A. Harris, D. P.

Hill, L. Issel-Tarver, A. Kasarskis, S. Lewis, J. C. Matese, J. E. Richardson, M. Ringwald, G. M. Rubin, and G. Sherlock. Gene Ontology: Tool for the unification of biology. *Nature Genetics*, 25(1):25–29, May 2000.

[8] F. Baader, I. Horrocks, and U. Sattler. Description logics as ontology languages for the Semantic Web. *Lecture Notes in Artificial Intelligence*, 2003.

[9] A. Balmin, V. Hristidis, and Y. Papakonstantinou. ObjectRank: Authority-based keyword search in databases. In *Proceedings of the 30th VLDB Conference*, Toronto, Canada, 2004.

[10] C. Baral, S. Kraus, and J. Minker. Combining multiple knowledge bases. *IEEE Transaction on Knowledge and Data Engineering*, 3(2):208–220, 1991.

[11] C. Baral, S. Kraus, J. Minker, and V. S. Subrahmanian. Combining knowledge bases consisting of first order theories. *Computational Intelligence*, 8(1):45–71, 1992.

[12] C. Baral, S. Kraus, J. Minker, and V. S. Subrahmanian. Combining default logic databases. *International Journal on Intelligent Cooperative Information Systems*, 3(3):319–348, 1994.

[13] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL Web ontology language reference. W3C recommendation. `http://www.w3.org/TR/owl-ref/`, 10 February 2004.

[14] S. Benferhat and S. Kaci. Logical representation and fusion of prioritized information based on guaranteed possibility measures: application to the

distance-based merging of classical bases. *Artificial Intelligence*, 148(1-2):291–333, 2003.

[15] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

[16] I. Bhattacharya and L. Getoor. Relational clustering for multi-type entity resolution. In *The 11th ACM SIGKDD Workshop on Multi Relational Data Mining (MRDM-05)*, 2005.

[17] I. Bhattacharya and L. Getoor. A latent dirichlet model for unsupervised entity resolution. In *The 6th SIAM Conference on Data Mining (SIAM SDM-06)*, 2006.

[18] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In *The 9th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, Paris, France, June 2004.

[19] P. Bonatti, Y. Deng, and V. S. Subrahmanian. An ontology-extended relational algebra. In *Proceedings of the IEEE International Conference on Information Reuse and Integration (IEEE IRI 2003)*, 2003.

[20] P. Bonatti, M. L. Sapino, and V. S. Subrahmanian. Merging heterogeneous security orderings. *Journal of Computer Security*, 5(1):3–29, 1997.

[21] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing ontologies. In *International Semantic Web Conference*, pages 164–179, 2003.

[22] D. Brickley and R. Guha. RDF vocabulary description language 1.0: RDF Schema. W3C recommendation. `http://www.w3.org/TR/rdf-schema/`, 10 February 2004.

[23] S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. In *Proceedings of the 7th WWW Conference*, Brisbane, Australia, 1998.

[24] D. Calvanese, G. D. Giacomo, and M. Lenzerini. A framework for ontology integration. In *Proc. of the First Semantic Web Working Symposium*, pages 303–316, 2001.

[25] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Description logics: Foundations for class-based knowledge representation. In *Proc. of the 17th IEEE Sym. on Logic in Computer Science (LICS 2002)*, pages 359–370, 2002.

[26] L. Cholvy and C. Garion. Answering queries addressed to several databases according to a majority merging approach. *J. Intell. Inf. Syst.*, 22(2):175–201, 2004.

[27] W. W. Cohen. Integration of heterogeneous databases without common domains using queries based on textual similarity. In *Proc. ACM SIGMOD*, pages 201–212, Seattle, WA, 1998.

[28] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string metrics for matching names and records. In *Proc. of the First Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, 2003.

[29] D. Connolly, F. van Harmelen, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. DAML+OIL (March 2001) reference description. W3C note. `http://www.w3.org/TR/daml+oil-reference`, 18 December 2001.

[30] http://www.ibm.com/software/data/db2/extenders/textinformation/index.html, 2001.

[31] DBLP XML records. Available at `http://dblp.uni-trier.de/xml/`, Nov 2003.

[32] Y. Deng, H. Kuno, and K. Smathers. Managing the evolution of simple and complex mappings between loosely-coupled systems. In *Proceedings of the Second Workshop on Semantics in Peer-to-Peer and Grid Computing at the Thirteenth International World Wide Web Conference*, New York, USA, 2004.

