

SUPPORTING LINK ANALYSIS USING ADVANCED QUERYING METHODS ON
SEMANTIC WEB DATABASES

by

KEMAFOR ANYANWU

(Under the Direction of Amit Sheth)

ABSTRACT

There is an increasing demand for technologies that can help organizations unearth actionable knowledge from their data assets. This demand continues to drive the flurry of activities in data mining research where the emphasis is on technologies that can identify patterns in data. However, in addition to the “patterns” view of data, other data and knowledge perspectives are required to support the broad range of complex analytical tasks found in contemporary applications. For example, in some applications in homeland security, bioinformatics, business and other investigative domains many tasks are focused on “connecting the dots”. For this genre of applications, support for identifying, revealing and analyzing links or relationships between groups of entities (*link analysis*) is crucial. Currently, mainstream database systems do not provide support for such analyses and current solutions rely on exporting their data from their databases into custom applications to be analyzed. This has the disadvantage of additional overhead and precludes the ability to exploit other mature technologies offered by today’s database systems.

This thesis argues for database support for link analysis by providing an appropriate interpretation for such information requests in a graph database model. It addresses several key

database issues with respect to supporting such queries. First, it identifies a number of querying constructs that are crucial to supporting linking analysis applications and proposes a formal query language called SPARQ2L that allows their expression. A formal semantics and characterization of the computational complexity of SPARQ2L's query constructs is also presented. Second, it proposes a database storage model that supports efficient processing of queries while being tolerant of data persistence. The storage model combines a graph linearization strategy rooted in algebraic techniques for solving path problems with a set of heuristics for node and edge clustering that aims to minimize external path lengths. Third, it proposes a novel relevance model SemRank which exploits the "machine processible semantics" of data in ascribing relative importance to query results and offers a flexible or "*modulative ranking*" model enabling serendipitous knowledge discovery.

INDEX WORDS: Link Analysis, Semantic Associations, Semantic Web Databases, Semantic Query Languages, RDF, SPARQL, SPARQ2L

SUPPORTING LINK ANALYSIS USING ADVANCED QUERYING METHODS ON
SEMANTIC WEB DATABASES

by

KEMAFOR ANYANWU

Bachelor of Science, University of Nigeria, Nigeria, 1989

A Dissertation Submitted to the Graduate Faculty of The University of Georgia in Partial
Fulfillment of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

ATHENS, GEORGIA

2007

© 2007

Kemafor Anyanwu

All Rights Reserved

SUPPORTING LINK ANALYSIS USING ADVANCED QUERYING METHODS ON
SEMANTIC WEB DATABASES

by

KEMAFOR ANYANWU

Major Professor: Amit Sheth

Committee: Jay Aronson
Liming Cai
John Miller

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August, 2007

DEDICATION

To Mum and Dad.

ACKNOWLEDGEMENTS

I give thanks to my Almighty God for allowing me the opportunity to pursue my dreams and providing me strength and endurance to see it to completion. It is by His grace that I stand.

I am greatly indebted to my advisor Amit Sheth. His tremendous drive and commitment to proved to be infectious and energized me at different points during my research. His guidance and support throughout my studentship and search for an academic position was unwavering and his insights with respect to those intangible elements that are necessary for success were extremely invaluable. His ready availability made it easy to get feedback and direction at different crucial points. To my advisory committee members, Jay Aronson, Liming Cai and John Miller, I say a big thank you for accepting the task of participating on my doctoral advisory committee and for your contributions and feedback throughout the process of writing this dissertation. I am greatly indebted to many of my professors particularly Bob Robinson and Bob Canfield, who although were not formally on my advisory committee, were very willing to have discussions with me and provide technical feedback and comments on papers that I was working. Special thanks to my professor Hamid Arabnia who always took time to offer a word or two of encouragement at crucial points during this journey. To my comrades, fellow student researchers at the Large Scale Distributed Systems Lab I say it was a wonderful time and I hope that our

paths cross again in the future. Special thanks go to Angela Maduko for collaborating with me on parts of my dissertation and her many contributions to its successful completion. It is my hope and prayer that you get to the finish line sometime very soon.

I am deeply indebted to my husband Okoronkwo and my three kids, Sachi, Sonna and Chide, for the sacrifices that they made to support my pursuit of this degree and the patience they showed throughout the process. This could not have been possible without their love and support.

Finally, I would like to acknowledge my late parents who planted this dream in my heart by exemplifying the value of reaching the heights of academic achievement, in particular my mum who by achieving this same dream, showed me that it is possible for a mother and wife to defeat the challenges of doctoral program in the sciences.

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENTS	v
LIST OF TABLES	x
LIST OF FIGURES	xi
CHAPTER	
1 INTRODUCTION	1
1.1 Link Analysis	1
1.2 Thesis Motivation and Summary	3
1.3 Research Challenges.....	5
1.4 Contributions	7
2 SEMANTIC WEB AND LINK ANALYSIS	9
2.1 Semantic Associations on the Semantic Web	9
2.2 Semantic Web Databases– State of the Art.....	12
2.3 Semantic Web Database Storage Systems	16
2.4 Query Result Ranking	16
3 QUERY LANGUAGE SUPPORT FOR PATH EXTRACTION QUERIES	18
3.1 Introduction	18
3.2 Query Language Design Goals.....	19

3.3	Motivating Examples	20
3.4	Preliminaries.....	22
3.5	The SPARQ2L Query Language.....	23
3.6	SPARQ2L Generalized Graph Patterns	27
3.7	The Semantics of Path Queries in SPARQ2L	28
3.8	SPARQ2L By Example.....	32
3.9	Comparison to other query languages	33
3.10	Computational Complexity of SPARQ2L.....	33
4	QUERY EVALUATION FRAMEWORK.....	36
4.1	Introduction	36
4.2	Design Goals	37
4.3	Preliminaries.....	37
4.4	Management of Path Sequences.....	41
4.5	Prunable Equivalence	43
4.6	Labeling Path Sequences.....	47
4.7	2-Color Path Sequences.....	50
5	QUERY PROCESSING	55
5.1	System Architecture Overview	55
5.2	Evaluation of Unconstrained Path Extraction Queries.....	57
5.3	Representation of Path Summaries.....	61
5.4	Performance Evaluation	62

6	RANKING RESULTS OF PATH EXTRACTION QUERIES.....	70
6.1	Philosophy of Ranking Model.....	72
6.2	Preliminaries.....	74
6.3	Information Gain and ρ -Path Semantic Associations	76
6.4	Refraction	81
6.5	S-Match	82
6.6	SemRank	83
6.7	Ordering Search Results using SemRank values	84
6.8	Overview of the SSARK System	84
6.9	Annotating Path Expression Trees	86
6.10	Retrieving Top-K Results.....	88
6.11	Evaluation.....	91
7	CONCLUSION AND FUTURE WORK	104
7.1	Overview of Thesis	104
7.2	Future Directions.....	105
	REFERENCES	106

LIST OF TABLES

	Page
Table 1: Subgraph Matching Queries in SPARQL and SQL	15
Table 2: Properties of the Datasets	63
Table 3: Example Semantic Associations.....	71
Table 4: Statistics of the top ranked relationship between the entity pairs.....	101
Table 5: : Top 3 Conventional and Discovery Relationships using the SemRank Algorithm	102

LIST OF FIGURES

	Page
Figure 1: RDF triples and graph	10
Figure 2: Semantic Associations.....	11
Figure 3: Subgraph Matching	13
Figure 4: ELIMINATE Algorithm.....	39
Figure 5: An Example Graph and its Path Sequence	41
Figure 6: Example database graph.....	44
Figure 7: Hierarchical labelling of an RDF graph	45
Figure 8: Core Graph Partitioning Algorithm.....	49
Figure 9 : 2-Color Code for Example Graph	53
Figure 10 : System Architecture	56
Figure 11 : Path Solve Algorithm	58
Figure 12: An illustration of the Path-Solve	60
Figure 14: NT-NT C-Queries (Time)	65
Figure 15: NT-NT C-Queries (#p- expressions	65
Figure 16 : T-T C-Queries(Time).....	65
Figure 17 : T - T C-Queries (#p-expressions).....	65
Figure 18 : NT - T C- Queries (Time)	64
Figure 19: NT - T C-Queries (#p-expressions).....	64
Figure 20 : NT - NT D-Queries (Time)	64

Figure 21 : NT - NT D - Queries(#p-expressions).....	64
Figure 22 : T- T D-Queries (Time).....	65
Figure 23 : T - T D-Queries (#p-expressions)	65
Figure 24 : NT - T D-Queries (Time).....	65
Figure 25 : NT - T D-Queries (#p-expressions).....	65
Figure 26 : NT - NT C-Queries (Time)	67
Figure 27 : NT - NT C - Queries (#p-expressions).....	67
Figure 28 : NT - NT D-Queries (Time)	67
Figure 29 : NT - NT D-Queries	67
Figure 30: An example RDF knowledge base	71
Figure 31: Query Interface.....	73
Figure 32: System Architecture with Ranking Module	85
Figure 33: APET showing Top-K evaluation	90
Figure 34: Schema Graph for Evaluation Testbed.....	93
Figure 14: NT-NT C-Queries (Time)	Error! Bookmark not defined.
Figure 36: Categorization of Relationships between Schwarzenegger and Agassi.....	97
Figure 37: Categorization of Relationships between Schwarzenegger and Bush.....	98
Figure 38: Categorization of Relationships between Jordan and Woods	98
Figure 39 : P-Values of the Maximum Number of Top 3 Rankings for the Experiments.....	99

1. INTRODUCTION

This thesis studies the data management and processing requirements for an emerging, important class of applications characterized by identifying and analyzing links in data. Traditional database management systems lack the appropriate primitives and access mechanisms for supporting this genre of analytical data processing. The motivation of the work presented here arises from these limitations of traditional data management systems and the need to develop intelligent data analysis techniques bolstered by the rapidly growing suite of Semantic Web technologies that offer the possibility of reasoning about the “meaning” of data.

1.1 Link Analysis

Much of nature’s existence can be viewed in terms of things and relationships between things. Even man made things such as document collections can all be viewed as a network of some kind e.g. the World Wide Web, citation network, etc. Correspondingly, many of the contemporary models for representing data are based on graph structures. For example, the eXtensible Markup Language - XML, a popular data representation and exchange format, can be regarded as graphs when considering IDREF/ID links, the Object Exchange Model - OEM is a rooted, edge-labeled, directed graph for Stanford’s LORE database system. On the Semantic Web, two major W3C standards, RDF and OWL, exhibit node and edge labeled graph models. In bioinformatics, many well-known projects, e.g., BioCyc (<http://biocyc.org>) are based on graph-structured data models. Even for some models which at first glance do not have an explicit graph structure, there is often an implicit graph structure. For example, the relational model, relationships are captured as attributes of a relation or via foreign key relations to other relations.

The ubiquitous nature of graph structures has given rise to an interesting area of research called *Link Analysis*. In general terms, link analysis refers to the process of analyzing links in data represented by a graph structure. The following excerpt from Wikipedia provides some insight:

“link analysis is a subset of network analysis, exploring associations between objects. ... It provides the crucial relationships and associations between very many objects of different types that are not apparent from isolated pieces of information”.

Some applications mentioned include criminal investigations where examining the addresses of suspects and victims, the telephone numbers they have dialed, their financial transactions during a given timeframe, and the familial relationships between these subjects can have a bearing on identifying the entire scope of a crime; fraud detection employed by banks and insurance agencies; in epidemiology and pharmacology by medical sector, search engines for relevance rating, and so on.

Given the scale of the datasets being considered these days, a problem that often arises is the problem of information overload. For example in link analysis studies, it wouldn't be surprising to find thousands of links connecting the subjects of interest. However, the fact is that many of the links or associations found may be mundane and unworthy of further consideration. Therefore, an important dimension of link analysis is importance determination enabling analysts to “zoom in” on important results. This is not always a black and white issue and very likely to be context or task dependent. The ranking models used on the World Wide Web to determine the relative importance of Web pages rely heavily on the structure of the Web. On the other hand, the World Wide Web Consortium (W3C) has embarked on an initiative to migrate the current Web to a more “Semantic Web”[14] where data on the Web is given well defined and machine-

understandable semantics to the degree that software agents can reason about it. Its framework constitutes of semantic data models that allow users create machine processible vocabularies of the concepts and relationship types in their domains. Then, Web resources may be more precisely described in terms of these domain vocabularies. This offers the opportunity to develop techniques for determining importance and relevance of Web content based on a deeper understanding of content so that links on the Web are not seen merely as “links” but what we would call *Semantic Associations*.

1.2 Thesis Motivation and Summary

The fundamental goals of link analysis are to be able to answers questions like “How are A, B, C and D related?” and “What are the “most important” relationships between X, Y and Z to me?”. From the perspective of graph structured databases, these questions can be viewed as *querying about the structure* of data i.e. extracting subgraph structures from a database graph given a set of *anchor points*. Hence, we refer to this class of queries as *Subgraph Extraction Queries*. The following two examples draw from real world scenarios and give an idea of the nature of these queries.

Scenario Example 1. (*Flight and Airport Risk Assessment*) To assess a potential threat to the safety of flights to certain airports, security officials would like to investigate all high risk passengers scheduled for such flights.

Find any relationships between passengers on flights to New York or Washington DC, who either purchased their tickets less than 24hrs before departure time or purchased their tickets by cash, particularly links associated with flight training.

Scenario Example 2. (*Analysis of gene interactions involved in advanced ovarian cancer adapted from [33]*). Microarray analysis data of tissue from advanced ovarian cancer revealed 1191 differentially expressed genes when compared to normal samples. Researchers will like to analyze these genes with respect to the biological pathways that they participate in to help understand the mechanism of action of the disease.

Show the interaction network for all genes that are differentially regulated in advance stage papillary serous ovarian cancer with respect to the signaling pathways. Constrain the results to those genes expressed in epithelial cells.

Other example applications arise in the transportation and telecommunications domains where network analysis is used for planning; in financial applications such as anti-money laundering analysis [63] and detecting potential biomedical patent infringement [54].

However, traditional database systems do not provide the primitives and query constructs to express path or subgraph extraction queries or reason about the relative importance of the results of such queries. Consequently, applications often have to rely on exporting their data from a database into an external application for performing this kind of analysis. There are clear disadvantages to this approach. Beyond the obvious overhead of exporting data from a database, it precludes the option of exploiting mature the database access mechanisms and advanced query optimization strategies in situations where certain subcomponents of complex analytical queries can be translated to traditional query paradigms.

The second aspect of link analysis is concerned with relevance ranking where the focus is on determining the relative importance of query results to the query. There are models that exist for ranking query results on structured databases e.g. relational databases and on the Web. These

models do not consider the “meaning” of the objects being ranked and rely user specified criteria and/or heuristics such as the topology of the Web.

This thesis is focused on providing support for link analysis tasks in Semantic Web databases. Specifically, it identifies and formalizes a number of query constructs required for a broad range of applications. Further, it proposes a formal query language SPARQ2L (*Recursively Protocol, RDF Query and Link Language*) that allows their expression capturing an important subclass of subgraph extraction queries called *path extraction queries*. Path extraction queries is a special of subgraph extraction queries where there are only two anchor points. It also proposes a storage model for general graphs that allows efficient query processing of path extraction queries that tolerates data persistence. Finally, it proposes a ranking model ascribing relative importance to query results. The ranking model called *SemRank* accounts for both structural and semantics properties of a data graph and introduces the notion of a “*modulative ranking*” model allowing users to adjust the way query results are ranked in order to suit their needs.

1.3 Research Challenges

There are a number of challenges that arise in trying to enable databases provide support for path extraction queries.

Query Language Support

Our examples show that often complex analytical queries will not be in the form “What relationships exist between A and B” but in-fact will specify additional constraints which results must satisfy e.g., connecting paths must contain some mandatory nodes or edges. In our example queries, we can observe constraints such as “*associated with flight training*”, “*tickets paid for by cash*”, “*genes expressed in epithelial cells*” and so on. While it may be possible to express some

of these constraints using the traditional *subgraph pattern matching* querying constructs offered by mainstream querying systems, newer querying constructs are needed to capture the full range of path extraction constraints needed by many applications. Therefore, our first challenge is to develop a formal query language that supports the synergistic expression of the subgraph matching and subgraph extraction query paradigms.

Query Processing Algorithms

Another challenge is the design of efficient query processing algorithms for processing complex analytical queries that involve both subgraph matching and path extraction. There has been a lot of advancement made towards optimally answering subgraph matching queries. On the other hand, efficient evaluation of subgraph extraction queries has received little attention. It is unlikely that we will be able to find polynomial time query processing algorithms for subgraph extraction queries. The reason for this suspicion is that some relatives of this problem are known to be NP-Hard. For example, the Steiner Tree problem which is a version of this problem where given a set V of points (vertices) we would like to interconnect them by a network (graph) of shortest length, where the length is the sum of the lengths of all edges is known to be NP-Hard. Even for the case of path extraction queries where we have only two anchors, the addition of certain classes of constraints leads to an NP-Hard problem. Consequently, the most feasible approach is likely to be one where query processing algorithms are heavily supported by other infrastructure such as index structures that may store preprocessed data.

Database Storage and Access Structures

In addition to the problem of developing indexing structures for quick access to preprocessed data and imposing favorable orderings on data to speed up database searches, database researchers must worry about the best way to store data on disk so that the disk access time is

minimized during query processing. This is important because frequent disk accesses can offset any advantages of a query processing algorithm that is optimal in terms of execution time. For path extraction queries, what we would like is to find a way to cluster graph nodes on disk so that a path may traverse through a large number of nodes but only traverse a few disk blocks. This problem has been shown [32] to be related to the bin packing problem which is NP-Hard for the case the graph is not a tree. Consequently, we can only hope to develop a set of heuristics that will perform well in most situations.

Model and Computational Infrastructure for Ranking Query Results

The ranking of query results requires two components. First is the ranking model which determines how importance values are assigned. Second is the infrastructure for computing importance values for each query result. For ranking query results from Semantic Web graph data models, we would like our ranking models to rely “semantics” of the links in the data. Further, it would be desirable for the ranking model to allow serendipitous knowledge discovery by allowing users encounter relevant information that they may not have thought of querying about directly. Finally, we would like to design algorithms and data structures that will allow us compute the Top-K query results based on our ranking model.

1.4 Contributions

- We present a classification of querying primitives that are important for supporting this class of queries and propose a formal query language called SPARQ2L (*Recursively Protocol And RDF Query and Link Language*) that supports their expression. SPARQ2L extends the current standard query language proposal for querying RDF databases with the ability to express a variety of query classes including *path extraction queries*, *reachability queries* and

recursive queries. Further, it supports the synergistic expression of the link analysis querying along with the traditional querying paradigms in a single query. A formal semantics and characterization of the computational complexity of SPARQ2L's query constructs is also presented. The contributions on query language support for link analysis queries are discussed in [9][7][5].

- We propose a storage model that transforms graphs into a linear representation that can be indexed and stored on disks. Heuristics are employed to minimize the external path length (i.e. number of disk blocks traversed by a path) of paths in a graph. The approach has its roots in algebraic methods for solving a system of linear equations which are given a graph theoretic interpretation. A query evaluation framework is also proposed that supports efficient path extraction query evaluation on disk resident graphs linearized using our storage model. The work on the storage and query processing model is discussed in [5].
- We present a ranking framework which includes a modulative rank model called SemRank and the computational infrastructure for computing Top-K queries based on SemRank values. SemRank model was developed on the premise that users need to be able to view their query results from different perspectives by demanding different orderings on their query results. The computational infrastructure for computing the Top-K SemRank results is based on the novel notion of a Semantic Summary analogous to the notion of structural summaries used for optimizing path expression query evaluation and a greedy pipelined Top-K algorithm for computing the approximate k best query results. The work on ranking query results is discussed in [6][7].

2. SEMANTIC WEB AND LINK ANALYSIS

2.1 Semantic Associations on the Semantic Web

At the core of the Semantic Web is a suite of modeling languages for describing domain vocabularies as well as individual Web resources and these languages are now in the process of standardization by the W3C. Two key modeling languages are RDF [50] and OWL [29] which allow Web resources to be described in terms of machine processible domain vocabularies. RDF is considered a foundational language while the OWL family of languages are generally more expressive languages rooted in formal logics. RDF has a companion specification called RDFS[37] which provides a special vocabulary for describing domain vocabularies. Our work focuses on the RDF data model and we will elaborate on it in the sequel.

RDF provides a simple data model for describing relationships between resources in terms of named properties and their values. Its central notion is that of a *resource* which refers to any concept or object and can be uniquely identified using an *Internationalized Resource Identifier (IRI)*. IRIs are a generalized form of Web URLs that can be used to identify all kinds of resources not just Web pages and are allowed to contain characters from the Universal Character Set. Two other fundamental notions are that of a *Statement* and a *Property*. A *Statement* which may be represented as a *triple* of the form (*Subject, Property, Object*) asserts that that a resource, the *Subject*, has a *Property* whose value is the *Object* (either another resource or a literal). Hence, a *property* denotes a binary relationship between two resources or between a resource and a literal value. For example (borrowed from [52]), suppose that we would like to represent the fact that a resource (a Web page) with identifier *http://www.example.org/index.html* has a

creator “John Smith”. This could be represented by the following RDF triple (*ex:index.html*, *ex:terms:creator*, *ex:staff/85740*). The prefixes *ex*, *ex:terms* and *ex:staff* are aliases for the following namespaces URI: <http://www.example.com/>, URI: <http://www.example.org/terms/> (for terms used by an example organization), and URI: <http://www.example.org/staffid/> (for the example organization's staff identifiers) respectively. A triple has an equivalent graph representation where the subject and object are nodes and the property is an edge connecting the subject to the object and the edge is labeled with the property name. Figure 1 shows a graph for our example triple and two additional triples.

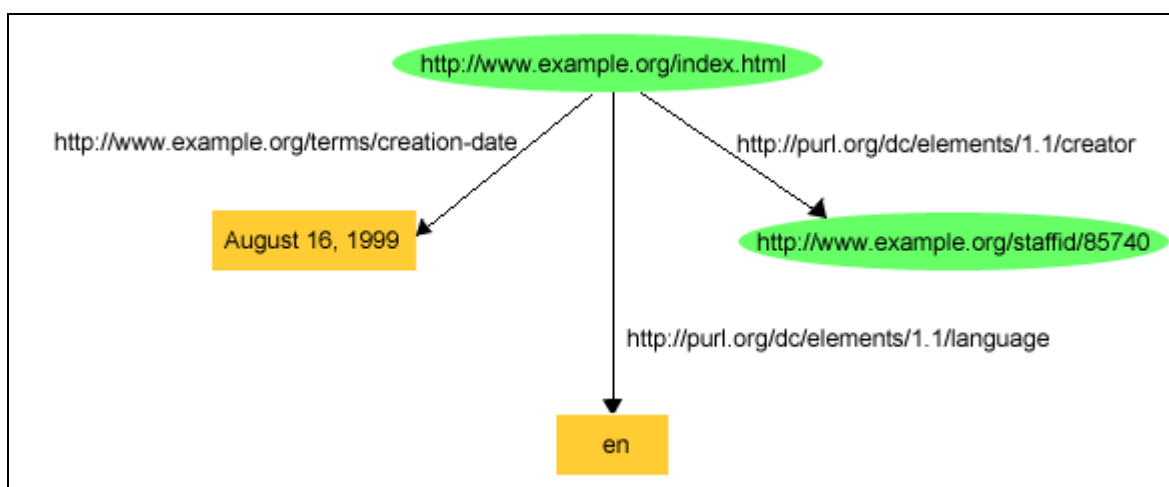


Figure 1: RDF triples and graph

In the same way, domain vocabularies are described using the special vocabulary provided by RDF’s companion specification RDFS. Domain vocabularies describe the types of entities i.e. *classes* (e.g. Students) and types of relations or *properties* in RDF parlance (e.g. author_of) in the domain. RDFS provides a special vocabulary of metaclasses and metaproperties for describing domain vocabularies. Classes/Properties are defined to be an instance of a metaclass *rdfs:Class/rdfs:Property*. Properties are further defined in terms of the classes whose instances

they may apply to (called their *domain*) and the classes whose instances they may take as values (called their *range*). In the RDFS model, both classes and properties can be organized into subsumption hierarchies. Therefore, the definition of classes/properties may also contain information about which classes/properties they are specializing using the *rdfs:subclassOf/rdfs:subpropertyOf* properties. Finally, domain entities or resources are classified based on the classes they belong to i.e. resource typing, using the same model and the *rdf:type* property. This model can be represented as a directed labeled graph, where nodes are labeled with the names of the classes they belong to, and edges are labeled with the names of the properties connecting the resources.

Based on this graph model, our earlier work [7] proposed a classification of so-called “semantic associations” that capture meaningful associations that exist in a database. Figure 2 shows three classes of associations in the context of semantic directed graph models. Figure 2 (a) shows a direct path connecting two entities *r1* and *r2* and we call this a ρ -pathAssociation.

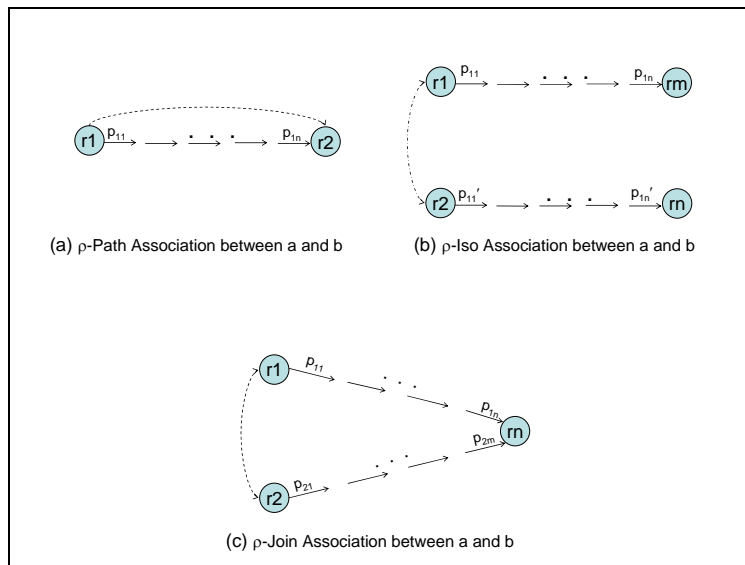


Figure 2: Semantic Associations

Figure 2(c) is called a ρ -joinAssociation between $r1$ and $r2$ indicating that a path originating from $r1$ meets or “joins” with a path originating from $r2$. The intuition here is that $r1$ and $r2$ meet at some point or have something common. Figure 2(b) shows a different kind of association where the entities need not be directly connected. This is called a ρ -isoAssociation capturing a kind of similarity relationship between entities. The paths $p = p_{11}, p_{12} \dots p_{1n}$ originating from $r1$ and $p' = p_{11}', p_{12}', \dots, p_{1n}'$ originating from $r2$ are semantically similar. This means that the corresponding edges in both paths are either exactly the same or specializations or generalizations of each other i.e. p_{1i} is the same/generalization/specialization of p_{1i}' .

