A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data

Matthew Steven Perry
Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all
Part of the Computer Engineering Commons, and the Computer Sciences Commons

Repository Citation
https://corescholar.libraries.wright.edu/etd_all/247

This Dissertation is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www.libraries.wright.edu, library-corescholar@wright.edu.
A Framework to Support Spatial, Temporal and Thematic Analytics over Semantic Web Data

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

By

MATTHEW S. PERRY
B.S., The University of Georgia, 2002

2008
Wright State University

__________________________
Amit P. Sheth, Ph.D.
Dissertation Director

__________________________
Thomas Sudkamp, Ph.D.
Director, Computer Science Ph.D. Program

__________________________
Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies

Committee on Final Examination

__________________________
Amit P. Sheth, Ph.D.

__________________________
Krishnaprasad Thirunarayan, Ph.D.

__________________________
Soon M. Chung, Ph.D.

__________________________
Christopher Barton, Ph.D.

__________________________
Kate Beard, Ph.D.
Spatial and temporal data are critical components in many applications. This is especially true in analytical applications ranging from scientific discovery to national security and criminal investigation. The analytical process often requires uncovering and analyzing complex thematic relationships between disparate people, places and events. Fundamentally new query operators based on the graph structure of Semantic Web data models, such as semantic associations, are proving useful for this purpose. However, these analysis mechanisms are primarily intended for thematic relationships. This dissertation proposes a framework built around the RDF data model for analysis of thematic, spatial and temporal relationships between named entities. We present a spatiotemporal modeling approach that uses an upper-level ontology in combination with temporal RDF graphs. A set of query operators that use graph patterns to specify a form of context are formally defined, and an extension of the W3C-recommended SPARQL query language to support these query operators is presented. We also describe an efficient implementation of the framework that extends a state-of-the-art commercial database system. We demonstrate the scalability of our approach with a performance study using both synthetic and real-world RDF datasets of over 25 million triples.
Contents

1 Introduction ........................................... 1
  1.1 Contributions .................................. 3
  1.2 Illustrative Examples ............................. 5
  1.3 Application Areas ................................ 7
    1.3.1 Semantic Sensor Web ....................... 7
    1.3.2 Event Web ................................ 8
  1.4 Outline .......................................... 9

2 Background .......................................... 10
  2.1 Semantic Web ................................... 10
  2.2 Ontologies and their Representation .............. 11
    2.2.1 RDF ..................................... 12
    2.2.2 OWL .................................... 15
  2.3 Semantic Analytics ................................. 15

3 Related Work ....................................... 18
  3.1 Data Modeling .................................. 18
    3.1.1 Ontologies and GIS ........................ 18
    3.1.2 Spatiotemporal Models ...................... 20
  3.2 Data Storage and Query Processing ............... 21
    3.2.1 Storing and Querying RDF ................... 22
    3.2.2 Spatial and Temporal Data on the Semantic Web 24
    3.2.3 Spatial and Temporal Query Processing .... 26

4 Modeling Approach .................................. 28
  4.1 Preliminaries .................................. 28
    4.1.1 RDF .................................... 28
    4.1.2 Temporal RDF ............................. 29
  4.2 Ontology-based Model ............................. 30
    4.2.1 Thematic Dimension ......................... 31
    4.2.2 Spatial Dimension .......................... 31
    4.2.3 Temporal Dimension ......................... 32
## Query Operator Design

5

5.1 Graph Patterns ............................................. 35
5.2 Spatial Operators .......................................... 37
5.3 Temporal Operators ....................................... 40
5.4 Computational Complexity ................................. 45

## Implementation Framework

6

6.1 Table Functions ........................................... 48
6.2 Storage and Indexing Scheme ............................... 53
   6.2.1 Existing Oracle Technologies ......................... 53
   6.2.2 Indexing Approach ................................... 54
      6.2.2.1 Spatial Indexing Scheme ....................... 55
      6.2.2.2 Temporal Indexing Scheme ..................... 55
6.3 Function Implementation .................................. 60
   6.3.1 Graph Pattern to SQL Translation .................... 60
   6.3.2 Spatial Functions .................................. 61
   6.3.3 Temporal Functions ................................. 65

## Experimental Evaluation

7

7.1 Repeatability of Experiments ............................. 69
7.2 Datasets .................................................. 70
   7.2.1 SynHist Dataset .................................... 70
   7.2.2 GovTrack Dataset .................................. 71
7.3 Experiments .............................................. 72
   7.3.1 Scalability with respect to Dataset Size ............. 74
      7.3.1.1 Basic temporal_extent ......................... 77
      7.3.1.2 Filtered temporal_extent ....................... 77
      7.3.1.3 temporal_eval .................................. 78
      7.3.1.4 Basic spatial_extent ......................... 78
      7.3.1.5 Filtered spatial_extent ....................... 79
      7.3.1.6 spatial_eval .................................. 80
   7.3.2 Scalability with respect to Graph Pattern Size ...... 80
      7.3.2.1 Filtered temporal_extent ....................... 80
      7.3.2.2 Filtered spatial_extent ....................... 84
      7.3.2.3 Basic temporal_extent ......................... 84
      7.3.2.4 Basic spatial_extent ......................... 85
   7.3.3 Scalability of Spatiotemporal Queries ............... 85
      7.3.3.1 Spatiotemporal Queries with respect to Dataset Size 85
      7.3.3.2 Spatiotemporal Queries with respect to Graph Pattern Size 88

## Query Language Support

8

8.1 The SPARQL Query Language .............................. 90
8.2 The SPARQL-ST Query Language ........................... 93
   8.2.1 Formal Syntax of SPARQL-ST ......................... 94
      8.2.1.1 Spatiotemporal Graph Patterns ................. 94
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.2.1.2</td>
<td>Spatial Built-in Conditions</td>
<td>95</td>
</tr>
<tr>
<td>8.2.1.3</td>
<td>Temporal Built-in Conditions</td>
<td>96</td>
</tr>
<tr>
<td>8.2.2</td>
<td>Formal Semantics of SPARQL-ST Queries</td>
<td>97</td>
</tr>
<tr>
<td>8.2.3</td>
<td>SPARQL-ST by Example</td>
<td>100</td>
</tr>
<tr>
<td>8.3</td>
<td>Design Decisions</td>
<td>104</td>
</tr>
<tr>
<td>8.3.1</td>
<td>Spatial Variables</td>
<td>104</td>
</tr>
<tr>
<td>8.3.2</td>
<td>Temporal Variables</td>
<td>106</td>
</tr>
<tr>
<td>8.3.3</td>
<td>Filter Expressions</td>
<td>107</td>
</tr>
<tr>
<td>8.4</td>
<td>Other Approaches for Querying Spatial and Temporal Data using SPARQL</td>
<td>107</td>
</tr>
<tr>
<td>8.5</td>
<td>SPARQL-ST Implementation Framework</td>
<td>110</td>
</tr>
<tr>
<td>8.5.1</td>
<td>Table Structures</td>
<td>110</td>
</tr>
<tr>
<td>8.5.2</td>
<td>Function Definition</td>
<td>111</td>
</tr>
<tr>
<td>8.5.3</td>
<td>Function Implementation</td>
<td>112</td>
</tr>
<tr>
<td>9</td>
<td>Conclusions and Future Directions</td>
<td>115</td>
</tr>
<tr>
<td>9.1</td>
<td>Future Work</td>
<td>116</td>
</tr>
<tr>
<td>A</td>
<td>Appendix</td>
<td>129</td>
</tr>
<tr>
<td>A.1</td>
<td>Queries used in Evaluation</td>
<td>129</td>
</tr>
<tr>
<td>A.1.1</td>
<td>Queries used for Table 7.3</td>
<td>129</td>
</tr>
<tr>
<td>A.1.2</td>
<td>Queries used for Table 7.4</td>
<td>136</td>
</tr>
<tr>
<td>A.1.3</td>
<td>Queries used for Figure 7.1</td>
<td>141</td>
</tr>
<tr>
<td>A.1.4</td>
<td>Queries used for Figure 7.2</td>
<td>150</td>
</tr>
<tr>
<td>A.1.5</td>
<td>Queries used for Figure 7.3</td>
<td>156</td>
</tr>
<tr>
<td>A.1.6</td>
<td>Queries used for Figure 7.4</td>
<td>163</td>
</tr>
<tr>
<td>A.1.7</td>
<td>Queries used for Table 7.5</td>
<td>170</td>
</tr>
<tr>
<td>A.1.8</td>
<td>Queries used for Table 7.6</td>
<td>171</td>
</tr>
<tr>
<td>A.1.9</td>
<td>Queries used for Figure 7.5</td>
<td>173</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Example RDF graph .................................................. 13
2.2 Example semantic associations ....................................... 17
3.1 Illustration of RDF storage schemes ................................. 23
4.1 Basic modeling approach .............................................. 30
4.2 GeoRSS GML-based ontology modeling basic spatial geometries .... 32
4.3 Temporal reification of the RDF statement (A B C) ............... 33
5.1 Example graph pattern ............................................... 36
6.1 Storage structures for RDF data ..................................... 54
7.1 Experiment GP1: filtered temporal_extent ............................ 81
7.2 Experiment GP2: filtered spatial_extent .............................. 81
7.3 Experiment GP3: highly selective basic temporal_extent .......... 82
7.4 Experiment GP4: highly selective basic spatial_extent .......... 82
7.5 Experiment ST1: basic spatial_extent plus temporal_extent .... 88
8.1 Illustration of subgraph matching querying paradigm ............ 90
8.2 Set of RDF triples in TURTLE format for illustration of example queries. 91
8.3 Set of triples representing a polygon .................................. 98
8.4 Table structures used for our SPARQL-ST implementation ....... 111
# List of Tables

2.1 RDFS Entailment Rules ........................................... 14
7.1 Characteristics of GovTrack and SynHist datasets .................. 70
7.2 Execution time for RDFS rules index creation and temporal inferencing .... 73
7.3 Experimental results for query execution time with respect to ontology size for GovTrack datasets ........................................... 75
7.4 Experimental results for query execution time with respect to ontology size for SynHist datasets ........................................... 76
7.5 Execution time for filtered spatial_extent plus filtered temporal_extent for GovTrack dataset ........................................... 87
7.6 Execution time for filtered spatial_extent plus filtered temporal_extent for SynHist dataset ........................................... 87
8.1 Results of Example 12 ........................................... 91
8.2 Results of Example 13 ........................................... 92
8.3 Results of Example 14 ........................................... 92
8.4 Results of Example 15 ........................................... 93
8.5 Results of Example 16 ........................................... 93
Acknowledgement

I am grateful for the many people who have helped me with this journey. I would first like to thank my advisor, Amit Sheth, for his guidance and support. His tremendous drive and dedication has shown me what it takes to be successful. I would also like to thank the members of my dissertation committee, Professors T. K. Prasad, Soon Chung, Christopher Barton and Kate Beard, for their support and insightful comments about my research.

I have had the privilege of working with a very active and energetic group of researchers during my studies, both at the LSDIS Lab and at the Kno.e.sis Center. I am thankful for collaborations and discussions with a variety of people including Boanerges Aleman-Meza, Farshad Hakimpour, Maciej Janik, Cartic Ramakrishnan and others. I would also like to thank Professors Budak Arpinar, Krzysztof Kochut and John Miller for their valuable feedback during project meetings.

I am thankful for interactions with Jagannathan Srinivasan, Souripriya Das, Melliyal Annamalai, Eugene Chong, Zhe Wu and Jay Banerjee at Oracle. The skills I learned during my internship and through later collaborations have been a valuable asset while completing this dissertation.

Finally, I would like to thank my family. Thank you to my wife Erin for her constant love and for her encouragement during difficult times. Thank you to my parents Marsha and Steve. They have always been there with love and support, and they made sure that I understood the value of education. Thank you to my brother Adam for his friendship and for his support over the years.
To my wife Erin. She has unselfishly made many sacrifices so that I could pursue this degree, and I could not have completed this dissertation without her love and support.
Introduction

Analytical applications are increasingly exploiting complex relationships among named entities as a powerful analytical tool. Such ‘connect-the-dots’ applications are common in many domains including national security, drug discovery, and medical informatics. Semantic Web Technologies (Herman et al., 2001) are well suited for this type of analysis. It is often necessary that the analysis process spans across multiple heterogeneous data sources, and ontologies and semantic metadata standards help facilitate aggregation and integration of this content. In addition, standard models for metadata representation on the web, such as Resource Description Framework (RDF) (Klyne and Carroll, 2004), model relationships as first class objects making it very natural to query and analyze entities based on their relationships. Researchers have consequently argued for graph-based querying of RDF (Angles and Gutierrez, 2005), and fundamentally new analytical operators based on the graph structure of RDF have emerged (e.g., semantic associations (Anyanwu and Sheth, 2003) and subgraph discovery (Ramakrishnan et al., 2005)). These operators allow querying for complex relationships among named entities where an ontology provides the context or domain semantics. We use the term *semantic analytics* to refer to this process of searching and analyzing semantically meaningful connections among named entities. Semantic analytics has been successfully used in a variety of settings, for example identifying conflict of interest (Aleman-Meza et al., 2006), detecting patent infringement (Mukherjea and Bamba, 2004) and discovering metabolic pathways (Kochut and Janik, 2007).

So far, semantic analytics tools have primarily focused on thematic relationships, but
spatial and temporal relationships are often critical components in analytical domains. In fact, most entities and events can be described along three dimensions: thematic, spatial and temporal. Consider the following event: *Fred Smith moved into the house at 244 Elm Street on November 16, 2007*. The thematic dimension describes *what* is occurring (the person Fred Smith moved to a new residence). The spatial dimension describes *where* the event occurs (the new residence is located at 244 Elm Street). The temporal dimension describes *when* the event occurs (the moving event occurred on November 16, 2007). Unfortunately, integrated semantic analytics over all three dimensions is not currently possible because of the following gaps in the state of the art:

- **Current GIS and spatial database technology does not support complex thematic analytics operations.** Traditional data models used for GIS excel at modeling and analyzing spatial and temporal relationships among geospatial entities but tend to model the thematic aspects of a given domain as directly attached attributes of geospatial entities. Thematic entities and their relationships are not explicitly and independently represented, making analysis of these relationships difficult.

- **Current semantic analytics technology does not support analysis of spatial and temporal relationships.** Semantic analytics research has focused on thematic relationships between entities. Thematic relationships can be explicitly stated in RDF graphs, but many important spatial and temporal relationships (e.g., distance and elapsed time) are implicit and require additional computation. Semantic analytics tools depend on explicit relations and must be extended if they are to use implicit spatial and temporal relations.

This dissertation describes a framework that aims to bridge these gaps. We demonstrate how to exploit the relationship-centric nature of Semantic Web data models to extend the state-of-the-art in modeling and querying spatial, temporal and thematic (STT) data.
Our modeling and querying paradigm is more flexible than current approaches. We use indirect thematic relationships to achieve a many-to-many mapping between thematic objects and spatial objects and use these relationships to define a notion of context so that we can query about an entity’s spatial and temporal properties with respect to different contexts.

In addition, demand for systems that can efficiently manage large amounts of Semantic Web data has reached a critical point. This demand is driven to a major extent by the existence of many large, real-world Semantic Web datasets. Some examples of publicly-available datasets include GovTrack (Tauberer, 2008) (data about activities of US Congress – 13 million triples), SwetoDBLP (Aleman-Meza et al., 2007; Kno.e.sis Center, 2008) (bibliography data focused on Computer Science publications – 11 million triples), DBPedia (Auer et al., 2008) (multi-domain data derived from Wikipedia content – 218 million triples) and UniProt (UniProt Consortium, 2008) (data describing functional aspects of proteins – over 1 billion triples). The development of a scalable system for management of STT Semantic Web data is thus a major component of this dissertation.

1.1 Contributions

The following advancements are required to progress from thematic analytics to STT analytics. First, we must adequately support spatial objects. Spatial objects (e.g., points, lines, polygons) are complex and cannot be represented as a single URI. We must define a way to represent spatial objects using Semantic Web languages and support the use of spatial indexes to evaluate queries involving these spatial objects. Second, we must define a way to model the temporal properties of facts expressed using Semantic Web languages. Third, inferencing procedures must be extended so that temporal properties are derived for inferred facts. Fourth, we must support additional computation for deriving implicit spatial and temporal relations. For example, computational geometry is needed to determine the distance between two spatial objects, and interval arithmetic is needed to determine the
intersection of a set of time intervals describing the valid times of a set of facts. Finally, we must extend query languages for Semantic Web data so that queries with spatial and temporal components can be expressed.

We propose a framework that extends current semantic analytics technology so that spatial and temporal data is supported in addition to thematic data. We address each of the issues described above. Specifically, we make the following contributions:

• An ontology-based spatiotemporal modeling approach using temporal RDF.

• A formalization of a set of spatial, temporal and thematic query operators for the proposed modeling approach that builds on a notion of context and supports computation of implicit spatial and temporal relations.

• A SQL-based implementation of the proposed query operators that involves a storage and indexing scheme for spatial and temporal RDF data and an efficient treatment of temporal RDFS inferencing.

• A query language, SPARQL-ST, that is an extension of the SPARQL RDF query language to support the identified query operators. SPARQL-ST adds a spatial variable type, constructs for manipulating temporal triples and new filtering capabilities involving spatial and temporal relations.

• A detailed performance study of our implementation using large synthetic and real-world RDF datasets. Our implementation demonstrates excellent scalability for very large RDF datasets in this evaluation (e.g., execution time of less than 500 milliseconds for a 10-hop graph pattern query over a 28 million triple dataset).
1.2 Illustrative Examples

We will give a brief introduction to our querying approach to illustrate some of our framework’s major concepts. These examples come from the battlefield intelligence domain. Suppose an intelligence analyst is assigned the task of monitoring the health of soldiers in order to detect possible exposure to a chemical or biological agent. In this case, the analyst may search for relationships connecting a sick soldier to potential chemical or biological agents by matching the soldier’s symptoms with known reactions to these agents. In addition, the analyst could further determine the likelihood of a particular chemical substance by querying for associations between the substance and enemy groups in the knowledge-base. For example, a member of the group may have worked at a facility that was reported to have produced the chemical. It is doubtful that such an analysis could produce definitive evidence of a biochemical attack, but incorporating spatial and temporal relationships could help in this regard. For instance, the analyst may want to limit the results to soldiers and enemies in close spatial proximity (e.g., find all soldiers with symptoms indicative of exposure to chemical X that fought in battles within 2 miles of sightings of any members of enemy group Y). We may pose the following SQL query involving the spatial_eval table function for such a search:

```sql
SELECT a as soldier
FROM TABLE (spatial_eval ('(?a has_symptom ?b)
  (Chemical_X induces ?b) (?a fought_in ?c)', '?c',
  '(?d member_of Enemy_Group_Y) (?d spotted_at ?e)', '?e',
  'geo_distance(distance=2 units=mile)'));
```

With this query, we are using the spatial_eval operator to specify (1) a relationship (context) connecting a soldier, a chemical agent and a battle location and (2) a relationship between members of an enemy organization and their known locations. We are then limiting the results based on the spatial proximity of the battles and enemy sightings. In addition, we
provide a *spatial_extent* operator that allows retrieving the spatial geometry associated with a thematic entity with respect to a given context and a *spatial_restrict* operator that can filter the results of a *spatial_extent* query using a spatial predicate, for example *find all soldiers participating in military events that take place within an input bounding box*.

We provide analogous *temporal_extent*, *temporal_restrict* and *temporal_eval* operators to query temporal aspects of connections between entities. The *temporal_extent* operator returns the temporal properties of a given relationship and the *temporal_restrict* operator allows optional filtering based on these temporal properties. For example, an analyst may want to find all soldiers that potentially came in contact with a particular soldier (i.e., *find all soldiers that were stationed in the same military base with Soldier 1 during January 2008 and also return the bases where contact may have occurred*). The corresponding query is shown below:

```sql
SELECT d as soldier, b as base
FROM TABLE (temporal_restrict(
   '(<Soldier_1> assigned_to ?a) (?a stationed_at ?b) (?c stationed_at ?b) (?d assigned_to ?c)',
   'DURING', '2008-01-01', '2008-01-31',
   'INTERSECT'));
```

In this query, we are specifying a thematic relationship connecting other soldiers to Soldier 1 via common military bases. We are additionally limiting the results to those connecting paths that are valid during the input time interval. The INTERSECT keyword indicates the type of temporal interval to use for a given result subgraph. In this case, we are interested in the time interval during which each edge (RDF statement) in the subgraph is valid. The *temporal_eval* operator acts as a temporal join for thematic subgraphs and can answer queries such as *find soldiers who exhibited symptoms after participating in a given military event*. In this query we are joining two subgraphs based on the temporal relation *after*.

Our implementation allows multiple operators to be used in a single SQL query. We
can therefore execute spatio-temporal-thematic queries that combine spatial and temporal operators. These possibilities are discussed in Section 6.1. In addition, we present the SPARQL-ST query language in Chapter 8 as an alternative to these SQL functions. Although we refer to our queries as spatial, temporal or spatiotemporal in the dissertation, all our queries involve a significant thematic component due to the graph patterns used in the queries.

1.3 Application Areas

Our framework will allow Semantic Web applications to more effectively use spatial and temporal data, and two promising application areas are the Semantic Sensor Web (Taylor and Ayyagari, 2006) and the Event Web (Jain, 2008). Both of these emerging application areas require management of Semantic Web data with heavy spatial and temporal components. Details of each area are given below.

1.3.1 Semantic Sensor Web

Sensor networks are becoming a popular way to collect and distribute observations about our world (Akyildiz et al., 2002). Sensors are being used in a variety of domains ranging from meteorology (University of Utah Department of Meteorology, 2008) to traffic planning (Ohio Department of Transportation, 2008). The Open Geospatial Consortium (OGC) describes the Sensor Web as “web-accessible sensor networks and archived sensor data that can be discovered and accessed using standard application protocols and application program interfaces” (Botts et al., 2007). In addition, the OGC has created the Sensor Web Enablement working group (Open Geospatial Consortium, 2008b), which has developed a suite of web service interfaces and metadata encodings for sensor webs. However, these interfaces are purely syntactic, and a more machine processable representation of infor-
1.3. APPLICATION AREAS

Information about sensor observations is needed to truly realize the Sensor Web. Researchers have proposed enhancing the Sensor Web framework with Semantic Web technologies to help alleviate this problem (Liu and Zhao, 2005; Henson et al., 2007; Sheth et al., 2008). The resulting Semantic Sensor Web will clearly require a solution for efficient storage and querying of semantic STT data.

1.3.2 Event Web

Nearly all human activity is rooted in space and time as events, and Professor Ramesh Jain (2008) has described vast collections of event data as the Web’s next evolution:

Event Web organizes data in terms of events and experiences and allows natural access from users perspectives. For each event, Event Web collects and organizes audio, visual, tactile, textual, and other data to provide people with an environment for experiencing the event from their perspective. Event Web also easily reorganizes events to satisfy different viewpoints and naturally incorporates new data types dynamic, temporal, and live. The current Web is document-centric hypertext. Unlike events, hypertext has no notion of time, space, or semantic structures other than often ad-hoc hyperlinks.

In Sheth and Perry (2008), we describe a web infrastructure that provides a means for realizing this interrelated web of events that can be traversed in any STT dimension. Our infrastructure is analogous to a GPS satellite system, which lets a GPS receiver automatically determine its location, speed, direction, and time. With such information, a GPS system can put a real-world event into its own spatial and temporal context. Similarly, the Event Web provides an infrastructure for placing Web data and documents into their own spatial and temporal context via services that enhance Web data and documents with spatial and temporal metadata.
Event registries are a core part of this infrastructure. Event registries store aggregated event data from various sources and provide STT query services so that clients can query and analyze event data. Clearly, modeling, storing and querying STT event data is critical part of these event registries and of the Event Web.

1.4 Outline

The remainder of the dissertation is organized as follows. Chapter 2 presents background information on the Semantic Web and semantic analytics. Chapter 3 surveys related work on modeling and querying STT data. Chapter 4 presents our spatiotemporal modeling approach, and Chapter 5 formalizes a set of query operators for our model. Chapter 6 describes an efficient implementation of our framework, and Chapter 7 presents a detailed performance evaluation of our implementation scheme. Chapter 8 proposes SPARQL-ST, an extension of SPARQL that enables STT queries. Chapter 9 concludes the dissertation and suggests directions for future research.
Background

We will first review background concepts. This includes an overview of the Semantic Web, a review of ontologies and ontology representation languages and an introduction to semantic analytics.

2.1 Semantic Web

The Semantic Web has received much attention recently. Its vision promises an extension of the current web in which all data is accompanied with machine-understandable metadata allowing capabilities for a much higher degree of automation and more intelligent applications (Berners-Lee et al., 2001). To make this idea more concrete, consider the statement “Wright State University is located in Dayton, OH.” The meaning of this statement is clear to a human with knowledge of colleges and universities and the geography of the United States. In addition, upon seeing this statement, other related information comes to mind such as professors who work at the University. In a Semantic Geospatial Web context (Egenhofer, 2002), this related information would be GIS data and services, such as road network data and facility locations for the Dayton area that could be combined with wayfinding services. The goal of the Semantic Web is to make the semantics of such data on the web equally clear to computer programs and also to exploit available background knowledge of related information. On the Semantic Web this statement would be accompanied with semantic metadata identifying an instance of the concept “University” with the
name “Wright State University”. Similarly, the instance of City and State, “Dayton, OH,” would unambiguously describe the university’s geographic location. Note the distinction between semantic metadata describing high-level concepts and relationships and syntactic and structural metadata describing low level properties like file size and format.

2.2 Ontologies and their Representation

Ontologies are central to realizing the Semantic Web, as they formally specify concepts and their relationships and provide the means to create semantic metadata for objects (documents, data files, databases, etc.). Ontology is defined as “a specification of a conceptualization” (Gruber, 1993). In database terms, we can divide an ontology into two parts: a schema and instance data. The schema models a domain by defining class types (e.g., University, City) and relationship types (e.g., located in). The schema is populated with instances of classes and relationships (e.g., Wright State University located in Dayton) to create facts representing knowledge of the domain.

The Semantic Web requires a standard, machine-processable representation of ontologies. The W3C has defined standard models and languages for this purpose. Here, we discuss the standard languages Resource Description Framework (RDF) (Klyne and Carroll, 2004) and Web Ontology Language (OWL) (McGuinness and Van Harmelen, 2004).

The use of ontologies represented with these languages is becoming very popular. Many such ontologies are available on the Web. These range from domain specific ontologies, for example Gene Ontology (Gene Ontology Consortium, 2008), NCI Cancer Ontology (Golbeck, 2008) and Glyco Ontology (Sahoo et al., 2006) in the Biological Sciences domain and ontologies from the UK’s Ordnance Survey (Ordnance Survey, 2008) and NASA’s SWEET ontologies (National Aeronautics and Space Administration, 2008) in the Geography domain, to general purpose ontologies such as WordNet (Miller et al.,
2.2. ONTOLOGIES AND THEIR REPRESENTATION

2.2.1 RDF

RDF has been adopted by the W3C as a standard for representing metadata on the Web. Resources in RDF are identified by Uniform Resource Identifiers (URIs) that provide globally-unique and resolvable identifiers for entities on the Web. These resources are described through participation in relationships. Relationships in RDF are called Properties and are binary relationships connecting resources to other resources or resources to Literals, that is, literal values such as Strings or Numbers. These binary relationships are encoded as triples of the form (subject, property, object), which denotes that a resource – the subject – has a property whose value is the object. These triples are referred to as statements. RDF also allows for anonymous nodes called blank nodes that can be used as the subject or object of a statement. We call a set of triples an RDF graph, as RDF data can be represented as a directed, labeled graph with typed edges and nodes. In this model, a directed edge labeled with the property name connects the subject to the object. An example RDF graph showing both schema statements and instance statements is shown in Figure 2.1.