[33] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, and A. Halevy. Learning to match ontologies on the Semantic Web. *VLDB Journal, Special Issue on the Semantic Web*, 12(4):303–319, 2003.

[34] X. Dong, A. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In *SIGMOD Conference*, Baltimore, USA, 2005.

[35] X. Dong, A. Y. Halevy, J. Madhavan, E. Nemes, and J. Zhang. Simlarity search for Web services. In *VLDB*, pages 372–383, Toronto, Canada, 2004.

[36] D. Florescu, D. Kossmann, and I. Manolescu. Integrating keyword search into XML query processing. In *Proceedings of the 9th International World Wide Web Conference*, Amsterdam, The Netherlands, 2000.

[37] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom. The TSIMMIS approach to mediation: Data models and languages. *Journal of Intelligent Information Systems*, 8(2):117–132, 1997.

[38] L. Gravano, P. Ipeirotis, N. Koudas, and D. Srivastava. Text joins in an RDBMS for Web data integration. In *WWW 2003*, pages 90–101, Budapest, Hungary, 2003.

[39] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. Approximate string joins in a database (almost) for free. In *VLDB 2001*, pages 491–500, 2001.

[40] G. Greco, S. Greco, and E. Zumpano. A logical framework for querying and repairing inconsistent databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(6):1389–1408, 2003.

[41] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[42] L. Guo, F. Shao, C. Botev, and J. Shanmugasundaram. XRANK: Ranked keyword search over XML documents. In *SIGMOD*, San Diego, USA, 2003.

[43] C. Gutiérrez, C. Hurtado, and A. Vaisman. Temporal RDF. In *European Semantic Web Conference (ESWC2005)*, pages 93–107, Heraklion, Greece, 2005.

[44] C. Gutiérrez, C. A. Hurtado, and A. O. Mendelzon. Foundations of Semantic Web databases. In *PODS*, pages 95–106, 2004.

[45] M. Halkidi, B. Nguyen, I. Varlamis, and M. Vazirgiannis. THESUS: Organizing Web document collections based on semantics and clustering. Technical Report 230, INRIA Project Gemo, 2003.

[46] P. Hayes. RDF semantics. `http://www.w3.org/TR/rdf-mt/`, 10 February 2004.

[47] J. Heflin. Towards the Semantic Web: Knowledge representation in a dynamic, distributed environment. Ph.D. Dissertation, 2001.

[48] J. Heflin and J. Hendler. Semantic interoperability on the Web. In *Proceedings of Extreme Markup Languages 2000*, pages 111–120, Alexandria, VA, 2000. Graphic Communications Association.

[49] M. A. Hernandez and S. J. Stolfo. The merge/purge problem for large databases. In *SIGMOD Conference*, pages 127–138, 1995.

[50] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, C. Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer, and E. Motta. OIL: The ontology inference layer. Technical Report IR-479. Vrije Universiteit Amsterdam, Faculty of Sciences. `http://www.ontoknowledge.org/oil/`, September 2000.

[51] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient IR-style keyword search over relational databases. In *Proceedings of the 29th VLDB Conference*, Berlin, Germany, 2003.

[52] V. Hristidis and Y. Papakonstantinou. DISCOVER: Keyword search in relational databases. In *Proceedings of the 28th VLDB Conference*, Hong Kong, China, 2002.

[53] A. Huang, Q. Xue, and J. Yang. TupleRank and implicit relationship discovery in relational databases. In *WAIM*, Chengdu, China, 2003.

[54] E. Hung, Y. Deng, and V. S. Subrahmanian. RDF aggregate queries and views. In *ICDE*, Tokyo, Japan, 2005.

[55] E. Hung, Y. Deng, and V. S. Subrahmanian. TOSS: An extension of TAX with ontologies and similarity queries. In *SIGMOD*, Paris, France, June, 2004.

[56] H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, and K. Thompson. TAX: A tree algebra for XML. In *Proceedings of DBPL'01*, Rome, Italy, 2001.

[57] M. A. Jaro. Probabilistic linkage of large public health data files. *Statistics in Medicine*, 14:491–498, 1995.

[58] J. J. Jiang and D. W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of International Conference on Research in Computationa Linguistics*, Taiwan, 1998.

[59] G. Karvounarakis, A. Magkanaraki, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, and K. Tolle. Querying the Semantic Web with RQL. *Computer Networks and ISDN Systems Journal*, 42(5):617–640, 2003.

[60] J. Keller, M. Popescu, and J. Mitchell. Taxonomy-based soft similarity measures in bioinformatics. In *Proceedings of the 2004 IEEE International Conference on Fuzzy Systems*, 2004.