2.2 Semantic Web Databases– State of the Art

In this section, we will elaborate on the features of present day Semantic Web systems and emphasize the missing elements necessary for the support of link analysis.

2.2.1 Querying Languages

There are now several RDF query languages including RDQL[61], TRIPLE[66] N3, RxPath[67], RQL[47] SeRQL[19], RDFQL, Versa and the current W3C standard RDF query language proposal SPARQL[58] which is a successor language to RDQL[61]. A query algebra RAL[35] has also been proposed as a vehicle for studying RDF query languages. These languages exhibit one of four styles: SQL-like languages SPARQL, RDQL, SeRQL; functional query languages such as RQL, SeRQL; rule-based languages such as TRIPLE, N3 and graph traversal languages like Versa and RxPath. CORESE[28] is a Semantic Web search engine based on conceptual graphs that offers an RDF query language that uses the triple syntax. It supports the expression of fixed path length queries but no other kinds of constraints are supported. PPARQL[3] is an extension to SPARQL that supports flexible graph matching on RDF databases, however a key

feature (path variables – to be elaborated on in the sequel) is lacking in this language. The query language proposal which goals are most similar to ours is SPARQLer[49]. We will elaborate on the specific differences between SPARQLer and our SPARQ2L in chapter three.

A majority of these languages, particularly the SQL-like languages mainly offer query constructs for what we will call the *subgraph matching querying* paradigm. This paradigm is one where a query is a description of a type of subgraph to be matched in the database i.e. a template, and result of the query is the set of all matches of the query subgraph found in the database. For example, let us look at the example in Figure 3. The big graph in the figure is a database graph that represents entities (*students, professors, publications, universities, etc*) and their relationships (*advisor, author_of, enrolled_in, etc.*).

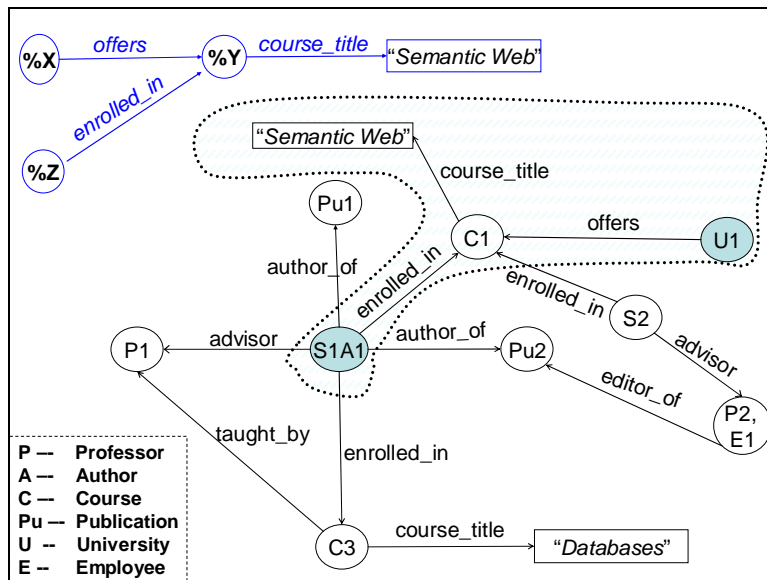


Figure 3: Subgraph Matching

In the figure, different instances of a particular type have a prefix stating their type and their suffix which assigns them a distinguishing identifier e.g. Pu2 can be thought of as “Publication2”. The legend at the bottom left shows the labels for all the entity types. The top part of the figure is a query graph i.e. the graph template to be matched and the nodes with labels beginning with a % sign are *query variables* (%X, %Y, %Z). Usually, a subset of the set of query variables are *return variables* i.e., variables whose bindings are in the query result. In our example, we choose (%X, %Z) as return variables. The encircled part of the data graph shows one matching (actually the only one) of the query graph in the database and the shaded circles are bindings for the return variables. Therefore, the result of our query consists of a 2-tuple (U1, S1A1). In effect, our query is finding the set of university-student pairs such that the student is enrolled in a course on the “Semantic Web” offered by the university.

Generally, these query patterns are specified in the WHERE clause of the query language. For example in Table 1, the left column shows the query for the query graph in the figure, specified in the query language called SPARQL[58]. In a SPARQL WHERE clause, there is a *triple pattern clause* corresponding to each node-edge-node subgraph of a query graph. The right column shows the same query in SQL where the graph structure for the relational model is more implicit (a relation for each entity type and each relation type). The specification node-edge-node query pattern is represented using several relational join operations. In some of the contemporary semi-structured data models OEM, XML, these query patterns are expressed as regular expressions called *path expressions*. In some cases, path expressions support more flexible matching allowing for more expressive queries. Two things to note about the subgraph matching query paradigm are that (i) queries explicitly specify the entity and relationship types and the nature of their relationship to one another, and the purpose of the query is to find the set

of entities linked or related in the manner specified, (ii) return query variables often map to entities i.e. nodes or single binary relations.

Table 1: Subgraph Matching Queries in SPARQL and SQL

SPARQL	SQL
SELECT %X, %Z	SELECT <i>S.name, U.name</i>
FROM http://somewhere	FROM <i>Universities U, Courses C, Students S,</i>
WHERE %X offers %Y	<i>Offers O, Enrollment E</i>
AND %Y course_title “Semantic Web”	WHERE <i>C.title = “Semantic Web”</i>
AND %Z enrolled_in %Y	AND <i>C.id = E.cid</i>
	AND <i>S.id = E.sid</i>
	AND <i>O.cid = E.cid</i>
	AND <i>U.id = O.uid</i>

In contrast, in the *path extraction querying paradigm*, a query specifies a pair of nodes and the result of the query is the set of paths connecting the given nodes. This means that for a path extraction query with source S1A1 and destination P1 in the database in Figure 3, the query return variable should be a *path variable* that maps to the paths $\{<(S1A1, enrolled_in, C3), (C3, taught_by, P1)>, <(S1A1, advisor, P1)> \}$. However, the support for path variables is lacking in most query languages and as our motivating examples demonstrate, it will be important enable the expression of both paradigms in a single query to support complex analytical queries.

2.3 Semantic Web Database Storage Systems

There are a number of available storage systems [3][13][20][27][39][74] for RDF data but only a few provide support for path extraction queries. A few [74] support bounded length and shortest path finding operations via a programmatic interface but without a query language. The most comprehensive support of path extraction queries is offered by [43]. Nonetheless, all of these systems are limited by the fact that they either purely main-memory databases and provide no support for evaluating queries on persistent databases or they rely on traditional relational database storage where these queries would have to be evaluated using very expensive multi-way joins. What would be desirable is to investigate the possibility of newer storage and indexing approaches that will support efficient processing of path queries. An effort in this direction is [11] which proposes an index structure called ρ -index for optimizing ρ -queries [7] whose foundation is path extraction queries. However, the ρ -index approach is based on a technique that transforms general graphs into trees blowing up the size of the resulting graph when there are many non-tree edges.

2.4 Query Result Ranking

The issue of relevance determination and ranking of search results has been investigated in the context of the Web [16][18][48][39][24], the Semantic Web [37][59][69] as well as other Semantic Networks [73], and in databases [34][41]. The approaches that have been used generally fall into three categories. The first category used primarily for Web page ranking is based on an apriori ranking model which is fixed in terms of how importance is determined and users have no input into altering or modifying these rankings to suit their needs. In these approaches typically, relevance is determined by a combination of how “close” a query is to the

description i.e. “matching”, and some *domain independent heuristics* used to assign apriori importance to objects. The second category [40][59] allows domain knowledge to be incorporate and applied universally across user searches. Here, a domain expert is allowed to “seed” the network with appropriate weights, essentially allowing *domain-specific* biases to be incorporated into relevance determination during user searches. With respect to allowing *user-specific* preferences the work on database preference queries [34][41] and Semantic Association Ranking [1] provide the flexibility of ordering and reordering query results based on some user adjustable criteria. However, the criteria are usually fixed which is limiting for a discovery environment where the nature of query results is difficult to anticipate. For example, for database preference queries the adjustable criteria are the attributes of database relations. In contrast, for path extraction queries the nature of the edges and nodes that form the paths are not known *a priori*. Consequently, novel approaches that allow such flexibility but do not rely solely on a fixed set of known attributes are desirable.

3. QUERY LANGUAGE SUPPORT FOR PATH EXTRACTION QUERIES

In this section, we discuss a class of queries called *path extraction queries* and present our proposal for a declarative query language, *SPARQ2L (Recursively Protocol, RDF Query and Link Language)*, supporting their expression in RDF databases. We present a formal algebraic syntax and compositional semantics for path extraction queries, and discuss some computational complexity issues. We also provide a comparison with a few other query languages with similar goals as our work. The work presented in this chapter appeared in references [9][7][5].

3.1 Introduction

In the context of graph structured data, the semantics of path extraction queries can be interpreted as paths in a graph connecting a pair of designated nodes. In many situations, it may be necessary for path extraction queries to specify restrictions that valid paths must satisfy e.g. must contain some mandatory nodes. As a more concrete example, consider the flight risk assessment scenario presented earlier in which passengers and CIA-watchlist organizations were the designated or “anchor nodes”. The query further restricts valid results to those associations or paths between anchor nodes that involve flight training, e.g. a passenger was previously enrolled in a flight training course, and also had financial links e.g. payments using an account that is linked to one of the watchlist organizations. Current RDF query languages, and most other mainstream graph data models such as XML for that matter, are unable to express such queries. The reason is the lack of two crucial elements necessary for supporting these kinds of queries, *path variables* and *path constraint expressions*. The notion of path variables was introduced in the chapter two. In this section will review various examples that justify the usefulness of these

notions and propose a classification of path constraint expressions that allows a variety of complex path extraction queries to be expressed. Then, we propose a query SPARQ2L which extends the current W3C query language standard proposal with the ability to express path extraction queries as well as other classes of graph traversal queries such as reachability queries and recursive queries. We present a formal algebraic syntax and compositional semantics for SPARQ2L and discuss some computational complexity issues for language. We also present a comparison of SPARQ2L and some other query languages which have similar goals.

3.2 Query Language Design Goals

The design of a query language presents opportunities for making a variety of choices. In the sequel, we will review the goals underlying the design choices for SPARQ2L.

2. Our first goal was to have an integrated query language that allowed for the expression of either traditional query paradigms or link analysis query paradigms such as path extraction queries. Further, we wanted the language to support synergistic expression of both paradigms to support complex knowledge-expository tasks where both paradigms may be required to fully capture would be necessary the information need.
2. A second goal was to avoid reinventing the wheel but rather build on existing and accepted query languages and methodologies, particularly those that have been or are being standardized.
2. While it was important to strengthen the expressiveness of our query language, it was equally important to keep the language simple and avoid having cumbersome query constructs even for complex queries.

Throughout the discussion we will highlight how the choices we made achieved these goals.

3.3 Motivating Examples

This section presents some concrete examples of demonstrating the need for the constructs introduced in SPARQ2L. We have drawn our examples from two popular domains, biological data analysis and social network analysis.

Biological pathway analysis

One area of research that has secured its place at the forefront of scientific research for the foreseeable future is called Bioinformatics. Here researchers are concerned with developing advanced analytical techniques for digesting the rapidly growing amount of available information on the biology of living things. Providing such technologies will enable biomedical researchers gain better insight into exact mechanics of life and guide them towards discovering better ways to treat diseases. The analysis of biological data is often done at three levels of abstraction: sequence data representing the two dimensional organization of the nucleotides/amino acids that make up a nucleic acid/protein; structure data which captures the exact location of amino acids within a protein's molecular structure i.e. three dimensional relationships; and pathway data which captures the activities of cell function and offers mechanistic view of cell function. The pathway level of abstraction contains some of the answers to the "how" questions that may be of interest to biologists, therefore providing tools for interrogating pathway data in complex ways should be very valuable. In particular, it could positively impact drug development processes by revealing for example, which pathways could be altered or inhibited to ameliorate diseases and suggest better drug targets with minimized side effects. The following two examples give an idea of the kinds of queries that researchers may want to pose.

Query 1a. In what ways does *substanceA* e.g. Mycobacterium Tuberculosis - MTB, affect the cellular response events in *pathwayB* e.g. PI3K signaling pathways.

This query can be expressed in terms of a path query from an MTB surface molecule to a cellular response event via a PI3K enzyme i.e. *path extraction with constraint on intermediate nodes/edges*.

Query 1b. Find all feedback loops involving the *compoundA* e.g. Methionine.

Some biological processes are driven by feedback loops and investigating such processes could be crucial to understanding role of specific substances in cellular behavior. From the point of view of path extraction queries, this could be expressed as a query for a *non-simple path with constraint on and intermediate node*.

Social network analysis

Another area where this has a wide range of application ranging from criminal investigations to business scenarios is the analysis of social networks. Some example queries that might be asked in these contexts include:

Query 2a. Find close connections between SalesPersonA & CIO-Y.

In this query, a salesperson is trying find what connections he/she may have to a Chief Information Officer of a company that could be exploited in trying to obtain an appointment for a sales presentation. This query can be interpreted as a *path query with length constraints*.

Query 2b. Find paths from an *organization* to a *bill* that involves some kind of *sponsorship* of *someone* linked to a *favorable vote* for a *bill*. The italicized words are entities and relationships that are of interest. This query has a typical “conflict of interest” query which has many applications. It can be interpreted as a *Path query with path pattern constraint*.

The discussion in this section presents some of the query constructs needed for link analysis informally. The rest of chapter will give a more formal interpretation of these constructs in the context of RDF databases.

3.4 Preliminaries

In Chapter 2, we introduced an RDF database informally. Here we will give it a more formal presentation.

Let I , L and B be pairwise disjoint infinite sets of IRIs, literals and blank nodes respectively.

Then, the set $RDFT = (I \cup B \cup L)$ is referred to as set of *RDF Terms*. An *RDF triple* is a 3-tuple $(s, p, o) \in (I \cup B) \times I \times (I \cup B \cup L)$ where s is the *subject*, p is the *predicate* and o is the *object*.

Definition 1 (RDF Database Graph). Let Σ be an alphabet consisting of labels and $G = (V, E)$ a directed graph, then the graph $G = (V, E, \lambda, \Sigma)$, $E \subseteq V \times V$, λ is a function $\lambda: E \rightarrow \Sigma$ (maps an edge to a label), is called a *directed edge-labeled graph*. Given an RDF triple $t = (s, p, o)$ there is an equivalent directed edge-labeled graph defined by $G_t = (\{s, o\}, \{(s, o)\}, \lambda, \{p\})$ which we call an *RDF triple graph*. Further, given a set of triples $T = \{t_1, t_2, \dots, t_k\}$ i.e. an RDF database, we call the graph $G_T = G_{t_1} \cup G_{t_2} \cup \dots \cup G_{t_k}$ is called an *RDF database graph*. In other words, an RDF database graph is the graph formed by the union of triple graphs for all triples in the database. From now on, we will use the pairs of terms (*triple, triple graph*) (database, database graph) synonymously. A *path* in an RDF database is defined in a natural way. A set of triples $T = \{t_1 = (x, p_1, o_1), t_2 = (s_2, p_2, o_2), \dots, t_k = (s_k, p_k, y)\}$ in an RDF database is called a *path* iff there exists only two triples $t_s, t_d \in T$ such that for every i there is a j that satisfies the two conditions:

- $s_i = o_j$ except perhaps s_s ,
- $o_i = s_j$ except perhaps o_d .

An RDF path is *non-simple* if there exists a resource o_i and subset T' of triples in T whose elements all have their object as o_i and $|T'| > 1$ i.e., a node is repeated on the path, otherwise it is *simple*. For convenience, we also define two functions (i) `triplesToResources()` which maps a set of triples to a set of IRIs, i.e. the IRIs of the resources (nodes and edges) on the path and (ii) the function `resourcesToTriples()` which maps a set of IRIs to a set of triples.

3.5 The SPARQ2L Query Language

The advantage of developing SPARQ2L as an extension of the SPARQL query language is that it is able to support both traditional pattern matching queries as well as our path extraction queries. This was achieved by introducing a generalization of the SPARQL *graph pattern* query construct to what we call a *generalized graph pattern*. We will present a formalization of these concepts using the algebraic syntax proposed in [44]. The concept of a graph pattern is based on that of a *triple pattern* and a set of algebraic operators {AND, FILTER, OPTIONAL, UNION}. Assume a set VT of variables that is disjoint from the sets I and L and whose elements range over the set I . An element x of VT is referred to as a *term variable* and is denoted by $?x$. A *triple pattern* is defined as any tuple in $(I \cup L \cup VT) \times (I \cup VT) \times (I \cup L \cup VT)$. For example, $(?x, \text{ex:email}, ?y)$ is a triple pattern with variables in the subject and object positions of the triple. A *graph pattern* is a set of triple patterns combined using certain binary operators.

Definition 2 (Graph Pattern). A *graph pattern* is defined inductively as:

- if P is a triple pattern then P is a graph pattern
- if $P1, P2$ are graph patterns then $(P1 \text{ AND } P2), (P1 \text{ UNION } P2), (P1 \text{ OPTIONAL } P2)$ are graph patterns

- if P is a graph pattern and F is a boolean condition on a term variable in P then $(P \text{ FILTER } F)$ is a graph pattern

Examples. The following are examples of graph patterns using the AND and FILTER operators.

$((?x, \text{ex:email}, ?y) \text{ AND } (?x, \text{ex:age}, ?z))$

$(((?x, \text{ex:email}, ?y) \text{ AND } (?x, \text{ex:age}, ?z)) \text{ FILTER } (?z > 15))$

Note that term variables range over the set I i.e. the set of resources. However, our queries require variables to range over paths in a data graph. In other words, set of subsets of I . We call this new class of variables, *path variables* and denoted an element p in that set by $??p$. Given the existence of path variables, we may then define a more general form of triple patterns called a *generalized triple pattern* which permits path variables in the property position of a triple pattern.

Definition 3 (Generalized Triple Pattern) A *generalized triple pattern* is a 3-tuple $qp \in (I \cup VT \cup L) \times VP \times (I \cup VT \cup L)$.

As observed from the motivating examples, beyond the need for supporting paths, there is a need to provide the ability for queries to specify that qualified paths meet must some additional conditions. In terms of query specification, this amounts to specifying constraints on the path variables analogously to how constraints are specified on term variables. From our experience, we have identified four important classes of constraints:

- Containment constraints on Nodes and Edges* – filtering path results based on the presence or absence of certain nodes or edges.

ii. *Pattern based constraints* – filtering path results based on the existence of a pattern on the path

iii. *Cost-based constraints* – filtering paths based on their costs.

To allow the expression of these constraints, we define the following *path built-in functions* on the set VP :

- $\text{containsAny} : (VP, 2^T) \rightarrow \text{boolean}$
- $\text{containsAll} : (VP, 2^T) \rightarrow \text{boolean}$
- $\text{isSimple} : VP \rightarrow \text{boolean}$
- $\text{containsPattern} : (VP, R(T)) \rightarrow \text{Boolean}$
- $\text{cost} : VP \rightarrow \mathbb{R}$

Intuitively, the first three functions return a truth value for path variable binding depending on whether the path contains any member of a given subset of resources (nodes or edges), contains all the members of the given set or is a simple path respectively. The fifth function, $\text{cost}()$, returns a real value which is the cost of a path. The function $\text{containsPattern}()$ takes as input a path pattern described using an extended regular expression mechanism and checks if the pattern exists on the path. Some other approaches [3][49] support regular expressions mechanisms defined over graph edges. However, our example query Query 2b shows an example where a pattern may involve nodes/entities (a person, a bill) and edges/properties (sponsors, votesFor). With the current approaches, one would have to split the pattern into different subpatterns where each of the required nodes would represent the final state of one expression and the start state for another subpattern. However, this would require an extra path aggregate operator to assemble the subpaths into a final result. Clearly, this is cumbersome.

Consequently, in SPARQ2L, we introduce the notion of extended regular expression mechanism we call *Triple Regular Expressions* or *TRE-expressions* which are actually regular expressions over triple patterns.

Definition 4 (Triple Regular Expressions) Recall that a regular expression over a set X can be defined inductively in the following way: if $x \in X$, then x , $(x)^*$, $(x)^+$, $(x)^?$ are regular expressions; if x and y are regular expressions, then so are $x \bullet y$, $x \mid y$. We define a *T-Regular Expression* or a *TRE* as a regular expression over the set T of extended triple patterns where either the subject, property or object of the triple could be replaced with $[\cdot, \cdot]$. The semantics of TRE expressions is described by example. A TRE $([s, \cdot], [\cdot, p], [\cdot, o])^+$ will match a path such that the subject of the first triple in the path is s , the property or edge in the last triple is p , the object of last triple is o , \cdot matches arbitrary intermediate nodes/edges on the path. On the other hand, A TRE $([s, \cdot], p, [\cdot, o])^+$ matches similarly as the previous expression except that all the properties i.e. edges on the path are p .

Example

Suppose that we have the following term variables $?o$, $?c$ and $?b$ ranging over the sets of organizations, congressmen and bills respectively. Our example query 2b could be interpreted as a path from an organization to a bill such that the organization has sponsored something linked to the official and that official is linked to a positive vote for the bill. Note that the sponsorship link to the congress official and the link from the official to the positive vote on the bill may not be direct ones. We can express this using the following TRE expression:

$$([?o, \cdot], [\cdot, \text{sponsors}], [\cdot, \cdot])^+ \bullet ([\cdot, \cdot], [\cdot, \cdot], [\cdot, ?c])^+ \bullet ([?c, \cdot], [\cdot, \text{votesFor}], [\cdot, ?b])^+$$

This pattern matches a path beginning with an organization followed by an arbitrary number of edges and intermediate nodes that are immediately followed by a sponsors edge that leads to

something sponsored (campaign, vacation, etc) that has a path to a congressman linked to a votesFor edge to a bill.

In the Definition 3, the set \mathcal{T} denotes the set of all possible triple patterns while $R(\mathcal{T})$ denotes the set of regular expressions over \mathcal{T} .

3.6 SPARQ2L Generalized Graph Patterns

Essentially, SPARQ2L graph patterns can be seen as composed of generalized triple patterns combined using standard SPARQL composition operators {AND, FILTER, OPTIONAL, UNION} as well as new operator called the PATHFILTER operator. The PATHFILTER operator allows for the expression of *path filter expressions* i.e. constraints on path variables. These expressions are based on the built-in functions described earlier.

Definition 5 (Path Built-in Condition) Let VT and VP (the sets of term and path variables respectively) be two pairwise disjoint sets of variables that are also disjoint from $I \cup L \cup B$. A *path built-in condition* is an expression built from $I \cup VP \cup L \cup R(\mathcal{T})$, the logical operators (\neg , \wedge), the comparison operators ($=$, \leq) and path built-in functions in the following manner:

If $??P \in VP$ is a path variable, c a constant, $M \subseteq I$ a set of IRIs and TP a TRE then following are path built-in conditions:

- 1) $\text{cost}(??P) = c$,
- 2) $\text{containsAny}(??P, M)$,
- 3) $\text{containsAll}(??P, M)$,
- 4) $\text{containsPattern}(??P, TP)$,
- 5) $\text{isSimple}(??P)$.

6) If BC_1 and BC_2 are path built-in conditions, then $(\neg BC_1)$ and $(BC_1 \wedge BC_2)$ are also path built-in conditions.

We can now define the notion of a SPARQ2L *Generalized Graph Pattern Expression* GGP is defined recursively as follows:

Definition 6 (Generalized Graph Pattern). A generalized graph pattern can be defined as

- ❖ if TP is a generalized triple pattern, then TP is a generalized graph pattern
- ❖ if GP is a graph pattern and GGP is a generalized path pattern then $(GGP \text{ AND } GP)$ is a generalized graph pattern expression
- ❖ if GGP is an generalized graph pattern expression and F is a path built-in condition then $(GGP \text{ PATHFILTER } F)$ is a path pattern

3.7 The Semantics of Path Queries in SPARQ2L

The discussion on the query constructs in SPARQ2L presents their semantics informally. However, the importance of a formal semantics specification for a query language cannot be overemphasized. In this section, we formalize the semantics of the query constructs in SPARQ2L. We point out that an understanding of the contents of this section is not required for subsequent discussions. Therefore readers that are not interested in the details of the formal semantics may skip this section.

In [44], the semantics of a SPARQL graph pattern is based on the concept of a *mapping*. A *mapping* μ is a partial function from VT to $RDFT = I \cup L \cup B$ and the function $\text{dom}(\mu)$ denotes the subset of VT in which μ is defined. Based on this notion, the semantics of a graph pattern is defined by a function $[[\cdot]]$ which takes a graph pattern expression and returns a set of *mappings*.

Consider the following database D =

{ (U1, offers, C1), (C1, course_title, "Semantic Web"), (S1A1, enrolled_in, C3),
 (U2, offers, C2), (C2, course_title, "Databases"), (S2, enrolled_in, C3)
 (U2, offers, C3), (C3, taught_by, P1) (S1A1, age, 19)
 (S2, age, 24) }

The following are SPARQL graph patterns and their solutions:

1. (?X, course_title, ?Y)

	?X	?Y
μ_1	C2	Databases
μ_2	C1	Semantic Web

2. ((?X, enrolled_in ?Y) AND ((?X, age, ?Z) AND (FILTER (?Z > 20))))

	?X	?Y	?Z
μ_1	S2	C3	24

We now extend these notions to SPARQ2L's generalized graph patterns.