RDF Schema (RDFS) (Brickley and Guha, 2004) provides a standard vocabulary for describing the classes and relationships used in RDF graphs and consequently provides the capability to define ontologies. Ontologies serve to formally specify the semantics of RDF data so that a common interpretation of the data can be shared across multiple applications. Classes represent logical groups of resources, and a member of a class is said to be an instance of the class. The RDFS vocabulary offers a set of built-in classes and properties. Two of the most relevant classes are rdfs:Class and rdf:Property, and some of the most relevant properties are rdf:type, rdfs:domain, rdfs:range, rdfs:subClassOf and rdfs:subPropertyOf. The rdf:type property is used to define class and property types (e.g., the triple (S, rdf:type,
2.2. ONTOLOGIES AND THEIR REPRESENTATION

Figure 2.1: Example RDF graph: a simple schema modeling politicians is shown in the upper part of the figure, and example instance data is shown in the lower part of the figure. Resources are represented with ovals and literals are represented with rectangles.

`rdfs:Class` asserts that \( S \) is a class. `rdf:type` is also used to denote instances of classes (e.g., \( (s, \text{rdf:type}, S) \) asserts that \( s \) is an instance of \( S \)). `rdfs:domain` and `rdfs:range` allow us to define the domain and range for a given property, and `rdfs:subClassOf` and `rdfs:subPropertyOf` allow us to create class and property hierarchies.

A set of entailment rules are also defined for RDF and RDFS (Hayes, 2004). Conceptually, these rules specify that an additional triple can be added to an RDF graph if the graph contains triples of a specific pattern. Such rules describe, for example, the transitivity of the `rdfs:subClassOf` property. The set of standard entailment rules defined by Hayes (2004) are shown in Table 2.1.

With respect to other data models, the unique aspects of the RDF model are (1) relationships are represented as first class objects rather than represented implicitly with, for example, foreign key constraints in the Relational model and (2) a formal semantics is specified according to the defined entailment rules for RDF and RDFS.
Table 2.1: RDFS Entailment Rules. The first column shows the rule name. The second column shows a set of RDF statements for a given RDF graph $G$, and the third column shows what statements should be added to $G$.

<table>
<thead>
<tr>
<th>Rule Name</th>
<th>If G Contains:</th>
<th>Then Add:</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs1</td>
<td>$uuu,aaa,lll.$</td>
<td>$_nnn,\text{rdf:type},\text{rdfs:Literal}.$ where $_nnn$ identifies a blank node allocated to $lll$</td>
</tr>
<tr>
<td>rdfs2</td>
<td>$aaa,\text{rdfs:domain},xxx.$ $uuu,aaa,yyy.$</td>
<td>$uuu,\text{rdf:type},xxx.$</td>
</tr>
<tr>
<td>rdfs3</td>
<td>$aaa,\text{rdfs:range},xxx.$ $uuu,aaa,vvv.$</td>
<td>$vvv,\text{rdf:type},xxx.$</td>
</tr>
<tr>
<td>rdfs4a</td>
<td>$uuu,aaa,xxx.$</td>
<td>$uuu,\text{rdf:type},\text{rdfs:Resource}.$</td>
</tr>
<tr>
<td>rdfs4b</td>
<td>$uuu,aaa,vvv.$</td>
<td>$vvv,\text{rdf:type},\text{rdfs:Resource}.$</td>
</tr>
<tr>
<td>rdfs5</td>
<td>$uuu,\text{rdfs:subPropertyOf},vvv.$ $vvv,\text{rdfs:subPropertyOf},xxx.$</td>
<td>$uuu,\text{rdfs:subPropertyOf},xxx.$</td>
</tr>
<tr>
<td>rdfs6</td>
<td>$uuu,\text{rdf:type},\text{rdfs:Property}.$</td>
<td>$uuu,\text{rdfs:subPropertyOf},uuu.$</td>
</tr>
<tr>
<td>rdfs7</td>
<td>$aaa,\text{rdfs:subPropertyOf},bbb.$ $uuu,aaa,yyy.$</td>
<td>$uuu,\text{bbb},yyy.$</td>
</tr>
<tr>
<td>rdfs8</td>
<td>$uuu,\text{rdf:type},\text{rdfs:Class}.$</td>
<td>$uuu,\text{rdfs:subClassOf}$ $\text{rdfs:Resource}.$</td>
</tr>
<tr>
<td>rdfs9</td>
<td>$uuu,\text{rdfs:subClassOf},xxx.$ $vvv,\text{rdf:type},uuu.$</td>
<td>$vvv,\text{rdf:type},xxx.$</td>
</tr>
<tr>
<td>rdfs10</td>
<td>$uuu,\text{rdf:type},\text{rdfs:Class}.$</td>
<td>$uuu,\text{rdfs:subClassOf},uuu.$</td>
</tr>
<tr>
<td>rdfs11</td>
<td>$uuu,\text{rdfs:subClassOf},vvv.$ $vvv,\text{rdfs:subClassOf},xxx.$</td>
<td>$uuu,\text{rdfs:subClassOf},xxx.$</td>
</tr>
<tr>
<td>rdfs12</td>
<td>$uuu,\text{rdf:type},\text{rdfs:ContainerMembershipProperty}.$</td>
<td>$uuu,\text{rdfs:subPropertyOf}$ $\text{rdfs:member}.$</td>
</tr>
<tr>
<td>rdfs13</td>
<td>$uuu,\text{rdf:type},\text{rdfs:Datatype}.$</td>
<td>$uuu,\text{rdfs:subClassOf},\text{rdfs:Literal}.$</td>
</tr>
</tbody>
</table>
2.2.2 OWL

OWL is designed to facilitate greater machine interpretability of data (i.e., more logical reasoning) than what is capable with RDF(S). OWL is based heavily on Description Logics and extends the fact stating abilities of RDF and the class and property defining abilities of RDFS with additional vocabulary (Horrocks et al., 2003). OWL allows defining classes as logical combinations (e.g., intersection, union, complement) and allows additional assertions about property types (e.g., we can state that a property is transitive, symmetric, functional, or the inverse of another property). Another important capability of OWL is the ability to define restrictions on the behavior of a property with respect to a given class. For example, we can define the class of Graduate Student as all individuals who are enrolled in at least 1 course of type Graduate Course. OWL provides three increasingly expressive sublanguages: OWL-Lite, OWL-DL and OWL-Full. OWL-DL is a subset of OWL that allows maximum expressiveness while guaranteeing computational completeness and decidability. OWL-Lite consists of a carefully chosen subset of OWL-DL that eliminates some computational complexity problems that may occur during the inferencing process. OWL-Full provides maximum expressiveness with no computational guarantees.

This dissertation focuses on RDF(S) rather than OWL because we are most interested in exploiting the rich web of named relationships in RDF graphs. We search for specific patterns of relationships and connection types between named entities rather than perform logical reasoning.

2.3 Semantic Analytics

The fundamental premise behind research in Semantic Analytics is that relationships are at the heart of semantics. One can observe the changing focus from documents to entities and on to relationships, and researchers have investigated a broad variety of issues related
to modeling, validating, discovering and exploiting various types of relationships between entities in content (Sheth et al., 2003a). These ideas lead to the concept of Metadata Reference Links (MREFs) that proposed associating semantic metadata with hypertext links (Shah and Sheth, 1998), the development of the InfoQuilt system (Sheth et al., 2003b) that investigated support for hypothesis validation style operations, and the OBSERVER system that focused on inter-ontological relationships and multi-ontology query processing (Mena et al., 1996).

More recently, and in step with the emergence of the Semantic Web, research on complex relationships lead to the definition of semantic associations (Anyanwu and Sheth, 2002). Semantic associations are based on intuitive notions such as connectivity and semantic similarity (see Figure 2.2). Anyanwu and Sheth (2003) gave a formalization of semantic associations using the RDF data model and defined a set of $\rho$-operators for querying semantic associations. The most fundamental of these operators is the $\rho$-path operator. The $\rho$-path operator asks the following question: “How is resource $X$ related to resource $Y$?” over an RDF graph and returns a set of paths connecting $X$ to $Y$. Researchers have also investigated the challenging issue of ranking these paths (Anyanwu et al., 2005; Aleman-Meza et al., 2005). Ranking semantic associations was necessitated by the sheer number of such associations even on moderate-size RDF graphs. Even a ranked list of associations could be a daunting task for a user to interpret and may in some cases cause a severe cognitive overload. In a related effort aimed at reducing such a cognitive overload, subgraph discovery techniques have been adapted to discover relatively small but informative subgraphs connecting the entities in the result of a given execution of the $\rho$-path operator (Ramakrishnan et al., 2005).

This dissertation is concerned with expanding work on semantic analytics from purely thematic relationships to thematic, spatial and temporal relationships. We show how to model STT data using RDF and provide a means to execute graph pattern queries involving spatial and temporal relationships.
Figure 2.2: Example semantic associations. James Larson (resource &r1) is $\rho$-path associated with Wright State University (resource &r4) because the two resources are connected by a path in the RDF graph, and resource &r5 is $\rho$-iso associated with resource &r2 because the two resources are involved in instances of two identical schema-level paths: they are both papers authored by university employees.
Related Work

This research lies at the intersection of many areas of Computer Science. It builds on traditional research in Databases involving indexing and query processing, especially from spatial and temporal databases and database techniques for managing RDF data. This research also builds on work in semantics and knowledge representation involving ontological modeling and inferencing. In addition, we build on spatiotemporal modeling research from the GIS community. This chapter reviews related work from these different areas and points out differences with our work. We divide related work into two categories: (1) data modeling and (2) data storage and querying processing.

3.1 Data Modeling

We first discuss the use of ontologies in Geographic Information Science (GIS) and then cover spatiotemporal modeling approaches.

3.1.1 Ontologies and GIS

There has been significant work regarding the use of geospatial ontologies in GIS. In general, Ontologies in GIS are seen as a vehicle to facilitate interoperability and to limit data integration problems both from different systems and between people and systems (Agarwal, 2005). Fonseca et al. (2002) present an architecture for an ontology-driven GIS in
which ontologies describe the semantics of geographic data and act as a system integrator independent of the data model used (e.g., object vs. field). Kuhn and Raubal (2003) introduced the concept of semantic reference systems, of which ontologies are a component, as a means to describe the same geographic information from varying perspectives. This includes notions of semantic transformation and projection of ontologies. These operations could potentially be used to present geographic information from different scales and granularities.

On the Web, the use of ontology for better search and integration of geospatial data and applications is embodied in the Geospatial Semantic Web (Egenhofer, 2002). From a Web context, Kolas et al. (2005) outline specific types of geospatial ontologies needed for integration of GIS data and services: base geospatial ontology, feature data source ontology, geospatial service ontology, and geospatial filter ontology. The base geospatial ontology provides core geospatial knowledge vocabulary while the remaining ontologies are focused on geospatial web services.

In summary, ontologies in the geospatial domain can be divided into two categories: (1) domain ontologies and (2) fundamental ontologies of geographic space. Domain ontologies define concepts of the Geography domain. Some typical examples of domain ontologies are available from the Ordnance Survey in the United Kingdom (Ordnance Survey, 2008). These include a hydrology ontology, an ontology of administrative geography and an ontology of buildings and places. Fundamental ontologies of space define concepts such as spatial regions, coordinate systems and topological relations.

Our work is complementary to the work on geo-ontologies. The spatial portion of our upper-level ontology (presented in Section 4.2) serves as a fundamental ontology of geographic space and the thematic portion of our upper-level ontology allows for integration of arbitrary domain-specific ontologies. Our work goes beyond the modeling aspects of geo-ontologies and provides a framework to efficiently utilize both geo-ontologies and
arbitrary domain-specific ontologies for querying in an information system.

### 3.1.2 Spatiotemporal Models

Spatiotemporal data models have received considerable attention in both the GIS and Database communities, and many good surveys exist (e.g., Pelekis et al. (2004) and Peuquet (2001)). In a recent survey, Pelekis et al. (2004) identify 10 distinct spatiotemporal data models. In general, our modeling approach differs through its extensive use of thematic relationships. We not only conceptually separate thematic entities from spatial entities, but we also utilize indirect thematic relationships to link thematic entities to spatial entities in a variety of ways (i.e., different contexts). Here, we review the modeling approaches that are most similar to ours.

Of the models discussed in the literature, the three domain model is conceptually the most similar to our RDF-based approach. The three domain model was introduced by Yuan (1994, 1996). This model represents semantics, space, and time separately. To represent spatiotemporal information in this model, semantic objects are linked via temporal objects to spatial objects. This provides temporal information about the semantic (thematic) properties of a given spatial region. This is analogous to temporal located at and occurred at relationships in our upper-level ontology (Section 4.2). The three domain model is quite similar to our approach in that it represents thematic entities as first-class objects rather than attributes of geospatial objects. The key difference is that the three domain model relies on direct connections from thematic entities to spatial regions whereas our model allows more flexibility through indirect connections composed of sequences of thematic relationships.

Our modeling approach also has similarities with object-oriented approaches. A recent proposal by Worboys and Hornsby (2004) combines the object-oriented and event-based modeling approaches to model dynamic geospatial domains. They define an upper-level ontology similar to the one we present in Section 4.2. They model the concept of a setting
and a situate function that maps entities and events to settings. Settings can be spatial, temporal, or spatiotemporal. In contrast to our work, the authors focus on geospatial objects and events and model what we would consider a thematic entity (e.g., an airplane) as a geospatial entity. That is, the separation between the thematic and spatial domains is not as strongly emphasized. Our RDF-based modeling approach provides a means to assign spatial properties to those entities not directly connected to a spatial setting and allows deeper analysis of purely thematic relationships.

General modeling approaches and languages have also been extended for spatiotemporal data. Tryfona and Jensen (1999, 2000) extended the entity-relationship model to create the spatiotemporal entity-relationship model (STER). Price et al. (2002) extended the Unified Modeling Language (UML) to create spatiotemporal UML. RDF is similar to these modeling languages in the sense that it is a general purpose ontology language and can model entities and relationships for a given domain. Our approach could therefore be seen as an extension of RDF (i.e. spatial types in combination with temporal triples) to allow for modeling spatial and temporal entities and relationships. RDF is different from these other languages in that it also serves as a model for storing and querying data in the form of RDF triples whereas UML and ER are primarily for conceptual modeling. We can thus query relationships directly as first class objects in RDF graphs, and we utilize this capability to design and implement relationship-based query operators. Furthermore, RDF statements carry well-defined semantics, and corresponding inferencing mechanisms must be supported.

### 3.2 Data Storage and Query Processing

We first review approaches to querying thematic RDF data and then discuss querying spatial and temporal data on the Semantic Web. This is followed by a review of querying spatial and temporal data using traditional database technology.
3.2. DATA STORAGE AND QUERY PROCESSING

3.2.1 Storing and Querying RDF

Many RDF query languages have been proposed in the literature. These include SQL-like languages (e.g., SPARQL (Prud’hommeaux and Seaborne, 2008), RDQL (Seaborne, 2004)), functional languages (e.g., RQL (Karvounarakis et al., 2002)), rule-based languages (e.g., TRIPLE (Sintek and Decker, 2002)) and graph traversal languages (e.g., RxPath (Souzis, 2004)). For a detailed comparison of these languages, see (Haase et al., 2004; Angles and Gutierrez, 2005). Recently, SPARQL has emerged as a W3C recommendation. SPARQL and our spatiotemporal extension of it, SPARQL-ST, is discussed further in Chapter 8. As an alternative to defining a new query language, an approach for querying RDF data directly in SQL has been proposed (Chong et al., 2005). This facilitates easy integration with other SQL queries against traditional relational data and saves the overhead of translating data from SQL to the RDF query language data format. Our implementation described in Chapter 6 follows this approach and introduces new SQL functions for spatial and temporal querying of RDF data.

A variety of systems for management of persistent RDF data have been presented in the literature. These systems usually rely on an underlying relational database representation. Three main types of storage schemes are commonly used (Theoharis et al., 2005): (1) schema-aware - one table per RDF(S) class or property (e.g., Sesame using PostgreSQL (Broekstra et al., 2002), the vertical partitioning scheme presented by Abadi et al. (2007)), (2) schema-oblivious - a single three-column (subject, predicate, object) table storing all statements (e.g., Jena (Wilkinson et al., 2003), 3Store (Harris and Gibbins, 2003), Sesame using MySQL (Broekstra et al., 2002), Oracle Semantic Data Store (Oracle, 2005b)) and (3) hybrid - one table storing class membership information and one table for each group of properties with the same range type such as Resource or integer (e.g., RDFSuite (Alexaki et al., 2001)). These different storage schemes are shown in Figure 3.1. Efficient evaluation of queries using these systems typically involves transformation into a SQL query against
3.2. DATA STORAGE AND QUERY PROCESSING

Figure 3.1: Illustration of RDF storage schemes adapted from Theoharis et al. (2005)

the underlying RDBMS representation, and traditional relational indexes are used to speed up query processing.

Alternate approaches persistently store RDF data using lower-level structures such as Hash Tables (Redland (Beckett, 2002)) and $B^+$-Trees (YARS (Harth and Decker, 2005)) and traverse these structures to evaluate queries.

All the previously mentioned techniques index RDF data based on a “collection of triples” conceptualization. The GRIN index, proposed by Udrea et al. (2007), exploits the graph structure of the RDF data. A GRIN index is a tree structure where leaf nodes represent a set of triples in the RDF graph and interior nodes are represented by a vertex, radius pair $(v, r)$ that represents all vertices in the RDF graph within $r$ hops of vertex $v$. Graph pattern queries are evaluated by traversing the tree to find all triples that may contain an answer to the query. A subgraph matching algorithm is then run over the identified portion of
the RDF graph. The initial implementation of GRIN used a main-memory representation, which was followed by a disk-based implementation using PostgreSQL (Pugliese et al., 2008).

Our approach uses an underlying relational database representation of RDF data that follows the schema-oblivious storage scheme. This storage scheme is augmented with additional structures for more efficient searching over spatial and temporal data. We utilize traditional spatial and temporal indexes in our query processing strategies and use composite $B^+$-tree indexes for efficient evaluation of graph pattern queries.

### 3.2.2 Spatial and Temporal Data on the Semantic Web

Work is somewhat limited with regards to incorporating spatial and temporal relationships into queries over Semantic Web data. Examples of querying geospatial RDF data are mostly seen in Web applications and semantic geospatial web services (Kammersell and Dean, 2006; Tanasescu et al., 2006). In general, this work mainly focuses on interoperability, and query processing proceeds by translating RDF representations of spatial features into geometric representations on the fly and then performing spatial calculations. In contrast, we look at how the relationship-centric nature of the RDF model can enable new query types and also address issues related to efficient query processing.

The SPIRIT spatial search engine (Jones et al., 2004) combines an ontology describing the geospatial domain with the searching and indexing capability of Oracle Spatial for the purposes of searching documents based on the spatial features associated with named places mentioned in the document. In contrast, our searching operators are intended for general purpose querying of ontological and spatial relationships.

Querying for temporal data in RDF graphs is less complicated, as RDF supports typed literals such as xsd:date, and corresponding query languages support filtering results based
3.2. DATA STORAGE AND QUERY PROCESSING

on literal values. However, this is far from supporting full temporal RDF as graphs discussed in this paper.

Gutierrez et al. (2005, 2007) introduced the concept of temporal RDF graphs and formally defined them. In addition, the authors briefly discussed aspects of a query language for temporal RDF graphs, but a through investigation of such a language has not been completed, and no implementation issues were mentioned. To the best of our knowledge, our work in (Perry et al., 2007) is the first to investigate efficient schemes for storing and querying temporal RDF and implementation of RDFS inferencing that incorporates the concept of valid time for RDF statements. Pugliese et al. (2008) present tGRIN an extension of the GRIN index for temporal RDF data. The tGRIN extension factors in the temporal distance between vertices in addition to the graph distance (number of edges). The authors approach using tGRIN, however, supports a more limited form of temporal RDFS inferencing than we do. Specifically, they only support inferences related to rdfs:subPropertyOf. Pugliese et al. also support a different form of temporal RDF queries than we support. Their queries involve temporal conditions on single edges of a graph pattern. In contrast, our queries involve temporal conditions on time intervals derived from multiple edges in a graph pattern (e.g., the intersection of the time intervals of each edge in a graph pattern).

Semantic Web researchers have proposed incorporating past work on qualitative spatial and temporal reasoning into the Semantic Web reasoning framework as an alternative to adding spatial and temporal capabilities to query languages. Hobbs and Pan (2004) translated a subset of Allen’s interval calculus (Allen, 1984; Allen and Ferguson, 1994) to OWL to create the OWL-Time ontology. Abdelmonty et al. (2005) demonstrated that OWL is insufficient to fully support the spatial reasoning required for a geo-ontology (e.g., it is very hard to define a class of HousesNearMotorways made up of individuals of type house that are within a specific distance of motorways). In a follow-on paper, Smart et al. (2007) showed how to use additional rules and specialized tools to help overcome the shortcomings of OWL. Our approach differs in that our implementation does not involve reasoning
over relative spatial and temporal relations (e.g., $(x \text{ before } y) \land (y \text{ before } z) \Rightarrow (x \text{ before } z)$).

Instead we support the computation of spatial and temporal relations using time values that are grounded to a timeline and spatial features that are grounded to a coordinate system.

### 3.2.3 Spatial and Temporal Query Processing

Management of spatial and temporal data has long been an area of interest (Guting, 1994; Guting et al., 2000; Ozsoyoglu and Snodgrass, 1995).

Processing temporal queries over relational data is well covered in the literature. Usually temporal information is stored as time intervals. Selection queries generally retrieve all intervals that intersect a given query interval. Various structures have been proposed for efficient execution of such queries (Salzberg and Tsotras, 1999). Another important task is interval join queries that join two relations based on overlapping intervals. Many approaches to evaluate these joins exist in the literature (Gao et al., 2005).

Processing spatial queries is also a well-researched topic. Spatial selection queries return a set of spatial objects that satisfy a spatial predicate (Aref and Samet, 1991). Various types of spatial index structures have been developed for such queries (e.g., the R-Tree (Beckman et al., 1990; Guttman, 1984) and quadtree (Samet, 1984)). Also important are spatial join queries, which join sets of spatial objects based on a spatial predicate. A variety of methods for evaluating spatial joins have been proposed (Arge et al., 2000; Brinkhoff et al., 1993; Gunther, 1993).

Work on indexing and querying spatiotemporal data or moving objects is also of interest (Guting et al., 2000). Indexing approaches usually optimize queries about future positions of spatiotemporal objects or queries about past states of the spatiotemporal objects (Hadjieleftheriou et al., 2002). Various approaches to indexing spatiotemporal objects appear in the literature (Mokbel et al., 2003).
3.2. DATA STORAGE AND QUERY PROCESSING

A key difference of the query types addressed here is our focus on thematic relationships. Rather than querying a set of spatial or temporal objects, we are querying thematic objects associated to spatial objects via a chain of thematic relationships (i.e. in a specific context). For example, the following relationships could represent a battle participation context: (Soldier, on_crew_of, Vehicle) (Vehicle, used_in, Battle) (Battle, occurred_at, Spatial Region). In other words, the spatial object associated with an entity is determined dynamically at run time. Therefore, we cannot create direct spatial indexes for these thematic entities. Similarly, we compute a temporal interval for a subgraph connecting multiple entities, also dynamically generated at run-time, making it infeasible to directly index the derived intervals. Rather than trying to improve upon existing indexing techniques for traditional queries over spatial and/or temporal objects, we focus on how to incorporate these indexing techniques into our query processing procedures.
Modeling Approach

Our ontology-based modeling approach is presented in this chapter. We give formalizations of RDF and Temporal RDF and present the core ontologies used in our modeling approach.

We use the running scenario of historical analysis of battlefield events of World War II to illustrate concepts in the remainder of the dissertation. We chose this scenario because it is easy to understand and because we have generated large synthetic datasets corresponding to this scenario that are used in our evaluation.

This work appears in Perry et al. (2006) and Perry and Sheth (2008).

4.1 Preliminaries

RDF graphs and temporal RDF graphs are formally defined in this section.

4.1.1 RDF

RDF has been adopted by the W3C as a standard for representing metadata on the Web. The RDF data model is defined as follows. Let $U$, $L$ and $B$ be pairwise disjoint sets of URIs, literals and blank nodes, respectively. The union of these sets $U \cup B \cup L$ is referred to as the set of RDF Terms $RT$. An RDF triple is a 3-tuple $(s, p, o) \in (U \cup B) \times U \times RT$ where $s$ is the subject, $p$ is the property and $o$ is the object. A set of RDF triples is referred to as
4.1 PRELIMINARIES

an RDF Graph, as RDF can be represented as a directed, labeled graph where a directed edge labeled with the property name connects a vertex labeled with the subject name to a vertex labeled with the object name.

4.1.2 Temporal RDF

In order to analyze the temporal properties of relationships in RDF graphs, we need a way to record the temporal properties of the statements in those graphs, and we must account for the effects of those temporal properties on RDFS inferencing rules. Gutierrez et al. (2005, 2007) introduced the notion of temporal RDF graphs for this purpose.

Temporal RDF graphs model linear, discrete, absolute time and are defined as follows (Gutierrez et al., 2007). Given a set of discrete, linearly ordered time points $T$, a temporal triple is an RDF triple with a temporal label $t \in T$. A statement’s temporal label represents its valid time. The notation $(s, p, o) : [t]$ is used to denote a temporal triple. The expression $(s, p, o) : [t_1, t_2]$ is a notation for $\{(s, p, o) : [t] \mid t_1 \leq t \leq t_2\}$. A temporal RDF graph is a set of temporal triples. For a temporal RDF graph $G_t$, $TRIPLES(G_t)$ denotes the set $\{(s, p, o) \mid \exists t \in T \text{ with } (s, p, o) : [t] \in G_t\}$.

The following example illustrates these concepts. Consider a soldier $s1$ assigned to the 1st Armored Division ($1stAD$) from April 3, 1942, until June 14, 1943, and then reassigned to the 3rd Armored Division ($3rdAD$) from June 15, 1943, until October 18, 1943. This would yield the following triples: $(s1, assigned\_to, 1stAD) : [04:03:1942, 06:14:1943]$, $(s1, assigned\_to, 3rdAD) : [06:15:1943, 10:18:1943]$.

We must also account for the effects of temporal labels on RDFS inferencing rules (see Section 6.2.2.2). To incorporate inferencing into temporal RDF graphs, a basic arithmetic of intervals is needed to derive the temporal label for inferred statements. For example, interval intersection would be needed for rdfs:subClassOf (e.g., $(x, rdfs:subClassOf, y) : [1, 4] \land (y, rdfs:subClassOf, z) : [3, 5] \Rightarrow (x, rdfs:subClassOf, z) : [3, 4]$).
4.2. ONTOLOGY-BASED MODEL

Here we discuss our ontology-based approach for modeling theme, space and time. We present an upper-level ontology defining a general hierarchy of thematic and spatial entity classes and associated relationships connecting these entity classes (see Figure 4.1). We intend for application-specific domain ontologies in the thematic dimension to be integrated into the upper-level ontology through subclassing of appropriate classes and relationships.

Figure 4.1 illustrates how a simple ontology from our historical analysis of WWII example would be integrated with this upper-level ontology. Temporal information is integrated into the ontology by labeling relationship instances with their valid times. A unique aspect of this approach is that we do not require the spatial properties of each thematic entity to be explicitly recorded. Instead, we utilize relationships in the thematic domain to indirectly provide spatial properties. This gives the benefit of greater flexibility in the integration of thematic and spatial information.
4.2. ONTOLOGY-BASED MODEL

4.2.1 Thematic Dimension

Our upper-level thematic ontology consists of a fundamental class hierarchy and a few basic relationships. In developing the class hierarchy, we first follow the approach of Basic Formal Ontology introduced by Grenon and Smith (2004) and distinguish between Continuants and Occurrents. Continuants are those entities that persist over time and maintain their identity through change. Examples from our historical battlefield analysis scenario could include a soldier, an aircraft or a city. Occurrents represent events and processes; they happen and then no longer exist. Examples are the bombing of a target or the execution of a training exercise. A second division of entities concerns spatial properties. Some Occurrents are inherently spatial such as a battle; others are not, such as the assignment of a soldier to a division. We therefore explicitly represent Spatial Occurrents and Non-Spatial Occurrents. Continuants also have varying spatial properties. We distinguish a special type of Continuant that we refer to as a Named Place. Named Places are entities that serve as locations for other physical entities and Spatial Occurrents. They have very static spatial behavior over time and are distinguished by a strong association with their spatial location. Examples of Named Places include a city, a zip code, a building, or a lake. In contrast to a Named Place, we distinguish another subclass of Continuant: Dynamic Entity. Dynamic Entities are those entities with dynamic spatial behavior whose identities are not as strongly associated with space. Examples include a person or a vehicle. We do not make further philosophical distinctions between these two types of Continuants as the final decision depends upon the domain and application.