[61] M. Klein, D. Fensel, F. van Harmelen, and I. Horrocks. The relation between ontologies and schema-languages: translating OIL-specifications in XML-schema. In *Proceedings of the ECAI'00 workshop on applications of ontologies and problem-solving methods*, Berlin, German, Aug 2000.

[62] S. Konieczny. On the difference between merging knowledge bases and combining them. In *Seventh International Conference on Principles of Knowledge Representation and Reasoning (KR'00)*, pages 135–144, 2000.

[63] S. Konieczny and R. P. Prez. Merging with integrity constraints. In *Fifth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU'99)*, pages 233–244, 1999.

[64] O. Lassila and R. R. Swick. Resource description framework (RDF) model and syntax specification. W3C recommendation. `http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/`, 22 February 1999.

[65] C.-J. Liau. A modal logic framework for multi-agent belief fusion. *ACM Trans. Comput. Logic*, 6(1):124–174, 2005.

[66] D. Lin. An information-theoretic definition of similarity. In *Proc. 15th International Conf. on Machine Learning*, pages 296–304. Morgan Kaufmann, San Francisco, CA, 1998.

[67] J. Lin. Integration of weighted knowledge bases. *Artif. Intell.*, 83(2):363–378, 1996.

[68] J. Lloyd. *Foundations of Logic Programming.* Springer Verlag, 1987.

[69] P. W. Lord, R. D. Stevens, A. Brass, and C. A. Goble. Investigating semantic similarity measures across the Gene Ontology: The relationship between sequence and annotation. *Bioinformatics*, 19(10):1275–1283, 2003.

[70] J. Lu, G. Moerkotte, J. Schue, and V. S. Subrahmanian. Efficient maintenance of materialized mediated views. In *Proc. 1995 ACM SIGMOD Conf. on Management of Data*, San Jose, CA, 1995.

[71] S. Luke, L. Spector, D. Rager, and J. Hendler. Ontology-based Web agents. In *Proceedings of the First International Conference on Autonomous Agents*, pages 59–66, New York, NY, 1997. Association of Computing Machinery.

[72] A. Maedche and S. Staab. Learning ontologies for the Semantic Web. In *SemWeb*, 2001.

[73] D. Maluf and G. Wiederhold. Abstraction of representation for interoperation. *Lecture Notes in AI*, 1315, 1997.

[74] F. Manola and E. Miller. RDF primer. W3C recommendation. `http://www.w3.org/TR/rdf-primer/`, 10 February 2004.

[75] D. L. McGuinness, R. Fikes, J. Rice, and S. Wilder. An environment for merging and testing large ontologies. In *Proc. 7th Int. Conf. on Principles of Knowledge Representation and Reasoning*, 2000.

[76] http://msdn.microsoft.com/library, 2001.

[77] G. A. Miller. WordNet: A lexical database for English. *Communications of the ACM*, 38(11):39–41, November 1995.

[78] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller. Introduction to WordNet: an on-line lexical database. *International Journal of Lexicography*, 3(4):235–244, 1990.

[79] M. Minsky. A framework for representing knowledge. *The Psychology of computer Vision*, 1975.

[80] P. Mitra, G. Wiederhold, and J. Jannink. Semi-automatic integration of knowledge sources. In *Proc. of the 2nd Int. Conf. On Information FUSION'99*, 1999.

[81] P. Mitra, G. Wiederhold, and M. Kersten. A graph-oriented model for articulation of ontology interdependencies. In *Proceedings Conference on Extending Database Technology 2000 (EDBT'2000)*, Konstanz, Germany, 2000.

[82] A. Monge and C. Elkan. The field-matching problem: algorithm and applications. In *Proc. of the Second International Conference on Knowledge Discovery and Data Mining*, 1996.

[83] National Library of Medicine. Medical subject headings fact sheet. http://www.nlm.nih.gov/pubs/factsheets/mesh.html.

[84] N. F. Noy and M. A. Musen. PROMPT: Algorithm and tool for automated ontology merging and alignment. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI-2000)*, 2000.

[85] http://technet.oracle.com/products/text/content.html, 2001.

[86] Parag and P. Domingos. Multi-relational record linkage. In *KDD-2004 Workshop on Multi-Relational Data Mining*, 2004.

[87] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In *Advances in Neural Information Processing (NIPS)*, 2002.

[88] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL Web ontology language semantics and abstract syntax. W3C recommendation. `http://www.w3.org/TR/owl-semantics/`, 10 February 2004.