Definition 7 (Mapping) A *mapping* ω is a partial function from $(VP \cup VN)$ to (2^{RDFT}) such that:

- if $vt \in VT$, $\omega(vt) = t \in 2^{\text{RDFT}}$ and $|t| = 1$

- if $vp \in VP$, $\omega(vp) = p \in 2^{\text{RDFT}}$ and $\text{resourcesToTriples}(p)$ maps to a path

For a generalized triple pattern tp , we denote by $\omega(tp)$, the tuple formed by substituting any variables in tp according to ω . The domain of ω is the subset of $VP \cup VT$ in which ω is defined and is denoted by $\text{dom}(\omega)$. Further, we extend the notion of *compatibility* defined in [44] to include compatibility between a mapping μ and a mapping ω . These extensions allow us to apply the notions of mapping *compatibility* and in a straightforward manner. We say that a mapping μ is *compatible* with a mapping ω if when $x \in \text{dom}(\mu) \cap \text{dom}(\omega)$, then $\mu(x) \in \omega(x)$.

Definition 8 (Join of mappings) Given two sets of mappings Ω and Θ , we can define function called their *join* $\triangleright\triangleleft$ as $\Omega \triangleright\triangleleft \Theta = \{\mu \cup \omega \mid \mu \in \Omega, \omega \in \Theta \text{ are compatible}\}$, in other words, the set of compatible mappings.

Definition 9. (Generalized Graph Pattern Solution) Let D be an RDF dataset over RDFT, TP a generalized triple pattern whose variables we denote by $\text{var}(TP)$ and GP_1 a graph pattern. Then, the solution $[[\cdot]]_D$ of a generalized graph pattern over D is defined as recursively as:

- i. $[[TP]]_D = \{ \omega \mid \text{dom}(\omega) = \text{var}(TP) \text{ and } \omega(TP) \text{ is either is triple or a path in } D \}$
- ii. $[[(GP_1 \text{ AND } GP_2)]]_D = [[GP_1]]_D \triangleright\triangleleft [[GP_2]]_D$

The formalization of the semantics of SPARQL FILTER expressions can be found in [44]. Therefore, we only need to formalize the semantics of the generalized graph patterns with PATHFILTER expressions. We say that a mapping ω satisfies a built-in condition PF written $\omega \models PF$ if, for $I \subseteq I$ and tr a TRE expression,

- 1) PF is $\text{containsAny}(??P, I)$ and $??P \in \text{dom}(\omega)$ and $I \cap \omega(??P) \neq \emptyset$.
- 2) PF is $\text{containsAll}(??P, I)$ and $??P \in \text{dom}(\omega)$ and $I \subseteq \omega(??P)$.

- 3) PF is `containsPattern(??P, tr)` and $??P \in \text{dom}(\omega)$ and `ground(tr)` is a subpath of $\omega(??P)$.
- 4) PF is `isSimple(??P)` and $??P \in \text{dom}(\omega)$ and for $x, y \in \omega(??P)$, $x \neq y$.
- 5) PF is $(\neg PF_1)$, PF_1 is a built-in condition and $\omega \not\models PF_1$
- 6) PF is $(PF_1 \wedge PF_2)$, PF_1 and PF_2 are built-in conditions and $\omega \models PF_1$ and $\omega \models PF_2$

Example

Consider another database $D1 =$

```
{
  (U1, offers, C1), (C1, course_title, "Semantic Web"), (S1A1, enrolled_in, C3),
  (U2, offers, C2), (C2, course_title, "Databases"), (S2, enrolled_in, C3)
  (U2, offers, C3), (C3, taught_by, P1) (S1A1, age, 19)
  (C3, rdf:type, Course) (S2, age, 24)
  (C2, rdf:type, Course) (S2, adviseeOf, P1)
  (S3, taOf, C3) }
```

The following is the solution of the following SPARQL graph pattern that finds paths between students and professors that involve a course.

- 1) $((?X, \text{rdf:type}, \text{Course}) \text{ AND } ((?Y, ??P, ?Z) \text{ PATHFILTER}(\text{containsAny}(??P, ?X)))$

	$??P$	$?X$	$?Y$	$?Z$
μ_1	{ (S3, taOf, C3), (C3, taught_by, P1) }	C3	S3	P1
μ_2	{ (S2, enrolled_in, C3), (C3, taught_by, P1) }	C3	S2	P1

Notice that the relationship between S2 and P1 is not returned because it doesn't meet containment constraint.

3.8 SPARQ2L By Example

This section presents the concrete syntax of SPARQ2L using examples.

Query 1a. (*Path Query with Constraint on Intermediate Nodes*) Find the paths of influence of Mycobacterium Tuberculosis MTB organism on PI3K signaling pathways.

```
SELECT ??p
WHERE {
    ?x ??p ?y .
    ?x bio:name "MTB Surface Molecule" .
    ?y rdf:type bio:Cellular_Response_Event .
    ?z rdf:type bio:PI3K_Enzyme .
    PathFilter(containsAny(??p, ?z)) }
```

Query 1b. (*Non-Simple Path Query*) Find any feedback loops (i.e. non simple paths) that involve the compound Methionine

```
SELECT ??p
WHERE {
    ?x ??p ?x .
    ?z compound:name "Methionine" .
    PathFilter(containsAny(??p, ?z)) }
```

Query 2a. (*Path Query with Path Length Constraint*). Find all close connections (< 4 hops) between SalesPersonA & CIO-Y.

```
SELECT ??p
```

```

WHERE {
  ?x ??p ?y .
  ?x foaf:name "salesPersonA".
  ?y company:is_CIO ?z.
  ?z company:name "CompanyY" .
  PathFilter( cost(??p) < 4 )
}

```

Query 2b. (*Path query with path pattern constraint*) Find paths from an *organization* to a *bill* that involves some kind of *sponsorship* of *someone* linked to a *favorable vote* for a *bill*.

```

SELECT ??p
WHERE {
  ?o ??p ?b .
  ?o rdf:type "organization" .
  ?b rdf:type "bill" .
  ?c rdf:type "congressofficial" .
  PathFilter( containsPattern (??p, ([?o, .], [. , sponsors], [. , .])+ • ( [. , .], [. , .], [. , ?c])+ • ([?c, .], [. , votesFor], [. , ?b])+ ) ) }

```

3.9 Comparison to other query languages

Some other query languages RxPath[67], Versa, PSPARQL[3], SPARQLer support flexible graph traversal queries over RDF databases however only SPARQLer supports path extraction queries directly. There are a few differences between our approach and the SPARQLer approach. The first but minor difference is that SPARQLer uses the standard SPARQL FILTER to express constraints on both term and path variables. To reduce the chances of users expressing semantically inconsistent we chose to propose a different filter clause for path variables. More important is the difference between the classes of queries that can be expressed. SPARQL path

extraction queries have flags for specifying the matching of undirected paths, a feature not currently supported in SPARQ2L. On the other hand, SPARQLer cannot express path extraction queries with containment constraints. Secondly, their regular expressions are defined over only properties i.e. edges and so patterns involving both nodes and edges cannot be specified. Thirdly, path variables are bound to RDF sequence types making it difficult to express path extraction queries about non-linear paths i.e. non-simple path such as Query1b.

3.10 Computational Complexity of SPARQ2L

In this section, we will comment on the computational complexity of the constructs introduced by SPARQ2L. For the computational complexity of the SPARQL query language see [44]. Generally, the problem finding simple paths in general graphs is intractable and many of the constructs introduced in SPARQ2L also have this property. For path extraction queries with the `containsAll(x_1, x_2, \dots, x_k)` predicate, we see that the HAMPATH problem can be considered to be a special case of this problem where $V(G) = \{x_1, x_2, \dots, x_n\}$. Therefore, we posit that such a query is NP-Hard in the size of k . However, we expect that for many applications k will be much less than the total number of nodes in the graph making it practical for those applications. The issue of finding paths subject to `containsAny(x_1, x_2, \dots, x_k)` is also no easier than DISJOINTPATH problem which is known to be NP-Hard. This is because we can consider the `containsAny(x_1, x_2, \dots, x_k)` problem to be equivalent to finding pairs of disjoint paths from the s to x_i and then from x_i to destination d . For the `containsPattern(pat)` expression we look the intractability results presented in [53] the authors study the problem of finding simple paths subject to constraints specified by a regular language. They show that there are conditions (regular expression and the graph are free of *conflicts*) under which such paths can be found in

polynomial time and give an algorithm. They further present a class of languages whose expressions can always be evaluated in time polynomial in the size of both the graph and the expression, and characterize the expressions for such languages syntactically.

4. QUERY EVALUATION FRAMEWORK

In this section, we present our storage model for general directed labeled graphs that supports efficient path extraction query processing on persistent graph databases. We present its theoretic graph foundations and propose some infrastructure for its realization.

4.1 Introduction

It is clear from the hardness results presented in the section on the computational complexity of SPARQ2L that practical implementations will need sophisticated infrastructure for the support of query processing, particularly for disk resident databases. Of the few RDF querying systems [38][74][49] that support some limited form of path extraction queries, most focus on main-memory based approaches and do not provide access structures or indexing mechanisms that would be effective for disk resident databases. For the RDF systems that rely on standard relational database backends, path extraction queries would need to be evaluated using multi-way join operations which is very prohibitive. The only effort that we are aware of that proposes an indexing strategy which may be adapted for disk resident graphs is [11]. This work is similar in spirit to ours and proposed an index structure called ρ -index is for optimizing the evaluation of ρ -path queries. The class of ρ -path queries was proposed in our earlier work and is equivalent to the class of path extraction queries discussed here.

We will now review some of the goals of our approach.

4.2 Design Goals

The design of data storage models offers the opportunity to make choices about time and space tradeoffs.

- We would like to identify methods to precompute and store some path information from a graph in a way that allows us reuse it for answering different queries. While this might increase the amount of information to be stored, our goal is to require no more than a size that is polynomial in the size of the graph.
- We would like to identify a suitable linear representation for directed graphs that will form the basis for a good disk based storage model that is
 - Systematically organized so that pruning strategies can easily be used during query processing.
 - Clusters related path information so that paths traverse as few disk blocks as possible leading to reduced disk access time during query processing.
- We would like to identify ways to index our linear representation using slight modifications of standard database indexing techniques for easy integration into mainstream systems.

4.3 Preliminaries

Our approach for systematically transforming a graph into a linear representation of precomputed path information is rooted in algebraic methods for solving a system of linear equations, in particular Gaussian elimination by LU decomposition. In this technique, a matrix representing a system of linear equations $Mx = b$ is decomposed into two triangular matrices L and U . Then, a solution for the system is found by first solving the system $Ly = b$ (frontsolving) then substituting y in the system $Ux = y$ and solving for the vector x (backsolving). Generally, this involves

processing each triangular matrix in a specific order. An advantage of this approach is that the triangular systems L and U can be used to solve for different right hand sides i.e. different values of b , allowing for the computationally dominant phase (the LU decomposition phase) to be reused for different problem instances. In [70][71], a graph theoretic interpretation (multiplication = concatenation and addition = union) is given to this technique leading to an efficient way to solve different path problems. The elements of the matrix after LU decomposition phase can be viewed as *partial path summaries* where for certain pairs of nodes, *some* of the path information between the nodes is computed. By “summaries” we mean a concise representation of path information as opposed to an enumeration of paths. A good analogy for path summarization is that of representing the set of strings in a regular language using a regular expression which allows an infinite language to be represented using a finite representation. A similar notion called a *path expression* is defined in [71] which a regular expression over the edges of a graph. We refer to path expressions here as *p-expressions* to avoid confusion with their usage in database literature. This algorithm for the LU decomposition phase is shown in Figure 4. It assumes that a graph G be *ordered* - $G_\alpha = (G, \alpha)$ where $\alpha : \{1, 2, \dots, N\} \rightarrow V(G)$ so that $\alpha(i)$ maps to some node v in G i.e. $v \in V(G)$. Conversely, $\alpha^{-1}(v)$ maps a node in G to an integer between 1 and N . Then given an ordered graph with n nodes and an $n \times n$ matrix M , the process begins by initializing $M[i, j]$ for $1 \leq i, j \leq N$ with a p-expression representing a union of the set of edges between the nodes $\alpha(i), \alpha(j)$. The function $\text{PE}(\)$ is a helper function such that $\text{PE}(x, y, \cup)$ and $\text{PE}(x, y, \bullet)$ returns the regular expressions $(x \cup y)$ and $(x \bullet y)$ respectively, while $\text{PE}(x, *)$ returns the regular expression $(x)^*$.

Algorithm 1**Procedure ELIMINATE****Input:** A directed graph $G = (V, E)$ and a matrix $M = n \times n$ where $n = |V|$ **Output:** A path sequence PS for G .

```

01  for each edge  $e = (u, v) \in E$ ,
02       $M[u, v] = PE ( M[u, v], PE(u, v), \cup );$ 
03  for  $v$  from 1 to  $n$ 
04       $M[v, v] = PE ( M[v, v], * );$ 
05      for each  $u > v$  such that  $M[u, v] \neq \emptyset$ 
06          for each  $w > v$  such that  $M[v, w] \neq \emptyset$ 
07               $M[u, w] = PE(M[u, w], PE( PE(u, v), PE(v, w), "\bullet"), \cup)$ 

```

Figure 4: ELIMINATE Algorithm

After the initialization phase, the p-expressions in the cells of the matrix are systematically expanded represent additional paths that satisfy the following certain conditions which hold true for all p-expressions computed at the end of the LU decomposition:

All elements of M satisfy one of two conditions: For $u, v \in V(G)$:

- and $\alpha^{-1}(u) \geq \alpha^{-1}(v)$, $M[\alpha^{-1}(u), \alpha^{-1}(v)]$ contains a p-expression representing exactly the paths from u to v that do not contain any intermediate vertex w such that $\alpha^{-1}(w) > \alpha^{-1}(v)$.
- and $\alpha^{-1}(u) < \alpha^{-1}(v)$, $M[\alpha^{-1}(u), \alpha^{-1}(v)]$ contains a p-expression representing exactly the paths from u to v that do not contain any intermediate vertex w such that $\alpha^{-1}(w) < \alpha^{-1}(u)$.

In general, this algorithm runs in $O(N^3)$ for a graph of size N , although some ideas for optimization are mentioned in [70].

By imposing a specific order on the elements of M yields what is known as a *path sequence* [70].

The notion of a path sequence provides a listing of p-expressions such that given any source s in

the corresponding graph, all path information for s can be computed in a single scan of the path sequence.

Definition 10. (Path Sequence) A *path sequence* [70] for an ordered graph database $G = (V, E, \alpha)$, is a sequence of path expressions $(P_1, v_1, w_1), (P_2, v_2, w_2), (P_3, v_3, w_3), \dots, (P_l, v_l, w_l)$ such that:

- for $1 \leq i \leq l$, P_i is a path expression of type (v_i, w_i) .
- for $1 \leq i \leq l$, if $v_i = w_i$ then $\lambda \in L(P_i)$ where λ is the empty string
- if p is a nonempty path in G then there is a unique sequence of indices $1 \leq i_1 \leq i_2 \leq \dots \leq i_k \leq l$ and a unique partition of p into nonempty paths $p = p_1, p_2, \dots, p_k$ such that $p_j \in L(P_{i_j})$ for $1 \leq j \leq k$.

To obtain a path sequence for a graph after the ELIMINATE algorithm we order the elements of the matrix in the following way: the sequence of p-expressions (X_i, u_i, v_i) where $\alpha^{-1}(u_i) \leq \alpha^{-1}(v_i)$ in increasing order on $\alpha^{-1}(u_i)$ is followed by the sequence of p-expressions (X_i, u_i, v_i) for $\alpha^{-1}(u_i) > \alpha^{-1}(v_i)$ in decreasing order on $\alpha(u_i)$. An example of a graph and its path sequence s shown in Figure 5.

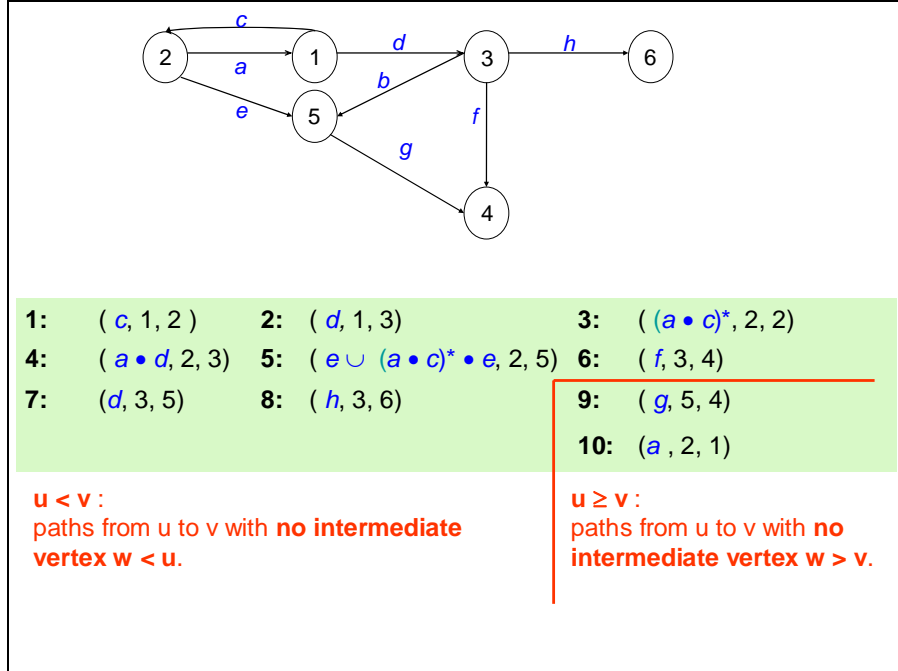


Figure 5: An Example Graph and its Path Sequence

The notion of a path sequence can also be defined in terms of the *strongly connected components* of G_α [70] which leads to a more efficient technique for computing path sequences by first decomposing it first into strong components.

Definition 11 (Path Sequence by Strong Components) Let G_1, G_2, \dots, G_k be a topologically ordered list of the strongly connected components of a graph G i.e., for $i > j$, there does not exist an edge from a component G_i to a component G_j . Further, let A_i be the path sequence for G_i and B_i be the elements: $\{ (X, \text{source}(e), \text{target}(e)) \mid \text{source}(e) \in V(G_i) \text{ and } \text{target}(e) \notin V(G_i) \}$ ordered arbitrarily. Then $A_1, B_1, A_2, B_2, \dots, A_{k-1}, B_{k-1}, A_k$ is a *path sequence* of G .

4.4 Management of Path Sequences

Path sequences have what we will call the *Single-Scan-Path-Preserving* property which means that for any given node u in a graph, evaluating path queries using a path sequence is done by

aggregating the precomputed path summaries in a single scan of the path sequence – to be elaborated on later. Therefore, to achieve our goal of indexing path sequences, what we need to do is to select an index structure that preserves this single scan property. A good candidate is the B+tree index used in standard databases. B+trees support range queries (retrieval of a range of values in a given interval) but in our own case we would need to perform additional operations (p-expression aggregation) on the p-expressions as they are retrieved. Searches using B+trees require key values. So the primary task we have is finding a strategy for assigning key values to p-expressions. We would like this key assignment strategy should as much as possible cluster p-expressions on the path sequence based on their likelihood of being relevant or irrelevant for the same class of queries. The goal of doing this is to minimize the width of the subinterval of the path sequence that needs to be retrieved from disk thereby reducing the number of disk accesses. This problem has been shown [32] to be NP-Hard using a reduction from the bin-packing problem. Therefore, we can only employ some heuristics to solve this problem. Our approach assigns close key values to p-expressions in the path sequence that are related based on their likelihood of being relevant or irrelevant for the same class of queries. A more fragmented organization of relevant and irrelevant p-expressions will lead to queries requiring many small relevant clusters that are scattered across the sequence and consequently many more disk seek operations. In the following section, we will discuss some relationships between nodes that allow us to consider them as related or “*Prunable Equivalent*” for some classes of queries.

4.5 Prunable Equivalence

Figure 6 shows an example RDF data graph with information about faculty, students, research projects, etc., and the relationships between them. The dotted circles are the strong components (maximal subgraph where there is a path connecting each pair of nodes) of the graph.

It is clear that for any non-singleton strong component i.e. strong component containing more than one node, if a path contains one of its constituent nodes as an intermediate node, then there is a path containing all of its nodes as intermediate nodes. This means that if, given a path query we can determine that any constituent node of a strong component is “irrelevant” to the query, we can conclude the irrelevance of all the other constituent nodes and edges and losslessly ignore their associated p-expressions. Consequently, we say that all the p-expressions associated with the nodes and edges of a non-singleton strong component are “*Prunable Equivalent*”.

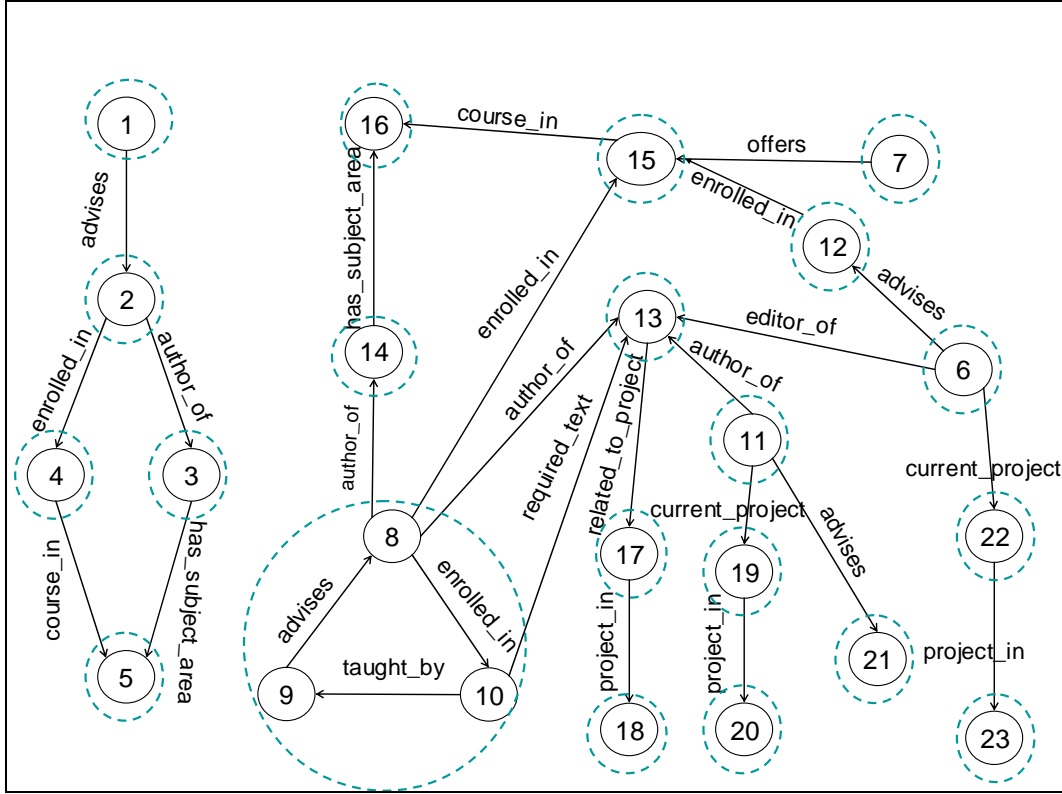


Figure 6: Example database graph

Strongly connected components enclosed in dotted circles

At a higher level of abstraction we may also consider equivalence relationships amongst groups of strong component nodes that form interesting subgraph structures. For example, consider the set of nodes $\{17, 18, 11, 19, 20, 21, 22, 23\}$ in the lower right portion of the graph. These nodes form tree subgraphs which dangle from a parent non-tree subgraph. We call these structures *dangling trees*. Some of properties of these dangling tree structures are that (i.) no paths connect any pair of dangling trees, (ii.) no paths connect a dangling tree back to the parent non-tree subgraph. From the point of view of a path query from a source s to a destination d , this implies that if we can somehow determine that d is not a node of a dangling tree T , then we can conclude the irrelevance of all the nodes in T and we can losslessly ignore all p-expressions associated with the strong component nodes and edges in T . Therefore, we say that the p-expressions of

strong component nodes and edges in a dangling tree are *Destination-Induced Prunable Equivalent*. This relationship trivially applies to disconnected component subgraphs of a graph e.g. the two subgraphs in the figure. Figure 7 shows the graph in Figure 6 with the different subgraphs enclosed in boxes. The dangling trees are in the boxes numbered 3, 4, 5 and 6, and the two disconnected non-tree subgraphs are labelled 1 and 2.