4.2.2 Spatial Dimension

The spatial portion of our upper-level ontology consists of a top-level class and two corresponding relations. Spatial Regions represents basic spatial geometries (i.e. georeferenced points, lines and polygons). The occurred at relation connects Spatial Occurrent to Spatial
4.2. ONTOLOGY-BASED MODEL

Figure 4.2: GeoRSS GML-based ontology modeling basic spatial geometries. Note that Geometric Aggregates contain collections of their respective Geometric Primitives (e.g., MultiPolygon contains a collection of Polygons). These relations and attributes of Coordinate Reference System have been left out of the figure for clarity.

Region, and located at connects Named Place to Spatial Region. These relations allow us to associate a thematic concept, such as the city of Berlin or the Battle of the Bulge, with its geospatial properties. Spatial properties of thematic entities can consequently be derived using the associated Spatial Regions.

The spatial features represented by the Spatial Region class are complex types that need to be fully modeled with a spatial ontology. Fortunately, there is movement towards standard ontologies for spatial geometries, for example work done as part of the Open Geospatial Consortium (OGC) Semantic Web Interoperability Experiment (Open Geospatial Consortium, 2008a) and the W3C geo incubator group (Lieberman, 2006). The existing OGC Geographic Markup Language (GML) specification serves as an excellent basis for these ontologies as discussed in Abdelmonty et al. (2005) and Kolas et al. (2005). We propose a spatial ontology based on the GeoRSS GML specification (Singh et al., 2008). The ontology models 2-dimensional spatial geometries and associated spatial reference system information. Figure 4.2 illustrates the RDF representation of this ontology.

4.2.3 Temporal Dimension

We use temporal RDF graphs (Gutierrez et al., 2007) to incorporate the time dimension into our model. Temporal information is represented by associating time intervals with re-
4.2. **ONTOLOGY-BASED MODEL**

Figure 4.3: Temporal reification of the RDF statement \((A B C)\). Constructs from the Owl-Time ontology are shown in gray.

relationship instances in the ontology. The time interval on the relationship denotes the times at which the relationship is valid. These time intervals are grounded to a discrete, linearly-ordered timeline. RDF reification is used to associate time intervals with RDF statements to realize temporal RDF graphs. RDF reification is a construct in RDF that allows one to make statements about statements, so we can assert that a given RDF statement has a given valid time. We use a portion of the OWL-Time ontology (Hobbs and Pan, 2004) to model the time intervals, and a new property *temporal* asserts that the reified statement is valid during the given time interval. Figure 4.3 illustrates this approach.
Query Operator Design

Our approach for querying over this ontology-based model utilizes the graph-centric structure of RDF data. For spatial aspects, we use subgraphs in the RDF graph to connect thematic entities (e.g., Dynamic Entities) to Spatial Regions. A given thematic entity can be connected to various Spatial Regions through a variety of different subgraphs, yielding a many-to-many mapping. Associated domain ontologies clarify the semantics of these subgraphs, and we refer to a given subgraph as a context. That is, a thematic entity has spatial properties with respect to a given context. Using our historical analysis of WWII example, a soldier could be associated with the spatial properties of his residence in one context (Soldier, lives_at, Residence) (Residence, located_at, Spatial.Region) or with the locations of his training facilities using a different context (Soldier, member_of, Military.Unit) (Military.Unit, trains_at, Base) (Base, located_at, Spatial.Region). For temporal aspects, we derive temporal intervals for these subgraphs through computations over the temporal values of the edges (temporal RDF triples) that make up the subgraph.

This chapter introduces and formalizes a set of query operators that follow the basic approach outlined above. We introduce spatial operators that allow (1) retrieving the spatial properties of an entity with respect to a given context (spatial.extent), (2) retrieving the set of entities whose associated Spatial Regions satisfy a spatial predicate (spatial.restrict) and (3) retrieving pairs of entities whose associated spatial regions satisfy a given spatial relation (spatial.eval). We introduce temporal operators that allow (1) deriving a temporal interval for a subgraph in the RDF graph (temporal.extent), (2) filtering a set of subgraphs
5.1. GRAPH PATTERNS

by evaluating a temporal predicate over their derived time intervals (temporal_restrict),
and (3) retrieving pairs of subgraphs whose time intervals satisfy a given temporal relation
(temporal_eval).

Our framework differs from traditional approaches to querying RDF data in that com-
putation of implicit relationships are supported. We do not rely on the existence of explicit
RDF statements asserting spatial and temporal relationships such as inside and after. In-
stead, we perform computations at query time to establish the existence of these relation-
ships that are implicit in the RDF dataset.

This work appears in Perry et al. (2006) and Perry and Sheth (2008).

5.1 Graph Patterns

Our querying approach relies on specifying a type of connection between resources in an
RDF graph. We use SPARQL-like graph patterns to express these connection types. Con-
ceptually, a graph pattern is a set of RDF triples where the subjects, properties and/or
objects may be replaced with variables. In general, a graph pattern query against an RDF
graph $G$ returns a set of mappings between the variables in the graph pattern and terms
(URIs, Blank Nodes and Literals) in $G$ such that replacing variables with their correspond-
ing terms results in a set of triples actually present in $G$. Figure 5.1 illustrates an example
graph pattern query. A formal syntax for SPARQL graph patterns and formal semantics for
SPARQL graph pattern queries is given in Perez et al. (2006). We present a fragment of this
formalization to define the general concept of a graph pattern, which we use to formally
define our proposed query operators.

Let $UL$ denote the union $U \cup L$ (recall that $U$ is the set of URIs and $L$ is the set of
Literals) and let $V_N$ be a set of variables disjoint from the set of RDF Terms $RT$.

A graph pattern is defined recursively as follows:
5.1. GRAPH PATTERNS

Figure 5.1: Example graph pattern from historical analysis of WWII scenario with resulting variable bindings.

- A tuple from \((UL \cup V_N) \times (U \cup V_N) \times (UL \cup V_N)\) is a graph pattern (triple pattern).

- If \(P_1\) and \(P_2\) are graph patterns, then \((P_1 \text{ AND } P_2)\) is a graph pattern.

The semantics of a graph pattern are defined in terms of a function \([[\cdot]]\), which takes a graph pattern expression and returns a set of mappings where a mapping \(\mu : V_N \rightarrow RT\) is a function from \(V_N\) to \(RT\). For a triple pattern \(tp\), we denote the set of variables in \(tp\) as \(\text{var}(tp)\), and we denote the triple obtained by replacing the variables in \(tp\) according to the mapping \(\mu\) as \(\mu(tp)\). For a graph pattern \(GP\), we denote the set of triples obtained by replacing the variables in \(GP\) according to \(\mu\) as \(\mu(GP)\), and we refer to this set of triples as an instance of \(GP\). For a mapping \(\mu\), the subset of \(V_N\) where it is defined is called its domain \(\text{dom}(\mu)\). Two mappings \(\mu_1\) and \(\mu_2\) are compatible if for all \(x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2)\), it is the case that \(\mu_1(x) = \mu_2(x)\). In other words the union \(\mu_1 \cup \mu_2\) is also a mapping. In addition, for two sets of mappings \(M_1\) and \(M_2\), the join is defined as:

\[ M_1 \bowtie M_2 = \{ \mu_1 \cup \mu_2 \mid \mu_1 \in M_1 \text{ and } \mu_2 \in M_2 \text{ and } \mu_1 \text{ and } \mu_2 \text{ are compatible mappings} \} \]

Let \(G\) be an RDF graph, \(tp\) a triple pattern and \(P_1, P_2\) graph patterns. The evaluation of a graph pattern over \(G\), denoted \([[\cdot]]_G\), is defined recursively as:

- \([[tp]]_G = \{ \mu \mid \text{dom}(\mu) = \text{var}(tp) \text{ and } \mu(tp) \in G \} \)
5.2. SPATIAL OPERATORS

We define our spatial operators using what we term a spatial context. Conceptually, a spatial context specifies a type of connection between a thematic entity and a spatial entity. Given a temporal RDF graph $G_t$, a spatial context is defined as a 2-tuple $(GP, v)$ where $GP$ is a graph pattern and $v \in \text{var}(GP)$ is a variable in $GP$ identifying a Spatial Region instance. That is, for each mapping $\mu \in [[GP]]_{\text{TRIPLES}(G_t)}$ with $\mu(v) = x$, there exists a triple $(x, \text{rdf:type}, \text{Spatial Region})$ in $\text{TRIPLES}(G_t)$. Note that $G$ in the previous section refers to a plain RDF graph, and here $G_t$ refers to a temporal RDF graph. Also recall that $\text{TRIPLES}(G_t)$ denotes the plain RDF graph created by removing the temporal information from $G_t$. As an example, consider the spatial context below that connects a soldier (?x) to a Spatial Region (?s).

```dbc
('(?x assigned_to ?y) (?y participates_in ?z)
(?z occurred_at ?s')
```

In the following, for a Spatial Region URI $sr$, we use $\text{geom}(sr)$ to refer to the actual spatial geometry (i.e. point, line, polygon) represented by $sr$ according to the spatial ontology described in Section 4.2. We use $S$ to denote the set of all possible spatial geometries.

The first spatial operator we define, $\text{spatial_extent}$, is intended to find the spatial properties of a thematic entity with respect to a given spatial context. The query “what are the spatial properties of the 101st Airborne Division with respect to battle participation” (Example 1) illustrates an example search using this operator. We can think of this operator as retrieving the spatial features corresponding to the identified Spatial Region in the result subgraphs of a graph pattern query.

$$\text{spatial_extent}((GP, v))_{G_t} \rightarrow \{ (\mu, s) \}$$
5.2. SPATIAL OPERATORS

Given:

a spatial context \((GP, v)\), a temporal RDF graph \(G_t\)

Find:

\[\{ (\mu, s) \mid \mu \in [GP]_{TRIPLES(G_t)} \text{ and } s = geom(\mu(v)) \}\]

Example 1.

\[\text{ANS} \leftarrow \text{spatial-extent}'(''(101st Airborne Division) participates in ?x)\)
\[\quad (?x occurred at ?s)', '=?s')_{G_t}\]

The next two spatial operators focus on spatial relationships. As a prerequisite, we define a spatial formula, which is used to express conditions on spatial relationships. Spatial formulas are built from qualitative spatial functions and metric spatial functions. A qualitative spatial function is a Boolean function \(qsf : S \times S \rightarrow \mathbb{B}\). Any of the following topological spatial relations identified by Egenhofer and Herring (1994) may be used as qualitative spatial functions in our formalization: disjoint, touch, overlap boundary disjoint, overlap boundary intersect, equal, contains, covers, inside, covered by. We define a qualitative spatial expression, \(qse\), as follows, where \(s_1, s_2 \in S \cup V_S\).

\[\langle qse \rangle := qsf(s_1, s_2)\]

A metric spatial function is a function \(msf : S \times S \rightarrow \mathbb{R}\). We use one metric spatial function \(distance : S \times S \rightarrow \mathbb{R}\), which returns the distance between two spatial geometries. Let \(V_S\) be a set of variables disjoint from \(V_N\) and \(RT\). We define a metric spatial expression, \(mse\), as follows, where \(s_1, s_2 \in S \cup V_S\) and \(r \in \mathbb{R}\).

\[\langle mse \rangle := \langle msf(s_1, s_2) \rangle \langle \text{comp} \rangle r\]
\[\langle \text{comp} \rangle := < | > | \leq | \geq | =\]

A spatial formula \(sf\) evaluates to a Boolean value for a given graph and is defined in terms of metric spatial expressions and qualitative spatial expressions. A spatial formula takes the following form.
5.2. SPATIAL OPERATORS

\( \langle sf \rangle ::= \langle mse \rangle | \langle qse \rangle | \langle sf \rangle \ AND \ \langle sf \rangle | \langle sf \rangle \ OR \ \langle langs f \rangle | \ NOT \ \langle sf \rangle \)

The spatial formulas used in our formalization are expressions containing exactly one free variable \( %s \) or exactly two free variables \( %s_1 \) and \( %s_2 \) and are denoted as \( sf(\%s) \) and \( sf(\%s_1, \%s_2) \).

The next spatial operator, \( spatial\_restrict \), is designed to retrieve thematic entities based on their spatial relationships with a given location in a given context. An example of this type of search is “which military units have spatial extents that are within 20 miles of (48.45 N, 44.30 E) in the context of battle participation?” Note that the variable \( %s \) used in the spatial formula is different from the variable \( v \) in the graph pattern that represents a Spatial Region instance, as \( v \) corresponds to a URI and \( %s \) corresponds to a spatial geometry. (Example 2).

\( spatial\_restrict ((GP, v), sf(\%s))_{Gt} \rightarrow \{(\mu, s)\} \)

Given:

- a spatial context \((GP, v)\), a spatial formula \( sf \) defined over \( S \) and a variable \( %s \),
- a temporal RDF graph \( G_t \)

Find:

\[ \{(\mu, s) \mid \mu \in [[GP]]_{TRIPLES(G_t)} \text{ and } s = geom(\mu(v)) \text{ and } sf \text{ evaluates to true for } %s = s\} \]

Example 2.

ANS ← \( spatial\_restrict((\text {?x participates in ?y}) (\text {?y occurred at ?s}), \text {?s}', \text {?s}',
\text {distance(%s, (48.45N, 44.30E)) ≤ 20 miles})_{Gt} \)

The final spatial operator, \( spatial\_eval \), investigates how thematic entities are related in space. We can think of this operator as a spatial join between thematic entities with respect to a given context. As an example, consider the query “which infantry unit’s operational area overlaps the operational area of the 3rd Armored Division?” (Example 3).
5.3. TEMPORAL OPERATORS

Given:
  
  a spatial context \((GP_1, v_1)\), a spatial context \((GP_2, v_2)\), a spatial formula \(sf\)
  
  defined over \(S\) and variables \(\%s_1, \%s_2\), a temporal RDF graph \(G_t\)

Find:
  
  \(\{ (\mu_1, s_1, \mu_2, s_2) \mid \mu_1 \in [[GP_1]]_{{TRIPLES(G_t)}} \land \mu_2 \in [[GP_2]]_{{TRIPLES(G_t)}} \land \)
  
  \(s_1 = geom(\mu_1(v_1)), s_2 = geom(\mu_2(v_2)) \land \)
  
  \(sf\) evaluates to \(true\) for \(\%s_1 = s_1, \%s_2 = s_2\) \}

Example 3.

\[
\text{ANS} \leftarrow \text{spatial eval}(''(?x_1 \text{ participates in } ?y_1) ('?y_1 \text{ occurred at } ?s_1)'', ''?s_1'').
\]

\[
(''(3rd Armored Division) \text{ participates in } ?y_2)(''?y_2 \text{ occurred at } ?s_2)'', ''?s_2'').
\]

\[
\text{overlap-boundary-intersect } (\%s_1, \%s_2) = \text{true})_{G_t}
\]

5.3 Temporal Operators

The basic idea behind our temporal operators is that we derive a time interval for a graph pattern instance using the time intervals associated with the triples in the graph pattern. These derived intervals are used to restrict graph pattern query results and to perform temporal joins between graph pattern instances.

We will first give some initial definitions. Let \(T\) be a set of totally ordered time points.

Let \(G_t\) be a temporal RDF graph defined over \(T\). For each statement \(e = (s, p, o) \in TRIPLES(G_t)\), let \(\text{temporal}(e) = \{ t \mid (s, p, o) : [t] \in G_t \}\). For a set of time points \(T' \subseteq T\), let \(\text{contig_intervals}(T') = \{ [t_i, t_j] \mid \forall t \in T : (if t_i \leq t \land t \leq t_j then t \in T') \land t_{i-1} \notin T' \land t_{j+1} \notin T' \}\).

Consider the following example:
Suppose:

\[ T = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\} \]

\[ T' = \{2, 3, 4, 7, 8\} \]

Then:

\[ \text{contig} \text{ intervals}(T') = \{[2, 4], [7, 8]\} \]

Given a set of temporal triples \( E = \{e_1, e_2, ..., e_n\} \), we define the interval expansion \( \text{int} \text{ expansion}(E) \), as the set

\[
\text{contig} \text{ intervals}(\text{temporal}(e_1)) \times \text{contig} \text{ intervals}(\text{temporal}(e_2)) \times ... \text{contig} \text{ intervals}(\text{temporal}(e_n))
\]

Consider the following example:

Suppose:

\[ E = \{e_1, e_2, e_3\} \]

\[ \text{contig} \text{ intervals}(\text{temporal}(e_1)) = \{[2, 4], [7, 8]\} \]

\[ \text{contig} \text{ intervals}(\text{temporal}(e_2)) = \{[1, 5], [7, 9]\} \]

\[ \text{contig} \text{ intervals}(\text{temporal}(e_3)) = \{[4, 5]\} \]

Then:

\[ \text{int} \text{ expansion}(E) = \{\{[2, 4], [1, 5], [4, 5]\}, \{[2, 4], [7, 9], [4, 5]\}, \{[7, 8], [1, 5], [4, 5]\}, \{[7, 8], [7, 9], [4, 5]\}\} \]

Given a set of time intervals \( I = \{(s_1, t_1), (s_2, t_2), ..., (s_n, t_n)\} \) defined over \( T \), let

\[ s_{\text{min}} = \min_{1 \leq i \leq n} s_i, \ s_{\text{max}} = \max_{1 \leq i \leq n} s_i, \ t_{\text{min}} = \min_{1 \leq i \leq n} t_i, \ \text{and} \ t_{\text{max}} = \max_{1 \leq i \leq n} t_i. \]
We define two values, \( \text{intersect} \) and \( \text{range} \), as follows:

\[
\text{intersect}(I) = \begin{cases} 
[s_{\text{max}}, t_{\text{min}}] & \text{if } s_{\text{max}} \leq t_{\text{min}}, \\
\text{null} & \text{if } s_{\text{max}} > t_{\text{min}}
\end{cases}
\]

\[
\text{range}(I) = \begin{cases} 
[s_{\text{min}}, t_{\text{max}}] & \text{if } s_{\text{min}} \leq t_{\text{max}}, \\
\text{null} & \text{if } s_{\text{min}} > t_{\text{max}}
\end{cases}
\]

Conceptually, \( \text{intersect}(I) \) is the largest time interval that intersects each interval in \( I \), and \( \text{range}(I) \) is the smallest interval that contains each interval in \( I \).

We will now extend these definitions from a set of intervals to a set of sets of intervals (e.g., what is returned from \( \text{int\_expansion} \)). Given a set of sets of time intervals \( I_S = \{I_1, I_2, ..., I_n\} \), we define \( \text{intersect}_S \) and \( \text{range}_S \) as follows:

\[
\text{intersect}_S(I_S) = \{i \mid \exists I \in I_S \text{ such that } i = \text{intersect}(I)\}
\]

\[
\text{range}_S(I_S) = \{i \mid \exists I \in I_S \text{ such that } i = \text{range}(I)\}
\]

The first temporal operator we define, \( \text{temporal\_extent} \), is intended to compute and return the derived time intervals for the results of a graph pattern query. This operator can return one of two time intervals: (1) the \( \text{intersect} \) interval that represents the time interval during which all statements in the graph pattern instance are valid and (2) the \( \text{range} \) interval that represents the time interval during which any statement in the graph pattern instance is valid. As an example consider the query “find all pairs of soldiers who were members of the 101st Airborne Division at the same time and return the times of the joint membership” (Example 4).

\( \text{temporal\_extent}(GP, IT)_{Gt} \rightarrow \{((\mu, i))\} \)

Given:
5.3. TEMPORAL OPERATORS

a temporal RDF Graph $G_t$, a graph pattern $GP$,
an interval type $IT \in \{\text{intersect}, \text{range}\}$

Find:

$$\{(\mu, i) \mid \mu \in [[GP]]_{\text{TRIPLES}(G_t)} \text{ and } i \in \text{intersect}_S/\text{range}_S(\text{int}\_\text{expansion}(\mu(GP)))\}$$

Example 4.

$$\text{ANS} \leftarrow \text{temporal}\_\text{extent} \left( \text{‘(?x assigned\_to \langle 101st Armored Division\rangle)\’, ‘intersect’}_G \right)$$

The remaining temporal operators examine temporal relationships. To specify conditions on these relationships, we define a temporal formula which is constructed from qualitative and metric temporal functions. For a given temporal RDF graph $G_t$ over time domain $T$, let $I$ denote the set of all time intervals over $T$. A qualitative temporal function is a Boolean function $qtf : I \times I \to \mathbb{B}$. Any of the thirteen interval relations identified by Allen (1983) can be used in qualitative temporal functions in our formalization. We define a qualitative temporal expression, $qte$, as follows, where $i_1, i_2 \in I \cup V_T$.

$$\langle qte \rangle := qtf(i_1, i_2)$$

A metric temporal function is a function $mtf : I \times I \to \mathbb{Z}$. We use one metric temporal function $\text{elapsed\_time} : I \times I \to \mathbb{Z}$, which is defined for two disjoint time intervals as the duration of time between the end of the earliest interval and the start of the latest interval. The function returns zero if the intervals are not disjoint.

Let $V_T$ be a set of variables disjoint from $V_N$, $RT$ and $V_S$. We define a metric temporal expression, $mte$, as follows, where $i_1, i_2 \in I \cup V_T$ and $z \in \mathbb{Z}$.

$$\langle mte \rangle := \langle mtf(i_1, i_2) \rangle \langle \text{comp} \rangle z$$

$$\langle \text{comp} \rangle := < | > | \leq | \geq | =$$
A *temporal formula* $tf$ evaluates to a Boolean value for a given graph and is constructed from qualitative temporal functions and metric temporal expressions. It takes the following form.

$$\langle tf \rangle ::= \langle mte \rangle \mid \langle qte \rangle \mid \langle tf \rangle \text{ AND } \langle tf \rangle \mid \langle tf \rangle \text{ OR } \langle tf \rangle \mid \text{ NOT } \langle tf \rangle$$

The temporal formulas used in our formalization are expressions containing exactly one free variable $#t$ or exactly two free variables $#t_1$ and $#t_2$ and are denoted as $tf(#t)$ and $tf(#t_1,#t_2)$.

The first relationship-based temporal operator, *temporal_restrict*, is concerned with the temporal properties of a single entity. This operator inquires about the properties of an entity at a given time. For example, one may ask “which members of the 3rd Armored Division participated in battles during September 1944?” (Example 5). The basic idea behind this operator is that we specify a graph pattern query and then restrict the set of results based on the temporal extents of the graph pattern instances.

$$\text{temporal}\_\text{restrict}(GP, IT, tf(#t))_{G_t} \rightarrow \{(\mu, i)\}$$

**Given:**
- a temporal RDF Graph $G_t$, a graph pattern $GP$,
- an interval type $IT \in \{\text{intersect}, \text{range}\}$,
- a temporal formula $tf$ defined over $I$ and a variable $#t$

**Find:**
- $\{ (\mu, i) \mid \mu \in [[GP]]_{\text{TRIPLES}(G_t)} \text{ and }$
- $i \in \text{intersect}_S/\text{range}_S(\text{int}\_\text{expansion}(\mu(GP))) \text{ and }$
- $tf$ evaluates to true for $#t = i$\}$

**Example 5.**

$$\text{ANS} \leftarrow \text{temporal}\_\text{restrict}(\forall \text{ (?x assigned to } \langle 3rd \text{ Armored Division} \rangle))$$

$$(\langle 3rd \text{ Armored Division} \rangle \text{ participates in } ?y), \text{ ‘intersect’,}$$

$${\text{during} (#t, [09:01:1944, 09:31:1944]) = true})_{G_t}$$
The final temporal operator, $\text{temporal}_\text{eval}$, allows for querying temporal relationships between entities. This operator can be thought of as a temporal join between graph pattern instances. This operator is designed for a query such as “which speeches by President Roosevelt were given within 1 day of a major battle?” (Example 6).

$$\text{temporal}_\text{eval}(GP_1, IT_1, GP_2, IT_2, tf(#t_1,#t_2))_{G_t} \rightarrow \{(\mu_1, i_1, \mu_2, i_2)\}$$

**Given:**

- a temporal RDF Graph $G_t$, a graph pattern $GP_1$,
- an interval type $IT_1 \in \{\text{intersect, range}\}$, a graph pattern $GP_2$, an interval type $IT_2 \in \{\text{intersect, range}\}$, a temporal formula $tf$ defined over $I$ and variables $#t_1, #t_2$

**Find:**

$$\{(\mu_1, i_1, \mu_2, i_2) \mid \mu_1 \in [[GP_1]]_{\text{TRIPLES}(G_t)} \text{ and } i_1 \in \text{intersect}_{S}/\text{range}_{S}(\text{int}\_\text{expansion}(\mu_1(GP_1))) \text{ and } \mu_2 \in [[GP_2]]_{\text{TRIPLES}(G_t)} \text{ and } i_2 \in \text{intersect}_{S}/\text{range}_{S}(\text{int}\_\text{expansion}(\mu_2(GP_2))) \text{ and } tf \text{ evaluates to true for } #t_1 = i_1 \text{ and } #t_2 = i_2\}$$

**Example 6.**

$$\text{ANS} \leftarrow \text{temporal}_\text{eval}(\langle \langle \text{President Roosevelt} \rangle \text{ gives } ?x \rangle, \langle \text{intersect}, \langle \text{intersect}, \text{temporal}_\text{distance}(#t_1, #t_2) \leq 1 \text{ day} \rangle \rangle)_{G_t}$$

### 5.4 Computational Complexity

The computational complexity of our operators is dominated by the complexity of the thematic component (i.e. the evaluation of the graph pattern query). The evaluation of a graph pattern $p$ over an RDF graph $G$ is equivalent to the subgraph isomorphism problem where the task is to determine if $p$ is isomorphic to a subgraph of $G$. The subgraph isomorphism problem is known to be NP-complete (Garey and Johnson, 1979).
Gutierrez et al. (2004) also discuss the simpler problem of testing the emptiness of a query answer set for a graph pattern query over an RDF graph in two forms, where \( q(G) \) denotes the set of mappings returned for a query \( q \) over an RDF graph \( G \):

1. Query complexity version: For a fixed graph \( G \), given a query \( q \), is \( q(G) \) nonempty?

2. Data complexity version: For a fixed query \( q \), given a graph \( G \), is \( q(G) \) nonempty?

The authors showed that this problem is NP-complete for the query complexity version and polynomial for the data complexity version. In addition, Gutierrez et al. (2005, 2007) showed that the asymptotic complexity of this problem is the same for both temporal and nontemporal RDF graphs.
Implementation Framework

This chapter describes the implementation of our spatial and temporal RDF query operators using Oracle’s extensibility framework (Oracle, 2005a). The implementation builds on Oracle's existing support for RDF storage and inferencing and support for spatial object types and indexes. The existing support for these features is the main reason we chose Oracle database for our implementation. We create SQL table functions for each of the previously discussed query operators. Additional structures are created to allow for spatial and temporal indexing of the RDF data for efficient execution of the table functions.

Our implementation uses procedural and declarative SQL and the built-in index structures of the DBMS. We do not depend on any lower-level interfaces of the DBMS, and no modifications to the database kernel are required. Our implementation could therefore be extended to another DBMS and is not restricted to Oracle. We will first give definitions of the table functions that correspond to the query operators defined in the previous chapter. This is followed by a discussion of our storage and indexing scheme and finally our query processing strategies.