[89] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. W3C candidate recommendation. http://www.w3.org/TR/rdf-sparql-query/, 6 April 2006.

[90] M. R. Quillian. Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12:410–430, 1967.

[91] R. Rada and E. Bicknell. Ranking documents with a thesaurus. *JASIS*, 40(5):304–310, 1989.

[92] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1):17–30, 1989.

[93] RDQL - a query language for RDF. W3C member submission. http://www.w3.org/Submission/RDQL/, 9 Jan 2004.

[94] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI*, pages 448–453, 1995.

[95] P. Schwarz, Y. Deng, and J. E. Rice. Finding similar objects using a taxonomy: A pragmatic approach. Draft, 2006.

[96] SIGMOD Record in XML. Available at http://www.acm.org/sigmod/record/xml/, Nov 2002.

[97] S. Staab, M. Erdmann, A. Maedche, and S. Decker. An extensible approach for modeling ontologies in RDF(S). In *Proceedings of the ECDL 2000 Workshop on the Semantic Web*, 2000.

[98] G. Stumme and A. Maedche. FCA-MERGE: Bottom-up merging of ontologies. In *IJCAI*, pages 225–234, 2001.

[99] V. S. Subrahmanian. Amalgamating knowledge bases. *ACM Trans. Database Syst.*, 19(2):291–331, 1994.

[100] A. Theobald and G. Weikum. The index-based XXL search engine for querying XML data with relevance ranking. In *EDBT*, pages 477–495, Prague, Czech Republic, 2002.

[101] A. Tomasic, L. Raschid, and P. Valduriez. Scaling access to heterogeneous data sources with DISCO. *IEEE TKDE*, 1998.

[102] O. Udrea, Y. Deng, E. Hung, and V. S. Subrahmanian. Probabilistic ontologies and relational databases. In *Proceedings of the Fourth International Conference on Ontologies, Databases and Applications of Semantics (ODBASE 2005)*, Agia Napa, Cyprus, 2005.

[103] O. Udrea, Y. Deng, E. Ruckhaus, and V. S. Subrahmanian. A graph theoretical foundation for integrating RDF ontologies. In *AAAI*, Pittsburgh, USA, July, 2005.

[104] O. Udrea, D. R. Recupero, and V. S. Subrahmanian. Annotated RDF. In *European Semantic Web Conference (ESWC2006), to appear*, 2006.

[105] United States Patent and Trademark Office. Patent classification home page. http://www.uspto.gov/web/patents/classification/.

[106] F. van Harmelen, I. Horrocks, and P. F. Patel-Schneider. A model-theoretic semantics for DAML+OIL (March 2001). W3C note. `http://www.w3.org/TR/daml+oil-model`, 18 December 2001.

[107] V. G. Voiskunskii. Evaluation of search results: A new approach. *Journal of the American Society for Information Science*, 48(2), Feb 1997.

[108] R. Volz, D. Oberle, and R. Studer. Towards views in the Semantic Web. In *The 2nd Int'l Workshop on Databases, Documents and Information Fusion (DBFUSION02)*, Karlsruhe, Germany, 2002.

[109] H. Wang, F. Azuaje, O. Bodenreider, and J. Dopazo. Gene expression correlation and Gene Ontology-based similarity: An assessment of quantitative relationships. In *The 2004 IEEE Symposium on Computational Intelligence in Bioinformatics and computational Biology (CIBCB-2004)*, 2004.

[110] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, pages 38–49, Mar 1992.

[111] G. Wiederhold. Intelligent integration of information. In *Proc. 1993 ACM SIGMOD Conf. on Management of Data*, pages 434–437, 1993.

[112] G. Wiederhold. Interoperation, mediation and ontologies. In *International Symp. on Fifth Generation Computer Systems, Workshop on Heterogeneous Cooperative Knowledge Bases, ICOT*, pages 33–48, 1994.

[113] World Intellectual Property Organization. International patent classification (sixth edition). http://www.wipo.int/classifications.

[114] Z. Wu and M. Palmer. Verb semantics and lexical selection. In *32nd. Annual Meeting of the Association for Computational Linguistics*, pages 133 –138, New Mexico State University, Las Cruces, New Mexico, 1994.

[115] Apache Xindice XML database. Available at `http://xml.apache.org/xindice/`.

[116] G. Zhou, R. Hull, R. King, and J. Franchitti. Using object matching and materialization to integrate heterogeneous databases. In *Proc. of 3rd Intl. Conf. on Cooperative Information Systems (CoopIS95)*, Vienna, Austria, 1995.