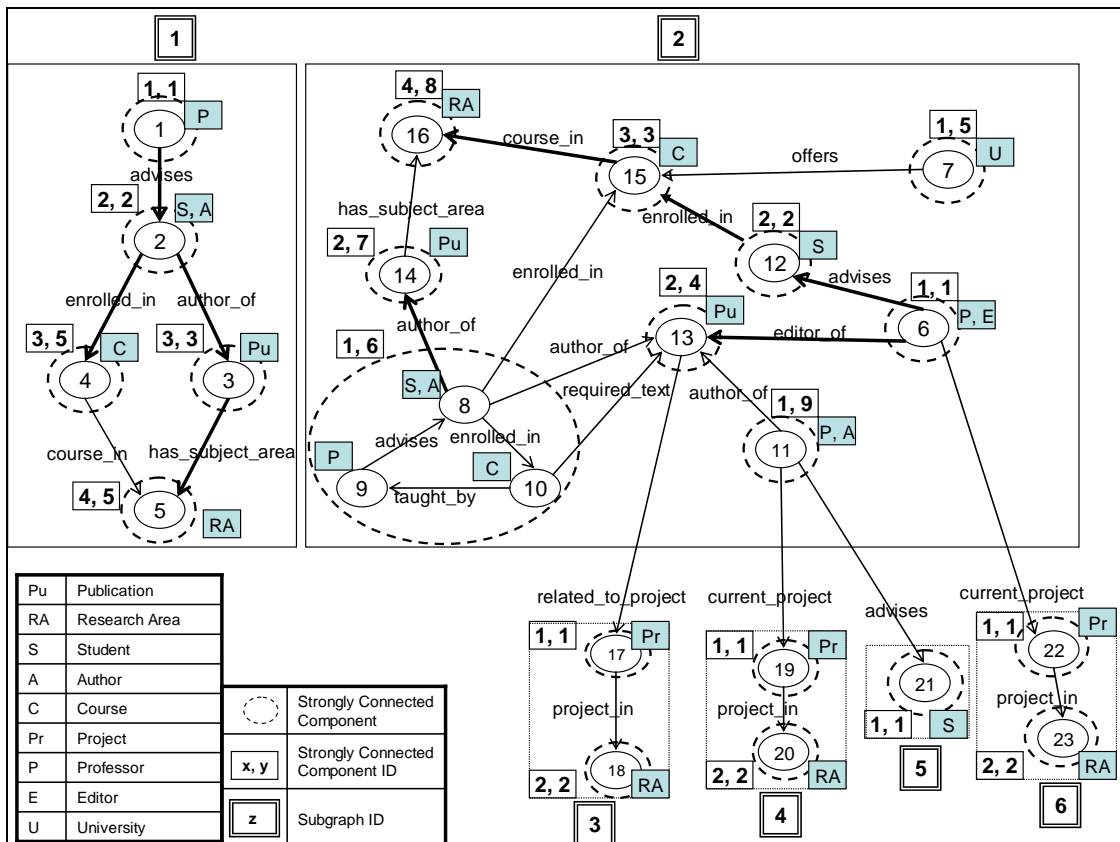


Figure 7: Hierarchical labelling of an RDF graph

Since the non-tree subgraphs are likely to be larger than dangling tree structures, it may be helpful to try to identify equivalence relationships within the structures so that we can refine the

partitioning that we have so far. One way to achieve this is to impose a tree structure of each of them and then classify nodes and edges based on their location in the tree structure. This leads us to a notion of an *optimal spanning tree*. Recall that a *spanning tree* is a tree subgraph of a graph that includes all the nodes in the graph.

Definition 12 (Optimal Spanning Tree). A spanning tree is called *optimal* if every edge (u, v) lies on the *longest path* from root of the tree to v . (In the case of a graph with multiple source nodes, we introduce a root node as the parent of all source nodes.)

In Figure 7, the darkened edges are the edges of the optimal spanning tree (the added root node is omitted from the figure). This optimal spanning tree structure has some properties that can be exploited during query processing.

Definition 13 (OST Property). For an optimal spanning tree T_G for a graph G as defined in Definition 12, any node u in T_G with OST-depth = k is not reachable from any node v with OST-depth = $l \geq k$.

This property allows us to define another prunable equivalence relationship on the nodes of a graph based on their depths in an optimal spanning tree. Given a query with source s and destination d with OST-depths k_s and k_d respectively, we can say that the p-expressions associated with the edges and strong component nodes at the OST-depth k_p , where $k_p \leq k_s$ or $k_p \geq k_d$ are *Treedepth-Induced Prunable Equivalent*.

The next section discusses how the partitioning using the different relationships is encoded using a labelling scheme.

4.6 Labeling Path Sequences

We would like to label the nodes and edges of a graph such that we can easily identify the groups of related nodes based on the relationships discussed in the previous section. To achieve this, we use a hierarchical labeling scheme that captures the above groupings at the three levels of abstraction. At the finest level, we have a *component identifier* which is a unique number assigned to each individual strong component that records the pre/post order visit time during a depth first traversal of the graph. In Figure 7, this is the second of the numbers in the rectangular boxes associated with strong components. The first number in the rectangle is the *level identifier* which is the depth of the strong component node in the optimal spanning tree. At the coarsest grouping we have what we call *subgraph identifiers* which identify disconnected non-tree subgraphs and the dangling tree subgraphs. The subgraph identifiers are the numbers in thick bordered boxes and the intervals of subgraphs identifiers for non-tree subgraphs and that of dangling tree subgraphs are non-overlapping. By default all the non-tree subgraphs are assigned subgraph identifiers lower than the tree subgraphs. The following statement states the properties of this labeling scheme that we exploit to determine relevance of strong component nodes and edges connecting them.

We now define a property that arises from our labeling that can be exploited during query processing.

Definition 14 (NonReachability Property *PropertyNR*). Given a path query q with s and d as source and destination nodes and i and j the subgraph identifiers of s and d respectively, and x and y be their level identifiers. Then

- i. $i \neq j$ implies that q 's result is empty.
- ii. $x > y$ implies that q 's result is empty.

iii. any node u with level identifier k such that $k < i$ or $k > j$ is not a member of q 's result set.

4.6.1 Graph partitioning algorithm

For ease of exposition, we will present the algorithm used to achieve the partitioning of a graph, as a multi-phase algorithm although the actual implementation merges some of the phases. An outline of the main steps in the algorithm is as follows: Given a directed labelled graph G

- 1) Find a topologically sorted list of the strong components of G
- 2) Find the disconnected components of G
- 3) For each disconnected component G_i
 - i) Find and put all dangling tree nodes in a set T_i and label nodes with the three identifiers
 - ii) Find an optimal spanning tree of the graph $G_i - T_i$ and label nodes with the three identifiers

There are well known algorithms to accomplish the first two steps, so we only present the algorithms for the later steps. To find the dangling trees associated with each non-tree subgraph we call the algorithm `partitionNodes()` given in Figure 8. The discussion of the algorithm will refer to the set of all T_i i.e. dangling tree nodes, as the *T-Set* and then the remaining nodes i.e. $G - T$ as the *N-Set*. The algorithm computes three sets the T-Set, N-Set and another set called the *R-Set*, which contains the roots of all the dangling tree structures in G . It assumes that the in-degree of every node is known. This can easily be computed in $O(|V|)$ time. The initialization steps of lines 1, 2 and 3 places every node in G in T-Set, then in the depth first traversal of G in line 4, the nodes of N-Set are determined and moved from T-Set to N-Set, using the *findSet* helper subroutine. The idea of the *findSet* subroutine follows from the definitions of the T-Set and N-Set of a graph i.e. 1) a node is in the T-Set of a graph if and only if it has only one parent

and all its descendants are in some tree structure of the graph and 2) a node is in the N-Set of a graph if it has more than one parent or at least one of its descendants is in some non-tree structure of the graph. Thus a leaf node which has one parent is in T-Set while that which has more than one parent is in N-Set. This is determined in lines 6 and 7 of *findSet*.

```

Algorithm 2
Procedure partitionNodes ( $G = (V, E)$ )
Input: A directed graph  $G = (V, E)$ 
Output:  $\{G_T, G_N\}$ 

01  T-Set =  $V$ ;
02   $\forall u \in V$  do      {initialization}
03       $u.inTS \leftarrow true, u.inNS \leftarrow false, u.visited \leftarrow false$ ;
04   $\forall u \in V$ , if ( $\neg u.visited$ ) then findSet( $u$ );

Procedure findSet (node  $u$ )
01   $u.visited \leftarrow true$ ;
02   $\forall v \in adj(u)$  do
03      if ( $\neg v.visited$ ) then findSet( $v$ )
04       $u.inTS \leftarrow u.inTS \& v.inTS$ ;
05       $u.inNS \leftarrow u.inNS \mid v.inTS$ ;
06  if ( $u.inTS \& u.indegree$  is 1) then  $u.inNS \leftarrow false$ ;
07  else if ( $u.inTS$ ) then  $u.inTS \leftarrow false$ ; move  $u$  to N-Set;
08       $\forall v \in adj(u)$  do if ( $v.inTS$ ) then copy  $v$  to R-Set
09  else move  $u$  to N-Set

```

Figure 8: Core Graph Partitioning Algorithm

For a non-leaf node u , the boolean expression of line 4 determines if all its descendants are in T-Set. If so, u is moved to N-Set if it has more than one parent. On the other hand, in line 9, u is

moved to N-Set if any of its descendants is in N-Set. Line 5 determines if u has some descendants in both T-Set and N-Set. If so, u 's children in T-Set are copied to R-Set in line 8.

Since the loops of lines 4 and 8 are executed $|\text{adj}(u)|$ times for each call to $\text{findSet}(u)$, the over all number of times they are executed is $O(|E|)$. Thus, the running time of *PartitionNodes* is $O(|V| + |E|)$.

4.7 2-Color Path Sequences

Based on the above labeling scheme for p-expressions we arrive at a sequential representation for a graph which associates key values with p-expressions of a path sequence. To further improve the clustering achieved by the labeling scheme, we co-index (assign same key value) all the p-expressions associated with a non-singleton strong component since they are prunable equivalent with respect to every query. This will allow for collectively retrieving or skipping over all the p-expressions associated with the strong component once its relevance is determined. For the same reason, we can co-index multiple edges connecting two strong components. For example, consider the components with rectangular boxes 1,6 and 2,4 which have two edges $\{\text{"author_of"}, \text{"required_text"}\}$ connecting them but involve different constituent nodes 8 and 10. By co-indexing both edges, we can collectively retrieve or skip their associated p-expressions once the relevance of the source and destination components have been determined. The prunable equivalence relationships based on dangling trees and optimal spanning tree depth are also accounted where possible. By clustering them apart from the rest of the graph, we ensure that we do not unnecessarily increase the external path length (number of disk blocks traversed by a path).

Next, we will propose a sequential representation for a graph we call a *2-Color Code* which consists of tuples of keys and sets of p-expressions.

Definition 15. (2-Color Label) Given a node n , with *subgraph identifier* s , *level identifier* l and *traversal identifier* t , we define a *2-Color label component* for n to be a triple of one of the forms (s, l, t) or (s, t, l) denoted by $s1t(n)$ and $st1(n)$ respectively.

We define a lexicographical ordering on 2-Color labels such that $(l_1, l_2, l_3) < (l_1, l_2, l_3)$ iff $l_1 < l_1$ or $l_1 = l_1$ & $l_2 < l_2$ or $l_1 = l_1$ & $l_2 = l_2$ & $l_3 < l_3$.

Definition 16. (2-Color Code). The *2-Color Code* C for a graph $G = (V, E)$ is a lexicographically ordered sequence of tuples of the form $(2cl_1, 2cl_2, \{p-ex_1, p-ex_2, \dots, p-ex_k\})$ where $2cl_i$ is a 2-Color label and $p-ex_i$ are p-expressions and the 2-Color labels in the tuple satisfy the following conditions:

- if v is a node in a dangling tree, then $(2cl_1 = st1(v), 2cl_2 = st1(v)) \in C$ otherwise $(2cl_1 = s1t(v), 2cl_2 = s1t(v)) \in C$
- if $e = (v_i, v_j)$ is an edge and v_j is in a dangling tree then $(st1(v_j), \text{label component of } v_i) \in C$ otherwise $(2cl_1 = s1t(v_i), 2cl_2 = st1(v_j)) \in C$

This definition says that every strong component node has both 2-Color label set to the same value, the difference is in what kind of 2-Color label they have. If the node is in a dangling tree it has a label of the form $st1()$ i.e. the traversal identifier comes before the level identifier. For other nodes, the level and traversal identifiers are reversed yielding $s1t()$ labels. In a similar way, for any edge whose destination node is in a dangling tree, the first 2-Color label is the label of the destination node while the second 2-Color label is for the source node. The ordering induced by a 2-Color Code defined as above can be summarized the following way.

Definition 17 (Order Property PropertyOP): For D and T , non-tree and tree subgraphs respectively of a graph G ,

- i. $u \in V(D)$ and $v \in V(T)$ implies $2CL(u) < 2CL(v)$. Similarly, $e \in E(D)$ and $f \in E(T)$ implies $2CL(e) < 2CL(f)$.
- ii. $u \in V(D)$ and $e = (u, v) \in E(D)$ implies $2CL(u) < 2CL(e)$ while $u \in V(T)$ and $e = (v, u) \in E(T)$ implies $2CL(u) > 2CL(e)$.

This says that the ordering induced by the 2-Color code is such that all p-expressions associated with non-tree subgraphs appear before the p-expressions of tree subgraphs. Also, within each disconnected subgraph the p-expressions are in level order for non-tree subgraphs and depth-first order for tree subgraphs. Finally, for non-tree subgraphs p-expressions associated with an edge appears after the p-expression for the edge's source node whereas the reverse is the case for tree subgraphs. These properties are exploited during query processing as will be seen in the next section.

Figure 9 shows a subset of the 2-Color code for our example graph with a number identifying each tuple in the code. The complete 2-Color code has 43 tuples. Every strong component has a pair of 2-Color labels which are the same. For the singleton component i.e. with only constituting of only one node, its corresponding set of p-expressions is empty e.g. element 1. Non-singleton strong components has a set of p-expressions of its constituent nodes and edges e.g. element 16.

```

1: [ (1,1,1), (1,1,1), { } ],      2: [ (1,1,1), (1,2,2), { (advises, 1, 2) } ],
.....
11: [ (2,1,1), (2,1,1), { } ],    12: [ (2,1,1), (2,2,2), { (advises, 6, 12) } ],
.....
16: [ (2,1,6), (2,1,6), { (enrolled_in, 8, 10),
                             (advises•enrolled_in, 9, 10),
                             ((taught_by•advises•enrolled_in)*, 10, 10),
                             (taught_by, 10, 9), (advises, 9, 8) } ],
17: [ (2,1,6), (2,2,4), { (author_of, 8, 13), (required_text, 10, 13)}, ]
20: .....
31: [ (3,1,1), (3,1,1), { } ],    32: [ (3,2,1), (3,1,1), { (project_in, 17, 18) } ],
34: [ (4,1,1), (2,9,1), { (current_project, 11, 19) } ],
35: [ (4,1,1), (4,1,1), { } ],
36: [ (4,2,1), (4,1,1), { (project_in, 19, 20) } ],  37: [ (4,2,2), (4,2,2), { } ],
38: [ (5,1,1), (2,9,1), { (advises, 11, 21) } ],    39: [ (5,1,1), (5,1,1), { } ],

```

Figure 9 : 2-Color Code for Example Graph

All other entries represent edges of two connected components that have at least one edge connecting them. For these entries the first elements are the 2-Color labels of the source and destination ordered based on the conditions in definition for the 2-Color code. The third entry is the set of p-expressions for the edges connecting the strong component. For example, entry 17 is for the set of edges connecting the strong component numbered 4 with the strong component numbered 6. The edges connecting them are “*author_of*” and “*required_text*” represented by the two p-expressions in the third element of the entry.

For a graph of n nodes, m edges and k strong components, the overall time for preprocessing includes the time to find strong components $O(m + n)$, the optimal spanning tree $O(k + m')$

where $m' < m$ is the number of edges connecting strong components, and the time to run the LU

decomposition algorithm for all strong components: $O \sum_{i=1}^k n_i'^3$ where n_i' is the number of nodes in strong component i .

To index these tuples of the 2-Color code, we insert them using the pair of 2-Color labels as keys and the set of p-expressions as labels. We note that the clustering induced by the labelling scheme is merely logical. To enforce this clustering to be reflected on disk, we exploit the fact that in BerkeleyDB, insertions are *appended* to a log file and are physically stored in the order that they are inserted. Therefore, we try as much as possible to insert related p-expressions in the index in correct path sequence order.

5. QUERY PROCESSING

While our work here doesn't address subgraph pattern matching queries specifically, part of our goal was to design a system such that we could easily integrate pattern matching query processing in our system. Therefore, our overall query processing strategy is designed to have two distinct phases – a pattern matching phase and a path extraction phase. The discussion here concerns the latter. The primary task in the query processing phase is to utilize the labels i.e. keys, to determine subsequences of the indexed path sequence that should be retrieved from disk and then compose partial path summaries into complete ones representing all result paths. Filtering algorithms may then be applied on the complete path summaries to enumerate all or a subset e.g. (Top-K paths based on SemRank) of the paths represented. The work on dealing with constraints is still ongoing and will be discussed briefly in the conclusion section.

5.1 System Architecture Overview

We begin with a discussion of the overall architecture of the system, highlighting the components needed for the preprocessing and query processing phases. The first step in our approach is to *load* RDF Schema and data documents into internal graph data structures. Then, different *preprocessing* steps are performed on the data which produces appropriate indexes on the data for each of the querying paradigms i.e. *Pattern Matching Indexes* stored in the *Pattern Match Store* (not currently supported), *Path Index* stored in the *Path Store* and statistical and structural summaries used for *Top-k queries* stored in the *System Catalog*.

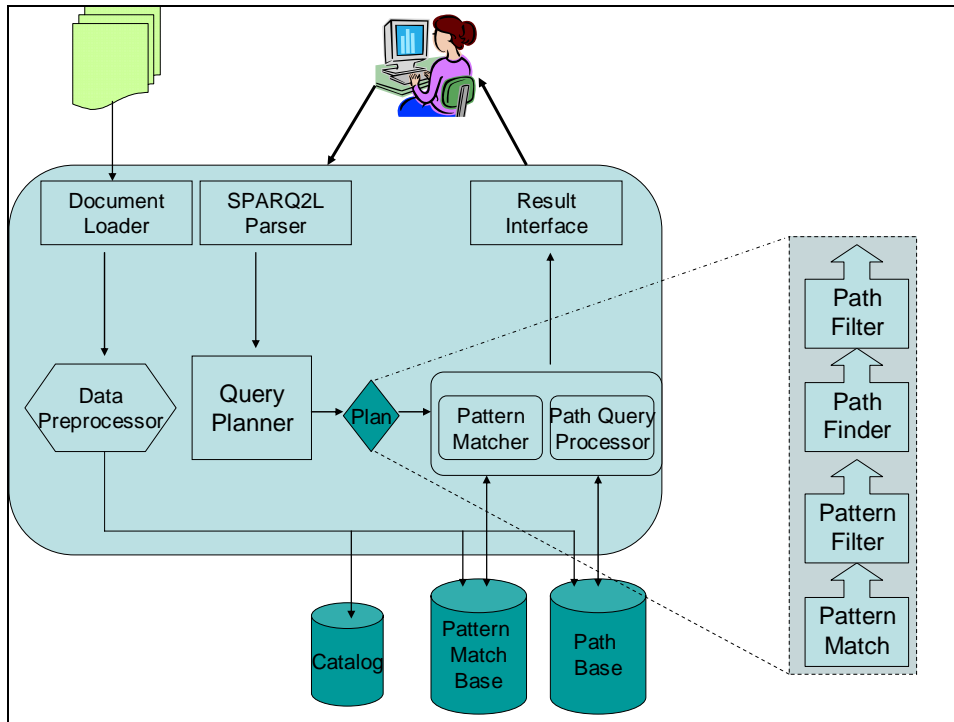


Figure 10 : System Architecture

Our storage layer uses the BerkeleyDB data storage system because of its flexibility with respect to accommodating non-relational storage models and arbitrary data types. The *Query Processor Module* consists of two kinds of query processors one for pattern matching queries and the other for path extraction queries. A query may be processed by multiple processors. For example, if a path extraction query specifies constraints described by a subgraph pattern, then the pattern match phase will be executed first and its results are fed into a subsequent path extraction query evaluation phase. The structure of the query plan looks like in this situation is shown in Figure 10.

Generally, for a path extraction query Q with constraints C , we evaluate Q and then apply a filtering algorithm on the results of Q to get those paths that satisfy C . The exception to this are cost based constraints which are evaluated directly. There are many reasons for taking this

strategy. First, we expect that users will often try multiple queries in the same context (in this case source and destination) that differ only in the specific constraints specified. For example, a user may first ask to find the shortest paths between x and y . Then, ask to find paths between x and y that contain a particular intermediate. Having computed a path summary for the unconstrained version of the query (i.e. representing all paths from x to y), we may then simply apply a different filter to extract the new set of results. This will save a lot of time because this path summary is likely to be in memory after the first query is executed, so we may not need to go back to the disk. Second, since many of the constraints supported by SPARQ2L are computationally hard, it may be helpful to focus on reducing the size of the input as much as possible. By first, computing a path summary for the unconstrained version representing all paths, we have a summary that represents a subgraph of the overall graph that is likely to be much smaller in size.

5.2 Evaluation of Unconstrained Path Extraction Queries

The Path Finder evaluates a query by successively retrieving the relevant p-expressions from disk and composing them into larger p-expressions that comprise the solution. Path Finder achieves this using the Path-Solve algorithm shown in Figure 11 below. The algorithm begins by initializing a matrix (*Result*) which keeps track of the composed p-expressions. To retrieve p-expressions from disk, the *openDBCursor* sub-routine returns a database *non-treeCursor*, *treeCursor* or *joinCursor*, depending on the subgraph in which the source and destination of the query is located.

A non-treeCursor (treeCursor) is always set to the 2-Color label for the strong component of the source (destination) node. The p-expressions needed to process the query are obtained using the

next cursor function, until the end of the cursor (i.e. the 2-Color label for the strong component of the destination (source)) is reached. To illustrate this, consider a query for paths from node 8 to node 16 in Figure 7. These nodes are in the same non-tree sub-graph thus a non-tree cursor set to 16th element in Figure 9 is returned by *openDBCursor*. In the *processNon-Tree()* sub-routine, the *next* cursor function returns the 17th to the 29th elements which are processed by calls to *processPE* sub-routine.

<pre> Algorithm 2 Path-Solve(Node s, Node d, int Result[]) 01 Result[id(s)] = ε //id(x) = α⁻¹(x) ie the id of node x 02 for each v ∈ V – {s} set Result[id(v)] = ∅ 03 cursor ← openDBCursor(s, d) 04 if (cursor is treeCursor) 05 Result ← processTree(cursor, Result, α⁻¹(d)) 06 else if (cursor is non-treeCursor) 07 Result ← processNon-Tree(cursor, Result) 08 else 09 Result ← processJoin(cursor, Result) 10 return Result openDBCursor(Node s, Node d) 01 if ((s & d) ∈ same non-tree sub-graph) 02 return non-treeCursor(s, d) 03 else if ((s & d) ∈ same tree sub-graph) 04 return treeCursor(s, d) 05 else if (s ∈ sub-graph g & d ∈ a dangling tree t of g) 06 return joinCursor(s, c, d) //c is the cut vertex of g ∪ t 07 else return null. </pre>	<pre> processNon-Tree(cursor, int Result[]) 01 while ((X_i, v_i, w_i) ← cursor→next()) <> null) 02 Result = processPE((X_i, v_i, w_i)) 03 return Result processTree(cursor, int Result[], int dest) 01 Result[dest] = ε 02 while ((X_i, v_i, w_i) ← cursor→next()) <> null) 03 Result[dest] = processPE((X_i, v_i, w_i)) • Result[dest] 04 (X_i, v_i, w_i) ← cursor→prev() 05 Result[dest] ← X_i • Result[dest] 06 cursor→set(v_i) 07 return Result processPE(int Result[], p-expression (X_i, v_i, w_i)) 01 if (v_i = w_i) then Result[w_i] ← Result[v_i] • X_i 02 else Result[w_i] ← Result[w_i] ∪ Result[v_i] • X_i 03 return Result </pre>
--	--

Figure 11 : Path Solve Algorithm

On the other hand, *processTree* proceeds in a bottom-up manner, successively using the *next* and *prev* cursor functions to obtain the relevant p-expressions. Since p-expressions obtained from a *treeCursor* begin with the destination up to the source, p-expressions of tree edges (obtained with the *prev* cursor function) are prepended to the p-expression in $Result[\alpha^l(d)]$.

Given *Result* and a p-expression (X_i, v_i, w_i) , *ProcessPE* computes a larger p-expression (X_i, u_i, w_i) by appending (X_i, v_i, w_i) to an existing p-expression in *Result* of type (X_j, u_i, v_j) (line 01). Further, a larger p-expression is computed as the union of any p-expression (X, u_i, w_i) which already exists in *Result*, but only if v_i is different from w_i . As an example, consider again the query for all paths from node 8 to node 16th. $Result[8]$ initially contains ϵ , as shown in Figure 12a. To process the first p-expression $(enrolled_in, 8, 10)$ of the 16th element, $Result[8]$ is concatenated to *enrolled_in* and stored in $Result[10]$, as shown in Figure 12b. Processing $(advises \bullet enrolled_in, 9, 10)$ implies concatenating $Result[9]$ to $advises \bullet enrolled_in$ and storing to $Result[10]$. However $Result[9]$ is null, thus $Result[10]$ remains unchanged. Figure 12c-j, show the changes made to *Result* as the rest of the p-expressions are processed.

A *joinCursor* retrieves p-expressions like a *treeCursor* until it meets p-expression for the cut vertex, after which it switches to a non-*treeCursor* behavior. To illustrate, consider a query for paths from node 11 to node 20. Node 11 is in a non-tree sub-graph which has a dangling tree that contains node 20. Thus, *openDBCursor* returns a *joinCursor* set to the 37th element in Figure 9. Its associated p-expression is empty, so that $Result[20]$ is unchanged. A call to the *prev* cursor function returns the p-expression $(project_in, 19, 20)$, which is prepended to ϵ , so that *project_in* is stored to $Result[20]$.

(a)	8 ε	(b)	10 enrolled_in	(c)	10 enrolled_in •(taught_by•advises•enrolled_in)*
(d)	9 (enrolled_in•(taught_by•advises•enrolled_in)*)•taught_by				
(e)	8 (((enrolled_in•(taught_by•advises•enrolled_in)*)•taught_by)•advises)*				
(f)	13 (((enrolled_in•(taught_by•advises•enrolled_in)*)•taught_by)•advises)*•author_of				
(g)	14 (((enrolled_in•(taught_by•advises•enrolled_in)*)•taught_by)•advises)*•author_of				
(h)	15 (((enrolled_in•(taught_by•advises•enrolled_in)*)•taught_by)•advises)*•enrolled_in				
(i)	16 ((((enrolled_in•(taught_by•advises•enrolled_in)*)•taught_by)•advises)*•author_of)•has_subject_area				
(j)	16 ((((enrolled_in•(taught_by•advises•enrolled_in)*)•taught_by)•advises)*•author_of)•has_subject_area \cup ((((enrolled_in•(taught_by•advises•enrolled_in)*)•taught_by)•advises)*•enrolled_in)•course_in				

Figure 12: An illustration of the Path-Solve

The cursor is then set to the 35th element, which also contains an empty p-expression. The next call to *prev* returns the 34th element with the p-expression $(current_project, 11, 19)$, which is prepended to *project_in* and stored back to *Result[20]*. Since this element represents the bridge edge, the cut vertex is noted and *set* sets the cursor to the 2-Color label of the source of the query, which is the 20th element. Since its associated p-expression is empty, no change is made to *Result[11]*. Since this element is the cut vertex, the result of the query is obtained with a final concatenation of *Result[11]* and *Result[20]*.