This work appears in Perry et al. (2007) and Perry and Sheth (2008).
6.1 Table Functions

We define four table functions: two spatial and two temporal. The following descriptions use the term *spatial geometry* to refer to an *SDO_GEOMETRY* object that would be stored in Oracle Spatial. We can think of a spatial geometry as the implementation of the class Spatial Region.

The *spatial_extent* table function implements the *spatial_extent* query operator described previously, and optional parameters are used to give the filtering functionality of the *spatial_restrict* operator. The signature for the table function is shown below:

```sql
spatial_extent (graphPattern VARCHAR,
    spatialVar VARCHAR, ontology RDFModels,
    <geom SDO_GEOMETRY>, <spatialRelation VARCHAR>)
returns AnyDataSet;
```

The *graphPattern* and *spatialVar* parameters represent the spatial context for the query, and *ontology* determines the temporal RDF graph to search against. This function returns a table with rows containing one column for each distinct variable in the graph pattern and one column for the spatial geometry. Each row contains the URI bound to each variable and the spatial geometry corresponding to the Spatial Region bound to *spatialVar*. Two optional parameters, a spatial geometry and a spatial relationship, can be used to filter the graph pattern instances. In this case, the table would only contain those graph pattern instances whose associated spatial geometries satisfy the specified spatial relation with the input spatial geometry. Our implementation currently supports the following spatial relationships: *disjoint*, *touch*, *overlap boundary intersect*, *overlap boundary disjoint*, *equal*, *contains*, *covers*, *inside*, *covered by*, *anyinteract* and *within distance*.

Example 7 shows a SQL query using the *spatial_extent* function that selects all soldiers who were on the crew of a vehicle used in a military event that occurred within 45 miles of
Example 7.

SELECT x
FROM TABLE (spatial_extent(
   '(?x <on_crew_of> ?y) (?y <used_in> ?z)
   (?z <occurred_at> ?l)', 'l',
   SDO_RDF_Models('military'), SDO_GEOMETRY(2001, 8265,
   SDO_POINT_TYPE(-71.796531, 44.304772, NULL), NULL, NULL),
   'GEO_DISTANCE(distance=45 unit=mile)'));

The `spatial_eval` table function implements the `spatial_eval` query operator defined previously. The signature for this table function is shown below:

```
spatial_eval (graphPattern VARCHAR,
   spatialVar VARCHAR, graphPattern2 VARCHAR,
   spatialVar2 VARCHAR, spatialRelation VARCHAR,
   ontology RDFModels)
return AnyDataSet;
```

`graphPattern` and `spatialVar` specify the first spatial context, and `graphPattern2` and `spatialVar2` specify the second spatial context. `spatialRelation` identifies the spatial relation for joining the two graph pattern instances. This function returns a table containing a column for each variable in `graphPattern` and `graphPattern2` and a column for each associated spatial geometry ($s_1$ and $s_2$). For each row in the resulting table, $s_1$ `spatialRelation` $s_2$ evaluates to `true`.

Example 8 shows a SQL query using the `spatial_eval` function that selects those platoons that train within 30 miles of `Platoon_12996`.

Example 8.

SELECT b
The **temporal_extent** table function implements both the **temporal_extent** and **temporal_restrict** operators discussed previously. Optional parameters are used to perform filtering based on temporal properties. The signature for the table function is shown below.

```
temporal_extent (graphPattern VARCHAR, intervalType VARCHAR, ontology RDFModels, <start DATE>, <end DATE>, <temporalRel VARCHAR>)
return AnyDataSet;
```

This function takes three parameters as input, specifically a graph pattern, a String value specifying the interval type (**INTERSECT** or **RANGE**), and a parameter specifying the temporal RDF graph to search against. The table returned contains a column for each variable in the graph pattern and two **DATE** columns that specify the start and end of the time interval computed for the graph pattern instance. Three optional parameters, two **DATE** values to identify the boundaries of a time interval and a temporal relationship, can be used to filter the found graph pattern instances. In this case, assuming the **DATE** columns in the returned table are named **stDate** and **endDate**, each row in the result satisfies the condition `[stDate, endDate] temporalRel [start, end]`. Our implementation currently supports seven temporal relationships: **before**, **after**, **during**, **overlap**, **during_inv**, **overlap_inv** and **anyinteract**.

Example 9 shows a SQL query using the **temporal_extent** function that selects all soldiers on the crew of a military vehicle and their corresponding platoons during the time interval [10:04:1942, 09:21:1944].
Example 9.

SELECT x, a
FROM TABLE (temporal_extent('(?x <on_crew_of> ?y)
  (?y <used_in> ?z) (?x <assigned_to> ?a)', 'INTERSECT',
  SDO_RDF_Models('military'),
  to_date('1942-10-04', 'yyyy-mm-dd'),
  to_date('1944-09-21', 'yyyy-mm-dd'),
  'DURING')));

The temporal_eval table function implements the temporal_eval operator described previously. It has the following signature:

temporal_eval (graphPattern VARCHAR,
  intervalType VARCHAR, graphPattern2 VARCHAR,
  intervalType2 VARCHAR, temporalRel VARCHAR,
  ontology RDFModels)
return AnyDataSet;

graphPattern and intervalType specify the left hand side of the join operation, while graphPattern2 and intervalType2 specify the right hand side. temporalRel identifies the join condition. This function returns a table containing a column for each variable in graphPattern and graphPattern2 and four DATE columns \([start_1, end_1, start_2, end_2]\) to indicate the derived time interval for each found graph pattern instance. For each row in the resulting table, \([start_1, end_1] \text{ temporalRel } [start_2, end_2]\) evaluates to true.

Example 10 shows a SQL query using the temporal_eval function that selects all pairs of soldiers \((s_1 \text{ and } s_2)\) such that \(s_1\) was leader of a platoon in Division.2186 and \(s_2\) was leader of a platoon in Division.2191 at overlapping times.

Example 10.

SELECT s1, s2
Multiple functions can be used in a single SQL query. This allows us to join the tables that result from a function execution and thus provides a mechanism for spatio-temporal-thematic queries. Example 11 shows a spatio-temporal-thematic query that selects all soldiers who were on the crew of a vehicle that was used in a military event that occurred within an input bounding box and also returns the times at which this particular spatial relationship holds.

**Example 11.** SELECT s.x, t.start_date, t.end_date

FROM

TABLE (spatial_extent(
  '(?x <on_crew_of> ?y) (?y <used_in> ?z)
  (?z <occurred_at> ?l)', 'l',
  SDO_RDF_Models('military'),
  SDO_GEOMETRY(2003, 8265,
  NULL, SDO_ELEM_INFO_ARRAY(1, 1003, 3),
  SDO_ORDINATE_ARRAY(-81.970263, 41.061209,
  -80.518693, 41.964041)),
  'GEO_RELATE(mask=inside)') s,
 TABLE (temporal_extent(
  \[52\]
6.2 Storage and Indexing Scheme

This section presents our storage and indexing scheme for spatial and temporal RDF data. We will first give an overview of existing Oracle capabilities for storing spatial geometries and RDF data and then present our spatial and temporal indexing schemes.

6.2.1 Existing Oracle Technologies

Oracle’s Semantic Data Store (Oracle, 2005b) provides the capabilities to store, inference over, and query semantic data, which can be plain RDF descriptions and RDFS-based ontologies. To store RDF data, users create a model (ontology) to hold RDF triples. The triples are stored after normalization in two tables: an RDFValues table that stores RDF terms and a numeric id and an RDFTriples table that stores the ids of the subject, predicate and object of each statement. Note that this follows the schema-oblivious storage scheme discussed in Section 3.2.1. Users can optionally derive a set of inferred triples based on user-defined rules and/or RDFS semantics. These triples are materialized by creating a rules index and stored in a separate InferredTriples table. For example, using RDFS semantics, all triples generated according to the RDFS entailment rules described in Section 2.2.1 would be inserted into the InferredTriples table. These storage structures are illustrated in Figure 6.1. A SQL table function is provided that allows issuing graph pattern
Figure 6.1: Storage structures for RDF data. Existing tables of Oracle Semantic Data Store are shown on the right, and our additional tables for efficiently searching spatial and temporal data are shown on the left.

queries against both asserted and inferred RDF statements.

Oracle Spatial (Oracle, 2005c) provides facilities to store, query and index spatial geometries. It supports the object-relational model for representing spatial geometries. A native spatial data type, SDO_GEOMETRY, is defined for storing vector data. Database tables can contain one or more SDO_GEOMETRY columns. Oracle Spatial supports spatial indexing on SDO_GEOMETRY columns, and provides a variety of procedures, functions and operators for performing spatial analysis operations.

6.2.2 Indexing Approach

In order to ensure efficient execution of graph pattern queries involving spatial and temporal predicates, we must provide a means to index portions of the RDF graph based on spatial and temporal values. Basically, this is done by building a table mapping Spatial Region instance URIs to their SDO_GEOMETRY representation and by building a modified RDFTriples table that also stores the temporal intervals associated with a triple. In order to build these indexes, users first load the set of asserted RDF statements into Oracle Semantic Data Store and build an RDFS rules index. After this step, users can run our indexing procedures to build spatial and temporal indexes for the RDF data.
6.2. STORAGE AND INDEXING SCHEME

6.2.2.1 Spatial Indexing Scheme

We provide the procedure `build_geo_index()` to construct a spatial index for a given ontology. This procedure first creates the table `SpatialData (value_id NUMBER, shape SDO_GEOMETRY)` for storing spatial geometries corresponding to instances of the class Spatial Region in the ontology. `value_id` is the id given to the URI of the Spatial Region instance in Oracle’s RDFValues table, and `shape` stores the `SDO_GEOMETRY` representation of the Spatial Region instance (see Figure 6.1). This table is filled by querying the ontology for each Spatial Region instance, iterating through the results and creating and inserting `SDO_GEOMETRY` objects into the spatial indexing table. Finally, to enable efficient searching with spatial predicates on this table, a spatial index (R-Tree) is created on the `shape` column.

6.2.2.2 Temporal Indexing Scheme

Our temporal indexing scheme is a bit more complicated, as it must account for temporal labels on statements inferred through RDFS semantics. However, we only need to handle a subset of the RDFS inferencing rules. Only a subset is required because we are not interested in handling temporal evolution of the ontology schema. What we need to handle are temporal properties of instance data. Specifically, we need to account for temporal labels of inferred `rdf:type` statements and statements resulting from `rdfs:subPropertyOf`. `rdf:type` statements result from the following rules (refer back to Section 2.2.1 for the complete set of RDFS inferencing rules):

- **rdfs2**: \((x, p, y) \land (p, rdfs:domain, a) \Rightarrow (x, rdf:type, a)\)
- **rdfs3**: \((x, p, y) \land (p, rdfs:range, b) \Rightarrow (y, rdf:type, b)\)
- **rdfs9**: \((x, rdf:type, y) \land (y, rdfs:subClassOf, z) \Rightarrow (x, rdf:type, z)\)

We infer instance statements from `rdfs:subPropertyOf` using the following rule:
6.2. STORAGE AND INDEXING SCHEME

In each case, if we assume that schema level statements in the ontology are eternally true, the temporal label of an inferred instance statement \( s \) is the union of the time intervals of all statements that can be used to infer \( s \).

This temporal inferencing serves an important purpose in our scheme. Consider the example of a *Battle* event \( b_1 \) that three platoons \( (p_1, p_2, p_3) \) participate in at different times:

\[
(p_1, \text{participates in}, b_1) : [1, 3] \\
(p_2, \text{participates in}, b_1) : [2, 5] \\
(p_3, \text{participates in}, b_1) : [1, 4]
\]

Using rule \( \text{rdfs3} \) for generating \( \text{rdf:type} \) statements, we infer:

\[
(b_1, \text{rdf:type}, \text{Battle}) : [1, 5]
\]

In this case, \([1, 5]\) is the interval union and represents the overall duration or lifetime of \( b_1 \). Note that we are using relationships between entities and an event to automatically infer the overall duration of the event.

We provide the procedure \( \text{build_temporal_index} \) \( (\text{ontology}, \text{rules_index_name}, \text{min_start_time}, \text{max_end_time}) \) to construct a temporal index for a given ontology and rules index. The \( \text{ontology} \) parameter identifies the temporal RDF graph stored in Oracle; \( \text{rules_index_name} \) identifies the RDFS rules index associated with the ontology; \( \text{min_start_time} \) and \( \text{max_end_time} \) specify the earliest date and the latest date in the associated time domain. The purpose of these boundary parameters is to act as the start time and end time of statements that are eternally valid. All schema-level statements in the ontology are considered eternally valid. All asserted instance level statements with missing or incomplete temporal properties are also considered eternally valid. The \( \text{build_temporal_index} \) procedure executes in three phases.

The first phase creates the temporary table \( \text{asserted_temporal_triples} \) \( (\text{subj_id NUM-} \)
6.2. STORAGE AND INDEXING SCHEME

The ontology is then queried to retrieve all temporal reifications. The subject, property, and object ids of each temporally reified statement and the start time and end time are inserted into this temporary table. Next, those statements with incomplete or missing temporal reifications are added to the *asserted_temporal_triples* table using *min_start_time* and *max_end_time* as a substitution for any missing temporal values. The final step of this phase scans the *asserted_temporal_triples* table and ensures that all asserted schema-level statements have \([min\_start\_time, max\_end\_time]\) as their valid time.

At this point, we have recorded the temporal values for each asserted statement, and the second and third phases perform the temporal inferencing process and create the final *TemporalTriples* table (see Figure 6.1). Algorithm 1 shows the temporal inferencing procedure. We first create a second temporary table *redundant_triples* (*subj_id NUMBER, prop_id NUMBER, obj_id NUMBER, start DATE, end DATE*). Then, we iterate through the *asserted_temporal_triples* table and add any inferred statements to the *redundant_triples* table. In this step, the temporal label of the asserted statement is directly assigned to the corresponding inferred statements. This procedure results in possibly redundant and overlapping intervals for each statement, so a third phase, shown in Algorithm 2, iterates through this table and cleans up the time intervals for each statement. The cleanup phase first sorts *redundant_triples* by (*subj_id, prop_id, obj_id, start_date*) and then makes a single pass over the sorted set to merge overlapping intervals having the same (*subj_id, prop_id, obj_id*) values. The final result of this process is a table *TemporalTriples* (*subj_id NUMBER, prop_id NUMBER, obj_id NUMBER, start DATE, end DATE*) that contains the complete set of asserted and inferred temporal triples.

The complexity of the temporal inferencing procedure is as follows. Assume we have \(n\) asserted triples in the dataset and \(c\) classes and \(p\) property types in the ontology schema. In the worst case, every property would be a subclass of every other property; every class would be a subclass of every class, and each property would have every class in its domain.
Algorithm 1 TemporalInference

1: create temporary table redundant_triples (subj_id, prop_id, obj_id, start, end)
2: for each row \( r \in \text{asserted}_{\text{temporal}}_{\text{triples}} \) do
3: \hspace{1em} if \( (r.prop = \text{rdf:}\text{type}) \) then
4: \hspace{2em} for each Class \( C \in \text{SuperClasses}(r.obj) \) do
5: \hspace{3em} insert row \( (r.subj, \text{rdf:}\text{type}, C, r.start\_date, r.end\_date) \)
6: \hspace{3em} into redundant_triples
7: \hspace{2em} end for
8: \hspace{1em} else
9: \hspace{2em} for each property \( P \in \text{SuperProperties}(r.prop) \) do
10: \hspace{3em} insert row \( (r.subj, P, r.obj, r.start\_date, r.end\_date) \)
11: \hspace{3em} into redundant_triples
12: \hspace{2em} end for
13: \hspace{1em} end if
14: \hspace{1em} end for
15: \hspace{1em} x \leftarrow \text{domain}(r.prop)
16: \hspace{1em} for each Class \( C \in \text{SuperClasses}(x) \cup \{x\} \) do
17: \hspace{2em} insert row \( (r.subj, \text{rdf:}\text{type}, C, r.start\_date, r.end\_date) \)
18: \hspace{2em} into redundant_triples
19: \hspace{1em} end for
20: \hspace{1em} y \leftarrow \text{range}(r.prop)
21: \hspace{1em} for each Class \( C \in \text{SuperClasses}(y) \cup \{y\} \) do
22: \hspace{2em} insert row \( (r.obj, \text{rdf:}\text{type}, C, r.start\_date, r.end\_date) \)
23: \hspace{2em} into redundant_triples
24: \hspace{1em} end for
25: \hspace{1em} end if
26: \hspace{1em} end for
Algorithm 2 MergeTemporalIntervals

1: create table
   TemporalTriples (subj_id, prop_id, obj_id, start, end)
2: sort redundant_triples by subj_id, prop_id, obj_id, start
3: r ← first row of redundant_triples
4: curr_row ← r
5: for each row r remaining in redundant_triples do
6:   if (r.subj_id = curr_row.subj_id and r.prop_id = curr_row.prop_id
    and r.obj_id = curr_row.obj_id) then
7:     if (r.start ≤ curr_row.end and r.end > curr_row.end) then
8:       curr_row.end ← r.end
9:     end if
10:    if (r.start > curr_row.end) then
11:      insert row (curr_row.subj_id, curr_row.prop_id,
12:                  curr_row.obj_id, curr_row.start, curr_row.end)
13:      into TemporalTriples
14:     end if
15:   else
16:     insert row (curr_row.subj_id, curr_row.prop_id,
17:                  curr_row.obj_id, curr_row.start, curr_row.end)
18:      into TemporalTriples
19:   end if
20: end for
21: insert into TemporalTriples
   SELECT (subj_id, prop_id, obj_id, min_start, max_end)
   FROM InferredTriples
   WHERE (subj_id, prop_id, obj_id)
   NOT IN TemporalTriples

59
and range. In this case, we would add $2c + p$ triples for every asserted triple, yielding $O(n(c + p))$ for Algorithm 1. In Algorithm 2, we must sort this set of statements and then make a single pass over the sorted set, yielding $O(n(c + p) \log(n(c + p)) + n(c + p))$. This gives an overall complexity of $O(n(c + p) \log(n(c + p)))$ for the temporal inferencing procedure.

### 6.3 Function Implementation

In this section we discuss the implementation of the SQL table functions defined previously. The table functions were implemented using Oracle’s `ODCITable` interface methods. With this scheme, users implement a `start()`, `fetch()` and `close()` method for the table function. In `start()`, the query parameters are parsed; a SQL query is prepared and executed, and a handle to the query is stored in a scan context parameter. The `fetch()` method fetches a subset of rows from the prepared query and returns them. This method is invoked as many times as necessary by the kernel until all result rows are returned. The `close()` method performs cleanup operations after the last `fetch()` call. We also implement an optional `describe()` method, which is used notify the kernel of the structure of the data type to be returned (i.e., columns of the table). This method is necessary because the number of columns in the return type depends on the graph pattern and cannot be determined until query compilation time.

#### 6.3.1 Graph Pattern to SQL Translation

Each of the table functions takes a graph pattern and ontology as input. The conversion of a graph pattern to a SQL query is therefore a central component of each function. The graph pattern is transformed into a self-join query against the `TemporalTriples` table corresponding to the input ontology. The graph pattern translation algorithm is shown in
Algorithm 3. The algorithm first parses the graph pattern and builds a mapping between tokens (i.e., variables and URIs) and a list of their occurrences in the graph pattern. To denote an occurrence, we record the triple pattern number and the position within the triple pattern (i.e. subject, predicate or object). We also build a mapping from URIs to their ids in the RDFValues table. We then use these mappings to build a self-join query over the TemporalTriples table with two sets of conditions in the where clause: (1) restrictions based on the ids of the URIs in the graph pattern and (2) join conditions based on variable correspondences between triple patterns. We must also join with the RDFValues table to resolve the ids of URIs bound to variables to actual URI Strings.

The example below illustrates the transformation process. The resulting SQL query assumes that the ids of on_crew_of and used_in are 1 and 2, respectively.

```sql
(?a <on_crew_of> ?b)(?b <used_in> ?c)
```

```sql
SELECT rv1.uri, rv2.uri, rv3.uri
FROM TemporalTriples tt1, TemporalTriples tt2,
    RDFValues rv1, RDFValues rv2,
    RDFValues rv3
WHERE tt1.prop_id = 1 and tt2.prop_id = 2
    and tt1.obj_id = tt2.subj_id
    and rv1.id = tt1.subj_id
    and rv2.id = tt1.obj_id
    and rv3.id = tt2.obj_id;
```

6.3.2 Spatial Functions

Spatial functions are implemented by augmenting the base graph pattern query discussed in the previous section.
Algorithm 3 Graph Pattern Translation

Input:

\(GP\): graph pattern
\(G_t\): temporal RDF graph

Output:

- \(selectStr\): select portion of SQL query
- \(fromStr\): from portion of SQL query
- \(whereStr\): where portion of SQL query
- \(varMap\): mapping between variables and a list of their occurrences in \(GP\)

1: \(selectStr ← 'SELECT'\)
2: \(fromStr ← 'FROM'\)
3: \(whereStr ← 'WHERE'\)
4: declare \(mapRecord\) as 2-tuple (\(triple\_pattern\_num, pos\))
5: declare Map \(uriMap\) (String, List of \(mapRecord\))
6: declare Map \(varMap\) (String, List of \(mapRecord\))
7: declare Map \(uriIdMap\) (String, Integer)
8: parse \(GP\) and populate \(uriMap, varMap\)
9: for each var \(v \in varMap\) do
   10: \(currList ← varMap(v)\)
   11: add 'tt.<currList(1).triple_pattern_num>.<currList(1).pos> as <v>' to \(selectStr\)
12: end for
13: for \(i = 1\) to \(numTriplePatterns\) do
   14: add 'TemporalTriples tt.<i>' to \(fromStr\)
15: end for
16: for \(i = 1\) to \(numVars\) do
   17: add 'RDFValues rv.<i>' to \(fromStr\)
18: end for
19: populate \(uriIdMap\) from \(RDFValues\)
20: for each URI \(u \in uriMap\) do
   21: \(currList ← uriMap(u)\)
   22: for \(i = 1\) to length(\(currList\)) do
      23: add 'tt.<currList(i).triple_pattern_num>.<currList(i).pos> = <uriIdMap(u)>' to \(whereStr\)
   24: end for
25: end for
26: for each var \(v \in varMap\) do
   27: \(currList ← varMap(v)\)
   28: for \(i = 1\) to length(\(currList\)) - 1 do
      29: add 'tt.<currList(i).triple_pattern_num>.<currList(i).pos> = tt.<currList(i+1).triple_pattern_num>.<currList(i+1).pos>' to \(whereStr\)
   30: end for
31: end for
Algorithm 4 shows the query processing procedure for \textit{spatial\_extent} function. We modify the base query as follows. First we identify the appropriate column (i.e., \textit{subj\_id}, \textit{prop\_id}, or \textit{obj\_id}) in the \textit{RDFTriples} table that corresponds to the position of the \textit{spatial\_variable} parameter. Then we add an additional join matching ids from the \textit{TemporalTriples} table with \textit{value\_ids} in the \textit{SpatialData} table to select the id of the \textit{SDO\_GEOMETRY} object. We must return the id, rather than the \textit{SDO\_GEOMETRY} object, from \textit{SpatialData} because object types cannot be returned from table functions. In the case of optional result filtering, we need to modify the where clause so that we filter the spatial features from \textit{SpatialData} according to the input spatial feature and spatial relation. This is done by adding the appropriate \textit{sdo\_relate} or \textit{sdo\_within\_distance} predicate available in Oracle Spatial. For example, given the query:

\begin{verbatim}
spatial\_extent (...) , sdo\_geometry (...),
  \textquote{\text{\textit{geo\_relate (inside)}}}
\end{verbatim}

we would modify the query as follows:

\begin{verbatim}
WHERE ... AND
  sdo\_relate (geo\_shape, sdo\_geometry (...), \textquote{\text{\textit{mask=inside}}} = \text{\textit{true}}).
\end{verbatim}

Algorithm 5 shows the query processing procedure for the \textit{spatial\_eval} function. We implement what is essentially a nested loop join (NLJ) using the basic \textit{spatial\_extent} and filtered \textit{spatial\_extent} operators. We first construct and execute a basic \textit{spatial\_extent} query in the \textit{start()} routine. Next, in the \textit{fetch()} routine, we consume a row from the \textit{spatial\_extent} query and then construct and execute the appropriate filtered \textit{spatial\_extent} query using the second pair of graph pattern and spatial variable parameters and the spatial relation parameter. This is repeated until all rows in the outer \textit{spatial\_extent} query are consumed.
Algorithm 4 spatial_extent

**Input:**
- GP: graph pattern
- svar: spatial variable identifier
- G_t: temporal RDF graph
- filterParams: optional filtering parameters

**Output:**
- rows: query results

```plaintext
1: GraphPatternTranslation (GP, G_t, selectStr, fromStr, whereStr, varMap)
2: add 'SpatialData.id as geom' to selectStr
3: add 'SpatialData' to fromStr
4: currList ← varMap(svar)
5: add t.t.<currList(1).triple_pattern_num>.<currList(1).pos> = SpatialData.value_id'
    to whereStr
6: if (filterParams are present) then
7:    parse filterParams and add appropriate sdo.relate or sdo.within_distance predicate to whereStr
8: end if
9: sctx ← parse (selectStr + fromStr + whereStr)
10: while sctx.results_remaining() do
11:    rows ← sctx.fetch_rows()
12: return rows
13: end while
```
6.3. FUNCTION IMPLEMENTATION

Algorithm 5 \textit{spatial\_eval}

\textbf{Input:}
\begin{itemize}
  \item $GP_1$: graph pattern
  \item $var_1$: spatial variable identifier
  \item $GP_2$: graph pattern
  \item $var_2$: spatial variable identifier
  \item $spatialRel$: spatial relation
  \item $G_t$: temporal RDF graph
\end{itemize}

\textbf{Output:}
\begin{itemize}
  \item $rows$: query result
\end{itemize}

\begin{enumerate}
  \item $sctx \leftarrow$ parse($\text{spatial\_extent}(GP_1, var_1, G_t)$)
  \item \textbf{while} $sctx$.results\_remaining() \textbf{do}
  \item \hspace{1em} $outer\_rows \leftarrow sctx$.fetch\_rows()
  \item \hspace{1em} \textbf{for} each row $r_1 \in outer\_rows$ \textbf{do}
  \item \hspace{2em} $inner\_rows \leftarrow \text{execute}(\text{spatial\_extent}(GP_2, var_2, G_t, r.geom, \text{inverse of spatialRel}))$
  \item \hspace{1em} \textbf{for} each row $r_2 \in inner\_rows$ \textbf{do}
  \item \hspace{2em} add $r_1.vars, r.geom, r_2.vars, r_2.geom$ to $rows$
  \item \hspace{1em} \textbf{end for}
  \item \hspace{1em} \textbf{end for}
  \item \hspace{1em} \textbf{return} $rows$
  \item \textbf{end while}
\end{enumerate}

6.3.3 Temporal Functions

The implementation of the temporal functions does not translate directly to a SQL query. We must do some extra processing of the base query results in the \textit{fetch()} routine to form a single time interval for each found graph pattern instance.