At the end of the Path-Solve algorithm, $Result[i]$ contains a p-expression of type $(X_i, \alpha^{-1}(s), w_i)$, representing all paths from node s to node i .

5.3 Representation of Path Summaries.

So far, we have discussed path summaries in terms of regular expressions. However, our implementation uses a tree representation called a *Path Expression Tree* or *PETree*.

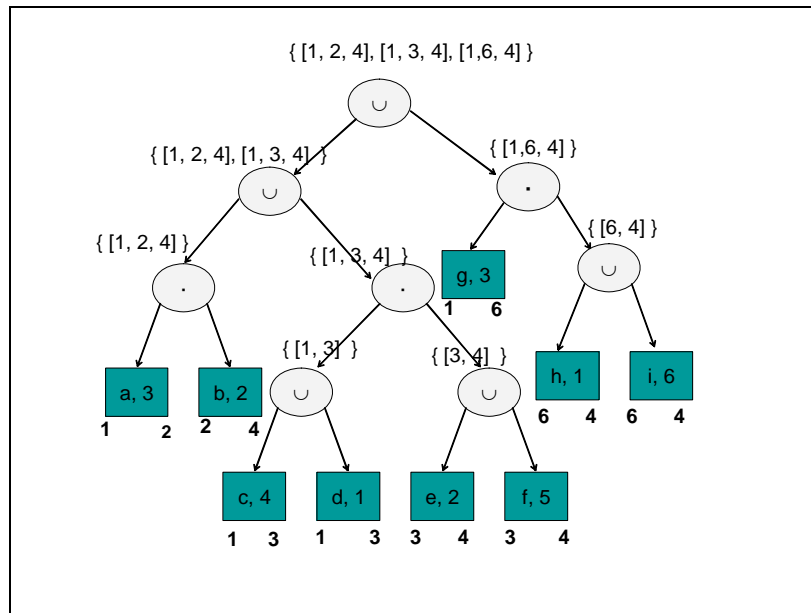


Figure 13 : An Example PETree

Basically, a PETree is an abstract syntax tree representation of paths from v_1 to v_2 , with internal nodes as operator nodes, concatenation (\bullet) and union (\cup) operators, and the leaves are the edges i.e. properties, on paths connecting v_1 to v_2 . An example of a PETree representing paths from node 1 to node 4 is shown in Figure 13. It represents 7 paths, all of length 2, $\{ a \bullet b, c \bullet e, c \bullet f, d \bullet e, d \bullet f, g \bullet h, g \bullet i \}$ from node 1 to node 4. Each internal node has two child subtrees and each leaf node has two nodes associated with it (only resource ids shown) which correspond to the

subject and *object* that are linked by the property. In the figure, they are labeled with sets of nodes that form paths they represent. The *subject* of a (●) internal node is subject of all its leftmost child leaf nodes and its object is the *object* of all its rightmost child leaf nodes. For example, the middle (●) internal node {labeled (1, 3, 4)}, has two leftmost (c and e) and rightmost leaf (d and f) nodes. Its subject is the subject of the c and e node which is the resource with id 1 and its object is the object of the d and f nodes, the resource with id 4.

The semantics of a (●) internal node is that the left child subtree represents all subpaths that may precede any of the subpaths represented in the right child subtree.

In general, the filtering phase involves traversing this tree and performing a set of operations to extract the desired paths. The section on ranking will demonstrate how this is done for extracting top-k paths.

5.4 Performance Evaluation

In this section, we describe an empirical evaluation of our query processing approach by comparing the performance when using our 2-Color Code (2CC) vs. five other several other randomly chosen topological orderings.

5.4.1 Implementation

We implemented our algorithms using Java 1.5, on a 1.8GHz Dual AMD Opteron processor with 10GB available RAM. We used Berkeley DB Java Edition for storage and indexing and performed all matrix implementations using the sparse matrix implementation of the Colt distribution. We used Brahms [38], an efficient main-memory storage to obtain a temporary graph representation of the RDF graphs in memory.

5.4.2 Datasets

We used a real world SwetoDBLP-Jan2006 [75] dataset and a synthetic dataset generated using the Lehigh University Benchmark with 6 Universities (UBA6). Table 2 below shows the properties of the datasets. SwetoDBLP has 9,921 non-tree sub-graphs with a total of about 300,000 scc nodes and 760,000 scc edges and 340,000 tree sub-graphs with a total of about 410,000 scc nodes. One of these non-tree subgraphs is very large (about 250,000 nodes and 660,000 edges) and the smallest sub-graph contains 2 scc nodes and 2 scc edges. It also contains about. The smallest and largest tree sub-graphs have a single scc node and 25 scc nodes respectively, with a maximum depth of 1. UBA6 however is more connected, containing a single non-tree subgraph with 118,195 strongly connected component (scc) nodes and 357,578 scc edges. Although it contains 61 tree subgraphs, each tree contains just a single scc node. Literal nodes and incident edges are ignored.

Table 2: Properties of the Datasets

	UBA6	SWETO_DBLP
Number of nodes	118,566	724,874
Number of edges	357,950	836,555
# of strong components	118,256	723,669
#of p-expressions	476,448	1,561,008

5.4.3 Performance Metrics

We evaluate the performance of the techniques by observing 1) the size of the reduced path sequence i.e. the number of p-expressions brought into memory from disk. This metric measures the goodness of our approach for identifying irrelevant p-expressions – those that were not retrieved from disk, 2) the query processing time including disk access time.

5.4.4 Query Workload

Our query workload consists of six different query types. First we have as positive (path exists) and negative (paths do not exist) and we denote positive or connected queries as (C-Queries) and negative or disconnected queries as (D-Queries). Then we identify queries based on whether their source/destination nodes are in tree or non-tree subgraphs. Queries with source and destination nodes in non-tree sub-graphs are denoted (NT-NT) queries, (NT-T) denotes source node is in a non-tree sub-graph and the destination node in a tree sub-graph and (T-T) denotes queries with both source and destination nodes in tree sub-graphs. We randomly selected 40 distinct source-destination pairs for each of the six categories and measured the average running time of all queries where the running time of a query is also an average over several executions of the query.

5.4.5 Experimental Results

Figure 14 - Figure 25 show the result of our experiments on the SwetoDBLP dataset. 2CC had the best performance for all the query workloads. The best performance of 2CC for C-Queries is observed in the T-T queries where only 4 p-expressions were brought into memory, with a total of 0.4 milliseconds query processing time. This is expected since the T-T queries for this dataset are single edge paths. For the D-Queries, 2CC performed very well using at most 0.025 milliseconds to determine that the result set is null. For these queries, no p-expression was

brought into memory. As was expected the performance of the other labeling schemes varied with the queries as seen in Figure 14 – Figure 15 and Figure 18 - Figure 19.

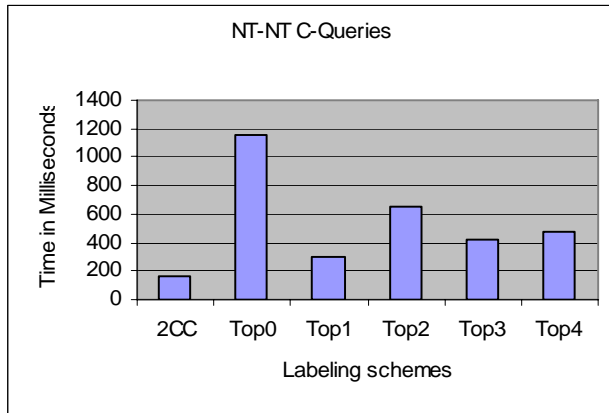


Figure 14: NT-NT C-Queries (Time)

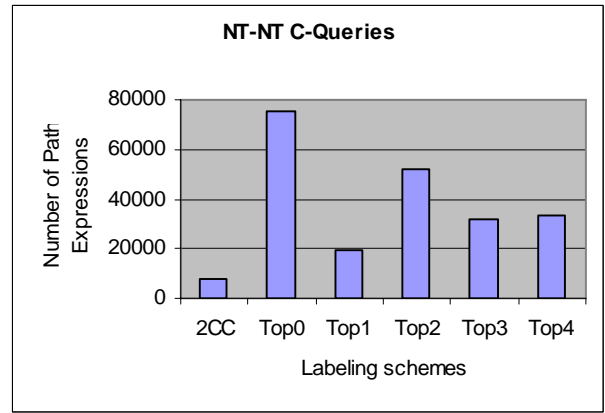


Figure 15: NT-NT C-Queries (#p-expressions)

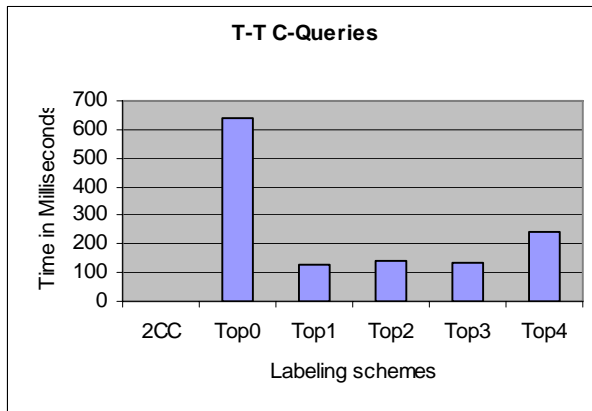


Figure 16 : T-T C-Queries(Time)

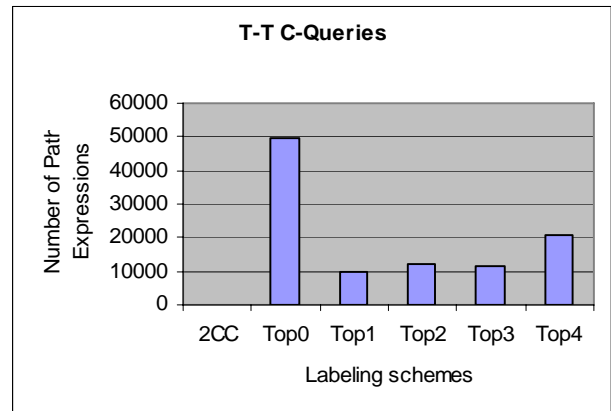


Figure 17 : T - T C-Queries (#p-expressions)

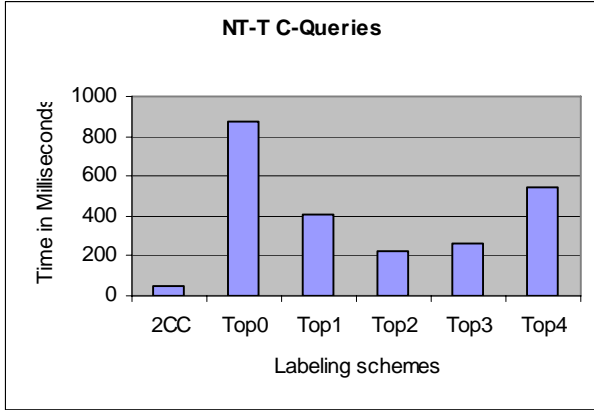


Figure 18 : NT - T C- Queries (Time)

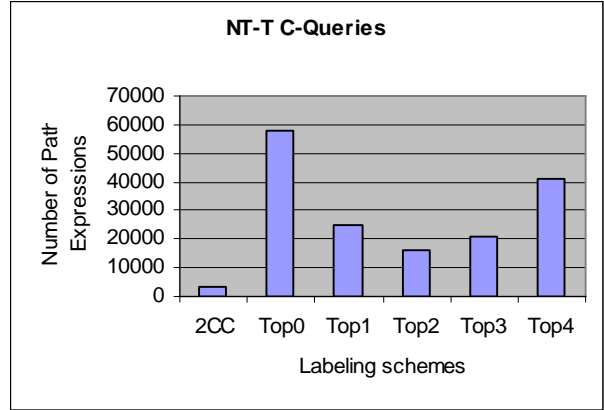


Figure 19: NT - T C-Queries (#p-expressions)

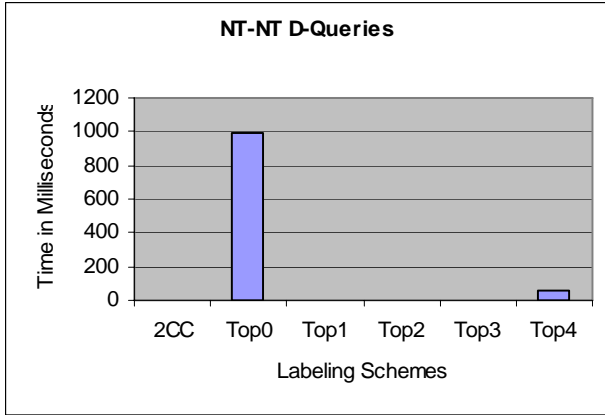


Figure 20 : NT - NT D-Queries (Time)

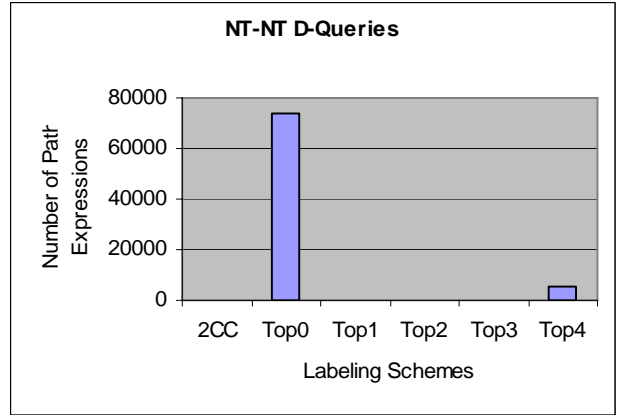


Figure 21 : NT - NT D - Queries(#p-expressions)

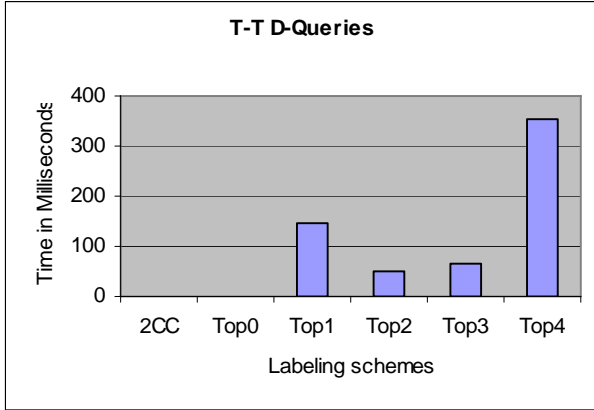


Figure 22 : T- T D-Queries (Time)

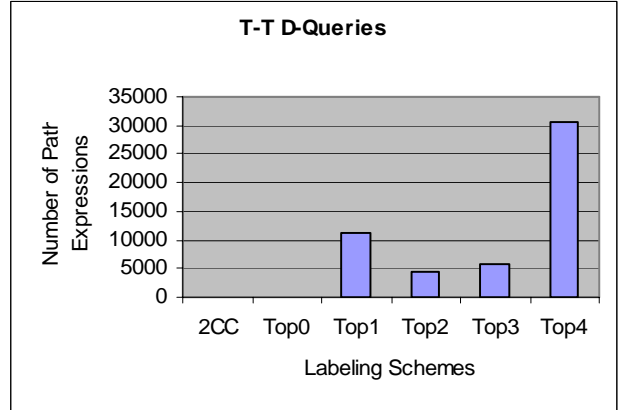


Figure 23 : T - T D-Queries (#p-expressions)

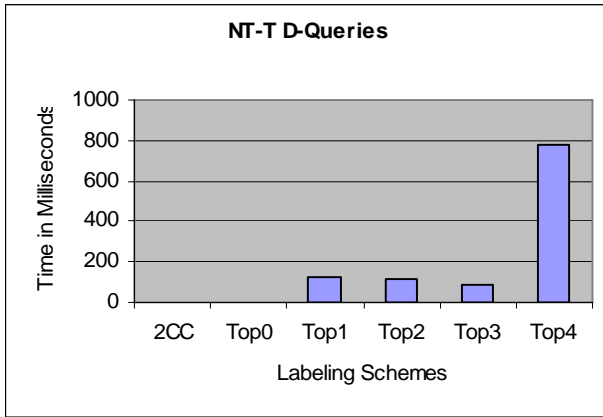


Figure 24 : NT - T D-Queries (Time)

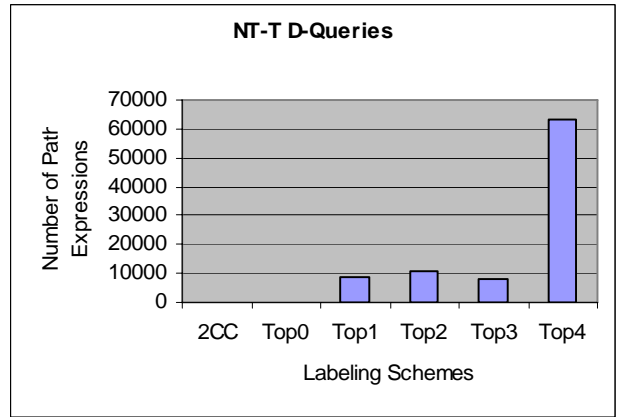


Figure 25 : NT - T D-Queries (#p-expressions)

While Top1 had the best performance amongst the topological labeling schemes for the NT-NT C-Queries, Top2 performed best for the NT-T C-Queries. Top0 performed worst for most of the query workloads but had the best performance in the NT-T D-Queries and T-T D-Queries (Figure 22 - Figure 25). We note that although performance of 2CC surpasses that of all the other topological labeling schemes, the ratio of the time performance is always larger than the ratio of

the size performance. This is as a result of the additional time the 2CC spends in pruning the p-expressions from the reduced path sequence based on properties PropertyNR and PropertyOP.

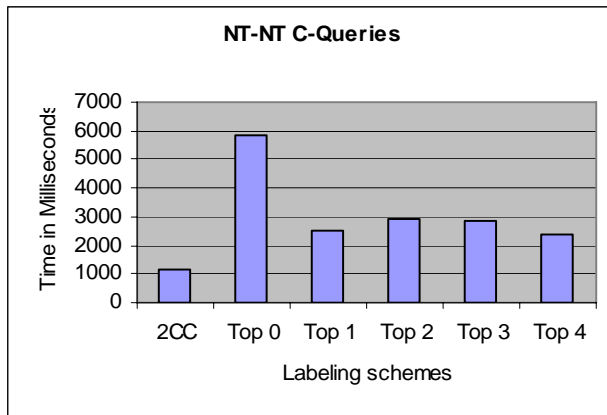


Figure 26 : NT - NT C-Queries (Time)

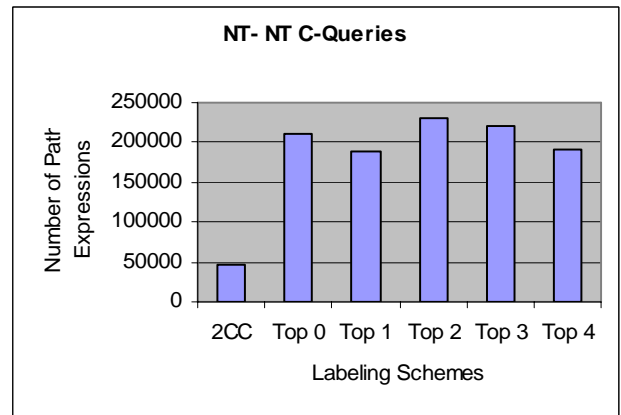


Figure 27 : NT - NT C - Queries (#p-expressions)

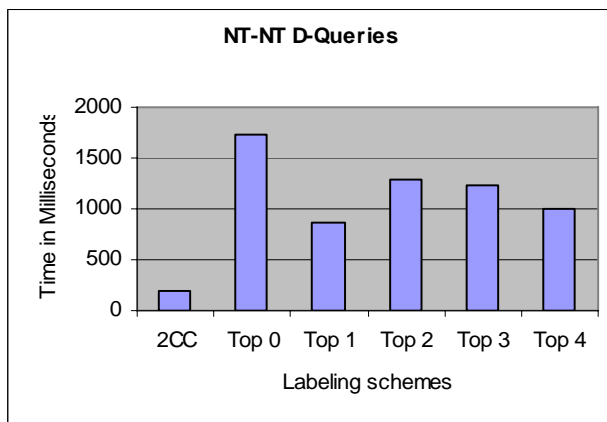


Figure 28 : NT - NT D-Queries (Time)

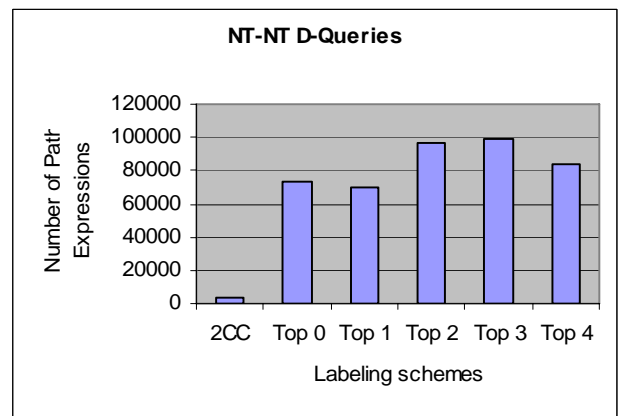


Figure 29 : NT - NT D-Queries

Figure 26 - Figure 29 show the results of our experiments on UBA6. As we mentioned earlier, the tree sub-graphs in UBA consist of only single nodes, so that T-T queries are meaningless for

this dataset. Furthermore, NT-T queries translate to either (a) finding the bridge edges or (b) finding paths through the bridge edges. Our experiments on the SwetoDBLP dataset showed that 2CC performs well for single edge paths. We observed a similar performance for this dataset and omit the results on the NT-T queries which can be inferred from the performance of Figure 27 the NT-NT queries. Figure 26 and Figure 27 show the performance of the labelling schemes for the NT-NT C-Queries. Again, 2CC performs best amongst all the labeling schemes. However, its performance on UBA6 is worse than on SwetoDBLP because UBA6 is a more connected graph, having only 1 non-tree sub-graph as opposed to SwetoDBLP which is fragmented into 9, 921 non-tree sub-graphs. Thus, query results in this workload have many more and longer paths. This is also reflected in the results of the NT-NT D-Queries shown in Figure 28 - Figure 29. Although 2CC also performed best, there were some queries for which determining non-reachability required more than a constant time check using labels leading to an average performance of 188 milliseconds for processing 3927 p-expressions. The disparity in the ratios of the time and size performances of 2CC to the other labelling schemes is also evident in the results. In spite of this, the time performance of 2CC is at least half of the time performance of the best topological labelling scheme (Top1) for both the C-Queries and the D-Queries.

6. RANKING RESULTS OF PATH EXTRACTION QUERIES

As mentioned earlier, traditional approaches for ranking information items are “query dependent” in that they rely on determining the best “match” to the query. The challenge with developing a query independent ranking model for path extraction query results is selecting the metrics to be used in the model. It would be fairly straightforward to choose metrics such as length of path or frequency of paths e.g., by shortest paths, longest paths, least frequently occurring paths, etc as the basis for ranking. However, each of these approaches makes an assumption about what is most relevant for *every* situation. In our experience, we have found that different applications have different needs and making assumptions that fix the ranking criteria may be limiting.

Our approach is based on the premise that generally searches fall into two broad categories. On one hand there are searches performed for investigative purposes which focus on uncovering things that are unusual – *Discovery Searches*. On the other, there are searches performed which are most interested in typical or conventional data items – *Conventional Searches*. Take for example the database in Figure 30. The top part of the figure shows four schema for four domains *University*, *Banking*, *Flight* and *Organization*, while the bottom part shows a set of resources described using those schemas i.e. the database. The links from the bottom part of the figure to the top part are the type links. In the RDF model, resources may belong to multiple types that are not related by inheritance.

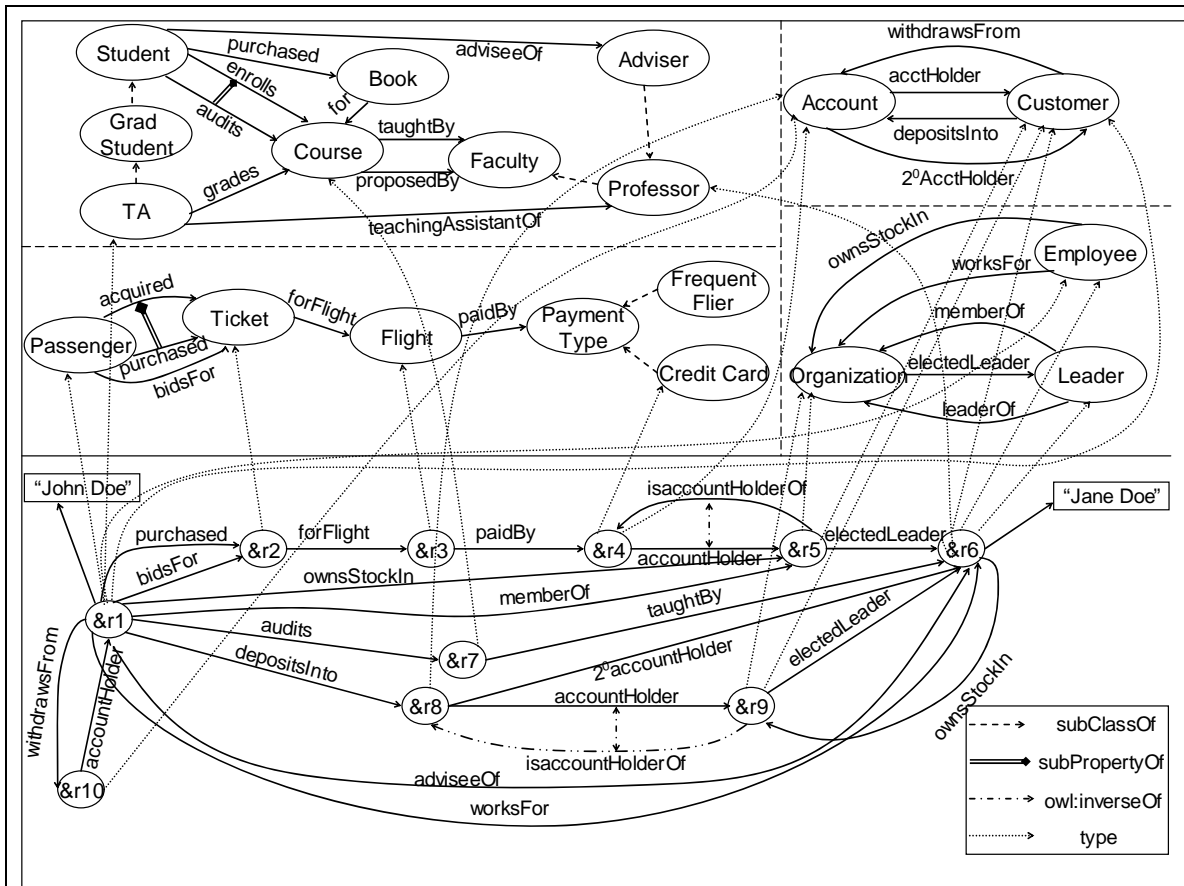


Figure 30: An example RDF knowledge base

Table 3 shows two path associations between resources $r1$ and $r6$, a student and her professor. The first row shows a path that connects $r1$ to $r6$ via the purchase of a ticket that was paid for by an organization (also an account holder) who is a leader of $r6$. The second row shows a relationship where $r1$ is directly linked to $r6$ through an advisee relationship.