Algorithm 6 shows the query processing strategy for the \textit{temporal\_extent} function. We first augment the basic graph pattern query in \textit{start()} to also select the start and end values for each temporal triple in the graph pattern instance. In the \textit{fetch()} routine, to compute the final temporal interval for each graph pattern instance, we examine the start and end times for each triple and select the earliest start and latest end (RANGE) or the latest start and earliest end (INTERSECT). In the case of INTERSECT, if the final start value is later than the final end value then the computed interval is not valid and is not included in the final result. When the optional filtering parameters are specified, we must...
perform additional checking of the found graph patterns to ensure they satisfy the filter condition. In addition to these extra computations in `fetch()`, we augment the base query in `start()` with a series of predicates involving the start and end times of each statement in the graph pattern. This is done to filter the results as much as possible in the base query to reduce subsequent overhead in `fetch()`. To illustrate these additional predicates, consider the following `temporal_extent` query and corresponding base query:

```sql
SELECT ...
FROM TABLE(temporal_extent{
  '(?x <on_crew_of> ?y) (?y <used_in> ?z)',
  'range', 1942, 1944, 'during'));
```

```sql
SELECT ...
FROM ..., TemporalTriples t1, TemporalTriples t2
WHERE ... AND t1.start > 1942 AND t2.end < 1944
AND t2.start > 1942 AND t2.end < 1944;
```

Algorithm 7 shows the query processing strategy for `temporal_eval`. The implementation of the `temporal_eval` operator is similar to the implementation of `spatial_eval`. We first build a basic `temporal_extent` query involving the first pair of graph pattern and interval type parameters, which is executed in the `start()` routine. Next, in `fetch()`, we consume a row from the basic `temporal_extent` query and execute an appropriate filtered `temporal_extent` query using the second pair of graph pattern and interval type parameters. This query uses the time interval from the current outer `temporal_extent` result and the inverse of the temporal relation parameter from the original `temporal_eval` query.
Algorithm 6 \textit{temporal\_extent}

\textbf{Input:}
\begin{itemize}
    \item $GP$: graph pattern
    \item $IT$: interval type
    \item $G_t$: temporal RDF graph
    \item $filterParams$: optional filtering parameters
\end{itemize}

\textbf{Output:}
\begin{itemize}
    \item $rows$: query result
\end{itemize}

1: $\text{GraphPatternTranslation}(GP, G_t, \text{selectStr}, \text{fromStr}, \text{whereStr}, \text{varMap})$

2: \textbf{for} each $i$ in 1 to $\text{graphPatternLen}$ \textbf{do}

3: \hspace{1em} add \textquote{tt\_<i>.start as st\_<i>, tt\_<i>.end as ed\_<i>}' to \text{selectStr}

4: \textbf{end for}

5: \textbf{if} ($filterParams$ are present) \textbf{then}

6: \hspace{1em} parse $filterParams$ and add appropriate constraints to $whereStr$

7: \textbf{end if}

8: $sctx \leftarrow \text{parse} (\text{selectStr} + \text{fromStr} + \text{whereStr})$

9: \textbf{while} $sctx.\text{results}\_\text{remaining}()$ \textbf{do}

10: \hspace{1em} $rows \leftarrow sctx.\text{fetch}\_\text{rows}()$

11: \hspace{1em} \textbf{for} each row $r \in rows$ \textbf{do}

12: \hspace{2em} \textbf{if} ($IT = \text{’RANGE’}$) \textbf{then}

13: \hspace{3em} curr\_interval $\leftarrow [\min(r.st), \max(r.ed)]$

14: \hspace{2em} \textbf{end if}

15: \hspace{1em} \textbf{if} ($IT = \text{’INTERSECT’}$) \textbf{then}

16: \hspace{2em} \textbf{if} ($\max(r.st) \leq \min(r.ed)$) \textbf{then}

17: \hspace{3em} curr\_interval $\leftarrow [\max(r.st), \min(r.ed)]$

18: \hspace{2em} \textbf{end if}

19: \hspace{1em} \textbf{end if}

20: \hspace{1em} \textbf{if} (curr\_interval is defined) \textbf{then}

21: \hspace{2em} \textbf{if} ($filterParams$ are present and curr\_interval, t\_interval satisfies filter condition) \textbf{then}

22: \hspace{3em} add $r.vars$, curr\_interval to $rows$

23: \hspace{2em} \textbf{end if}

24: \hspace{2em} \textbf{if} ($filterParams$ are not present) \textbf{then}

25: \hspace{3em} add $r.vars$, curr\_interval to $rows$

26: \hspace{2em} \textbf{end if}

27: \hspace{1em} \textbf{end if}

28: \hspace{1em} \textbf{end for}

29: \hspace{1em} \textbf{return} $rows$

30: \textbf{end while}
Algorithm 7 \textit{temporal\_eval}

\textbf{Input:}
- \(GP_1\): graph pattern
- \(IT_1\): interval type
- \(GP_2\): graph pattern
- \(IT_2\): interval type
- \(temporalRel\): temporal relation
- \(G_t\): temporal RDF graph

\textbf{Output:}
- \(rows\): query results

```
1: \(sctx \leftarrow \text{parse}(\text{temporal\_extent}(GP_1, IT_1, G_t))\)
2: \textbf{while} \(sctx.\text{results}\_\text{remaining}()\) \textbf{do}
3: \(outer\_rows \leftarrow sctx.\text{fetch\_rows}()\)
4: \textbf{for each row} \(r_1 \in outer\_rows\) \textbf{do}
5: \(inner\_rows \leftarrow \text{execute}(\text{temporal\_extent}(GP_2, IT_2, G_t, r_1.\text{interval}, \text{inverse of temporalRel}))\)
6: \textbf{for each row} \(r_2 \in inner\_rows\) \textbf{do}
7: \(\text{add} r_1.\text{vars}, r.\text{interval}, r_2.\text{vars}, r_2.\text{interval} \text{ to rows}\)
8: \textbf{end for}
9: \textbf{end for}
10: \textbf{return} \(rows\)
11: \textbf{end while}
```
Experimental Evaluation

The experimental evaluation of our implementation is described in this chapter. All code was written in PL/SQL, and all experiments were conducted using Oracle 10g Release 2 running on a Sun Fire V490 server with four 1.8 GHz Ultra Sparc IV processors and 8GB of main memory. The operating system used was 64-bit Solaris 9. The database used an 8 KB block size and was configured with a 512 MB buffer cache and a `pga_aggregate_target` size of 512 MB. The times reported for each query were obtained as follows. The query was run once initially to warm up the database buffers and then timed for 10 consecutive executions. We report the mean execution time over these 10 consecutive executions. Times were obtained by querying for `systimestamp` before and after query execution and computing the difference.

This work appears in Perry and Sheth (2008).

7.1 Repeatability of Experiments

It is important that the experimental results presented in this dissertation are repeatable. We have prepared an accompanying website to facilitate this repeatability: http://knoesis.wright.edu/students/mperry/dissertation/Test-Details.html. All queries and datasets used in our experiments can be downloaded from this website. In addition, the configuration of the database server used for testing is described above, and the queries used in our experiments
also appear in Appendix A.1.

7.2 Datasets

We conducted experiments using two RDF datasets. One consisted of synthetically generated RDF data corresponding to historical analysis of WWII (SynHist), and the other (GovTrack) consisted of real-world RDF data from the political domain that we obtained from http://www.govtrack.us/data/rdf/. Table 7.1 shows the characteristics of these datasets.

7.2.1 SynHist Dataset

Five synthetically generated datasets (SH1 - SH5) were used in our experiments. The datasets correspond to a historical battlefield analysis ontology schema that we created. The ontology schema defined 15 class types and 9 property types. Each dataset was created in three phases. First we populated the thematic portion of the ontology. Second we added spatial information, and in the final step we generated temporal labels for the statements in the populated ontology.
7.2. DATASETS

To populate the thematic portion of the battlefield analysis ontology, we used the ontology population tool described in Perry (2005). This tool inputs an ontology schema and relative probabilities for generating instances of each class and property type. Based on these probabilities, it generates instance data, which, in effect, simulates the population of the ontology. We integrated these RDF graphs with the upper-level ontology described in Section 4.2 by adding a handful of rdfs:subClassOf statements to each RDF dataset.

To add spatial aspects to this dataset, we randomly assigned a spatial geometry to each instance of Spatial Region in the ontology. We used year 2000 census block group boundary polygons from the United States Census Bureau (2001) for the spatial geometries. Differently-sized sets of contiguous US States were chosen in proportion with the ontology size.

The final phase of dataset generation assigned temporal labels to statements in the ontology. Temporal intervals were randomly assigned to each asserted instance statement. Start times and end times for each interval were randomly selected with uniform probability from two overlapping date ranges. We ensured that each interval was valid (i.e., start time earlier than end time) before adding it to the dataset.

### 7.2.2 GovTrack Dataset

The GovTrack RDF dataset contains data about activities of the US Congress. More specifically, it contains data describing politicians, bills, voting records, political organizations, political offices, and terms held by politicians. The ontologies used for this dataset contained 74 classes and 139 properties. 22 classes and 47 properties were actually used in the instance data.

Some transformations and enhancements of the dataset were needed to make it appropriate for experimentation. We integrated the ontologies used with the upper-level ontology described in Section 4.2 using rdfs:subClassOf statements. The GovTrack data contained
a significant amount of temporal information. However, this information was encoded using separate properties rather than as temporal RDF. For example, an instance of the class Term would have a start_date property and an end_date property. A preprocessing step was therefore needed to transform the dataset into a temporal RDF graph. This step would, for example, remove the existing start_date and end_date statements for a Term and then add the temporal label [start_date, end_date] to all statements involving the Term. To enhance the dataset with spatial data, we linked Congressional_District instances with their corresponding boundary polygons available from the United States Census Bureau (2001). We used boundary files for the 106th - 110th Congress.

We created three differently-sized subsets of the GovTrack data (GT1 - GT3). GT1 contained information on bills and voting from the 106th Congress. GT2 used the 106th and 107th Congress, and GT3 used the 106th - 110th Congress.

7.3 Experiments

Our experiments were designed to characterize the overall scalability of our approach with respect to (1) dataset size and (2) graph pattern complexity.

For testing, $B^+$-Tree indexes were created on each column of the TemporalTriples table and on the value_id column of the SpatialData table, and an R-Tree index was created on the shape column of SpatialData. We also created four composite $B^+$-Tree indexes on the TemporalTriples table to allow for efficient index-based joins: (prop_id, subj_id, obj_id) and (prop_id, obj_id, subj_id) for spatial operators and (prop_id, subj_id, obj_id, start, end) and (prop_id, obj_id, subj_id, start, end) for temporal operators.

Table 7.2 shows the execution time for creating RDFS rules indexes using Oracle Semantic Data Store and for executing our temporal inferencing procedure. Times were obtained using the timing option of SQLPlus. The results show that the time required for
temporal inferencing is comparable to the time required for RDFS rules index creation. In addition, the procedures take longer on the GovTrack dataset due to its larger ontology schema. The larger schema is also responsible for the greater number of inferred statements relative to the number of asserted statements.

In the following, we refer to two different graph pattern types: unselective and selective. An unselective graph pattern contains constant URIs in the predicate position in each triple pattern and variables in each subject and object position, for example:

```
(?x <usgov:cosponsor> ?y) (?x <usgov:sponsor> ?z)
(?x <usgov:inCommittee> ?c)
```

A selective graph pattern has constant URIs in each predicate position and additionally contains a constant URI in the subject and/or object position in at least one triple pattern, for example:

```
(?p <usgov:hasRole> ?y)
(?y <usgov:forOffice> <usgov:congress/senate/va>)
```
7.3.1 **Scalability with respect to Dataset Size**

Tables 7.3 and 7.4 summarize the results of our experimentation with respect to dataset size. These experiments were designed to test the general scalability of our operators for the GovTrack and SynHist datasets.
Table 7.3: Experimental results for query execution time with respect to ontology size for GovTrack datasets

<table>
<thead>
<tr>
<th>Query</th>
<th>Operator</th>
<th>Relation</th>
<th>Num Triples</th>
<th>Num Vars</th>
<th>Result Size</th>
<th>Execution Time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>GT1</td>
</tr>
<tr>
<td>G1</td>
<td>T-Ext</td>
<td></td>
<td>3</td>
<td>4</td>
<td>1000</td>
<td>386</td>
</tr>
<tr>
<td>G2</td>
<td>T-Ext</td>
<td></td>
<td>5</td>
<td>6</td>
<td>1000</td>
<td>562</td>
</tr>
<tr>
<td>G3</td>
<td>T-Ext</td>
<td></td>
<td>3</td>
<td>3</td>
<td>94</td>
<td>35</td>
</tr>
<tr>
<td>G4</td>
<td>T-Ext</td>
<td></td>
<td>5</td>
<td>5</td>
<td>94</td>
<td>53</td>
</tr>
<tr>
<td>G5</td>
<td>T-Filter</td>
<td>INT/DURING</td>
<td>3</td>
<td>4</td>
<td>451</td>
<td>375</td>
</tr>
<tr>
<td>G6</td>
<td>T-Filter</td>
<td>INT/AFTER</td>
<td>5</td>
<td>6</td>
<td>483</td>
<td>424</td>
</tr>
<tr>
<td>G7</td>
<td>T-Eval</td>
<td>INT/DURING</td>
<td>3 / 3</td>
<td>3 / 3</td>
<td>90</td>
<td>568</td>
</tr>
<tr>
<td>G8</td>
<td>T-Eval</td>
<td>INT/BETWEEN</td>
<td>3 / 2</td>
<td>3 / 2</td>
<td>120</td>
<td>196</td>
</tr>
<tr>
<td>G9</td>
<td>S-Ext</td>
<td></td>
<td>3</td>
<td>4</td>
<td>1000</td>
<td>392</td>
</tr>
<tr>
<td>G10</td>
<td>S-Ext</td>
<td></td>
<td>5</td>
<td>6</td>
<td>1000</td>
<td>540</td>
</tr>
<tr>
<td>G11</td>
<td>S-Ext</td>
<td></td>
<td>3</td>
<td>3</td>
<td>437</td>
<td>155</td>
</tr>
<tr>
<td>G12</td>
<td>S-Ext</td>
<td></td>
<td>5</td>
<td>5</td>
<td>428</td>
<td>230</td>
</tr>
<tr>
<td>G13</td>
<td>S-Filter</td>
<td>INSIDE</td>
<td>3</td>
<td>4</td>
<td>166</td>
<td>721</td>
</tr>
<tr>
<td>G14</td>
<td>S-Filter</td>
<td>ANYINTERACT</td>
<td>5</td>
<td>6</td>
<td>559</td>
<td>1088</td>
</tr>
<tr>
<td>G15</td>
<td>S-Filter</td>
<td>INSIDE</td>
<td>3</td>
<td>4</td>
<td>283</td>
<td>215</td>
</tr>
<tr>
<td>G16</td>
<td>S-Filter</td>
<td>ANYINTERACT</td>
<td>5</td>
<td>6</td>
<td>442</td>
<td>506</td>
</tr>
<tr>
<td>G17</td>
<td>S-Eval</td>
<td>ANYINTERACT</td>
<td>4 / 1</td>
<td>4 / 2</td>
<td>99</td>
<td>7827</td>
</tr>
<tr>
<td>G18</td>
<td>S-Eval</td>
<td>w/in DIST</td>
<td>4 / 2</td>
<td>4 / 2</td>
<td>24</td>
<td>24448</td>
</tr>
<tr>
<td>G19</td>
<td>S-Eval</td>
<td>ANYINTERACT</td>
<td>4 / 1</td>
<td>4 / 2</td>
<td>15</td>
<td>80</td>
</tr>
<tr>
<td>G20</td>
<td>S-Eval</td>
<td>w/in DIST</td>
<td>4 / 2</td>
<td>4 / 2</td>
<td>73</td>
<td>790</td>
</tr>
</tbody>
</table>
Table 7.4: Experimental results for query execution time with respect to ontology size for SynHist datasets

<table>
<thead>
<tr>
<th>Query</th>
<th>Operator</th>
<th>Relation</th>
<th>Num Triples</th>
<th>Num Vars</th>
<th>Result Size</th>
<th>Execution Time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mil1</td>
<td>Mil2</td>
<td>Mil3</td>
<td>Mil4</td>
</tr>
<tr>
<td>H1</td>
<td>T-Ext</td>
<td></td>
<td>3</td>
<td>4</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>H2</td>
<td>T-Ext</td>
<td></td>
<td>5</td>
<td>6</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>H3</td>
<td>T-Ext</td>
<td></td>
<td>3</td>
<td>3</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>H4</td>
<td>T-Ext</td>
<td></td>
<td>5</td>
<td>5</td>
<td>178</td>
<td></td>
</tr>
<tr>
<td>H5</td>
<td>T-Filter</td>
<td>INT/OVERLAP</td>
<td>3</td>
<td>4</td>
<td>251</td>
<td></td>
</tr>
<tr>
<td>H6</td>
<td>T-Filter</td>
<td>INT/OVERLAP</td>
<td>5</td>
<td>6</td>
<td>280</td>
<td></td>
</tr>
<tr>
<td>H7</td>
<td>T-Eval</td>
<td>INT/OVERLAP</td>
<td>3 / 2</td>
<td>3 / 3</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>H8</td>
<td>T-Eval</td>
<td>INT/ANYINTERACT</td>
<td>3 / 3</td>
<td>3 / 3</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>H9</td>
<td>S-Ext</td>
<td></td>
<td>3</td>
<td>4</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>H10</td>
<td>S-Ext</td>
<td></td>
<td>5</td>
<td>6</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>H11</td>
<td>S-Ext</td>
<td></td>
<td>3</td>
<td>3</td>
<td>183</td>
<td></td>
</tr>
<tr>
<td>H12</td>
<td>S-Ext</td>
<td></td>
<td>5</td>
<td>5</td>
<td>224</td>
<td></td>
</tr>
<tr>
<td>H13</td>
<td>S-Filter</td>
<td>OVERLAP</td>
<td>3</td>
<td>4</td>
<td>449</td>
<td></td>
</tr>
<tr>
<td>H14</td>
<td>S-Filter</td>
<td>w/in DIST</td>
<td>5</td>
<td>6</td>
<td>136</td>
<td></td>
</tr>
<tr>
<td>H15</td>
<td>S-Eval</td>
<td>w/in DIST</td>
<td>3 / 1</td>
<td>3 / 2</td>
<td>130</td>
<td></td>
</tr>
<tr>
<td>H16</td>
<td>S-Eval</td>
<td>w/in DIST</td>
<td>2 / 2</td>
<td>2 / 3</td>
<td>57</td>
<td></td>
</tr>
</tbody>
</table>
7.3. EXPERIMENTS

7.3.1.1 Basic temporal_extent

Queries G1 - G4 and H1 - H4 tested the scalability of the temporal_extent operator for the GovTrack and SynHist datasets. Query G1, G2 and H1, H2 measure the response time (i.e. time to return the first 1000 rows) for an unselective graph pattern query, and G3, G4 and H3, H4 tested the execution time for a selective graph pattern query. For both query types and both datasets, query execution time is near constant as the dataset size grows. This is a result of the index-based nested loop join (NLJ) strategy used by the DBMS, which tends to have execution times proportional to the result set size. The 5-triple queries are slower than the 3-triple queries as a result of the additional joins needed to evaluate the query.

7.3.1.2 Filtered temporal_extent

Query G5, G6 and H5, H6 tested the scalability of the temporal_extent operator with filtering. These queries used an unselective graph pattern in combination with very selective temporal conditions. The queries show relatively constant execution time for the GovTrack dataset but show more of a linear growth for the SynHist dataset. In each case, the DBMS uses an index-based NLJ strategy over the composite indexes containing start date and end date information.

These particular queries represent a challenging case for the temporal_extent operator. Because the INTERSECT / RANGE interval derived for a graph pattern instance is constructed dynamically from the temporal labels of each edge in the graph pattern instance, we cannot directly index these derived values. We must instead apply the temporal filtering condition to each graph pattern instance as it is being constructed, which can lead to a very large set of intermediate results that are later discarded. The unnecessary intermediate results are generated because, in many cases, we cannot exclude a graph pattern instance until it is fully constructed and the final derived time interval is known. We try to alleviate this problem by placing limited temporal constraints on each triple pattern in the graph pattern.
These initial constraints can reduce the number of intermediate results generated, but the amount of reduction depends on the specific interval type and temporal relation used. This issue is further explored in Section 7.3.2.2.

The difference in the scalability of the queries over the GovTrack dataset is a result of the characteristics of the time intervals in each dataset. The triples in the SynHist dataset have much longer time intervals with respect to the maximum start and end times of the whole dataset as compared to the GovTrack dataset. As a result, the temporal filtering conditions that can be placed on each triple in the graph pattern are ultimately less selective, leading to larger growth in intermediate results as the dataset size increases.

7.3.1.3 temporal

Queries G7, G8 and H7, H8 tested the scalability of the \textit{temporal\_eval} operator. Selective graph patterns were used for both the left hand side (LHS) and right hand side (RHS) graph pattern in G7, G8 and H7. H8 used a LHS graph pattern and an unselective RHS graph pattern. The results show that execution times for G8 and H7 are relatively constant across each dataset, but queries G7 and H8 show a linear growth in execution time. The growth in execution time for H7 is a result of the larger sets of intermediate results generated by the unselective RHS graph pattern as the dataset size grows. The results for G7 are a result of the DURING temporal relation. This particular relation only allows weak temporal constraints on each triple pattern, leading to a growth in intermediate results. This is explored further in Section 7.3.2.2.

7.3.1.4 Basic spatial\_extent

Queries G9 - G12 and H9 - H12 tested the scalability of the \textit{spatial\_extent} operator. G9, G10 and H9, H10 measured the response time (first 1000 rows) for unselective graph pattern queries, and G11, G12 and H11, H12 measured the execution time of selective graph
7.3. EXPERIMENTS

Pattern queries. For both query types and both datasets, query execution time is near constant as the dataset size grows. This is a result of the index-based NLJ strategy used by the DBMS, which tends to have execution times proportional to the result set size. The 5-triple queries are slower than the 3-triple queries as a result of the additional joins needed to evaluate the query. The query execution times are roughly equivalent to those for basic temporal_extent queries, as the extra join with the SpatialData table needed for the spatial queries is offset by the extra overhead of deriving INTERSECT / RANGE time intervals for the temporal queries.

7.3.1.5 Filtered spatial_extent

Queries G13 - G16 and H13, H14 tested the scalability of the filtering capability of the spatial_extent operator. Each query used an unselective graph pattern in combination with a selective spatial predicate. For each query, execution times are relatively constant across each dataset, which is a result of the index-based NLJ strategy used by the DBMS. The slower times reported in G13 and G14 are a result of the very complex spatial geometries used to represent congressional districts, which increase the time needed to perform the spatial filtering using the R-Tree index. Queries G15 and G16 used the same graph patterns and filtering parameters but were run over a modified dataset substituting random census block group polygons for the congressional district polygons. The execution times are significantly faster using these spatial geometries.

In the SynHist dataset, we see that the spatial filtering queries scale better than temporal filtering queries. Unlike INTERSECT/RANGE intervals, the spatial geometries can be indexed because they are not dynamically created. The spatial filtering queries consequently scale better because we can consistently reduce the search space using the spatial index and do not get as much growth in intermediate results as the dataset size increases.
7.3. **EXPERIMENTS**

7.3.1.6 **spatial_eval**

Queries G17 - G20 and H15, H16 tested the scalability of `spatial_eval`. G17, G19 and H15 used selective LHS graph patterns and unselective RHS graph patterns. G18, G20 and H16 used selective RHS and LHS graph patterns. In each case, execution times are relatively constant across each dataset due to the index-based join strategy and the consistent filtering from the spatial index. The execution times of G17 and G18 are much slower due to the complexity of the congressional district polygons. To evaluate a `spatial_eval` query over the GovTrack dataset, we must compute spatial relations between two complex spatial geometries, which is an expensive operation. We had better performance with filtered `spatial_extent` queries because we were computing spatial relations between a complex spatial geometry in the dataset and a simple spatial geometry specified in the query. G19 and G20 are the same `spatial_eval` queries using census block group polygons, which yield much faster execution times.

7.3.2 **Scalability with respect to Graph Pattern Size**

Our next experiments are designed to test the scalability of various operators with respect to query complexity: that is, the size of the graph pattern used. We have focused on `temporal_extent` and `spatial_extent` operators, as their functionality forms the basis of our implementation.

7.3.2.1 **Filtered temporal_extent**

Experiment GP1 tested the scalability of a filtered `temporal_extent` query as the complexity of the graph pattern used in the query increased. We used unselective graph patterns and very selective temporal predicates in each case. We ran one set of queries over the SH5 dataset and one set of queries over the GT3 dataset.
7.3. EXPERIMENTS

Figure 7.1: Experiment GP1: filtered temporal_extent with respect to graph pattern size for SynHist (SH5) and GovTrack (GT3) datasets.

Figure 7.2: Experiment GP2: filtered spatial_extent with respect to graph pattern size for SynHist (SH5) and GovTrack (GT3) datasets.
Figure 7.3: Experiment GP3: highly selective basic temporal extent with respect to graph pattern size for SynHist (SH5) and GovTrack (GT3) datasets.

Figure 7.4: Experiment GP4: highly selective basic spatial extent with respect to graph pattern size for SynHist (SH5) and GovTrack (GT3) datasets.
The key to the performance of filtered temporal_extent queries is the amount the search space can be reduced by placing partial temporal constraints on each triple pattern in the graph pattern. As we noted earlier, the effectiveness of these partial temporal constraints depends on the particular interval type and temporal relation used in a query.

The objective of this experiment was to characterize the performance of filtered temporal_extent queries in both the worst-case scenario (very limited initial temporal filtering) and the best-case scenario (complete initial temporal filtering). An INTERSECT interval type in combination with a DURING temporal relation represented the worst-case. In this situation, we can only enforce that the valid time interval of each triple does not end before the query interval starts or start after the query interval ends. In contrast, with a RANGE interval type and a DURING temporal relation, we can enforce that each triple starts after the query interval starts and ends before the query interval ends. These conditions completely filter out any unwanted graph pattern instances, and this query represents a best-case. Figure 7.1 shows the execution times for a best-case and worst-case query for unselective graph patterns varying in size from one triple to seven triples. We can see that execution time grows roughly linearly in each case, but performance is significantly worse with the INTERSECT temporal relation. The performance is better for the GovTrack dataset because of the nature of the temporal intervals in each dataset as we discussed in Section 7.3.1.2. The execution time for queries over the SynHist dataset tends to grow more rapidly at first and then taper off as the graph pattern gets more complex. This trend is a result of the selectivity of the graph pattern itself. In this dataset, there are fewer instances of the more complex graph patterns. This slows the growth in intermediate results, so not as much additional temporal filtering is needed in the fetch() method.
7.3. EXPERIMENTS

7.3.2.2 Filtered spatial_extent

Experiment GP2 tested the scalability of filtered spatial_extent queries. The graphs in Figure 7.2 show the execution times for queries involving unselective graph patterns and selective spatial filtering conditions. As the graph pattern size grows, the query execution times show linear scalability on both datasets and are much faster than the worst-case temporal queries. Because the spatial values in our dataset are not dynamically derived, we can effectively index them. The faster execution times result from the more effective spatial indexing. The spatial index is used initially to select the nodes satisfying the spatial filtering condition, which reduces the search space for evaluating the rest of the graph pattern. The queries over the GovTrack dataset have slower execution times because spatial computations are more expensive for the complex spatial geometries in the GovTrack dataset.

7.3.2.3 Basic temporal_extent

Experiment GP3 tested the scalability of basic temporal_extent queries using selective graph patterns. Figure 7.3 shows query execution time for basic temporal_extent queries as graph pattern size ranges from 1 triple to 10 triples. The number of result rows returned from the query is also shown in the graphs. These graphs show that performance is quite good for selective graph pattern queries even as the graph patterns grow relatively large. In each case, the execution times grow roughly linearly as the graph pattern size increases when the effects of the result set size are taken into account. The DBMS starts with the most selective triple pattern and uses an index-based join to construct the rest of the graph pattern instance. The initial selection dramatically cuts down the search space and results in the fast execution times for these queries.
7.3. EXPERIMENTS

7.3.2.4 Basic spatial_extent

Experiment GP4 tested the scalability of basic \textit{spatial_extent} queries using selective graph patterns. Figure 7.4 shows the execution time of basic \textit{spatial_extent} queries as graph pattern size ranges from 2 to 10 triples. The result set size of each query is also shown in the figure. Execution time grows linearly as graph pattern size increases when the result set size is taken into account. Again, the DBMS starts with the most selective triple pattern and grows the graph pattern instance from there using an index-based NLJ strategy. The initial selection reduces the search space and is responsible for the good performance that we see. The times reported in this experiment are a bit slower than those in GP3 due to the larger result set sizes.

7.3.3 Scalability of Spatiotemporal Queries

We performed some basic experiments to demonstrate the scalability of spatiotemporal queries that combine a spatial operator and a temporal operator in a single SQL query.