Table 3: Example Semantic Associations

1	$\&r1 \xrightarrow{\text{purchased}} \&r2 \xrightarrow{\text{forFlight}} \&r3 \xrightarrow{\text{paidBy}} \&r4$ $\xrightarrow{\text{accountHolder}} \&r5 \xrightarrow{\text{leader}} \&r6$
2	$\&r1 \xrightarrow{\text{adviseeOf}} \&r6$

The determination of which association is most important may depend on the application. For investigative types of scenarios the first association may be of the most interest, whereas for causal browsers performing a conventional search the later might be of more interest.

6.1 Philosophy of Ranking Model

Due our philosophy of discovery vs. conventional type searches, our ranking model is based on metrics that measure the predictability of a query result. This way, a query result that is highly predictable can be ranked highest when a conventional search is performed whereas the least predictable results can be ranked highest when a discovery type search is performed. Also, as mentioned earlier, users may modulate from one type of search to the other which alters the ranking to the appropriate type. The modulation in search types is done using a sliding bar (see Figure 31).

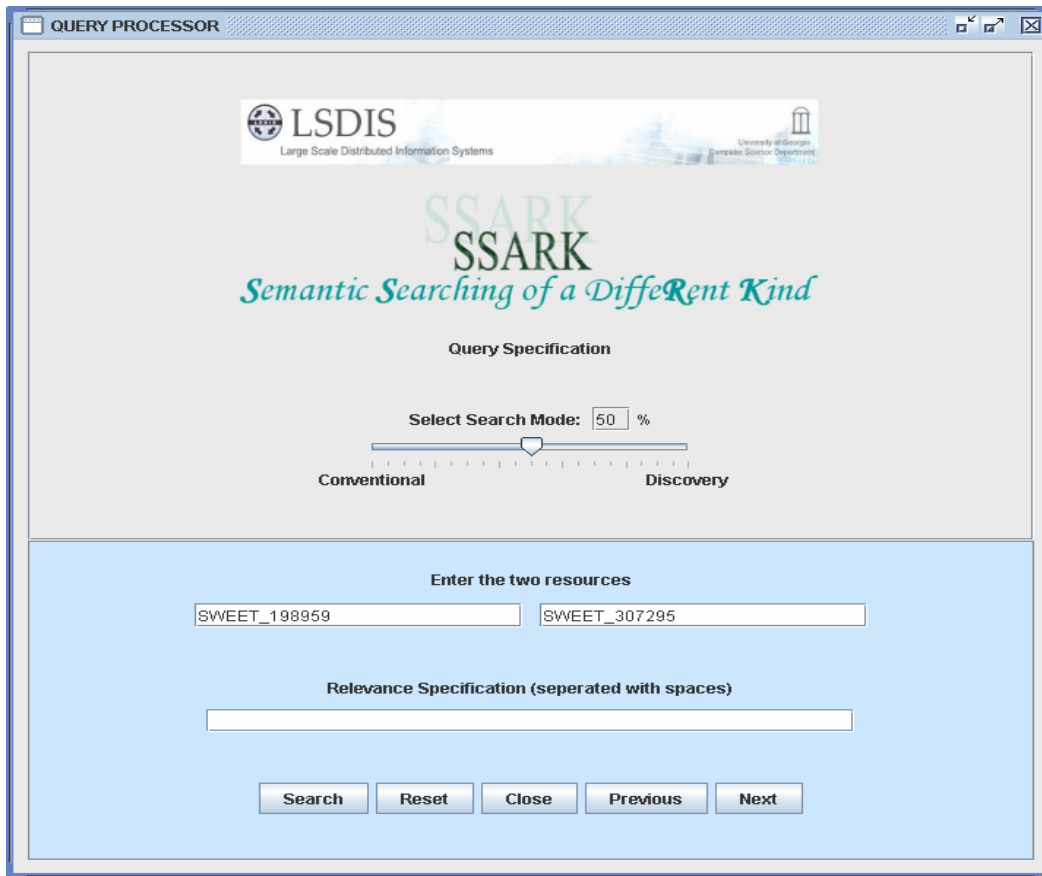


Figure 31: Query Interface

With respect to measuring predictability we have two metrics. The first is based on an information theoretic concept that measures the information content of a result. In other words, how much information a user would gain by being informed about the existence of the result. This is closely related to the likelihood that a user could have guessed that such an association exists or the predictability of the association. The second metric measures predictability based on how discrepant the structure of the result is from the possibilities that can be gleaned from the schema. The idea is that typically a schema that models a domain is expected to capture much of what is expected in terms of relationships in a domain. As we shall see later, the flexibility of the RDF model allows information to be captured that is not represented by the schema leading to situations where paths at the data layer that cannot be predicted by just looking at the schemas..

This means that even when users have access to a domain model they may not necessarily be able to predict all of the reality in the domain. For example in Figure 30, the path type made up of the edge sequence or *property sequence* `purchased•forFlight•paidBy•accountholder•electedLeader` does not occur anywhere at the schema layer but occurs in a path from *r1* to *r6*. Therefore, our model would rank such paths high for discovery searches. The third component of our model is based on the realization that users may have some bits and pieces of information that they can provide to a query engine to help describe their search need. We allow such input in the form of keywords. If keywords are provided, then a match value is determined based on the semantic proximity between the keywords and entity and relation types in a path result.

The third unique component of our approach is that it allows users to adjust or “modulate” the ranking of their results. This allows users to look at their results from different perspectives and also enhances the chances of allowing them stumble upon previously unknown information. The next section will elaborate on the philosophy behind our ranking model.

6.2 Preliminaries

We need to extend the graph model in Chapter 3 to include explicit links between the schema layer and the data layer i.e. typing of entities and relations. The RDF model is unique in that it allows inheritance on both classes and property (relations). So that one relation type can be define to be a specialization of on another. For example, in Figure 30 we have that “audits” property that links a student to a course is a specialization of the “enrolls” property. The link between entities and their classes as well as property types and their instances is represented by

an interpretation function $\llbracket \cdot \rrbracket$ which maps a class or property to its instances. We also define the following auxillary interpretation functions:

For a class c and a property p , $\llbracket \cdot \rrbracket$, $\llbracket \cdot \rrbracket^\wedge$, $\llbracket \cdot \rrbracket^+$ are functions on c and p such that:

- i) $\llbracket c \rrbracket^\wedge = \{ r \mid r \text{ is } \textit{rdf:type} \text{ of } c \}$ i.e. only proper instances of c
- ii) $\llbracket p \rrbracket^\wedge = \{ [r_1, r_2] \mid r_1 \in \llbracket c_i \rrbracket^\wedge, c_i \in p.\textit{domain}() \ \& \ r_2 \in \llbracket c_j \rrbracket^\wedge, c_j \in p.\textit{range}() \}$ i.e. only proper instances of p
- iii) $\llbracket c \rrbracket^+ = \llbracket c \rrbracket^\wedge \cup \{ \llbracket c' \rrbracket^\wedge \}$ where c is *subClassOf* c' i.e. proper instances of c and the superclasses of c .
- iv) $\llbracket p \rrbracket^+ = \llbracket p \rrbracket^\wedge \cup \{ \llbracket p' \rrbracket^\wedge \}$ where p is *subPropertyOf* p' i.e. proper instances of p and the superproperties of p .
- v) $\llbracket c \rrbracket = \llbracket c \rrbracket^\wedge \cup \{ \llbracket c' \rrbracket^\wedge \}$ where c' is *subClassOf* c i.e. proper instances of c and the subclasses of c .
- vi) $\llbracket p \rrbracket = \llbracket p \rrbracket^\wedge \cup \{ \llbracket p' \rrbracket^\wedge \}$ where p' is *subPropertyOf* p i.e. proper instances of p and the subproperties of p .

To make the rest of the discussion easier, we will use an alternate notation for path. This will define a path not as a sequence of triples but as a sequence of properties i.e. binary relations that form the path.

Definition 18 (Property Sequence) A *Property Sequence* $PS = P_1, P_2, \dots, P_n$ is a finite sequence of properties whose interpretation is given by:

$$\llbracket PS \rrbracket \subseteq \times_{i=1}^n \llbracket P_i \rrbracket \text{ where for } ps \text{ an instance of } PS, \text{ i.e. } ps \in \llbracket PS \rrbracket,$$

1. $ps[i] = [r_1, r_2] \in [[P_i]]$ for $1 \leq i \leq n$. (the notation $ps[i][0]$ and $ps[i][1]$ refer to r_1 and r_2 respectively.)
2. $ps[i][1] = ps[i+1][0]$.

6.3 Information Gain and ρ -Path Semantic Associations

In information theory, the amount of information contained in an event is measured by the negative logarithm of the probability of occurrence of the event. Thus if χ is a discrete random variable or an event that has outcomes or possible values x_1, x_2, \dots, x_n occurring with probabilities pr_1, pr_2, \dots, pr_n , i.e. $\Pr(\chi = x_i) = pr_i$, with $pr_i \geq 0$ and $\sum_{\text{all } i} pr_i = 1$, the amount of information gained or uncertainty removed by knowing that χ has the outcome x_i is given by

$I(\chi = x_i) = -\log pr_i$. The maximum information content of χ is attained when $pr_i = 1/n$ for all i .

This is given by $I(\chi) = \log n$.

Based on this we can build a model for measuring the information content of a semantic association by considering the occurrence of edge as an event and RDF properties as its outcomes. We begin with defining the notion for a property and then extend it to a sequence of properties of a path. Assume that P is the set of all property types in a description base and χ is a discrete random variable with sample space $[[P]]$. Then for any valid property $p \in P$, the probability that $\chi = p$ is given by

$$\Pr(\chi = p) = \frac{|[[p]]^\wedge|}{|[[P]]^\wedge|}$$

We refer to this probability as the specificity SP of the property p . The specificity of a property is a measure of its uniqueness relative to all other properties in the description base. The

information content $I(\chi = p)$ of the occurrence of a property p in the description base due to its specificity is: $I_S(p) = I(\chi = p) = -\log \Pr(\chi = p)$

It is possible to develop a similar measure which exploits the semantics of RDF and RDFS. Given that RDF resources are typed, any two resources $r1$ and $r2$ have a finite number of valid properties from that may connect them. Using this information, we can estimate the information content of a property p linking $r1$ and $r2$ with respect to only the valid properties as possibilities. What we then expect is that information content will be larger in situations where the number of valid properties is large and smaller in situations with fewer valid properties.

The valid properties that can link two resources include those that have been explicitly defined in a schema and those that may be inferred from the semantics of RDFS. In particular, the semantics of multiple domains/ranges on a property p implies that a resource must belong to all the domain/range classes even if that membership is not explicitly stated. We introduce the concept of a Representative Ontology Class (ROC) as a concise summary of related classes by virtue of the equivalence of their interpretations. For example, in Figure 30 the classes Book and Ticket belong to an ROC which represents the set of things that can be purchased. We now make this notion more precise.

Definition 19 (Representative Ontology Classes) For an ontology O with the set of classes C and properties P and $|C| = n$, let S be a $n \times n$ matrix with the following entries:

$$S_{ij} = \bigcup_{p, c_i \in \text{domain}^{\wedge}(p) \wedge c_j \in \text{range}^{\wedge}(p)}$$

where $\text{domain}^{\wedge}/\text{range}^{\wedge}$ refer to classes in the proper domain/range of a property (i.e. excluding their subclasses). We can define an equivalence relation \sim such that:

$$\sim(i, j) \text{ iff } S_{ik} = S_{jk} \text{ and } S_{ki} = S_{kj} \text{ for all } k.$$

\sim partitions the elements of S into the set L of equivalence classes of \sim , where each equivalence class represents the set of classes that are equivalent with respect to their outgoing and incoming properties. It corresponds to the set of classes that should have the same interpretations. Each $l \in L$ is called a *Representative Ontology Class* (ROC). The set of all possible properties semLinks that can directly connect two ROCs X and Y is given by $\text{semLinks}(X, Y) \rightarrow S_{i(X),i(Y)}$

where $i(l) \in \{i: C_i \in l\}$. $i(l)$ is called a *representative* for the ROC l and C_i is called a member of l . Since the members of each equivalence class are equivalent with respect to their outgoing and incoming properties, then it suffices to pick a representative class for X and Y , say $C_{i(X)}$ and $C_{i(Y)}$ respectively.

Now, given any two resources $r1$ and $r2$, we can measure the probability distribution of the types of properties that can connect them in the instance base. If we let π be the set of all possible properties that may connect $r1$ and $r2$, then π clearly depends on the types of $r1$ and $r2$. If C_1, C_2, \dots, C_m and D_1, \dots, D_n are the classes of $r1$ and $r2$ respectively and if X_1, X_2, \dots, X_k , and Y_1, Y_2, \dots, Y_p are ROCs that C_i and D_j belong to respectively, then

$$\pi = \cup_{\text{semLinks}(X_i, Y_j)} \text{ and } \theta = \bigcup \{ [[P]]^+ \mid P \in \pi \}$$

θ represents the interpretation of all the valid properties that may connect $r1$ and $r2$. Thus, the probability that $\chi \in \theta$ is given by:

$$\Pr(\chi \in \theta) = \frac{|\theta|}{|[[P]]^\wedge|}$$

For a given valid property p in the description base, if $\chi \in \theta$, the probability that $\chi = p$ is given by:

$$\Pr(\chi = p | \chi \in \theta) = \frac{\Pr(\chi = p, \chi \in \theta)}{\Pr(\chi \in \theta)} = \frac{|[[p]]^\wedge|}{|\theta|}$$

We refer to this probability as the θ -Specificity SP_θ of property p . The θ -specificity of a property is a measure of its uniqueness relative to all other properties in the description base whose domain and range belong to the same ROC's respectively. The information content of the occurrence of a valid property p $I(\chi=p | \chi \in \theta)$ in the description base due to its θ -Specificity can then be defined as

$$I_{\theta-s}(p) = I(\chi=p | \chi \in \theta) = -\log \Pr(\chi = p | \chi \in \theta).$$

To illustrate these concepts, for the property “purchased” connecting r_1 and r_2 in Figure 30 above, $ROC_1 = \{\text{Student, Passenger}\}$, $ROC_2 = \{\text{Book, Ticket}\}$ with $\theta = \{[[\text{purchased}]]^+, [[\text{bidsFor}]]^+\}$, so that the size of $\theta = |[[\text{purchased}]]^\wedge| + |[[\text{acquired}]]^\wedge| + |[[\text{bidsFor}]]^\wedge|$. If there are 20, 40, 80 instances of the properties purchased, acquired and bidsFor respectively and a total of 1000 property instances in the description base, then the specificity of purchased is 0.02 while its θ -specificity is 0.143.

We can extend these ideas to capture the information content of a ρ -path association. Let $PS = p_1, p_2, \dots, p_n$, be a property sequence and $ps \in [[PS]]$ a path. It is clear that ps occurs as frequently as the *least* frequently occurring property p_i in PS . We define the information content

of ps due to its specificity as $I_S(ps) = \max_{\forall i} \{I_S(p_i)\}$. Intuitively, this means that a path is as informative as its most informative edge with respect to the entire description base.

For the information content of ps due its θ -specificities we must somehow combine the different θ -specificities of the various properties on the path. However, since the θ -specificities of the edges are based on different probability distributions, we cannot meaningfully compared them

with one another, so we must first normalize the values by taking the ratio of the observed information content to the maximum possible information content (as described earlier). This results in normalized θ -specificity values, $NI_{\theta-S}$. It is important that our combination function doesn't bias towards longer paths, therefore a sum function is not a good combination function. Also, we must ensure that the combination function distinguishes between paths with more uniform θ -specificity distributions than those with non-uniform θ -specificity distributions. To see why this is so, take for example two paths ps_1 and ps_2 that have the same average $NI_{\theta-S}$, but ps_1 has θ -specificity values for all its edges about equal while ps_2 has a range of θ -specificity values for its edges from low to high. Then it seems that ps_2 that has some edges with low information content (i.e. some weak links) which should be easier to predict than ps_1 which has all its edges with an equal level of predictability. Therefore, to measure the information content of ps with respect to the θ -specificity of the properties p_1, p_2, \dots, p_n , we modify the value gotten from the average of the θ -specificities to:

$$I_{\theta-S}(ps) = \min_{\forall_i} \{NI_{\theta-S}(p_i)\} + \frac{\left(\sum_{\forall_i} NI_{\theta-S}(p_i) \right) - \min_{\forall_i} \{NI_{\theta-S}(p_i)\}}{n-1}$$

This implies that information content due to θ -specificity is a combination of the information gained from the weakest point along the path and an average of the rest.

Finally, to get the total information gained by knowing a path occurred we combine the values of information content due to both specificity and θ -specificity:

$$I(ps) = I_S(ps) + I_{\theta-S}(ps).$$

6.4 Refraction

As mentioned earlier, the multiple classification of resources allowed in the Semantic Web data models like RDF, can create paths at the description layer the do not occur at the schema layer, especially when multiple schemas are used to describe a set of resources. We use the term Refraction to refer to a deviation from a path's representation at the schema layer. In other words, a description layer path starts and proceeds along exactly as described at the schema layer and then changes direction or refracts at some point.

In order to make this notion of refraction more precise, we need a representation of all the paths that are possible based on what is explicitly defined in or inferable from a schema. Then for a given path in the description base, any sequence of edges not represented would be considered a refraction due to a multiple classification of a node. We propose the notion of a Semantic Summary as such a representation. It is analogous to the concept of DataGuides [36] and other structural summaries [54] used to optimize the evaluation of path expression queries in semi-structured data models. A semantic summary is a graph in which the vertices are ROCs.

Definition 20 (Semantic Summary) A Semantic Summary for an ontology O with sets C/P of classes/properties is a graph $G_S = (V_S, E_S, \lambda, <)$

1. V_S is the set of ROCs of O as defined in Definition 1.
2. $E_S = \{(x, y) \mid S_{i(x), i(y)} \neq \emptyset \text{ and } x, y \in V_S\}$
3. $\lambda : E_S \rightarrow 2^P$
 - i. $(x, y) \in E_S, \lambda(x, y) = \text{semLinks}(x, y)$
4. $<$ is a subsumption relation on nodes in V_S such that for two ROCs x and y , x is subclassOf y if for $c_i \in x, \exists c_j \in y$ such that c_i subclassOf c_j .

Two edges (u, v) and (w, x) of a semantic summary are said to be adjacent if $v = w$.

Given a semantic summary $G_S = (V_S, E_S, \lambda, <)$ and a property sequence $PS = p_1, p_2, \dots, p_n$, if $e_i, e_j \in E_S$ and e_i and e_j are not adjacent in G_S and $p_i \in \lambda(e_i)$ and $p_{i+1} \in \lambda(e_j)$, then we say that there is a refraction from p_i to p_{i+1} . Formally, for a path sequence $PS = p_1, p_2, \dots, p_n$, $\text{refraction}(p_i, p_{i+1})$

$$= \begin{cases} 1 & \text{if } \exists e_i, e_j \text{ such that } e_i \text{ is adjacent to } e_j \wedge p_i \in \lambda(e_i) \wedge p_{i+1} \in \lambda(e_j) \\ 0 & \text{otherwise} \end{cases}$$

We use the term refraction count RC to refer to the number of refractions on a path, given by

$$\text{RC}(PS) = \frac{\sum_{i=1}^{n-1} \text{refraction}(p_i, p_{i+1})}{n-1} \quad \text{for } n \geq 2, \quad 0 \text{ otherwise.}$$

For example, in Figure 30, the path $p_{r1, r6} = \&r1 \xrightarrow{\text{depositsInto}} \&r8 \xrightarrow{\text{accountHolder}} \&r9 \xrightarrow{\text{electedLeader}} \&r6$ with the property sequence $PS = \text{depositsInto} \bullet \text{accountHolder} \bullet \text{electedLeader}$, refracts from accountHolder to electedLeader because the resource $\&r9$ is multiply classified as an instance of both Organization and Customer and $\text{RC}(PS) = 1$.

6.5 S-Match

In order to integrate IR style search with ρ -queries, we allow users to augment their queries with keywords. A Semantic Match (match of property or super/subproperty) of a keyword and a property occurring in Semantic Association) increases the rank value for that Semantic Association. The degree of the match and hence its S-Match value is determined by the proximity of the properties in the property hierarchy. This approach is very similar to that used in determining the similarity of concepts in an ontology [51] [60]. Given a property sequence $PS = p_1, p_2, p_3 \dots p_n$ and a set of keywords $K = \{k_1, k_2, k_3 \dots k_m\}$, the degree of a match between k_i and p_j is given by $\text{SemMatch}(k_i, p_j) = 0 < (2^d)^{-1} \leq 1$, where d is the minimum distance between

the properties in a property hierarchy. If two keywords match on the same property we take the maximum of their SemMatch values. Then for a path $ps \in [[PS]]$, its S-Match value is given by

$$S - Match(ps) = \sum_{i=1}^n \max_{j=1}^k \{SemMatch(p_i, k_j)\}$$

For example, in Figure 30 above, the minimum distance between “audits” and “enrolls” is 1, so that $SemMatch(audits, enrolls) = \frac{1}{2}$. Given $K = \{audits, taughtBy\}$ and a property sequence $PS = enrolls \bullet taughtBy$, $S-Match(ps) = \frac{1}{2} + 1 = 1\frac{1}{2}$.

6.6 SemRank

All the factors discussed above when combined together give the SemRank value of a Semantic Association. However, the exact nature in which they are combined is dependent on the search mode μ which varies from 0 to 1, with 0 indicating purely Conventional and 1 indicating purely Discovery modes respectively. Based on this, we build a modulative model for SemRank in which the mode μ specifies how each of the factors contribute to the rank of an association. The query mode μ modulates the contribution of the information content of an association to its rank as shown below:

$$I_{\mu}(ps) = (1-\mu)(I(ps))^{-1} + \mu I(ps)$$

This leads to higher rank values being assigned to the most unpredictable paths at the purely discovery mode, and lower rank values being assigned at the purely conventional mode. The query mode μ modulates the refractive count of an association as shown below $RC_{\mu}(ps) = \mu RC(ps)$. Since predictable paths are desired at the purely conventional mode, paths ranked highest at this mode do not have refractions, as the formula above shows. Both the purely discovery and purely conventional modes seek to retrieve paths whose component properties

have high S-Match values with the keywords provided by the user. Therefore, S-Match is not modulated by μ . The SemRank formula combines these three factors to assign a rank to any Semantic Association, adapting itself as the mode changes. It is defined for a ρ -path association ps as:

$$\text{SEMRANK}(\text{SA}) = I_{\mu}(\text{ps}) \times (1 + \text{RC}_{\mu}(\text{ps})) \times (1 + \text{S-Match}(\text{ps}))$$

Using this model, we can provide a flexible ranking approach for ranking complex relationships.

6.7 Ordering Search Results using SemRank values

The approach for obtaining an ordering on Semantic Associations resulting from a search will depend largely on the strategy for computing SemRank values. Possible strategies include integrating query processing, SemRank computation and result ordering into a single phase or performing the last two steps in a separate phase after query evaluation. The choice of the strategy to be adopted is dependent on whether exact orderings are required or whether approximately correct orderings are acceptable. In the case of approximately correct orderings, we trade correctness for efficiency. This happens because in the case of exact orderings it may be necessary to completely compute the SemRank values of all Semantic Associations and then sort them in order. When there is a large number in the result set, this may prove to be inefficient.

In this section, we will discuss an approach for computing SemRank values for Semantic Associations and an approximate Top-K ordering algorithm used in our SSARK system.

6.8 Overview of the SSARK System

The approach used in the SSARK prototype system implementation consists of three phases supported by the architecture shown in Figure 32.

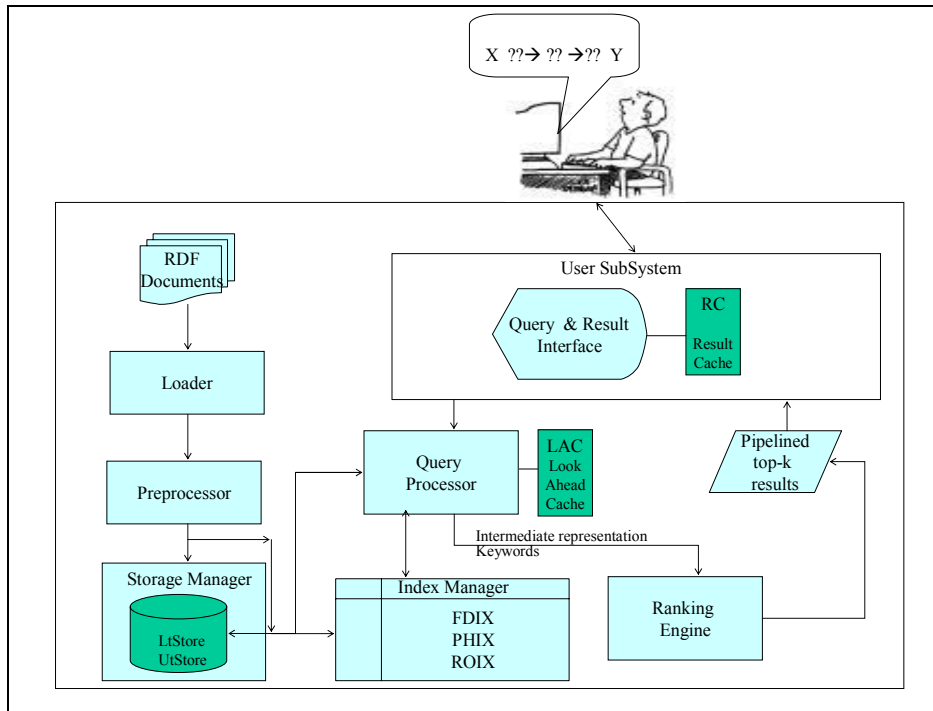


Figure 32: System Architecture with Ranking Module

In the preprocessing phase, RDF documents are loaded and preprocessed into an intermediate representation by the Loader and Preprocessor. The intermediate representation of an RDF graph produced by the preprocessor is called a path sequence. A path sequence is a sequence of subgraph representations that is amenable to efficient query evaluation. The persistence of a path sequence is managed by the Storage Manager which allows for its storage in a database. When a query with a pair of resources is given, the Query Processor selects relevant subsequences of a path sequence and composes them to generate an annotated summary of the Semantic Associations called an Annotated Path Expression Tree (APET). The discussion of query evaluation is outside the scope of this paper. For the sake of brevity, the rest of the discussion will focus on ranking only ρ -pathAssociations, even though the other types of associations have also been investigated and are being prototyped.

The APET generated by the query processor is a tree representation of the regular expression that represents all paths found between the resources specified in the query. It is a K-ary and-or tree where leaves are the labeled edges of the paths and internal nodes are operator (union, concatenation) nodes with K children. Figure 33 shows an example APET. A semantic transformation process is performed during query evaluation to ensure that cycles are not represented in an APET. A discussion of the semantic transformation process is outside the scope of this paper but can be summarized thusly: For any cycle c with paths (a) from vertex v_1 to vertex v_2 and (b) from v_2 back to v_1 , c is broken up into two paths from v_1 to v_2 . The first path is equivalent to (a) and the second is equivalent to $((b)^{-1})^R$ i.e., the reverse of the path (b) back to v_1 with the properties in (b) substituted with their inverse properties. Also during query evaluation the leaves of an APET (i.e. path edges) are annotated with their SemRank values. Then during the ranking phase, the Ranking Engine uses a pipelined Top-K algorithm to extract approximately the Top-K paths represented in the summary. The sequel elaborates on the structures used to support the SemRank computation as well as the Top-K algorithm.

6.9 Annotating Path Expression Trees

During query evaluation, the leaves of an APET are annotated with a set of values that contribute to their SemRank values. These values are either retrieved directly or computed from indexes in the Index subsystem. We will now summarize the roles of the indexes used in the computation of SemRank values:

- Frequency Distribution Index (FDIX): FDIX maps each property p to a tuple $(|[[p]]^{\wedge}|, |[[p]]^+|)$ where $|[[p]]^{\wedge}|$ is the size of p 's proper extent and $|[[p]]^+|$ includes the size of the

proper extent of p 's superproperties. These values are used for calculating Specificity and θ -Specificity.

- Representative Ontology IndeX (ROIX): The Representative Ontology Index is a hierarchical index that maps resources to classes and then classes to ROCs. It also stores the semLinks that link the ROCs, i.e. the labels on the edges linking the ROCs in the semantic summaries. This information is used to determine the refraction count of a path.
- Property Hierarchy IndeX (PHIX): Each property in the RDF data model may participate in a number of subsumption hierarchies. The idea is to index these properties in such a way that the distance between two entities in the hierarchy can be measured in constant time. To this effect, we index the properties in each hierarchy in a manner similar to the Dewey Decimal Coding (DDC) where each node is assigned an id that preserves its relative position amongst its siblings, prefixed by the id of its parent. For all the ids of all properties to be unique, each hierarchy is assigned a hierarchy id. Determining the distance between two properties then amounts to summing up the number of strings in the two ids beyond their Least Common Ancestor. For example given the ids 0.1.3.4.5.6 and 0.1.3.4.8, their LCA is 0.1.3.4. Beyond this, the first id has two strings (5.6) and the second has one (8), so the distance between them is three. Because a property may have more than one id, (since it may participate in more than one subsumption hierarchy), PHIX maps every property p to a set of ids. Using this index we can efficiently measure the distance between a keyword given a query and the properties on the resulting path which determines the SemMatch value of the path.

6.10 Retrieving Top-K Results

After the query evaluation has returned an APET, the ranking engine extracts the top-K paths represented in the APET. An exact SemRank ordering may require an exponential time algorithm since all paths must be assigned a value first before the paths are ordered. Here, we use a practical approach that finds an approximately correct ordering thereby sacrificing total correctness for efficiency. The algorithm proceeds in two phases. In the first phase, the top-K paths are computed based on all values except the refraction count values. Then the second phase reranks the paths from the first phase based on their refraction count. It is clear that this will not always result in the totally correct SemRank ordering, but we expect that what we get is an approximation that is suitable for most applications. The reason for this is that during the top-K computation as will be discussed shortly, paths are composed iteratively into subpaths in a bottom up manner from the leaves so that the entire path is composed when the iteration is at the root of the APET. This means that properties of a path that are used in SemRank computation such as the refraction count can only be computed at the end of a path building phase as opposed to the other factors (e.g. specificity) that are properties of the edges themselves and are known once an edge is encountered.

The algorithm proceeds bottom-up computing top-k paths for nodes based on the top-K paths computed for its children. Each non-leaf node maintains a list for storing its Top-K paths, as well as a max-priority queue implemented using the heap data structure with which it orders its Top-K paths. The idea is to obtain the Top-K paths for each node by accessing only a minimal prefix of the Top-K paths of its children nodes, which are ordered in non-increasing order of SemRank values. At an 'or' node, during the first iteration of the algorithm, the first path from each child's Top-K paths are first enqueued into the max-priority q, then k paths are extracted from the

queue. For any path extracted from the queue, the next path from its list is enqueued. For subsequent iterations, we update the queue with the first new entry of a list if it was updated after its previously last entry had been enqueued. As such, not more than k paths are accessed from each child's list during each iteration of the algorithm. On the other hand, at an 'and' node, during the first iteration of the algorithm, the first path from each list is concatenated (preserving the order of the lists) to obtain the first of the k paths. To obtain the remaining $k-1$ paths, we first initialize the queue by obtaining and enqueueing a concatenation of the second path from each list with the first path from all other lists, then we extract $k-1$ paths from the queue. For each path p_i extracted from the queue, if p_i is a concatenation of paths $l_{a,1}, l_{b,2}, \dots, l_{c,m}$ from lists l_1, l_2, \dots, l_k respectively (where $l_{j,k}$ refers to the j th path from list k), we create a new path p_{i+1} by concatenating paths $l_{a_m,1}, l_{b_m,2}, \dots, l_{c_m,k}$ (where $l_{j_m,k}$ is equal to $l_{j+1,k}$ if k equals m , and $l_{j,k}$ if k is not equal to m). Having created the path p_{i+1} , we only insert it into the queue if

- there does not exist any path $p_j = p_{i+1}$ with $l_{j_m,k}$ equal to $l_{j-1,k}$ when k equals m that is still in the queue and
- p_{i+1} is not already in the queue.

For other iterations, the queue may need to be re-initialized if it is empty. This algorithm is analogous to the ranked join algorithms described in [56] except that we have taken some measures to optimize queue operations. All possible paths have been extracted from the queue of both the 'and' and 'or' nodes when these queues become empty. In general, this technique can be applied to retrieve ranked paths from any tree representation of path expressions, irrespective of the relevance model used for ordering, as long as a monotone combining function is used in the join step. In the second phase, the rank of the Top- K paths retrieved from the first phase are re-

computed this time including the refractive index of the paths, then the paths are re-ordered based on the new rank values.

To illustrate this, suppose we want to retrieve the top-2 paths from the APET shown in Figure 33(a) using the approximate retrieval technique.

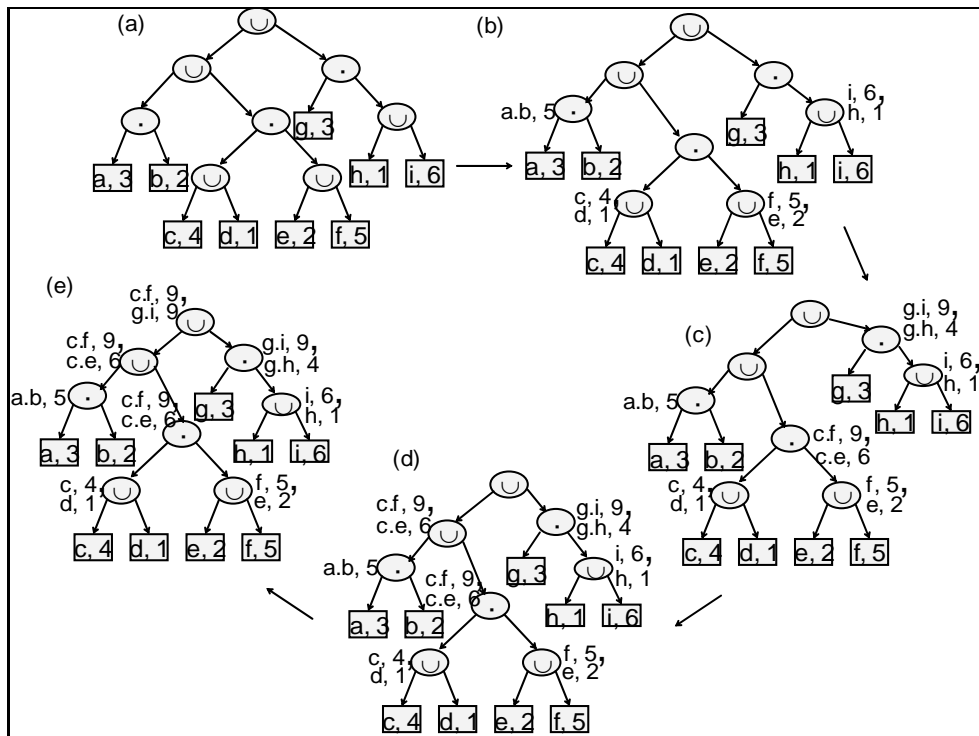


Figure 33: APET showing Top-K evaluation

For this example, we obtain the rank of a path by summing the ranks of its sub-paths. Figure 33(b) shows the state of the APET after all sub-trees of height 1 have been processed. Processing continues in a similar manner until the list of the root node of the APET gets updated with the Top-K paths. Figure 33(e) shows the state of the APET after the first ranking phase. The top-2 paths c.f and g.i both have the same ranks (9). If we assume that there is refraction from g to i and none from c.f, then during the second ranking phase, the ranks of these paths would be re-

computed to incorporate refraction. The new rank of g.i would then be 18 (9×2), so that after they are re-ordered, g.i precedes c.f.

Given an APET, let $R = p_1, p_2, \dots, p_n$ be all paths in the APET in non-increasing order of SemRank and let $R' = p_{1'}, p_{2'}, \dots, p_{k'}$ be the Top-K paths obtained from the APET using the approximate retrieval technique. We define the approximation error or cost of approximation as the average distance between the index of a path in R' and its occurrence in R . This is given by

$$\frac{1}{k} \sum_{p_{j'}=p_i} |j' - i|$$

6.11 Evaluation

There are an increasing number of publicly available RDF data sets ranging from those that are narrowly focused (e.g. FOAF [80], DBLP [78], ODP [79]) to those with broader scope covering multiple domains (e.g. TAP [77], SWETO). However, most of these data sets presented limitations that made them unsuitable as evaluation testbeds for SemRank. Key limitations are identified below.

- i) In some cases, the data is organized as taxonomies with very few non-hierarchical property types connecting resource types.
- ii) In the large datasets, a large proportion of the property instances were classification relationships i.e. instances of the `rdf:type` property and instances of very few types of non-hierarchical property types.
- iii) For the data sets covering a broader scope, the relationships that were captured often resulting in a large number of disconnected components, so that they were few relationships overall across resources.

- iv) Another important issue is that evaluation testbed(s) must have data distributions that model reality

We speculate that the reason for these problems is twofold. First, since most existing querying paradigms are focused on finding entities based on type information given in a query, much of the data collection has focused on classification information and very few other property types. Second, most Semantic Web data sets are still, understandably, in the early evolutionary phase. However, there are consequences of these issues for realistically evaluating SemRank. First, in most cases data sets did not contain paths connecting entities beyond just direct binary relationships. In the cases where paths were found they were typically no longer than 3 edges (i.e. properties). Furthermore, where there were multiple paths found connecting resources they were typically multiple instances of the same type of paths. In other words, because the limited set of property data available in these data sets, most paths were typically similar. For example, two researchers may have multiple paths connecting them but all the paths would be either about coauthorship or about them publishing papers at the same conference. But paths are fundamental to the notion of importance in SemRank. Its determination of importance is based on distinguishing “important” relationships from “less important” relationships for a given search mode, so that data sets with limited path information are inadequate for evaluating SemRank. Another problem is that the SemRank model is very sensitive to data distributions so it is important that the distributions in data sets reflect reality. Unfortunately, because a lot of these data sets are still being developed, the nature of their property distributions does not always reflect reality, but rather reflect the stage of their development. For example, a data set may contain data from a few sources that have a lot of specific types of information but little about others. This will cause the other types of property types to have a low frequency count. In the

context of the SemRank model, relationships involving such properties would wrongly be interpreted as a rare relationships and ranked high at the discovery mode even though the relationships in reality are fairly common.

For our evaluation, we used the SWETO dataset with data extracted from real world sources. However, we augmented the data set with synthetic data to make up for the gaps between data missing from the SWETO data set (either because it was not available on the Web to be extracted or had not yet been extracted.) In this way, we reduce the risk of entities or relationships being wrongly inferred as rare. Special care was taken to ensure that the synthetic data mirrored reality by using publicly available statistical data.

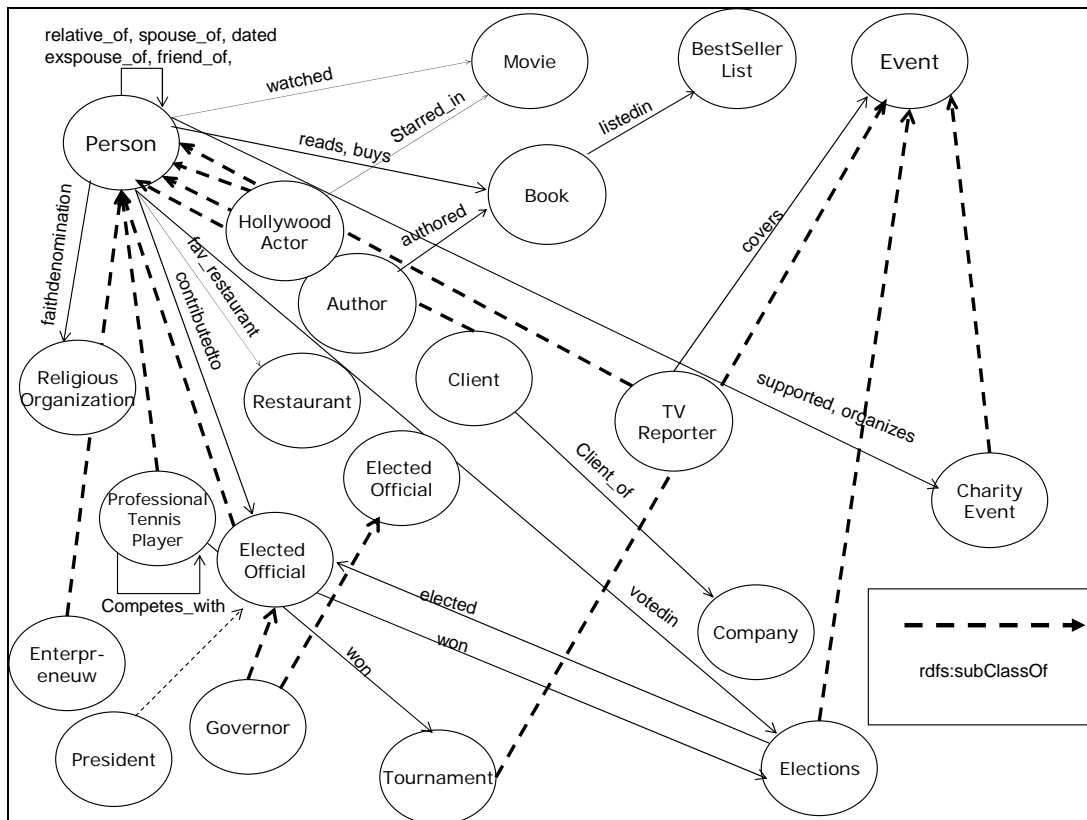


Figure 34: Schema Graph for Evaluation Testbed

For example, when generating additional information about politicians, information about numbers of US state Governors, Congressmen, etc and number of the population of people residing in the US was used as a guide. Figure 34 shows the schema for our evaluation test bed. Although we did not generate data to the same scale, we used these numbers to ensure that the proportions of class and property instances relative to one another mirrored reality. For example, there are more *Congressmen* than there are *Governors* and there are far more *votedIn* relationships between *Persons* and *Elections* (reflecting the number of votes involved in an election) than there are about *elected* relationships between *Elections* and *Elected Officials* (reflecting the number of people actually elected).

Experiments

We have implemented the SemRank algorithm using Java 1.5. To evaluate the performance of SemRank for ranking Semantic Associations, we sampled 58 human subjects from very different backgrounds. To ensure that most human subjects had some basic knowledge about the entities involved in our scenario, we observed several paths across domains such as Entertainment, Sports and Politics, between pairs of well known real world resources such as Arnold Schwarzenegger {Governor, Entertainer} and Andre Agassi {Professional Tennis Player}, Arnold Schwarzenegger and George W. Bush {Politician} and Michael Jordan {Professional Basket-ball} and Tiger Woods {Professional Golf Player}.

The experiment was conducted by giving the subjects a subset (average 8) of all the relationships found to exist between the chosen pairs of entities. We restrict the number of relationships to be ranked by the human subjects to a manageable size to ensure that the human subjects are not overwhelmed when assigning ranks to the relationships. Also, to avoid bias which may be

introduced by any specific knowledge about the domains which the subjects may have, we ensured that the subset of relationships presented to the subjects were heterogeneous.

Since SemRank does not have an absolute rank for all possible situations as in fixed ranking schemes, but a relative rank based on a continuous search mode spectrum, we conducted our experiments for the two intuitive limits of the spectrum; Conventional, and Discovery. Specifically, for each observed pair of entities, the human subjects were given a list of randomly ordered ranked paths relating the entities. They were then instructed to categorize each of the paths as either Conventional or Discovery path, to reflect their perception of the information portrayed by the path as “*not surprising*” or “*really surprising*”. The subjects were instructed to mark paths for which they were unsure about categorizing them as Conventional or Discovery as “*Uncertain*”. In addition to these, the subjects were also instructed to indicate the paths they perceived as the top 3 Conventional paths as well as the top 3 Discovery paths. In this way, we hope to capture an overall sense of what users consider conventional or discovery and see how it correlates with the ranking produced by SemRank at these modes.

QUERY PROCESSOR					
Results for Mode 0					
<i>$\rho(r1(\text{Arnold Schwarzenegger}), r6(\text{Andre Agassi}))$</i>					
(4)	✕	r2(Arnold Schwarzenegger) <u>supports</u> → r12(GrandSlamForChildren) ← <u>organized</u> r6(Andre Agassi)			0.868438269
(7)	✕	r2(Arnold Schwarzenegger) <u>spouseOf</u> r1(Maria Shriver) <u>faithDenomination</u> → r13(Catholic) ← <u>faithDenomination</u> r9(Brooke Shields) <u>ex-spouseOf</u> r6(Andre Agassi)			0.851910857
(6)	✕	r2(Arnold Schwarzenegger) <u>spouseOf</u> r1(Maria Shriver) <u>dinedAt</u> → r10(Madre's Restaurant) ← <u>dinedAt</u> r9(Brooke Shields) <u>ex-spouseOf</u> r6(Andre Agassi)			0.849771221
(3)	✕	r2(Arnold Schwarzenegger) <u>clientOf</u> → r11(NetJets) ← <u>clientOf</u> r5(Pete Sampras) <u>competedWith</u> r6(Andre Agassi)			0.841345706
(8)	✕	r2(Arnold Schwarzenegger) <u>spouseOf</u> r1(Maria Shriver) <u>authorOf</u> r14(Book) <u>listedIn</u> → r15(BestSellerList) ← <u>listedIn</u> r16(Book) <u>authorOf</u> r9(Brooke Shields) <u>ex-spouseOf</u> r6(Andre Agassi)			0.82324056
(5)	→	r2(Arnold Schwarzenegger) <u>spouseOf</u> r1(Maria Shriver) <u>friendOf</u> r9(Brooke Shields) r8(#3697) <u>ex-spouseOf</u> r6(Andre Agassi)			0.807238719
(2)	✕	r2(Arnold Schwarzenegger) <u>spouseOf</u> r1(Maria Shriver) <u>relativeOf</u> → r7(Edward Kennedy) ← <u>receivedCampaignContributionsFrom</u> r6(Andre Agassi)			0.805644261
(1)	✕	r2(Arnold Schwarzenegger) <u>starredIn</u> → r3(Last Action Hero) ← <u>starredIn</u> r4(Bridgette Wilson) <u>ex-spouseOf</u> r5(Pete Sampras) <u>friendOf</u> r6(Andre Agassi)			0.794968057

Figure 35: Relationships Used in Experiment

Results

To effectively analyze the performance of the SemRank ranking at a search mode with respect to the ranks assigned by the subjects at the same mode, we first analyze the consistency of the rankings of relationships between each pair of entities by the human subjects. As discussed earlier, we presented the relationships between the entity pairs to 58 human subjects. However, as is expected with evaluations involving human subjects from disparate backgrounds, we had some incomplete as well as inconsistent responses for the set of relationships between the investigated entity pairs. With the elimination of the rankings obtained from subjects which had some incomplete or inconsistent responses, the number of human subjects sampled reduced to

33, 41 and 31 for the relationships between Arnold Schwarzenegger and Andre Agassi (S – A), Arnold Schwarzenegger and George W. Bush (S – B) and Michael Jordan and Tiger Woods (J – W) respectively. The figures below show graphical representations of the rankings of the relationships between the pairs of entities, as obtained from the human subjects.

In Figure 36 below, we observe that the human subjects who ranked the relationships between Schwarzenegger and Agassi assert that the relationships denoted by Paths 4, 5 and 2 are conventional with about 82%, 67% and 55% agreements respectively. On the other hand, they purport that Paths 8, 6, 7, 3 and 1 are discovery type relationships with 79%, 61%, 55%, 55% and 52% agreements respectively.

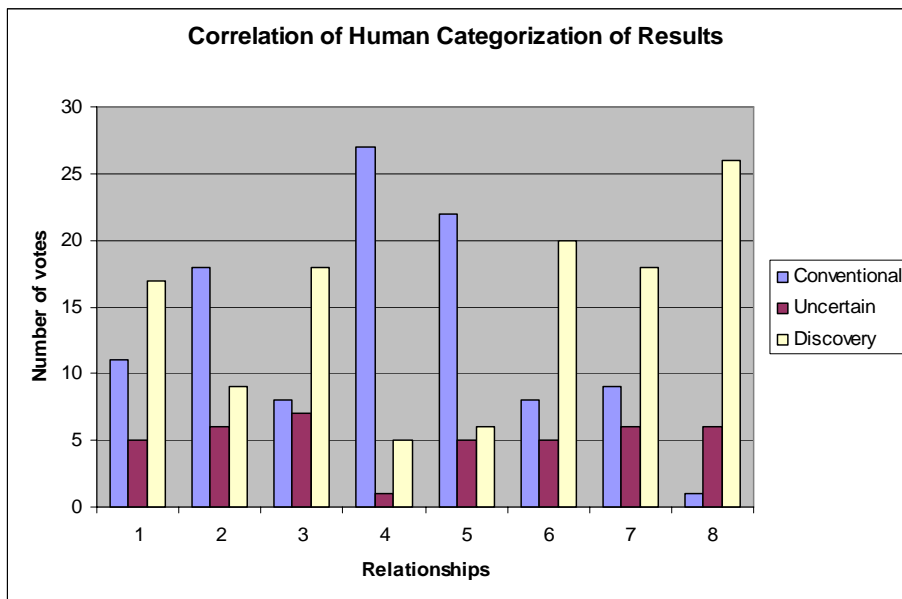


Figure 36: Categorization of Relationships between Schwarzenegger and Agassi

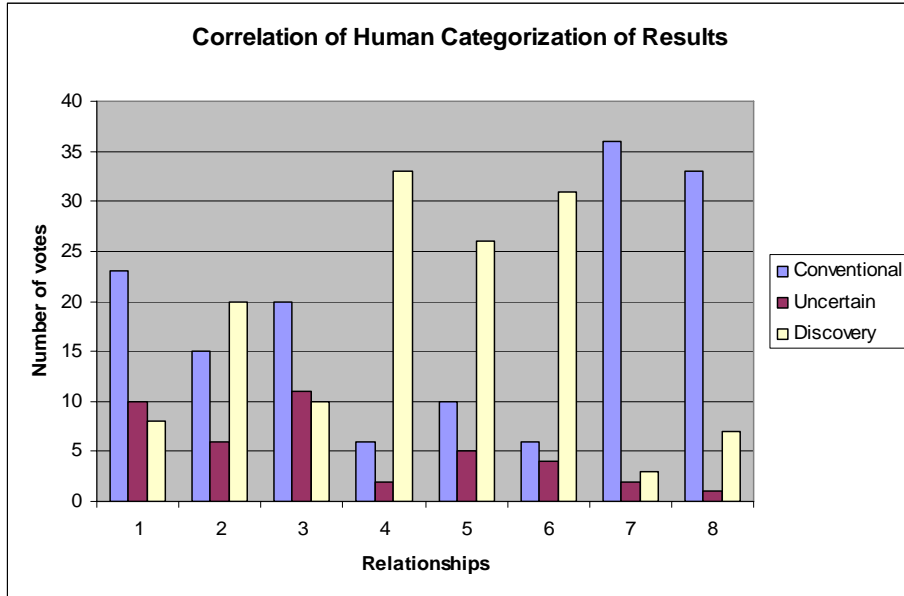


Figure 37: Categorization of Relationships between Schwarzenegger and Bush

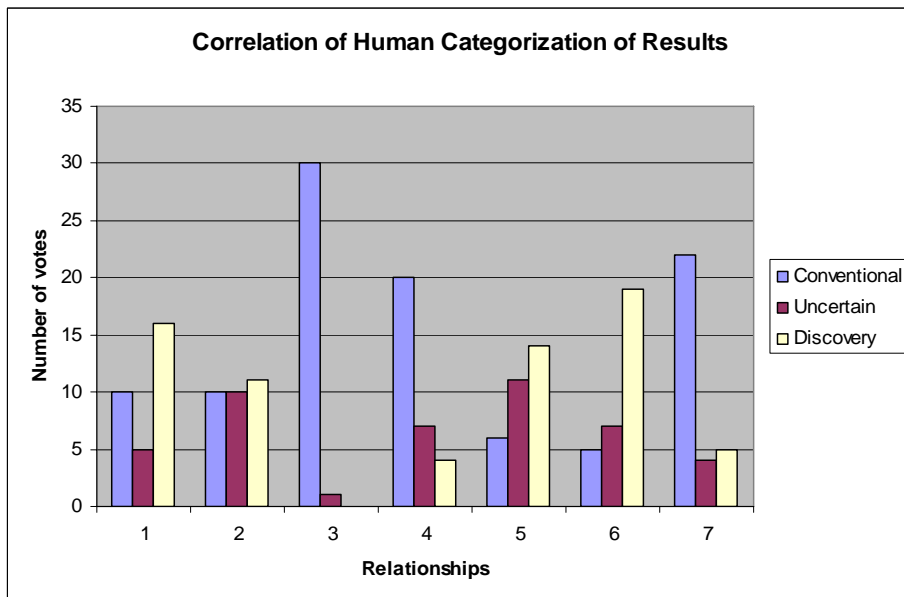


Figure 38: Categorization of Relationships between Jordan and Woods

Although we get some sense of what relationships the human subjects perceive as Conventional or Discovery type relationships, we wish to determine whether there is consistency in ranking of these relationships amongst the subjects. That is, we wish to determine if the subjects deem certain relationships to be more (for example) “conventional” than others in some objective sense or is this determination largely subjective?