7.3.3.1 Spatiotemporal Queries with respect to Dataset Size

Our first spatiotemporal experiment tested scalability with respect to dataset size. Tables 7.5 and 7.6 show the execution times for a query involving both a filtered \textit{temporal_extent} operator and a filtered \textit{spatial_extent} operator. Each query used one filtered \textit{spatial_extent} operator invocation and one filtered \textit{temporal_extent} operator invocation. The same unselective graph pattern was used in each operator invocation, and the results of each operator invocation were joined based on equality of variable values (i.e. along the lines of the spatiotemporal query example in Section 6.1). The results show that execution times are significantly slower than queries involving a single operator because the results for each individual function invocation must be retrieved and then joined based on variable corre-
spondences to form the final result. This slowdown occurs for both datasets. However, the queries show good scalability with respect to dataset size. Execution time is near constant as the dataset size increases for the GovTrack dataset, but the execution time grows linearly for the SynHist dataset. The growth in execution time for the SynHist dataset is due to the scalability of queries involving a filtered *temporal_extent* operator on this dataset as discussed previously.
### Table 7.5: Execution time for filtered spatial extent plus filtered temporal extent for GovTrack dataset

<table>
<thead>
<tr>
<th>Query Operator</th>
<th>Relation</th>
<th>Graph Pattern</th>
<th>Result Size</th>
<th>Execution Time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STG1</td>
<td>ST-Filter</td>
<td>INSIDE INT/DURING</td>
<td>3</td>
<td>122</td>
</tr>
<tr>
<td>STG2</td>
<td>ST-Filter</td>
<td>ANYINT INT/DURING</td>
<td>5</td>
<td>397</td>
</tr>
</tbody>
</table>

### Table 7.6: Execution time for filtered spatial extent plus filtered temporal extent for SynHist dataset

<table>
<thead>
<tr>
<th>Query Operator</th>
<th>Relation</th>
<th>Graph Pattern</th>
<th>Result Size</th>
<th>Execution Time (msec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STH1</td>
<td>ST-Filter</td>
<td>OVERLAP INT/OVERLAP</td>
<td>3</td>
<td>43</td>
</tr>
<tr>
<td>STH2</td>
<td>ST-Filter</td>
<td>w/in DIST INT/OVERLAP</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>
7.3. EXPERIMENTS

Figure 7.5: Experiment ST1: basic spatial_extent plus temporal_extent with respect to graph pattern size for SynHist (SH5) and GovTrack (GT3) datasets.

7.3.3.2 Spatiotemporal Queries with respect to Graph Pattern Size

Experiment ST1 tested the scalability of a spatiotemporal query with respect to graph pattern complexity. The spatiotemporal queries involved both a spatial_extent operator invocation and a temporal_extent operator invocation. Within a spatiotemporal query, the same selective graph pattern was used for each operator and the results of the two operator invocations were joined on equality of variable values. Figure 7.5 shows the execution times for one such spatiotemporal query of each graph pattern size. The results of this experiment show that execution time tends to grow linearly with graph pattern complexity when result set size is taken into account. Execution times are roughly twice as long as a query involving a single operator (i.e. as in experiments GP3 and GP4), as results for both function invocations must be retrieved and then joined.
Query Language Support

It is important that our STT querying approach fits with the Semantic Web community’s existing querying framework. SPARQL is the current World Wide Web Consortium (W3C) recommended query language for RDF data (Prud’hommeaux and Seaborne, 2008). This chapter presents SPARQL-ST, an extension of SPARQL that allows querying spatial RDF data and temporal RDF graphs.

The remainder of this chapter is organized as follows. First, we give an introduction to the main features of the SPARQL query language. This is followed by a formal syntax and semantics for SPARQL-ST and an illustration of the concrete syntax of SPARQL-ST through a series of examples. We then give explanations for our major design decisions and discuss other approaches for querying spatial and temporal data with SPARQL. The implementation of SPARQL-ST is presented at the end of the chapter.

This chapter refers to concepts introduced earlier in the dissertation, more specifically the spatial ontology from Chapter 4 and terminology, such as \texttt{interval\_extension()}, from Chapter 5. Readers may want to refer back to these chapters, if needed, while reading this chapter.
8.1 The SPARQL Query Language

SPARQL is a declarative query language with a familiar SQL-like syntax. We will introduce the major concepts of SPARQL with a series of examples. For full details of the language, the reader is referred to its W3C specification document (Prud’hommeaux and Seaborne, 2008).

SPARQL queries follow what can be thought of as a subgraph matching querying paradigm. A query specifies a particular pattern or type of subgraph, and the results of the query represent all instances of the subgraph in a given RDF graph. In SPARQL, these subgraph types are specified with a set of triple patterns called a graph pattern. A triple pattern is essentially an RDF triple where the subject, predicate and/or object has been replaced with a variable. A graph pattern matches a subgraph in an RDF graph when one can substitute terms in the RDF graph for variables in the graph pattern and the result is an RDF graph equivalent to the matched subgraph. Figure 8.1 illustrates this subgraph matching paradigm.

Example 12 shows a simple SPARQL query involving a single triple pattern. The query selects all resources that are of type usbill:HouseBill. An optional PREFIX clause is used to specify abbreviations for namespaces used in the query as a shorthand for the full URI. A SELECT clause specifies the variables whose values should appear in the result.
8.1. THE SPARQL QUERY LANGUAGE

Figure 8.2: Set of RDF triples in TURTLE format for illustration of example queries.

The graph pattern used in the query appears in the WHERE clause. Variables in the graph pattern are identified with a “?” prefix. Table 8.1 shows the result of this query for the RDF data in Figure 8.2. This RDF data will be used for the remainder of the examples in this section.

Example 12.

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX usbill: <http://www.rdfabout.com/rdf/schema/usbill/>

SELECT ?b
WHERE {
  ?b rdf:type usbill:HouseBill .
}
```

Table 8.1: Results of Example 12

<table>
<thead>
<tr>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://.../usgov/congress/106/bills/h2916">http://.../usgov/congress/106/bills/h2916</a></td>
</tr>
<tr>
<td><a href="http://.../usgov/congress/107/bills/h3041">http://.../usgov/congress/107/bills/h3041</a></td>
</tr>
</tbody>
</table>

Example 13 shows a SPARQL query involving a graph pattern that consists of multiple triple patterns, and the result is shown in Table 8.2. The query selects all resources of type `usbill:HouseBill` and their sponsors. We have left out the PREFIX clause in this example and all remaining examples for brevity.
Example 13.

SELECT ?b, ?p
WHERE { ?b rdf:type usbill:HouseBill .
    ?b usbill:sponsor ?p }

Table 8.2: Results of Example 13

<table>
<thead>
<tr>
<th>b</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://.../106/bills/h2916">http://.../106/bills/h2916</a></td>
<td><a href="http://.../people/N000002">http://.../people/N000002</a></td>
</tr>
<tr>
<td><a href="http://.../107/bills/h3041">http://.../107/bills/h3041</a></td>
<td><a href="http://.../people/D000275">http://.../people/D000275</a></td>
</tr>
</tbody>
</table>

Example 14 illustrates a graph pattern involving a String literal. Literals are expressed in graph patterns using double quotes. The query selects all resources of type *usbill:HouseBill* that were introduced in the 107th congress. The result of this query is shown in Table 8.3.

Example 14.

SELECT ?b
WHERE { ?b rdf:type usbill:HouseBill .
    ?b usbill:congress "107" }

Table 8.3: Results of Example 14

<table>
<thead>
<tr>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://.../107/bills/h3041">http://.../107/bills/h3041</a></td>
</tr>
</tbody>
</table>

Example 15 illustrates SPARQL’s **OPTIONAL** clause. Specifying that part of a graph pattern is optional essentially means that a result should return values for variables in the optional part of the graph pattern if possible, but subgraphs should not be discarded if the optional part of the graph pattern is not matched. This **OPTIONAL** construct is analogous to an outer join in relational databases. The example query below selects all resources that are of type *usbill:HouseBill* and also returns the committees associated with the bills if this information is available. Table 8.4 shows the result of this query. Note that bill *h2916* appears in the result even though no committee was found for this bill.
Example 15.

SELECT ?b, ?c
WHERE { ?b rdf:type usbill:HouseBill .
    OPTIONAL { ?b usbill:inCommittee ?c } }

Table 8.4: Results of Example 15

<table>
<thead>
<tr>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://.../bills/h2916">http://.../bills/h2916</a></td>
<td><a href="http://.../HouseCommerce">http://.../HouseCommerce</a></td>
</tr>
<tr>
<td><a href="http://.../bills/h3041">http://.../bills/h3041</a></td>
<td></td>
</tr>
</tbody>
</table>

The SPARQL FILTER construct is used to restrict the results of a graph pattern match according to some expression. Many built-in operators for use in these expressions are listed in the SPARQL specification. Example 16 uses a regular expression matching operator to restrict the results of the graph pattern query to those with a label containing the word “handgun.” The result of this query is shown in Table 8.5.

Example 16.

SELECT ?b
WHERE { ?b rdf:type usbill:HouseBill .
    ?b rdfs:label ?l .
    FILTER (regex(?l, "handgun", "i")) }

Table 8.5: Results of Example 16

<table>
<thead>
<tr>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://.../106/bills/h2916">http://.../106/bills/h2916</a></td>
</tr>
</tbody>
</table>

8.2 The SPARQL-ST Query Language

This section presents the SPARQL-ST query language extension. We first give a formal syntax for SPARQL-ST and then give a formal semantics for SPARQL-ST queries. The
section concludes by presenting the concrete syntax of SPARQL-ST using a set of example queries.

8.2.1 Formal Syntax of SPARQL-ST

In this section, we give a formalization of the SPARQL-ST syntax. Our extension is based on the formalization of the SPARQL syntax given by Perez et al. (2006). We introduce spatial variables and temporal variables, which are used to form spatiotemporal graph patterns. We also introduce spatial built-in conditions and temporal built-in conditions.

8.2.1.1 Spatiotemporal Graph Patterns

Let $UL$ denote the union $U \cup L$ (recall that $U$ is the set of URIs and $L$ is the set of Literals) and let $V_N$ be a set of variables. Let $V_S$ be a set of spatial variables, and let $V_T$ be a set of temporal variables. $V_N$, $V_S$, $V_T$, and $RT$ (the set of RDF terms) are pairwise disjoint. A spatial triple pattern is a 3-tuple from $(UL \cup V_N \cup V_S) \times (U \cup V_N) \times (UL \cup V_N \cup V_S)$. A spatiotemporal triple pattern is a 4-tuple from $(UL \cup V_N \cup V_S) \times (U \cup V_N) \times (UL \cup V_N \cup V_S) \times (V_T)$.

A spatiotemporal graph pattern is defined recursively as follows:

- if $st$ is a spatial triple pattern, then $st$ is a spatiotemporal graph pattern
- if $stt$ is a spatiotemporal triple pattern, then $stt$ is a spatiotemporal graph pattern
- if $SP_1$ and $SP_2$ are spatiotemporal graph patterns, then $(SP_1 \text{ AND } SP_2)$ is a spatiotemporal graph pattern
- if $SP$ is a spatiotemporal graph pattern and $R$ is a SPARQL built-in condition, then the expression $(SP \text{ FILTER } R)$ is a spatiotemporal graph pattern
8.2. THE SPARQL-ST QUERY LANGUAGE

- if $SP$ is a spatiotemporal graph pattern and $SR$ is a spatial built-in condition, then
  the expression ($SP$ SPATIAL FILTER $SR$) is a spatiotemporal graph pattern

- if $SP$ is a spatiotemporal graph pattern and $TR$ is a temporal built-in condition, then
  the expression ($SP$ TEMPORAL FILTER $TR$) is a spatiotemporal graph pattern

The syntax for SPARQL built-in conditions is given in Perez et al. (2006) and remains
unchanged. Spatial built-in conditions and temporal built-in conditions are described be-
low.

8.2.1.2 Spatial Built-in Conditions

SPARQL-ST requires that we express spatial constraints on spatial variables. We intro-
duce spatial built-in conditions for this purpose. Spatial built-in conditions are built from
qualitative spatial functions and metric spatial functions.

A qualitative spatial function is a Boolean function $qsf : S \times S \rightarrow \mathbb{B}$. Recall that $S$
the set of all possible spatial geometries. Any of the following topological spatial relations
identified by Egenhofer and Herring (1994) may be used as qualitative spatial functions in
our formalization: disjoint, touch, overlap boundary disjoint, overlap boundary intersect,
equal, contains, covers, inside, covered by. We define a qualitative spatial expression, $qse$,
as follows, where $s_1, s_2 \in S \cup V_S$.

$$\langle qse \rangle := qsf(s_1, s_2)$$

A metric spatial function is a function $msf : S \times S \rightarrow \mathbb{R}$. We use one metric spatial
function $distance : S \times S \rightarrow \mathbb{R}$, which returns the distance between two spatial geometries.
We define a metric spatial expression, $mse$, as follows, where $s_1, s_2 \in S \cup V_S$ and $r \in \mathbb{R}$.

$$\langle mse \rangle := \langle msf(s_1, s_2) \rangle \langle \text{comp} \rangle r$$

$$\langle \text{comp} \rangle := < | > | \leq | \geq | =$$
A spatial built-in condition \( sf \) evaluates to a Boolean value for a given graph and is defined in terms of metric spatial expressions and qualitative spatial functions. A spatial built-in condition takes the following form.

\[
\langle sf \rangle ::= \langle mse \rangle \mid \langle qse \rangle \mid \langle sf \rangle \text{ AND } \langle sf \rangle \mid \langle sf \rangle \text{ OR } \langle sf \rangle \mid \text{NOT } \langle sf \rangle
\]

### 8.2.1.3 Temporal Built-in Conditions

To express constraints on temporal variables in SPARQL-ST, we introduce temporal built-in conditions. Temporal built-in conditions are built from qualitative and metric temporal functions. For a given temporal RDF graph \( G_t \) over time domain \( T \), let \( I \) denote the set of all time intervals over \( T \).

As a prerequisite, we define a temporal primitive \( tp \) as follows, where \( V'_T \subseteq V_T \), \( vt \in V_T \) and \( i \in I \).

\[
\langle tp \rangle ::= \text{intersect}(V'_T) \mid \text{range}(V'_T) \mid vt \mid i
\]

A qualitative temporal function is a Boolean function \( qtf : I \times I \rightarrow \mathbb{B} \). Any of the thirteen interval relations identified by Allen Allen (1983) can be used in qualitative temporal functions in our formalization. We define a qualitative temporal expression, \( qte \), as follows.

\[
\langle qte \rangle ::= qtf(\langle tp \rangle, \langle tp \rangle)
\]

A metric temporal function is a function \( mtf : I \times I \rightarrow \mathbb{Z} \). We use one metric temporal function \( \text{elapsed time} : I \times I \rightarrow \mathbb{Z} \), which is defined for two disjoint time intervals as the duration of time between the end of the earliest interval and the start of the latest interval. The function returns zero if the intervals are not disjoint. We define a metric temporal expression, \( mte \), as follows, where \( z \in \mathbb{Z} \).
8.2. THE SPARQL-ST QUERY LANGUAGE

A temporal built-in condition \( tf \) evaluates to a Boolean value for a given graph and is constructed from qualitative temporal functions and metric temporal expressions as follows:

\[
\langle tf \rangle := \langle mte \rangle \mid \langle qte \rangle \mid \langle tf \rangle \land \langle tf \rangle \mid \langle tf \rangle \lor \langle tf \rangle \mid \text{NOT} \langle tf \rangle
\]

8.2.2 Formal Semantics of SPARQL-ST Queries

The semantics of a SPARQL-ST spatiotemporal graph pattern query is based on the concept of a mapping. Here, we extend the concept of a mapping from Perez et al. (2006) to also include spatial and temporal variables. Conceptually, our extension maps spatial variables to a set of RDF triples rather than a single URI and maps temporal variables to a time interval rather than a single URI. For illustration, an example set of triples that represent a valid Spatial Region instance is shown in Figure 8.3. Recall that for a set \( A \), \( 2^A \) denotes the powerset of \( A \). A mapping \( \mu \) is a function from \( (V_N \cup V_S \cup V_T) \) to \( (RT \cup 2^{(U \cup B) \times U \times RT}) \cup I \) such that:

- if \( vn \in V_N \) then \( \mu(vn) = rt \in RT \)
- if \( vs \in V_S \) then \( \mu(vs) = g \in 2^{(U \cup B) \times U \times RT} \) and \( g \) forms a valid Spatial Region instance
- if \( vt \in V_T \) then \( \mu(vt) = i \in I \)

For a mapping \( \mu \), the subset of \( (V_N \cup V_S \cup V_T) \) where it is defined is called its domain \( \text{dom}(\mu) \). Two mappings \( \mu_1 \) and \( \mu_2 \) are compatible if, for all \( x \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2) \), it
is the case that $\mu_1(x) = \mu_2(x)$. In other words, the union $\mu_1 \cup \mu_2$ is also a mapping. In addition, for two sets of mappings $M_1$ and $M_2$, the join is defined as:

$$M_1 \bowtie M_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \text{ and } \mu_2 \in M_2 \text{ and } \mu_1 \text{ and } \mu_2 \text{ are compatible mappings}\}$$

The semantics of a spatiotemporal graph pattern are defined in terms of a function $[[\cdot]]$, which takes a spatiotemporal graph pattern and returns a set of mappings. Before we can define this function, we need to introduce some additional constructs to handle spatial and temporal aspects of graph patterns. Because a spatial variable maps to a collection of triples, we introduce a function, head, that reduces this set of triples to a single URI. We also define functions, triple and ttriple, which allow us to go from a mapping to a single RDF triple or temporal RDF triple. These single triples are used to formally define the function $[[\cdot]]$.

We will first define the function $\text{head} : (RT \cup 2^{((U \cup B) \times U \times RT)}) \rightarrow RT$. This function is defined as follows:

- if $t \in RT$ then $\text{head}(t) = t$
- if $t \in 2^{((U \cup B) \times U \times RT)}$ then $\text{head}(t) = s \in RT$ such that $(s, \text{rdf:type, Spatial Region}) \in t$

Conceptually, if $t$ is a single URI, $\text{head}(t)$ returns this single URI, and if $t$ is a collection of triples representing a Spatial Region instance, $\text{head}(t)$ returns the top level
8.2. THE SPARQL-ST QUERY LANGUAGE

URI of the Spatial Region instance. For the example in Figure 8.3, the top level URI is geo: polygon.123.

For a mapping \( \mu \) and a spatial triple pattern \( sp \), we denote the triple obtained by replacing the variables \( v \) in \( sp \) with the value \( \text{head}(\mu(v)) \) as \( \text{triple}(\mu, sp) \). For a mapping \( \mu \) and a spatiotemporal triple pattern \( stp \), we denote the temporal triple obtained by replacing the variables \( v \in V_N \cup V_S \) in \( stp \) with the value \( \text{head}(\mu(v)) \) and the variables \( t \in V_T \) in \( stp \) with the value \( \mu(t) \) as \( t \_ \text{triple}(\mu, stp) \).

Let \( G_t \) be a temporal RDF graph, \( sp \) a spatial triple pattern, \( stp \) a spatiotemporal triple pattern and \( SP_1, SP_2 \) spatiotemporal graph patterns. The evaluation of a spatiotemporal graph pattern over \( G_t \), denoted \([\cdot]_{G_t}\), is defined recursively as:

- \([sp]_{G_t} = \{ \mu \mid \text{dom}(\mu) = \text{var}(sp) \text{ and } \text{triple}(\mu, sp) \in \text{TRIPLES}(G_t) \}\)
- \([stp]_{G_t} = \{ \mu \mid \text{dom}(\mu) = \text{var}(sp) \text{ and for } (s, p, o) : [t_1, t_2] = t \_ \text{triple}(\mu, stp) \text{ it is the case that } (s, p, o) \in \text{TRIPLES}(G_t) \text{ and } [t_1, t_2] \in \text{contig. intervals}(\text{temporal((s, p, o))})\}
- \([SP_1 \text{ AND } SP_2]_{G_t} = [[SP_1]]_{G_t} \bowtie [[SP_2]]_{G_t}\)

The semantics of spatial built-in conditions and temporal built-in conditions are defined as follows. A mapping \( \mu \) satisfies a spatial built-in condition \( sf \) written \( \mu \models sf \) if \( \text{var}(sf) \subseteq \text{dom}(\mu) \) and \( sf \) evaluates to true when each variable \( vs \in V_S \) in \( sf \) is replaced with \( \text{geom}(\mu(vs)) \). A mapping \( \mu \) satisfies a temporal built-in condition \( tf \) written \( \mu \models tf \) if \( \text{var}(tf) \subseteq \text{dom}(\mu) \) and \( tf \) evaluates to true when each variable \( vt \in V_T \) in \( tf \) is replaced with \( \mu(vt) \).

Given a temporal RDF graph \( G_t \), a spatiotemporal graph pattern \( SP \), a spatial built-in condition \( SR \) and a temporal built-in condition \( TR \),

- \([SP \text{ SPATIAL FILTER } SR]_{G_t} = \{ \mu \in [[P]]_{G_t} \mid \mu \models SR \}\)
8.2. **THE SPARQL-ST QUERY LANGUAGE**  

- \[ [SP TEMPORAL FILTER TR]_{G_t} = \{ \mu \in [P]_{G_t} \mid \mu \models TR \} \]

### 8.2.3 SPARQL-ST by Example

This section presents the concrete syntax of SPARQL-ST using examples. Temporal variables are identified using a ‘#’ prefix, and spatial variables are identified using a ‘%’ prefix. The constructs `intersect()` and `range()` refer to the `intersect` and `range` intervals defined in Chapter 5.

**Example 17.** (Basic Temporal Query) Find all politicians who were senators of Ohio at the same time, and return the times of joint senatorship. This query selects each pair of senators and the `intersect` interval representing the time during which all temporal triples in that result were valid. The existence of such an interval implies that the senators held the position at the same time.

```sparql
SELECT ?s1, ?s2, intersect(#t1, #t2, #t3, #t4)
WHERE {
  ?s1 usgov:hasRole ?a #t1 .
  ?a usgov:forOffice usgov:senate/oh #t2 .
  ?s2 usgov:hasRole ?b #t3 .
  ?b usgov:forOffice usgov:senate/oh #t4 }
```

**Example 18.** (Temporal Filter Query) Find all house members who sponsored a bill after April 2, 2008. This query uses the `TEMPORAL FILTER` construct to ensure that all triples in a result are valid (i.e. they have a valid `intersect` interval at some time after April 2, 2008.

```sparql
SELECT ?p, ?b
WHERE {
  ?p usgov:hasRole ?r #t1 .
  "temporal filter" -- TEMPORAL FILTER
}```
Example 19. (Temporal Filter Query) Find all politicians who have been on the Senate Energy and Natural Resources Committee and any bills they sponsored while on the committee. This query uses a TEMPORAL FILTER to ensure that the sponsorship relation is valid during the valid time of the corresponding membership relation.

```
SELECT ?p, ?b 
WHERE { 
  usgov:SenateEnergyandNaturalResources foaf:member ?p #t1 .
  TEMPORAL FILTER (during(#t2, #t1)) 
}
```

Example 20. (Basic Spatial Query) Find the congressional district spatial geometries for all politicians who voted for bill number 88. This query simply selects the spatial variable representing the appropriate Spatial Region instance.

```
SELECT ?p, %g 
WHERE { 
  ?v usgov:hasBallot ?b .
  ?v usgov:billNo "88" .
  ?b usgov:hasOption "Aye" .
  ?p usgov:hasRole ?r .
}
```
Example 21. (Filtered Spatial Query) Find all politicians representing congressional districts within a given bounding box. This query uses a SPATIAL FILTER involving the inside function to ensure each returned congressional district falls within the given geographical area.

```
SELECT ?p
WHERE {
  ?p usgov:hasRole ?r .
  ?r usgov:forOffice ?o .
  ?o usgov:represents ?c .
  ?c stt:located_at %g .
  SPATIAL FILTER (inside(%g, GEOM(POLYGON ((
    -75.14734 40.884813, -70.77847 40.884813,
    -70.77847 42.3525606, -75.14734 42.3525606,
    -75.14734 40.884813)) ))
}
```

Example 22. (Filtered Spatial Query) Find all politicians that represent areas within 100 miles of the district represented by Nancy Pelosi. This query uses a SPATIAL FILTER involving a distance function over two variables. This SPATIAL FILTER clause serves a spatial join condition for the two disjoint graph patterns in the query.

```
SELECT ?n
WHERE {
  ?p usgov:hasRole ?r .
```
8.2. \textit{THE SPARQL-ST QUERY LANGUAGE}

\begin{verbatim}
?r usgov:forOffice ?o .
?o usgov:represents ?q .
?q stt:located_at %g .
?a foaf:name "Nancy Pelosi" .
?a usgov:hasRole ?b .
?c usgov:represents ?d .
?d stt:located_at %h .
SPATIAL FILTER (distance(%g, %h) <= 100 miles) }
\end{verbatim}

\textbf{Example 23.} (Spatiotemporal Query) At what times does John Linder represent a district that borders a district represented by a member of a different political party, and who is the other representative? This query uses a \texttt{SPATIAL FILTER} to join two disjoint graph patterns as in example 22, but it also selects the \texttt{intersect} interval that represents the times the spatial relation holds.

\begin{verbatim}
SELECT ?n, intersect(#t1, #t2, #t3, #t4, #t5, #t6, #t7, #t8, #t9, #t10, #t11, #t12)
WHERE {
  ?l foaf:name "John Linder" #t1 .
  ?l usgov:hasRole ?r #t3 .
  ?o usgov:represents ?q #t5 .
  ?q stt:located_at %g #t6 .
  ?a foaf:name ?n #t7 .
  ?a usgov:hasRole ?b #t9 .
\end{verbatim}
8.3 Design Decisions

In this section, we review the major design decisions for SPARQL-ST. We discuss possible alternatives and give reasons for our decisions.

8.3.1 Spatial Variables

The introduction of spatial variables is a major component of our SPARQL extension. These variables represent complex spatial objects and map to a set of RDF triples. Two possible alternatives to introducing a new variable type are (1) specifying all parts of the spatial object in a graph pattern and (2) utilizing the concept of named graphs to represent spatial objects.

Example 24 illustrates the first alternative where the relevant parts of a spatial object are specified in a graph pattern.

Example 24.

SELECT ?positions
WHERE { <http://.../house/106/nh> usgov:represents ?x .
?lr geo:lrPosList ?positions }
8.3. DESIGN DECISIONS

We see the following problems with this approach. First, the relevant portions of a spatial object that need to be returned from the query will vary. For example, if one is selecting the position lists of a multipolygon, it is unclear how to specify this in a graph pattern, as the number of polygons making up each multipolygon will vary. Second, it is unclear how to reference a spatial object in a spatial filter expression. That is, what parts of the graph pattern should be passed to a spatial function in the spatial filter expression? A special variable type solves both of these problems.

Another alternative is to use named graphs to represent spatial objects. A named graph is created by associating a set of RDF triples with some URI $u$. This set of triples can then be collectively referred to by the identifier $u$. This strategy would require a modification to our modeling approach that substitutes graph for Spatial Region in our upper-level ontology, and a given named graph would contain all triples for one Spatial Region (i.e. there would be a one-to-one mapping between named graphs and Spatial Regions). A query using this approach is shown in Example 25. This query returns all triples making up each named graph (Spatial Region) in the result.

**Example 25.**

```sparql
SELECT ?g, ?s, ?p, ?o
WHERE {
  <http://.../house/106/nh> usgov:represents ?x .
  ?x stt:located_at ?g .
  GRAPH ?g { ?s, ?p, ?o } }
```

We feel that this solution is problematic because it uses the named graph construct in an unintended way, which makes the semantics of our STT modeling approach less clear. In addition, using a named graph as input to a spatial function could lead to unexpected errors if the input named graph did not represent a Spatial Region.
8.3. DESIGN DECISIONS

8.3.2 Temporal Variables

Another key aspect of our approach is using temporal variables to specify a quad to represent a temporal triple pattern. An alternative would be to use SPARQL as it is and use the RDF reification triples to extract valid times for triples. Example 26 illustrates this approach.

Example 26.

```
SELECT ?x, ?t11, ?t12, ?t21, ?t22
WHERE {
    ?x usgov:hasRole ?r .
    ?r usgov:forOffice ?o .
    ?s1 rdf:subject ?x .
    ?s1 rdf:predicate usgov:hasRole .
    ?s1 rdf:object ?r .
    ?s1 stt:temporal ?t1 .
    ?st1 owlTime:inCalendarClockDataType ?t11 .
    ?t1 owlTime:ends ?end1 .
    ?end1 owlTime:inCalendarClockDataType ?t12 .
    ?s2 rdf:subject ?r .
    ?s2 rdf:predicate usgov:forOffice .
    ?s2 rdf:object ?o .
    ?st2 owlTime:inCalendarClockDataType ?t21 .
    ?end2 owlTime:inCalendarClockDataType ?t22 }
```

This approach is problematic for the following reasons. First, it is extremely verbose,
as it takes eight triple patterns to retrieve the valid times for each statement. Second, the semantics of the temporal queries are lost because the query will simply match triples in the RDF dataset, and the concepts of temporal RDFS inferencing, \textit{interval\_extension()}, etc. are lost. In addition, special temporal variables make it clear that one is querying a temporal RDF graph rather than a plain RDF graph.

### 8.3.3 Filter Expressions

We define separate \textsc{spatial\_filter} and \textsc{temporal\_filter} expressions in our query language extension. We could have used the existing \textsc{sparql\_filter} construct and allowed spatial and temporal functions within a standard filter expression. We chose to define new filter expression types because it makes the semantics of the query language extension easier to understand and to formalize.

### 8.4 Other Approaches for Querying Spatial and Temporal Data using SPARQL

Extensions of the SPARQL query language are abundant in the literature. These range from extensions for computing semantic associations (Anyawu et al., 2007; Kochut and Janik, 2007) to extensions for enabling skyline queries (Siberski et al., 2006). To the best of our knowledge, no extensions of SPARQL for temporal RDF graphs or spatial RDF data exist at this date. Ways to handle spatial and temporal data in standard SPARQL have been discussed, however.

The only discussion of querying spatial data using SPARQL appears in a paper by Kolas and Self (2007) in the Semantic Web in use track of ISWC 2007. The authors describe a prototype system for integrated storage and querying of spatial and semantic data. The
system is queried using standard SPARQL syntax. They use the GeoRSS RDF vocabulary to model spatial objects and use a set of qualitative topological relationships based on the Region Connection Calculus (Cohn et al., 1997) to specify spatial relationships in queries. The query below uses their approach to find gas stations within 1 mile of 38°N, 77°W.

```
SELECT ?x
WHERE { ?x rdf:type gas:GasStation .
   ?p gml:radius "1" .
   ?g rdf:type gml:Point .
   ?g gml:pos "38 -77" }
```

In contrast to this approach, we introduce special spatial variables and specify spatial constraints using a SPATIAL FILTER clause instead of encoding the spatial constraint within the graph pattern. Without introducing spatial variables this approach would suffer from the shortcomings described previously. In addition, encoding the constraint “within 1 mile of 38°N, 77°W” as a part of a graph pattern implies different semantics for a graph pattern query than those specified for SPARQL. A graph pattern should match an explicit subgraph in the RDF graph being queried (e.g., according to SPARQL semantics, the rcc:part relation from the above example should explicitly exist in the RDF graph). With Kolas and Self’s approach, the graph pattern is matched by performing spatial computations on the fly to determine the existence of the relationship. In contrast, we would specify such a constraint using a SPATIAL FILTER expression. Their implementation only supported the relations connected and part, and no performance results were pre-
There have also been proposals for adding geospatial capabilities to SPARQL using the extensibility features of the Jena Semantic Web framework and its ARQ SPARQL engine (Hewlett-Packard Development Company, 2008). For example, code implementing property functions that extend ARQ for geospatial relations appears at http://geospatialweb.googlecode.com/svn/trunk/jenaext/src/org/geospatialweb/arqext/. The following example query uses a nearby() property function to select hotels near a certain point.

```
SELECT ?n
WHERE {
  ?s geo:nearby(51.45, -2.583) .
  ?s rdf:type ex:Hotel .
  ?s ex:name ?n }
```

Again, such an approach does not use spatial variables, so it will suffer from the shortcomings we mentioned earlier. This approach also does not conform to the SPARQL specification because property functions are an ARQ-specific feature that are not part of the SPARQL specification.

There are currently no extensions of SPARQL for temporal RDF graphs. However Gutierrez et al. (2005, 2007) and Pugliese et al. (2008) discuss aspects of querying temporal RDF graphs. Gutierrez et al. (2005, 2007) briefly present a query language for temporal RDF graphs through a series of examples. The authors state that the query language needs a built in arithmetic language to reason about time and intervals and a construct to form maximal validity intervals for a given triple. In our proposal, the TEMPORAL FILTER clause provides the needed temporal reasoning capabilities, and the need for maximal intervals is taken care of by the interval_extension() function used to define the semantics of our SPARQL extension. Pugliese et al. (2008) formally define a temporal RDF query.
The query is essentially a graph pattern involving triple patterns associated with either a temporal variable or a temporal constraint. The temporal query specified by Pugliese, et al. also supports the notion of a maximal interval for each triple. An additional feature we support over these proposals is the ability to perform temporal computations over temporal intervals derived from the maximal intervals of multiple triples. We use the notions of intersect and range to provide this capability. Furthermore, none of these querying approaches have been defined in the context of the SPARQL W3C recommendation.

8.5 SPARQL-ST Implementation Framework

We have implemented SPARQL-ST using the extensibility framework of Oracle 10g. The implementation builds on Oracle’s existing support for storage and querying of RDF data and spatial data. We provide a single SQL table function, sparql_st, that inputs a valid SPARQL-ST query and returns a table of the resulting variable mappings.

8.5.1 Table Structures

Our SPARQL-ST implementation uses a slightly modified version of the storage scheme presented in Section 6.2. The revised storage scheme is shown in Figure 8.4. The TemporalTriples table remains unchanged, but an additional column (rdf_serialization) has been added to the SpatialData table that stores a CLOB containing a serialization of all the RDF triples associated with a particular Spatial Region. This column is populated during the spatial indexing procedure described in Section 6.2.2.1. Storing this additional information allows us to quickly look up and return the appropriate mapping for a spatial variable.
8.5. SPARQL-ST IMPLEMENTATION FRAMEWORK

8.5.2 Function Definition

The definition of the \( \text{sparql}\_\text{st} \) table function is shown below.

\[
\text{sparql}\_\text{st}(\text{queryStr VARCHAR}) \text{ return AnyDataSet} ;
\]

The function inputs a query in SPARQL-ST syntax as a single \texttt{VARCHAR} parameter. The result table contains a single row for each variable in the \texttt{SELECT} clause of the SPARQL-ST query. Example 27 illustrates a SPARQL-ST query in this scheme.

\textbf{Example 27.}

\begin{verbatim}
SELECT *
FROM TABLE (sparql_st ( 
    'SELECT ?x, ?z, ?y, %l
     WHERE {
         ?x <assigned_to> ?z .
         ?x <participates_in> ?y .
         ?y <occurred_at> %l .
     SPATIAL FILTER(overlapbdyintersect(%l, GEOM(
         POLYGON((-122.84501 42.240328, 
            -122.8075 42.240328, -122.8075 42.3764, 
            -122.84501 42.3764, -122.84501 42.240328))))
}

\end{verbatim}
8.5.3 Function Implementation

Our implementation of the \textit{sparql\_st} table function builds off our existing \textit{spatial\_extent}, \textit{spatial\_eval}, \textit{temporal\_extent}, and \textit{temporal\_eval} functions. Each of these existing functions was modified to input and evaluate an arbitrary spatial filter expression and an arbitrary temporal filter expression. These modified functions evaluate the arbitrary filter expressions by applying the filter row-by-row in their \textit{fetch} method.

Our basic approach for the \textit{sparql\_st} function is to select a single condition from the spatial or temporal filter expression and use this single condition for an index-based execution of one of our existing STT operators and then evaluate the remaining portions of the filter expressions on a row-by-row basis over the results of the initial index-based query. For example, consider the query below.

\begin{verbatim}
SELECT ...
WHERE {
  SPATIAL FILTER(overlap(%s1, <spatial_constant>)
    and touch(%s1, %s2))
}
\end{verbatim}

We would push down the first relation “\textit{overlap(%s1, <spatial\_constant>)}” to the base query in the \textit{start()} method of the modified \textit{spatial\_extent} function, and then the second relation “\textit{touch(%s1, %s2)}” would be evaluated as each row is returned in the \textit{fetch()} method.

The basic execution steps for the \textit{sparql\_st} function are shown in Algorithm 8. We first parse the SPARQL-ST query and extract the relevant portions of the query. Next, we determine the number of connected components in the query’s graph pattern. If there is a single connected component, we execute the appropriate \textit{spatial\_extent} or \textit{temporal\_extent} query. In the case of two connected components, we expect a spatial or temporal filter ex-
Algorithm 8 \textit{sparql\_st}

\textbf{Input:} 
$qStr$: SPARQL-ST query string

\textbf{Output:} 
\textit{rows}: query result

1: parse $qStr$ and populate $\text{graphPattern, spatialFilter, temporalFilter}$
2: $\text{numComps} \leftarrow$ number of connected components in $\text{graphPattern}$
3: if ($\text{numComps} = 1$) then
4: $\text{comp}_1 \leftarrow \text{graphPattern}$
5: $\text{sIdxCond} \leftarrow$ extract any atomic spatial filter appropriate for index-based evaluation
6: $\text{tIdxCond} \leftarrow$ extract any atomic temporal filter appropriate for index-based evaluation
7: if ($\text{sIdxCond}$ is not null) then
8: $\text{baseQuery} \leftarrow \text{spatial\_extent}($comp$_1, \text{sIdxCond}, \text{spatialFilter}, \text{temporalFilter})$
9: else
10: if ($\text{tIdxCond}$ is not null) then
11: $\text{baseQuery} \leftarrow \text{temporal\_extent}($comp$_1, \text{tIdxCond}, \text{spatialFilter}, \text{temporalFilter})$
12: else
13: $\text{baseQuery} \leftarrow \text{temporal\_extent}($comp$_1, \text{spatialFilter}, \text{temporalFilter})$
14: end if
15: end if
16: else
17: $\text{comp}_1 \leftarrow \text{getComponent1($graphPattern$)}$
18: $\text{comp}_2 \leftarrow \text{getComponent2($graphPattern$)}$
19: $\text{sJoinCond} \leftarrow$ extract any atomic spatial join condition appropriate for index-based evaluation
20: $\text{tJoinCond} \leftarrow$ extract any atomic temporal join condition appropriate for index-based evaluation
21: if ($\text{sJoinCond}$ is not null) then
22: $\text{baseQuery} \leftarrow \text{spatial\_eval}($comp$_1, \text{comp}_2, \text{sJoinCond}, \text{spatialFilter}, \text{temporalFilter})$
23: else
24: $\text{baseQuery} \leftarrow \text{temporal\_eval}($comp$_1, \text{comp}_2, \text{tJoinCond}, \text{spatialFilter}, \text{temporalFilter})$
25: end if
26: end if
27: $\text{sctx} \leftarrow \text{parse (baseQuery)}$
28: while $\text{sctx}\text{.results\_remaining()}$ do
29: $\text{rows} \leftarrow \text{sctx}\text{.fetch\_rows()}$
30: return $\text{rows}$
31: end while
pression to join the separate components. This join condition is identified, and we execute
the appropriate _spatial_eval_ or _temporal_eval_ query.
Conclusions and Future Directions

This dissertation showed how the relationship-centric nature of the RDF data model can extend the state-of-the-art in modeling and querying spatial, temporal and thematic data.

One of our main conceptual contributions with respect to data modeling is the use of graph patterns to specify a form of context. These contexts allow a many-to-many mapping between thematic and spatial objects, and the temporal properties of these graph patterns, modeled using temporal RDF graphs, adds a temporal dimension to our contexts. As a result of the many-to-many mapping between thematic and spatial objects, our modeling approach for STT data is more flexible than other approaches.

Our modeling approach was complemented by formally-defined query operators that allow graph pattern (thematic) queries with spatial and temporal constraints. These operators extend the state-of-the-art in querying RDF data because spatial and temporal constraints for graph pattern queries are not currently supported. In addition, these operators extend the state-of-the-art in querying traditional spatial and temporal data because graph patterns allow more complex queries in the thematic dimension than what is currently supported.

It is beneficial that the work in this dissertation is done in the context of the Semantic Web community. SPARQL is the current W3C-recommended query language for RDF data, and we formally defined SPARQL-ST, an extension of SPARQL to support spatial and temporal RDF data. This extension brings our STT querying in line with current standards.
9.1. **FUTURE WORK**

July 10, 2008

for querying RDF data.

While formal aspects of data modeling and querying are necessary for this research, one cannot overlook the importance of an efficient implementation of our framework. We have shown that our STT data model can be efficiently stored and our query operators can be efficiently implemented by extending a state-of-the-art commercial database system. In addition, we provided an efficient algorithm and implementation of temporal RDFS inferencing. We performed extensive testing of our implementation using both real-world and synthetic RDF datasets of over 25 million triples. Queries showed good scalability both in terms of dataset size and graph pattern complexity, for example execution time of less than 500 milliseconds for a 10-hop graph pattern over a 28 million triple dataset.

### 9.1 Future Work

A promising direction for future work is to extend this framework from graph pattern queries to semantic association queries (Anyanwu et al., 2007; Anyanwu and Sheth, 2003, 2002; Anyanwu et al., 2005; Kochut and Janik, 2007). Whereas graph pattern queries follow a subgraph matching querying paradigm, semantic association queries follow a path extraction querying paradigm. A semantic association query specifies a set of anchor points in an RDF graph and returns paths connecting the anchor points as a result. Usually, we would want to limit the types of paths returned (e.g., paths less than a certain length or paths containing edges of a certain type). For many domains it would be desirable to place constraints on the spatial and temporal properties of paths. Consider the following examples adapted from Anyanwu et al. (2007).

**Flight and Airport Risk Assessment:** To assess a potential threat to the safety of flights to certain airports, security officials may try to identify high-risk passengers scheduled for such flights.
9.1. FUTURE WORK

Find any relationships between passengers on flights to New York or Washington, DC, who purchased their tickets either less than 24hrs before departure or with cash. Limit results to connections associated with flight training, particularly very recent flight training (within the last three months).

Earlier attempts to support such computations, for example Sheth et al. (2005), did not consider spatial and temporal dimensions.

Conflict of Interest: To evaluate any potential conflict of interest between a defendant and potential jurors, one would like to search for social connections between them.

Find any close (i.e., less than 4-hop) explicit or implicit social connections between juror X and defendant Y, where an explicit social connection is a foaf:knows relationships in social network data, and an implicit connection is defined as close spatial and temporal proximity through employment or residency.

Here, again, earlier work on such applications, for example Aleman-Meza et al. (2006), did not consider spatial and temporal dimensions.

Analysis of Biological Pathways: Used by scientists to study complex interactions in living cells, biological pathways are usually represented as graphs where nodes represent molecules and edges represent reactions (Krishnamurthy et al., 2003). Researchers like to analyze genes with respect to the biological pathways in which they participate (Donninger et al., 2004). Spatial aspects (e.g., location of pathway components in the cell) and temporal aspects (e.g., time lag) of these interactions are also of interest to researchers (Saraiya et al., 2005).

Find all pathways connecting gene X to gene Y involving components that move between cell compartments over time.

Using our notion of context, possible spatial extensions could allow (1) constrained semantic association searches based on spatial predicates (e.g., find all paths connecting...
to \( y \) such that some intermediate node \( z \) is connected to location \( l \) with respect to context \( c \), (2) semantic association searches connecting thematic entities to a region of space (e.g., find all paths connecting \( x \) to location \( l \) with respect to context \( c \)) and (3) semantic associations searches using implicit spatial relations as links in the resulting paths (e.g., find all paths connecting \( x \) to \( y \) such that two intermediate nodes \( z_1 \) and \( z_2 \) satisfy spatial relation \( r \) with respect to context \( c \)).

Possible temporal extensions could allow semantic association searches restricted by (1) temporal constraints on adjacent edges in the path (e.g., find all paths connecting \( x \) to \( y \) such that the valid times for each pair of adjacent edges overlap) and (2) temporal constraints on intervals computed for the entire path (e.g., find all paths connecting \( x \) to \( y \) such that the entire path is valid during a specific time period).


BIBLIOGRAPHY


BIBLIOGRAPHY

July 10, 2008


Appendix

A.1 Queries used in Evaluation

Note: There are six table functions defined and used in these queries. *temporal_restrict* was presented as *temporal_extent* with filtering parameters in the dissertation for simplicity of presentation. The same applies for *spatial_restrict* (*spatial_find*). An extra parameter is defined to allow passing an optimizer hint to the database. In addition, for spatial functions extra parameters are defined to pass in the name of the *SpatialData* table and the name of the id and value_id columns in the *SpatialData* table.

A.1.1 Queries used for Table 7.3

*GovTrack Dataset*

temporal extent low selectivity

3 hop

```sql
select count(*) from table (temporal_extent(''
  (?x <http://.../usbill/cosponsor> ?y)
  (?x <http://.../usbill/sponsor> ?z)
  (?x <http://.../usbill/inCommittee> ?c)
',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT')) where rownum < 1001;
```

5 hop

```sql
select count(*) from table (temporal_extent(''
  (?a <http://.../politico/hasRole> ?b)
  (?x <http://.../usbill/cosponsor> ?a)
  (?b <http://.../politico/forOffice> ?c)
  (?x <http://.../usbill/sponsor> ?y)
  (?x <http://.../usbill/inCommittee> ?z)
',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
```

129
A.1. QUERIES USED IN EVALUATION

July 10, 2008

'tINTERSECT')}) where rownum < 1001;

**temporal extent high selectivity**

3 hop

```sql
select count(*) from table (temporal_extent('  (?p <http://.../politico/hasRole> ?y)  (?y <http://.../politico/forOffice>  <http://.../usgov/congress/senate/va>)  (<http://.../usgov/congress/senate/va>  <http://purl.org/dc/terms/isPartOf> ?x)'',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'tINTERSECT'));
```

5 hop

```sql
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'tINTERSECT'));
```

**filtered temporal extent**

3 hop

```sql
select count(*) from table (temporal_restrict('  (?x <http://.../politico/hasRole> ?y)  (?y <http://.../politico/forOffice> ?z)  (?z <http://purl.org/dc/terms/isPartOf> ?p)'',
to_date('1997-01-01', 'yyyy-mm-dd'),
to_date('2000-09-21', 'yyyy-mm-dd'),
'DURING',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'tINTERSECT', '/** FIRST_ROWS */'));
```

5 hop

```sql
```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

```
select count(*) from table (temporal_restrict('  (?p <http://.../usgovt/party> ?a)
  (?p <http://xmlns.com/foaf/0.1/name> ?b)
  (?p <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)',
to_date('2005-12-31', 'yyyy-mm-dd'),
to_date('2006-01-01', 'yyyy-mm-dd'),
'AFTER',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT', '/** FIRST_ROWS */')));
```

temporal eval

query 1

```
select count(*) from table (temporal_eval('  (?x <http://.../politico/hasRole> ?y)
  (?y <http://.../politico/forOffice> ?z)
  (?z <http://.../politico/represents>
    <http://.../usgov/geo/us/tx/cd/103/29>),
  'INTERSECT',
  ' (?a <http://.../politico/hasRole> ?b)
    (?b <http://.../politico/forOffice> ?c)
    (?c <http://purl.org/dc/terms/isPartOf>
      <http://.../usgov/congress/senate>),
  'INTERSECT',
  ' DURING_INV',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'/** FIRST_ROWS */')));
```

query 2

```
select count(*) from table (temporal_eval('  (?x <http://.../politico/hasRole> ?y)
  (?y <http://.../politico/forOffice> ?z)
  (?z <http://.../politico/represents>
    <http://.../usgov/geo/us/tx/cd/103/29>),
  'INTERSECT',
  ' (?a <http://.../politico/hasRole> ?b)
    (?b <http://.../politico/forOffice>
      <http://.../usgov/congress/senate/ky>),
  'INTERSECT',
  ' BEFORE',
```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'/*+ FIRST_ROWS */'));

spatial extent low selectivity

3 hop

select count(*) from table (spatial_extent(''
  (?y <http://.../politico/represents> ?z)
  (?z <http://.../usgovt/congress> ?x)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)',
  'l',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_small_geo_mod',
  'id')) where rownum < 1001;

5 hop

select count(*) from table (spatial_extent(''
  (?p <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
  (?z <http://.../usgovt/congress> ?b)'',
  'l',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_small_geo_mod',
  'id')) where rownum < 1001;

spatial extent high selectivity

3 hop

select count(*) from table (spatial_extent(''
  (?y <http://.../usgovt/congress> "109")
  (?y <http://.../politico/represents> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)',
  'l',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_small_geo_mod',
  'id'));
A.1. QUERIES USED IN EVALUATION

5 hop

```
```

Filtered spatial extent

3 hop

```
select count(*) from table (spatial_find('' ?x <http://.../politico/forOffice> ?y) (?y <http://.../politico/represents> ?z) (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'', 'l', 'POLYGON ((-75.14734 40.884813, -70.77847 40.884813, -70.77847 42.3525606, -75.14734 42.3525606, -75.14734 40.884813))', 8265, 'GEO_RELATE(mask=inside)'', SDO_RDF_Models('gov_track'), SDO_RDF_Rulebases('RDFS'), 'gov_small_geo_mod', 'id', 'shape', '/** LEADING(S) FIRST_ROWS */'));
```

5 hop

```
select count(*) from table (spatial_find('' ?p <http://.../politico/hasRole> ?x) (?p <http://.../usgovt/party> ?a) (?x <http://.../politico/forOffice> ?y) (?y <http://.../politico/represents> ?z) (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'', 'l', 'POLYGON ((-75.14734 40.884813, -70.77847 40.884813, -70.77847 42.3525606, -75.14734 42.3525606, -75.14734 40.884813))', 8265, 'GEO_RELATE(mask=anyinteract)'', SDO_RDF_Models('gov_track'),
```

133
A.1. QUERIES USED IN EVALUATION

July 10, 2008

SDO_RDF_Rulebases('RDFS'),
'gov_small_geo_mod',
'id',
'shape', '/*+ LEADING(S) FIRST_ROWS */');

simple dataset

3 hop

select count(*) from table (spatial_find(' (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/reports> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)',
  'l',
  'POLYGON ((-81.281509 41.497129, -80.519239 41.497129,
    -80.519239 41.977523, -81.281509 41.977523,
    -81.281509 41.497129))', 8265,
  'GEO_RELATE(mask=inside)',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_simp_geo_mod',
  'id',
  'shape', '/*+ LEADING(S) FIRST_ROWS */'));

5 hop

select count(*) from table (spatial_find(' (?p <http://.../politico/hasRole> ?x)
  (?p <http://.../usgovt/party> ?a)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/reports> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)',
  'l',
  'POLYGON ((-84.281509 39.497129, -80.519239 39.497129,
    -80.519239 41.977523, -84.281509 41.977523,
    -84.281509 39.497129))', 8265,
  'GEO_RELATE(mask=anyinteract)',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_simp_geo_mod',
  'id',
  'shape', '/*+ LEADING(S) FIRST_ROWS */'));

spatial eval

select count(*) from table (spatial_eval('
A.1. QUERIES USED IN EVALUATION

July 10, 2008

```sql
SELECT COUNT(*) FROM TABLE (SPATIAL_EVAL(''
  (http://.../usgov/congress/people/P000014
  <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l),
  'l',
  (?c <http://lsdis.cs.uga.edu/STT#located_at> ?d), 'd',
  'GEO_RELATE(mask=anyinteract)',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_small_geo_mod',
  'id', 'shape', '/*+ LEADING(S) FIRST_ROWS */'));
```

```sql
SELECT COUNT(*) FROM TABLE (SPATIAL_EVAL(''
  (http://.../usgov/congress/people/A000358
  <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l),
  'l',
  (?b <http://.../usgovt/congress> "106")
  (?b <http://lsdis.cs.uga.edu/STT#located_at> ?d), 'd',
  'GEO_DISTANCE(distance=5 unit=mile)',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_small_geo_mod',
  'id', 'shape', '/*+ LEADING(S) FIRST_ROWS */'));
```

**simple dataset**

```sql
SELECT COUNT(*) FROM TABLE (SPATIAL_EVAL(''
  (http://.../usgov/congress/people/P000014
  <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l),
  'l',
  (?c <http://lsdis.cs.uga.edu/STT#located_at> ?d), 'd',
  'GEO_RELATE(mask=anyinteract)',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_simp_geo_mod',
  'id', 'shape', '/*+ LEADING(S) FIRST_ROWS */'));
```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

select count(*) from table (spatial_eval(' 
  (<http://.../usgov/congress/people/A000358>
  <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)', 'l',
  '(?b <http://.../usgovt/congress> "106")
  (?b <http://lsdis.cs.uga.edu/STT#located_at> ?d)', 'd',
  'GEO_DISTANCE(distance=25 unit=mile)',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov.simp_geo_mod',
  'id', 'shape', '/*+ LEADING(S) FIRST_ROWS */'));

A.1.2 Queries used for Table 7.4

SynHist Dataset

low selectivity temporal extent queries

3 hop

select count(*) from table (temporal_extent(' 
  (?x <http://lsdis.cs.uga.edu/military#leader_of> ?y)
  (?z <http://lsdis.cs.uga.edu/military#assigned_to> ?y)
  (?z <http://lsdis.cs.uga.edu/military#participates_in> ?l)',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT')) where rownum < 1001;

5 hop

select count(*) from table (temporal_extent(' 
  (?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?z)
  (?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (?a <http://lsdis.cs.uga.edu/military#leader_of> ?z)
  (?z <http://lsdis.cs.uga.edu/military#platoon_of> ?b)',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT')) where rownum < 1001;
A.1. QUERIES USED IN EVALUATION  

July 10, 2008

**high selectivity temporal extent**

**3 hop**

select count(*) from table (temporal_extent(''
  (?x <http://lsdis.cs.uga.edu/military#assigned_to>
   <http://lsdis.cs.uga.edu/military/Platoon_12923>)
  (<http://lsdis.cs.uga.edu/military/Platoon_12923>
   <http://lsdis.cs.uga.edu/military#trains_at> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'),
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT'));

**5 hop**

select count(*) from table (temporal_extent(''
  (?x <http://lsdis.cs.uga.edu/military#assigned_to>
   <http://lsdis.cs.uga.edu/military/Platoon_12905>)
  (<http://lsdis.cs.uga.edu/military/Platoon_12905>
   <http://lsdis.cs.uga.edu/military#trains_at> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
  (<http://lsdis.cs.uga.edu/military/Platoon_12905>
   <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  (?a <http://lsdis.cs.uga.edu/military#leader_of>
   <http://lsdis.cs.uga.edu/military/Platoon_12905> ''),
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT'));

**high selectivity temporal restrict**

**3 hop**

select count(*) from table (temporal_restrict(''
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)'
  ,
  to_date('1940-03-22 00:26:01', 'yyyyMMdd hh24:mi:ss'),
  to_date('1941-12-01 10:22:00', 'yyyyMMdd hh24:mi:ss'),
  'OVERLAP',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT', '/**+ FIRST_ROWS */'));