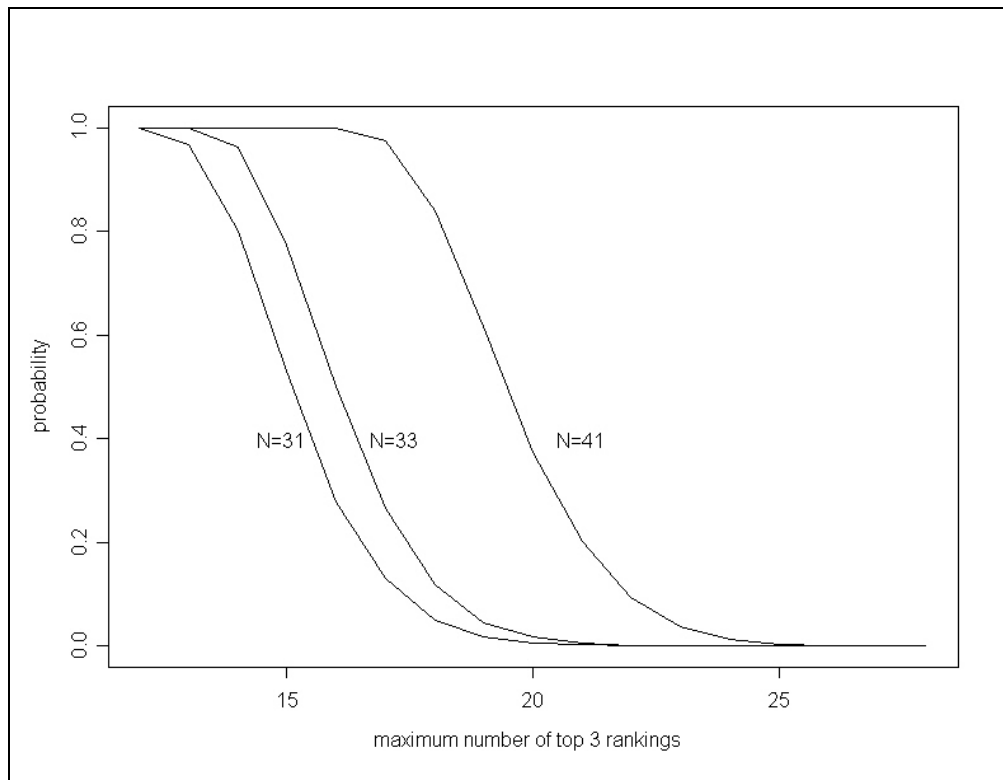


Figure 39 : P-Values of the Maximum Number of Top 3 Rankings for the Experiments

To do this, we conducted a statistical test of significance as follows:

H₀: the ranking of relationships by the subjects is completely subjective and that any ranking of relationships is as likely as any other.

H₁: certain relationships are more Conventional or Discovery type relationships in some sense that can be consistently recognized by the human subjects.

Level of significance $\alpha = 0.05$.

Test Statistic Z = the maximum number of times any relationship is ranked in the top 3.

Decision Rule: Reject H_0 if the probability of the observed Z under the null hypothesis is less than 0.05 otherwise do not reject H_0 .

Since the human subjects in the experiment were asked to choose the top three Conventional and the top three Discovery type relationships, if the null hypothesis is true then the ranking of relationships is random and each relationship will be placed in the top three approximately $3/8$ of the time. However, if some relationship is ranked in the top three significantly more often than would be expected by chance, then we reject the null hypothesis.

To assess the significance of the rankings, we adopted a simulation approach to obtain a distribution for each of the relationships between the entity pairs under the null hypothesis. In particular, we focused on simulating the distribution of the relationships between the entity pairs with the maximum number of occurrence in the top 3 Conventional and top 3 Discovery relationships. Each repetition of the simulation had the following steps:

- 1) For each sample individual, using a uniform random number generator, randomly generate rankings for the relationships between an entity pair;
- 2) For each relationship i , record the number y_i of times it was placed in the top 3;
- 3) Find the maximum Z of the y_i 's.

Steps 1) - 3) were repeated 5,000 times to generate an approximate distribution for Z under the null hypothesis. Having obtained the distribution for Z , we then compare the observed Z from the experiment to that of this distribution at a 95% level of significance. Thus if its probability is less than 0.05, we reject the null hypothesis.

Figure 39 above shows the probabilities of occurrence of the relationship with the most frequency in the top 3 Conventional or top 3 Discovery relationships for sample sizes 31, 33 and 41, drawn from the distribution of Z obtained from the simulation. In Table 4 below, we show the observed Conventional and Discovery relationships which occurred most frequently in the top 3 Conventional and top 3 Discovery relationships, for each of the entity pairs. For the observed number of times these relationships occurred in the top 3 Conventional or top 3 Discovery relationships, Table 4 shows the probabilities with which these relationships appeared the same number of times in the top 3 Conventional or top 3 Discovery relationships, in the simulation. Thus from Table 4, we observe that for each of the experiments conducted, there was strong evidence of non-randomness on the ranking of relationships.

Table 4 : Statistics of the top ranked relationship between the entity pairs

Experiment	Sample Size	Path ranked most often in top 3.	Observed max number of times in top 3 (p-value)
S – A (Conventional)	33	4	28 (<0.0001)
S – A (Discovery)	33	8	23 (<0.0004)
S – B (Conventional)	41	7,8	33 (<0.0001)
S – B (Discovery)	41	4,6	32 (<0.0001)
J – W (Conventional)	31	3	30 (<0.0001)
J – W (Discovery)	31	6	26 (<0.0001)

That is, in each case there were relationships that were placed in the top three significantly more often than by chance, so that we reject the null hypothesis and conclude that the top Conventional and Discovery relationships were not randomly chosen by the human subjects but

however were consistently recognized as such. So far, we have established that the relationships with the maximum occurrence in the top 3 Conventional and top 3 Discovery type relationships as ranked by the subjects were chosen in a consistent manner. Based on this, we now analyze the performance of the SemRank Algorithm at the search modes 0 and 1, corresponding to the Conventional and Discovery modes. Table 5 below shows the top 3 Conventional and Discovery type relationships between the entity pairs obtained using the SemRank algorithm.

Table 5 : Top 3 Conventional and Discovery Relationships using the SemRank Algorithm

Entity Pairs	Top 3 Conventional Relationships	Top 3 Discovery Relationships
S – A	7	3
	4	1
	2	8
S – B	8	6
	7	5
	3	4
J – W	7	4
	6	2
	5	3

From this table, we observe the top Conventional and top Discovery type relationships as obtained using the SemRank algorithm do not correspond with those professed by the subjects, in all cases. For example, human subjects consider paths 4 and 8 from S – A as the top Conventional and top Discovery relationships respectively, but this is not the case using the

SemRank algorithm. However, if we increase the window size and observe the top 2 Conventional and top 2 Discovery relationships, we obtain path 4. A further increase of the window size to the top 3 Conventional and top 3 Discovery relationships reveals path 8.

On the other hand, for the relationships from S – B, human subjects deem paths 7 or 8 and 4 or 6 as the top Conventional and top Discovery relationships respectively. Using a window size of 1, the rankings obtained using the SemRank algorithm reveals paths 8 and 6 as the top Conventional and Discovery relationships respectively, which correspond with the rankings obtained using the SemRank algorithm with a window size of 1. We observe that in both cases (S – A and S – B), a window of size 3 is the maximum window needed to reveal the top Conventional and top Discovery relationships from the rankings of the human subjects. Thus, the performance of the SemRank algorithm is not unreasonable, especially since the kind of prior knowledge that users have about a situation greatly determines what they consider unusual or surprising. However, for the relationships from J – W, the SemRank algorithm produces a ranking that is almost opposite that produced by the human subjects. For example, where the subjects consider paths 3 and 6 as the top Conventional and top Discovery type relationships respectively, these relationships appear as Discovery and Conventional type relationships in the SemRank ranking. This aberration elucidates the weakness of the SemRank algorithm which largely stems from what prior knowledge users may have, as this determines what they consider surprising or not. However, the elegance of the SemRank approach is that if a user finds a particular ranking unsuitable or unintuitive with respect to their context, they can easily change the rankings with a simple act of moving the sliding bar in the query interface.

7. CONCLUSION AND FUTURE WORK

7.1 Overview of Thesis

This thesis studied the problem of supporting link analysis in Semantic Web databases and addressed a number of challenges in enabling advanced querying paradigms for its support. We began with some background and motivation in Chapter 2, where we presented some motivating examples that highlighted some of the differences with traditional querying and ranking paradigms and the requirements of querying and ranking paradigms for link analysis in knowledge discovery applications. In Chapter 3, we present a query language called SPARQ2L that supports the expression of path extraction queries on RDF databases, an important class of queries for link analysis. SPARQ2L extends the almost standardized query language SPARQL, with path variables and path constraint expressions. Its set of path constraint expressions allows expressiveness of a broad set of applications. We demonstrated SPARQ2L's expressiveness empirically and show by comparison to other query languages its superiority in this aspect. In Chapter 4, we studied the problem of developing a storage model for efficiently evaluating path extraction queries on disk resident databases. The storage model is based on a linearization of certain classes of path summaries of paths in a graph and heuristics for clustering related path summaries. We discuss "relatedness" based on the notions of prunability and prunable equivalence which allows us to reason about the relevance or irrelevance of groups of path summaries to a query. In Chapter 5, we discuss the issue of query processing using our storage model and demonstrate empirically the effectiveness of our overall approach using a variety of positive and negative query classes on synthetic and real life datasets.

The final chapter, Chapter 7, dealt with the problem of managing the results of queries. We presented a ranking model called SemRank which is based on classifying searches in a domain-

independent manner and offering flexible and user-adjustable query result ordering to suit particular classes of searches. SemRank is a rich ranking model combining both semantic, structural and information theoretic criteria in the determination of importance. We discuss the design of computational infrastructure for computing the Top-K query results based on SemRank orderings and present a human-based evaluation of SemRank that showed a reasonable correlation to humans' query result expectations.

7.2 Future Directions

In the future, we would like to further study the applications landscape for link analysis queries and extend the expressiveness of SPARQ2L to support an even broader class of queries. We would also like to investigate the problem of extending our storage and query processing model to support the evaluation of all the constraint classes proposed in SPARQ2L and provide a tighter coupling of our computational infrastructure with the computational infrastructure used in current DBMSs. Finally, we would like to investigate the problem of designing distributed storage and query evaluation infrastructure for supporting link analysis in distributed environments.

REFERENCES

- [1] Aleman-Meza, B., Halaschek, C., Arpinar, I., and Sheth, A. Context-Aware Semantic Association Ranking. In Proceedings of SWDB'03: 33-50, Berlin, Germany 2003
- [2] Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., Tolle, K. The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases. In SemWeb 2001.
- [3] Alkhateeb, F., Baget J. F., Euzenat, J. Complex Path Queries for RDF. ISWC2005 poster paper, Nov 6 – 10, 2005, Galway, (Ireland).
- [4] Angles, R., Gutierrez, C. Querying RDF Data from a Graph Database Perspective Proceedings of the ESWC2005 conference, May 2005, Heraklion, Greece.
- [5] Anyanwu, K., Maduko, A., Sheth, A. SPARQ2L: Towards Supporting Subgraph Extraction Queries in RDF Databases. Proceedings of the WWW Conference 2007, May 7-12, 2005, Banff, Canada
- [6] Anyanwu, K., Maduko, A., Sheth, A. SemRank: Ranking Complex Relationship Search Results on the Semantic Web. Proceedings of the WWW Conference 2005, May 10-14, 2005, Chiba, Japan
- [7] Anyanwu, K., Maduko, A., Sheth, A. From Link Analysis Ranking to *Relationship* Analysis Ranking - Adding Semantics to the Mix. To be resubmitted for second round reviews to the Journal of Web Semantics.
- [8] Anyanwu, K., Sheth, A. ρ -Queries: enabling querying for Semantic Associations on the Semantic Web. Proceedings of the WWW 2003 conference. May 20 – 24, 2003, Budapest Hungary,
- [9] Anyanwu, K., Sheth, A. The ρ operator: Discovering and Ranking Semantic Associations on the Semantic Web, ACM SIGMOD Record, v.31 n.4, December 2002

- [10] Bailey, J. Bry, F., Furche, T., Schaffert, S. Web and Semantic Web Query Languages: A Survey.
- [11] Barton, S: Designing Indexing Structure for Discovering Relationships in RDF Graphs. DATESO 2004
- [12] Baeza-Yates, R. A., Ribeiro-Neto, B. A. Modern Information Retrieval ACM Press Addison-Wesley 1999
- [13] Beckett, D. The design and implementation of the Redland RDF application framework. Computer Networks, 39(5):577--588, 2002
- [14] Berners-Lee, T., Hendler, J., Lassila, O. The Semantic Web. Scientific American, May 2001.
- [15] Bonstrom, V., Hinze, A., Schweppe, H. "Storing RDF as a Graph," *la-web*, p. 27, (LA-WEB'03), 2003.
- [16] Borodin, A., Roberts, G. O., Rosenthal, J. S., Tsaparas, P. (2005). Link Analysis Ranking Algorithms Theory And Experiments. ACM Transactions on Internet Technologies (TOIT), 5(1).
- [17] Brickley, D., Guha, R.V. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Working Draft, 2002.
- [18] Brin, S., Page, L. The Anatomy of a Large-Scale Hypertextual Web search engine. Proceedings of the WWW1998 conference, pp. 107--117. Brisbane, Australia.
- [19] Broekstra, J. *SeRQL*: Sesame RDF query language. In M. Ehrig et al., editors, SWAP Deliverable 3.2 Method Design, pages 55--68. 2003

- [20] Broekstra, J., Kampman, A. van Harmelen, F. 2001. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. In D. Fensel, J. Hendler, H. L., and Wahlster, W., eds., *Semantics for the WWW*. MIT Press.
- [21] Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K. JENA: Implementing the Semantic Web Recommendations. *WWW (ATPP) 2004*: 74-83
- [22] Chaudhuri, S., Das, G., Hristidis, V., Weikum, G. Probabilistic Ranking of Database Query Results. *VLDB 2004*
- [23] Chaudhuri, S., Ramakrishnan, R., Weikum, G. Integrating DB and IR Technologies: What is the Sound of One Hand Clapping? *CIDR 2005*: 1-12
- [24] Chitrapura, K. P., Kashyap, S. R. Node ranking in labeled directed graphs. *CIKM 2004*: 597-606
- [25] Cho, J., Roy, S. "Impact of Web Search Engines on Page Popularity." *In Proceedings of the World-Wide Web Conference (WWW)*, May 2004.
- [26] Cohen. S., Mamou, J., Kanza, Y., Sagiv, Y. XSearch: A Semantic Search Engine for XML, *VLDB 2003*.
- [27] Corby, O., Dieng-Kuntz, R., Faron-Zucker, C., Gandon. F. "Searching the Semantic Web: Approximate Query Processing Based on Ontologies," *IEEE Intelligent Systems*, vol. 21, no. 1, pp. 20-27, Jan/Feb, 2006.
- [28] Corby, O., Dieng-Kuntz, R., Faron-Zucker, C.. *Querying the Semantic Web with the corese search engine*. ECAI/PAIS2004, Valencia (ES), August 2004. IOS Press.
- [29] Dean, M., Schreiber, G. (editors). *OWL Web Ontology Language Reference*. W3C Recommendation, <http://www.w3.org/TR/owl-ref/> 2004.

- [30] Deerwester, S., Dumais, S. T., Landauer, T. K., Furnas, G. W. and Harshman, R. A.
"Indexing by latent semantic analysis." *Journal of the Society for Information Science*, 41(6),
391-407, 1990.
- [31] Ding, L., Finin, T., Joshi, A., Pan, R., Scott Cost, R., Peng, Y., Reddivari, P., Doshi, V.,
Sachs, J. Swoogle: A search and metadata engine for the semantic web. In *Proceedings of the
Thirteenth ACM Conference on Information and Knowledge Management*, 2004.
- [32] Diwan, A. A., Rane, S., Seshadri, S., Sudarshan, S. Clustering techniques for minimizing
external path length., *Proceedings of the International Conference on Very Large Databases*,
Morgan Kauffman, 1996.
- [33] Donninger, H., Bonome, T., Radonovich, M., Pise-Masison, C. A., Brady, J., Shih, J. H.,
Barrett, J., Birrer, M. J. Whole genome expression profiling of advance stage papillary serous
ovarian cancer reveals activated pathways. *Oncogene* (2004) **23**, 8065-8077
- [34] Fagin, E. Wimmers, L. A formula for incorporating weights into scoring rules, *Theoretical
Computer Science* 239, 2000, pp. 309-338 (Special issue for selected papers from the 1997
International Conference on Database Theory).
- [35] Frasincar, F., Houben, G., Vdovjak, R., Barna, P. RAL: An Algebra for Querying RDF.
World Wide Web 7(1): 83-109 (2004)
- [36] Goldman, R., Widom, J. Dataguides: Enabling query formulation and optimization in
semistructured databases, *VLDB*, 1997.
- [37] Guha, R. V., McCool, R., Miller, E. Semantic search. *WWW 2003*: 700-709.
- [38] Halaschek, C., Aleman-Meza, B., Arpinar, B., Sheth, A. Discovering and Ranking Semantic
Associations over a Large RDF Metabase. *VLDB 2004 demo paper*.

- [39] Harth, A., Decker, S. Optimized index structures for querying RDF from the web. In LAWEB 2005.
- [40] Haveliwala, T. H. Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search. IEEE Trans. Knowl. Data Eng. 15(4): 784-796 (2003)
- [41] Hristidis, V., Papakonstantinou, Y., Balmin, A. Keyword Proximity Search on XML Graphs. IEEE ICDE, 2003.
- [42] Hristidis, V., Koudas, N., Papakonstantinou, Y. PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. SIGMOD Conference 2001
- [43] Janik, M., Kochut, K. BRAHMS: A WorkBench RDF Store and High Performance Memory System for Semantic Association Discovery. ISWC2005: 431-445
- [44] Perez, J., Arenas, M., Gutierrez, C. Semantics and Complexity of SPARQL. ISWC'06, Athens, GA, USA, 2006.
- [45] Haase, P., Broekstra, J., Eberhart, A., Volz, R. A Comparison of RDF Query Languages. ISWC'04: 502-517
- [46] Karger, D., Bakshi, K., Huynh, D., Quan, D., Sinha, V. Haystack: A General-Purpose Information Management Tool for End Users Based on Semistructured Data. CIDR 2005: 13-26
- [47] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, and M. Schol. RQL: A Declarative Query Language for RDF. WWW'02, Honolulu, Hawaii, USA, May7-11 2002.
- [48] Kleinberg, J. Authorative sources in a hyperlinked environment. J. ACM, 48:604-632, 1999.
- [49] Kochut, K., Janik, M. "SPARQLer: Extended Sparql for Semantic Association Discovery", Fourth European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, 3-7 June 2007
- [50] Lassila, O., Swick, R. RDF Model and Syntax Specification. W3C Recommendation

- [51] Lin, D. An Information-Theoretic Definition of Similarity, Proceedings of the Fifteenth International Conference on Machine Learning, p.296-304, 1998.
- [52] Mannola, F, Miller, E. RDF Primer. W3C Recommendation 2004.
- [53] Mendelzon, A.O. and Wood, P.T., Finding Regular Simple Paths In Graph Databases. In *15th Conference on Very Large Databases*, (Amsterdam, The Netherlands, 1989), Morgan Kaufman pubs. (Los Altos CA).
- [54] Milo, T., Suciu, D. "Index structures for Path Expressions ". In Proc. of the 7th Intl. Conf. on Database Theory, January 1999.
- [55] Mukherjea, S., Bamba, B. BioPatentMiner: An Information Retrieval System for BioMedical Patents. VLDB 2004: 1066-1077
- [56] Natsev, A., Chang, Y. C., Smith, J. R., Li, C. S., Vitter, J. S. Supporting incremental join queries on ranked inputs. In Proc. of VLDB 2001.
- [57] Nie, Z., Zhang, Y., Wen, J., Ma, W. Object-Level Ranking: Bringing Order to Web Objects. Proceedings of the 14th international World Wide Web conference. WWW2005, pp 567-574. May 10-14, 2005, in Chiba, Japan
- [58] Prud'hommeaux E., Seaborne, A. SPARQL Query Language for RDF. W3C Candidate Rec. 6 April 2006.
- [59] Rochas, C., Schwabe, D., Poggi de Aragao, M. A Hybrid Approach for Searching in the Semantic Web. WWW2004.
- [60] Rodriguez, M., Egenhofer, M. Determining Semantic Similarity Among Entity Classes from Different Ontologies, IEEE TKDE , 15 (2): 442-456, 2003.
- [61] Seaborne, A. *RDQL — A Query Language for RDF*, WWW Consortium, Member Submission SUBM-RDQL-20040109, January 2004.

- [62] Shannon, C.E. (1948), A Mathematical Theory of Communication, Bell Syst. Tech. J., 27, 379-423, 623-656.
- [63] Sheth, A., Aleman-Meza, B., Arpinar, I., B., Ramakrishnan, C., Halaschek, C., Bertram, C., Warke, Y., Avant, D., Arpinar, F. S., Anyanwu, K., Kochut, K. Semantic Association Identification and Knowledge Discovery for National Security Applications, JDM,16 (1), Jan-March 2005.
- [64] Sheth, A., Arpinar, B., Kayshap, V. Relationships at the heart of Semantic Web: Modeling, Discovering and Exploiting Complex Relationships. Enhancing the Power of Internet Studies in Fuzziness and Soft Computing. M. Nikravesh, B. Azvin, R. Yager and L. Zadeh, Springer-Verlag, 2003.
- [65] Siberski, W., Pan, J. Z., Thaden, U. Querying the Semantic Web with Preferences. ISWC 2006: 612-624
- [66] Sintek, M., Decker, S. TRIPLE - an RDF query, inference and transformation language. In DDLP, 2001.
- [67] Souzis, A. RxPath specification proposal. <http://rx4rdf.liminalzone.org/RxPathSpec>
- [68] Sparck Jones, K., Walker, S., Robertson, S. E. A Probabilistic Model of Information Retrieval: development and comparative experiments. *Information Processing and Management* **36**, Part 2 809-840 (2000).
- [69] Stojanovic, N., Mädche, A., Staab, S., Studer, R., Sure, Y. SEAL -- A Framework for Developing SEMantic PortALs. In: K-CAP 2001 – In Proc. of ACM Conference on Knowledge Capture, October 21-23, 2001
- [70] Tarjan, R. E. “Fast Algorithms for Solving Path Problems”. JACM, Vol. 28, No. 3, July 1981, pp. 594-614

- [71] Tarjan. R.E. A Unified Approach to Path Problems. JACM, 28:3:577--593, 1981.
- [72] Weikum, G., Graupmann, J., Schenkel, R., Theobald, M. Towards a Statistically Semantic Web. ER 2004 : 3-17
- [73] Zhuge, H., Zheng, P. Ranking Semantic-linked Network, WWW2003, Budapest, May, 2003.
- [74] [Oracle® Spatial Resource Description Framework \(RDF\) 10g Release 2 \(10.2\) Manual](#)
- [75] SWETO-DBLP <http://lsdis.cs.uga.edu/projects/semdis/swetodblp/>
- [76] Customer Identification and Risk Assessment (CIRAS), [Semagix Inc.](#)
http://www.semagix.com/solutions_ciras.html
- [77] TAP. <http://tap.stanford.edu/>
- [78] DBLP: An RDF ontology for DBLP. <http://www.semanticweb.org/library/>
- [79] ODP RDF dump <http://rdf.dmoz.org/>
- [80] FOAF. <http://www.foaf-project.org/>
- [81] Google. <http://www.google.com>