**5 hop**
A.1. QUERIES USED IN EVALUATION

July 10, 2008


temporal eval


138
low selectivity spatial extent

3 hop


5 hop


high selectivity spatial extent

3 hop


5 hop
A.1. QUERIES USED IN EVALUATION

July 10, 2008


filtered spatial extent

3 hop

select count(*) from table (spatial_find('  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?z)  (?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)  (?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)" ', 'l', 'POLYGON((-122.84501 42.240328, -122.8075 42.240328, -122.8075 42.3764, -122.84501 42.3764, -122.84501 42.240328))", 8265, 'GEO_RELATE(mask=overlapbdyintersect)" , SDO_RDF_Models('syn_hist'), SDO_RDF_Rulebases('RDFS'), 'd1_military_mod_geo', 'id', 'shape', '/*+ LEADING(s) FIRST_ROWS */'));

5 hop

A.1. QUERIES USED IN EVALUATION

July 10, 2008

'd1_military_mod_geo',
'id',
'shape', '/*+ LEADING(s) FIRST_ROWS */');

spatial_eval
select count(*) from table (spatial_eval('  (?x <http://lsdis.cs.uga.edu/military#on_crew_of>
   <http://lsdis.cs.uga.edu/military/Air_Vehicle_12691>)
   (<http://lsdis.cs.uga.edu/military/Air_Vehicle_12691>
   <http://lsdis.cs.uga.edu/military#used_in> ?z)
   (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)'
   ',
   'l',
   '(?c <http://lsdis.cs.uga.edu/STT#occurred_at> ?d)', 'd',
   'GEO_DISTANCE(distance=10 unit=mile)',
   SDO_RDF_Models('syn_hist'),
   SDO_RDF_Rulebases('RDFS'),
   'd1_military_mod_geo',
   'id', 'shape', '/*+ LEADING(s) FIRST_ROWS */'));

select count(*) from table (spatial_eval('  (<http://lsdis.cs.uga.edu/military/Platoon_12996>
   <http://lsdis.cs.uga.edu/military#trains_at> ?z)
   (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'
   ',
   'l',
   '(?b <http://lsdis.cs.uga.edu/military#trains_at> ?c)
   (?c <http://lsdis.cs.uga.edu/STT#located_at> ?d)', 'd',
   'GEO_DISTANCE(distance=30 unit=mile)',
   SDO_RDF_Models('syn_hist'),
   SDO_RDF_Rulebases('RDFS'),
   'd1_military_mod_geo',
   'id', 'shape', '/*+ LEADING(s) FIRST_ROWS */'));

A.1.3 Queries used for Figure 7.1

SynHist Dataset
temporal filter ... worst case
1 hop
select count(*) from table (temporal_restrict('  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)',
  to_date('1943-11-15 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
A.1. QUERIES USED IN EVALUATION

July 10, 2008

to_date('1943-12-24 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
'DURING',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT', '/** FIRST_ROWS */')));

2 hop
select count(*) from table (temporal_restrict(
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  to_date('1943-11-25 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
  to_date('1943-12-24 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
  'DURING',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT', '/** FIRST_ROWS */')));

3 hop
select count(*) from table (temporal_restrict(
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  to_date('1943-12-12 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
  to_date('1943-12-24 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
  'DURING',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT', '/** FIRST_ROWS */')));

4 hop
select count(*) from table (temporal_restrict(
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?a)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  to_date('1943-12-14 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
  to_date('1943-12-23 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
  'DURING',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT', '/** FIRST_ROWS */')));
5 hop

```sql
select count(*) from table (temporal_restrict('
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (?a <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  to_date('1943-12-15 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
  to_date('1943-12-24 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
  'DURING',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT', /*+ FIRST ROWS */'));
```

6 hop

```sql
select count(*) from table (temporal_restrict('
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (?a <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  (?b <http://lsdis.cs.uga.edu/military#battalion_of> ?c)
  to_date('1943-12-15 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
  to_date('1943-12-24 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
  'DURING',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT', /*+ FIRST ROWS */'));
```

7 hop

```sql
select count(*) from table (temporal_restrict('
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (?a <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  (?b <http://lsdis.cs.uga.edu/military#battalion_of> ?c)
  (?c <http://lsdis.cs.uga.edu/military#trains_at> ?t)
  to_date('1943-12-15 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
  to_date('1943-12-24 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
  'DURING',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT', /*+ FIRST ROWS */'));
```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

SDO_RDF_Rulebases('RDFS'),
'INTERSECT', '/*+ FIRST_ROWS */');

temporal filter ... best case

1 hop

select count(*) from table (temporal_restrict(' (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)' ,
   to_date('1943-11-15 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
   to_date('1943-12-24 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
   'DURING',
   SDO_RDF_Models('syn_hist'),
   SDO_RDF_Rulebases('RDFS'),
   'RANGE', '/*+ FIRST_ROWS */'));

2 hop

select count(*) from table (temporal_restrict(' (?y <http://lsdis.cs.uga.edu/military#used_in> ?z) (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)' ,
   to_date('1943-05-30 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
   to_date('1943-12-24 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
   'DURING',
   SDO_RDF_Models('syn_hist'),
   SDO_RDF_Rulebases('RDFS'),
   'RANGE', '/*+ FIRST_ROWS */'));

3 hop

   to_date('1943-03-15 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
   to_date('1943-12-24 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
   'DURING',
   SDO_RDF_Models('syn_hist'),
   SDO_RDF_Rulebases('RDFS'),
   'RANGE', '/*+ FIRST_ROWS */'));

4 hop

A.1. QUERIES USED IN EVALUATION

July 10, 2008

5 hop
select count(*) from table (temporal_restrict('(?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (?a <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  to_date('1943-01-15 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
  to_date('1944-03-30 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
  'DURING',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'RANGE', '/*+ FIRST_ROWS */')));

6 hop
select count(*) from table (temporal_restrict('(?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (?a <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  (?b <http://lsdis.cs.uga.edu/military#battalion_of> ?c)
  to_date('1943-01-15 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
  to_date('1944-06-24 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
  'DURING',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'RANGE', '/*+ FIRST_ROWS */')));

7 hop
select count(*) from table (temporal_restrict('(?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  to_date('1943-01-15 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
  date('1944-01-15 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
  'DURING',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'RANGE', '/*+ FIRST_ROWS */'));
A.1. QUERIES USED IN EVALUATION

July 10, 2008

(?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)
(?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(?a <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
(?b <http://lsdis.cs.uga.edu/military#battalion_of> ?c)
(?c <http://lsdis.cs.uga.edu/military#trains_at> ?t),
to_date('1943-06-15 00:26:01', 'yyyy-mm-dd hh24:mi:ss'),
to_date('1945-03-24 10:22:00', 'yyyy-mm-dd hh24:mi:ss'),
'DURING',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'RANGE', '/*+ FIRST_ROWS */');

GovTrack Dataset

temporal filter ... worst case

1 hop

select count(*) from table (temporal_restrict(''
   (?x <http://.../politico/forOffice> ?y),'',
to_date('1990-12-31', 'yyyy-mm-dd'),
to_date('1994-01-01', 'yyyy-mm-dd'),
'DURING',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT', '/*+ FIRST_ROWS */'));

2 hop

select count(*) from table (temporal_restrict(''
   (?x <http://.../politico/forOffice> ?y)
   (?y <http://.../politico/represents> ?z),'',
to_date('1990-12-31', 'yyyy-mm-dd'),
to_date('1992-12-25', 'yyyy-mm-dd'),
'DURING',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT', '/*+ FIRST_ROWS */'));

3 hop

select count(*) from table (temporal_restrict(''
   (?p <http://.../politico/hasRole> ?x)
   (?x <http://.../politico/forOffice> ?y)
   (?y <http://.../politico/represents> ?z),'',
to_date('1990-12-31', 'yyyy-mm-dd'),
to_date('1992-12-01', 'yyyy-mm-dd'),
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT', '/*+ FIRST_ROWS */'));

146
4 hop

```sql
select count(*) from table (temporal_restrict(''
   (?p <http://xmlns.com/foaf/0.1/name> ?b)
   (?p <http://xmlns.com/foaf/0.1/name> ?x)
   (?x <http://.../politico/hasRole> ?y)
   (?y <http://.../politico/forOffice> ?y)
   (?y <http://.../politico/represents> ?z)
   to_date('1990-12-31', 'yyyy-mm-dd'),
   to_date('1992-12-01', 'yyyy-mm-dd'),
   'DURING',
   SDO_RDF_Models('gov_track'),
   SDO_RDF_Rulebases('RDFS'),
   'INTERSECT', '/*+ FIRST_ROWS */'));
```

5 hop

```sql
select count(*) from table (temporal_restrict(''
   (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
   (?p <http://xmlns.com/foaf/0.1/name> ?b)
   (?p <http://xmlns.com/foaf/0.1/name> ?x)
   (?x <http://.../politico/hasRole> ?y)
   (?y <http://.../politico/forOffice> ?y)
   (?y <http://.../politico/represents> ?z)
   to_date('1990-12-31', 'yyyy-mm-dd'),
   to_date('1992-12-01', 'yyyy-mm-dd'),
   'DURING',
   SDO_RDF_Models('gov_track'),
   SDO_RDF_Rulebases('RDFS'),
   'INTERSECT', '/*+ FIRST_ROWS */'));
```

6 hop

```sql
select count(*) from table (temporal_restrict(''
   (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
   (?n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f)
   (?p <http://xmlns.com/foaf/0.1/name> ?b)
   (?p <http://.../politico/hasRole> ?x)
   (?x <http://.../politico/forOffice> ?y)
   (?y <http://.../politico/represents> ?z)
   to_date('1990-12-31', 'yyyy-mm-dd'),
   to_date('1992-12-01', 'yyyy-mm-dd'),
   'DURING',
   SDO_RDF_Models('gov_track'),
   SDO_RDF_Rulebases('RDFS'),
   'INTERSECT', '/*+ FIRST_ROWS */'));
```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

7 hop

select count(*) from table (temporal_restrict(''
   (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
   (?n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f)
   (?n <http://www.w3.org/2001/vcard-rdf/3.0#Given> ?g)
   (?p <http://xmlns.com/foaf/0.1/name> ?b)
   (?p <http://.../politico/hasRole> ?x)
   (?x <http://.../politico/forOffice> ?y)
   (?y <http://.../politico/represents> ?z)',
  to_date('1990-12-31', 'yyyy-mm-dd'),
  to_date('1992-12-01', 'yyyy-mm-dd'),
  'DURING',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT', '/*+ FIRST_ROWS */'));

temporal filter ... best case

1 hop

select count(*) from table (temporal_restrict(''
   (?x <http://.../politico/forOffice> ?y)',
  to_date('2006-12-31', 'yyyy-mm-dd'),
  to_date('2030-01-01', 'yyyy-mm-dd'),
  'DURING',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'RANGE', '/*+ FIRST_ROWS */'));

2 hop

select count(*) from table (temporal_restrict(''
   (?x <http://.../politico/forOffice> ?y)
   (?y <http://.../politico/represents> ?z)',
  to_date('2006-12-31', 'yyyy-mm-dd'),
  to_date('2030-01-01', 'yyyy-mm-dd'),
  'DURING',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'RANGE', '/*+ FIRST_ROWS */'));
A.1. QUERIES USED IN EVALUATION

July 10, 2008

3 hop

select count(*) from table (temporal_restrict(''  
  (?p <http://.../politico/hasRole> ?x)  
  (?x <http://.../politico/forOffice> ?y)  
  (?y <http://.../politico/represents> ?z)',  
  to_date('2006-12-31', 'yyyymm-dd'),  
  to_date('2030-01-01', 'yyyymm-dd'),  
  'DURING',  
  SDO_RDF_Models('gov_track'),  
  SDO_RDF_Rulebases('RDFS'),  
  'RANGE', '/*+ FIRST_ROWS */'));

4 hop

select count(*) from table (temporal_restrict(''  
  (?p <http://.../usgovt/party> ?a)  
  (?p <http://xmlns.com/foaf/0.1/name> ?b)  
  (?p <http://.../politico/hasRole> ?x)  
  (?x <http://.../politico/forOffice> ?y)',  
  to_date('1958-08-31', 'yyyymm-dd'),  
  to_date('2030-01-01', 'yyyymm-dd'),  
  'DURING',  
  SDO_RDF_Models('gov_track'),  
  SDO_RDF_Rulebases('RDFS'),  
  'RANGE', '/*+ FIRST_ROWS */'));

5 hop

select count(*) from table (temporal_restrict(''  
  (?p <http://.../usgovt/party> ?a)  
  (?p <http://xmlns.com/foaf/0.1/name> ?b)  
  (?p <http://.../politico/hasRole> ?x)  
  (?x <http://.../politico/forOffice> ?y)  
  (?y <http://.../politico/represents> ?z)',  
  to_date('1958-08-31', 'yyyymm-dd'),  
  to_date('2030-01-01', 'yyyymm-dd'),  
  'DURING',  
  SDO_RDF_Models('gov_track'),  
  SDO_RDF_Rulebases('RDFS'),  
  'RANGE', '/*+ FIRST_ROWS */'));

6 hop
A.1. QUERIES USED IN EVALUATION

select count(*) from table (temporal_restrict('(?p <http://.../usgovt/party> ?a)
  (?p <http://xmlns.com/foaf/0.1/name> ?b)
  (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
  (?p <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  ,
to_date('1958-08-31', 'yyy-mm-dd'),
to_date('2030-01-01', 'yyy-mm-dd'),
'DURING',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'RANGE', '/*+ FIRST_ROWS */')));

7 hop
select count(*) from table (temporal_restrict('(?p <http://.../usgovt/party> ?a)
  (?p <http://xmlns.com/foaf/0.1/name> ?b)
  (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
  (?n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f)
  (?p <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  ,
to_date('1958-08-31', 'yyy-mm-dd'),
to_date('2030-01-01', 'yyy-mm-dd'),
'DURING',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'RANGE', '/*+ FIRST_ROWS */')));

A.1.4 Queries used for Figure 7.2

SynHist Dataset
spatial filter
select count(*) from table (spatial_find('(?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  ,
'1',
'POLYGON((-124.55244 42.991794, -120.4635 42.991794,
  -120.4635 44.271004, -124.55244 44.271004,
  -124.55244 42.991794))', 8265,
'GEO_RELATE(mask=inside)',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id',
'shape', '/*+ LEADING(s) FIRST_ROWS */');

2 hop
select count(*) from table (spatial_find('  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)  
    (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)',
  '1',
  'POLYGON((-124.55244 42.991794, -119.4635 42.991794, 
            -119.4635 44.71004, -124.55244 44.71004, 
            -124.55244 42.991794))', 8265,
  'GEO_RELATE(mask=inside)',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'd5_military_mod_geo',
  'id',
  'shape', '/*+ LEADING(s) FIRST_ROWS */'));

3 hop
select count(*) from table (spatial_find('  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)  
    (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)  
    (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)',
  '1',
  'POLYGON((-124.55244 42.991794, -119.4635 42.991794, 
            -119.4635 44.05004, -124.55244 44.05004, 
            -124.55244 42.991794))', 8265,
  'GEO_RELATE(mask=inside)',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'd5_military_mod_geo',
  'id',
  'shape', '/*+ LEADING(s) FIRST_ROWS */'));

4 hop
select count(*) from table (spatial_find('  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)  
    (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
A.1. QUERIES USED IN EVALUATION

July 10, 2008

5 hop

```
5 hop
select count(*) from table (spatial_find(')
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (?a <http://lsdis.cs.uga.edu/military#platoon_of> ?b)')

5 hop
select count(*) from table (spatial_find(')
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (?a <http://lsdis.cs.uga.edu/military#platoon_of> ?b))
```

6 hop

```
6 hop
select count(*) from table (spatial_find(')
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (?a <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  (?b <http://lsdis.cs.uga.edu/military#battalion_of> ?c)')
```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

'GEO_RELATE(mask=inside)',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
d5_military_mod_geo',
'id',
'shape', '/*+ LEADING(s) FIRST_ROWS */'));

7 hop

select count(*) from table (spatial_find('  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to> ?a)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (?a <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  (?b <http://lsdis.cs.uga.edu/military#battalion_of> ?c)
  (?c <http://lsdis.cs.uga.edu/military#trains_at> ?t)
', 'l',
'POLYGON((-124.55244 42.991794, -119.4635 42.991794,
  -119.4635 43.65004, -124.55244 43.65004,
  -124.55244 42.991794))', 8265,
'GEO_RELATE(mask=inside)',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
d5_military_mod_geo',
'id',
'shape', '/*+ LEADING(s) FIRST_ROWS */'));

GovTrack Dataset

spatial filter

1 hop

select count(*) from table (spatial_find('  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
', 'l',
'POLYGON ((-75.14734 40.884813, -70.77847 40.884813,
  -70.77847 42.3525606, -75.14734 42.3525606,
  -75.14734 40.884813))', 8265,
'GEO_RELATE(mask=inside)',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'gov_all_geo_mod',
'id',
'shape', '/*+ LEADING(s) FIRST_ROWS */'));
A.1. QUERIES USED IN EVALUATION

2 hop

```sql
select count(*) from table (spatial_find(''
    (?y <http://.../politico/represents> ?z)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)',
    'POLYGON ((-75.14734 40.884813, -70.77847 40.884813,
    -70.77847 42.3525606, -75.14734 42.3525606,
    -75.14734 40.884813))', 8265,
    'GEO_RELATE(mask=inside)',
    SDO_RDF_Models('gov_track'),
    SDO_RDF_Rulebases('RDFS'),
    'gov_all_geo_mod',
    'id',
    'shape', '/**+ LEADING(s) FIRST_ROWS */')));
```

3 hop

```sql
select count(*) from table (spatial_find(''
    (?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)',
    'POLYGON ((-75.14734 40.884813, -70.77847 40.884813,
    -70.77847 42.3525606, -75.14734 42.3525606,
    -75.14734 40.884813))', 8265,
    'GEO_RELATE(mask=inside)',
    SDO_RDF_Models('gov_track'),
    SDO_RDF_Rulebases('RDFS'),
    'gov_all_geo_mod',
    'id',
    'shape', '/**+ LEADING(s) FIRST_ROWS */')));
```

4 hop

```sql
select count(*) from table (spatial_find(''
    (?p <http://.../politico/hasRole> ?x)
    (?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)',
    'POLYGON ((-75.14734 40.884813, -70.77847 40.884813,
    -70.77847 42.3525606, -75.14734 42.3525606,
    -75.14734 40.884813))', 8265,
    'GEO_RELATE(mask=inside)',
    '```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

```
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'gov_all_geo_mod',
'id',
'shape', '/*+ LEADING(s) FIRST_ROWS */')));

5 hop
select count(*) from table (spatial_find(''
  (?p <http://xmlns.com/foaf/0.1/name> ?b)
  (?p <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'',
  'l',
  'POLYGON ((-75.14734 40.884813, -70.77847 40.884813,
    -70.77847 42.3525606, -75.14734 42.3525606,
    -75.14734 40.884813))', 8265,
  'GEO_RELATE(mask=inside)' ,
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_all_geo_mod',
  'id',
  'shape', '/*+ LEADING(s) FIRST_ROWS */')));

6 hop
select count(*) from table (spatial_find(''
  (?p <http://xmlns.com/foaf/0.1/name> ?b)
  (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
  (?p <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'',
  'l',
  'POLYGON ((-75.14734 40.884813, -70.77847 40.884813,
    -70.77847 42.3525606, -75.14734 42.3525606,
    -75.14734 40.884813))', 8265,
  'GEO_RELATE(mask=inside)' ,
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_all_geo_mod',
  'id',
  'shape', '/*+ LEADING(s) FIRST_ROWS */')));
```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

7 hop

select count(*) from table (spatial_find(
  (?p <http://xmlns.com/foaf/0.1/name> ?b)
  (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
  (?n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f)
  (?p <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
  'l',
  'POLYGON ((-75.14734 40.884813, -70.77847 40.884813,
    -70.77847 42.3525606, -75.14734 42.3525606,
    -75.14734 40.884813))', 8265,
  'GEO_RELATE(mask=inside)',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_all_geo_mod',
  'id',
  'shape', '/*+ LEADING(s) FIRST_ROWS */')));

A.1.5 Queries used for Figure 7.3

SynHist Dataset

temporal extent

1 hop

select count(*) from table (temporal_extent(
  (?x <http://lsdis.cs.uga.edu/military#assigned_to>
    <http://lsdis.cs.uga.edu/military/Platoon_153649>)
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT'));

2 hop

select count(*) from table (temporal_extent(
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to>
    <http://lsdis.cs.uga.edu/military/Platoon_153649>)
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT'));

156
'INTERSECT'));

3 hop


4 hop


5 hop


6 hop

select count(*) from table (temporal_extent('(?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
A.1. QUERIES USED IN EVALUATION

July 10, 2008

(?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
  <http://lsdis.cs.uga.edu/military/Platoon_153649>)
(?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  
  <http://lsdis.cs.uga.edu/military/Platoon_153649> 
  <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  (?b <http://lsdis.cs.uga.edu/military#battalion_of> ?c),
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT'));

7 hop

select count(*) from table (temporal_extent(' 
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to>
  <http://lsdis.cs.uga.edu/military/Platoon_153649>)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  <http://lsdis.cs.uga.edu/military/Platoon_153649> 
  <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  (?b <http://lsdis.cs.uga.edu/military#battalion_of> ?c)
  (?c <http://lsdis.cs.uga.edu/military#trains_at> ?t)
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT'));

8 hop

select count(*) from table (temporal_extent(' 
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to>
  <http://lsdis.cs.uga.edu/military/Platoon_153649>)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  <http://lsdis.cs.uga.edu/military/Platoon_153649> 
  <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  (?b <http://lsdis.cs.uga.edu/military#battalion_of> ?c)
  (?c <http://lsdis.cs.uga.edu/military#trains_at> ?t)
  (?x <http://lsdis.cs.uga.edu/military#participates_in> ?e)
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT'));
A.1. QUERIES USED IN EVALUATION

9 hop

select count(*) from table (temporal_extent(''
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to>
    <http://lsdis.cs.uga.edu/military/Platoon_153649>)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (<http://lsdis.cs.uga.edu/military/Platoon_153649>
    <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  (?b <http://lsdis.cs.uga.edu/military#battalion_of> ?c)
  (?c <http://lsdis.cs.uga.edu/military#trains_at> ?t)
  (?x <http://lsdis.cs.uga.edu/military#participates_in> ?e)
  (?e <http://lsdis.cs.uga.edu/STT#occurred_at> ?m)'
, SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT')));

10 hop

select count(*) from table (temporal_extent(''
  (?x <http://lsdis.cs.uga.edu/military#on_crew_of> ?y)
  (?y <http://lsdis.cs.uga.edu/military#used_in> ?z)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to>
    <http://lsdis.cs.uga.edu/military/Platoon_153649>)
  (?z <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (<http://lsdis.cs.uga.edu/military/Platoon_153649>
    <http://lsdis.cs.uga.edu/military#platoon_of> ?b)
  (?b <http://lsdis.cs.uga.edu/military#battalion_of> ?c)
  (?c <http://lsdis.cs.uga.edu/military#trains_at> ?t)
  (?x <http://lsdis.cs.uga.edu/military#participates_in> ?e)
  (?e <http://lsdis.cs.uga.edu/STT#occurred_at> ?m)
  (?b <http://lsdis.cs.uga.edu/military#trains_at> ?q)'
, SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT')));

GovTrack Dataset
temporal extent

1 hop

select count(*) from table (temporal_extent(''
  (?y <http://.../politico/forOffice>
    <http://.../usgov/congress/senate/va>)'
, SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT'));
A.1. QUERIES USED IN EVALUATION

July 10, 2008

INTERSECT')});

2 hop

select count(*) from table (temporal_extent('(?p <http://.../politico/hasRole> ?y)
(?y <http://.../politico/forOffice>
<http://.../usgov/congress/senate/va>)',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT')));

3 hop

select count(*) from table (temporal_extent('(?p <http://.../politico/hasRole> ?y)
(?y <http://.../politico/forOffice>
<http://.../usgov/congress/senate/va>)
(http://.../usgov/congress/senate/va>
<http://purl.org/dc/terms/isPartOf> ?x)',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT')));

4 hop

select count(*) from table (temporal_extent('(?p <http://.../politico/hasRole> ?y)
(?y <http://.../politico/forOffice>
<http://.../usgov/congress/senate/va>)
(http://.../usgov/congress/senate/va>
<http://purl.org/dc/terms/isPartOf> ?x)
(?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT')));

5 hop

select count(*) from table (temporal_extent('(?p <http://.../politico/hasRole> ?y)
(?y <http://.../politico/forOffice>
<http://.../usgov/congress/senate/va>)
(http://.../usgov/congress/senate/va>
<http://purl.org/dc/terms/isPartOf> ?x)
A.1. QUERIES USED IN EVALUATION

July 10, 2008

6 hop

select count(*) from table (temporal_extent(
(??p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
(??n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f)
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT'));

7 hop

select count(*) from table (temporal_extent(
(??p <http://.../politico/hasRole> ?y)
(??y <http://.../politico/forOffice>
<http://.../usgov/congress/senate/va>)
(??p <http://.../usgov/congress/senate/va>
<http://purl.org/dc/terms/isPartOf> ?x)
(??p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
(??n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f)
(??n <http://www.w3.org/2001/vcard-rdf/3.0#Given> ?g)
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT'));

8 hop

select count(*) from table (temporal_extent(
(??p <http://.../politico/hasRole> ?y)
(??y <http://.../politico/forOffice>
<http://.../usgov/congress/senate/va>)
(??p <http://.../usgov/congress/senate/va>
<http://purl.org/dc/terms/isPartOf> ?x)
A.1. QUERIES USED IN EVALUATION

July 10, 2008

(?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
(?n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f)
(?n <http://www.w3.org/2001/vcard-rdf/3.0#Given> ?g)
(?p <http://xmlns.com/foaf/0.1/name> ?q)
(?p <http://xmlns.com/foaf/0.1/gender> ?r)
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT'));

9 hop

select count(*) from table (temporalExtent('
  (?p <http://.../politico/hasRole> ?y)
  (?y <http://.../politico/forOffice>
    <http://.../usgov/congress/senate/va>)
  (http://.../usgov/congress/senate/va
    <http://purl.org/dc/terms/isPartOf> ?x)
  (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
  (?n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f)
  (?n <http://www.w3.org/2001/vcard-rdf/3.0#Given> ?g)
  (?p <http://xmlns.com/foaf/0.1/name> ?q)
  (?p <http://xmlns.com/foaf/0.1/gender> ?r)
  (http://.../usgov/congress/senate/va
    <http://.../politico/represents> ?t)
  ',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT')));

10 hop

select count(*) from table (temporalExtent('
  (?p <http://.../politico/hasRole> ?y)
  (?y <http://.../politico/forOffice>
    <http://.../usgov/congress/senate/va>)
  (http://.../usgov/congress/senate/va
    <http://purl.org/dc/terms/isPartOf> ?x)
  (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
  (?n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f)
  (?n <http://www.w3.org/2001/vcard-rdf/3.0#Given> ?g)
  (?p <http://xmlns.com/foaf/0.1/name> ?q)
  (?p <http://xmlns.com/foaf/0.1/gender> ?r)
  (http://.../usgov/congress/senate/va
    <http://.../politico/represents> ?t)
  (http://.../usgov/congress/senate/va
    <http://www.w3.org/2000/01/rdf-schema#label> ?u)' ,
A.1. QUERIES USED IN EVALUATION

SDO_RDF_Models(‘gov_track’),
SDO_RDF_Rulebases(‘RDFS’),
‘INTERSECT’));

A.1.6 Queries used for Figure 7.4

SynHist Dataset
spatial extent
2 hop
select count(*) from table (spatial_extent(''
  (<http://lsdis.cs.uga.edu/military/Soldier_3013>
  <http://lsdis.cs.uga.edu/military#participates_in> ?y)
  (?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)'', 'l' ,
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'd5_military_mod_geo',
  'id' ));

3 hop
select count(*) from table (spatial_extent(''
  (?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to>
  <http://lsdis.cs.uga.edu/military/Platoon_12918>)
  (?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)'', 'l' ,
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'd5_military_mod_geo',
  'id' ));

4 hop
select count(*) from table (spatial_extent(''
  (?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to>
  <http://lsdis.cs.uga.edu/military/Platoon_12918>)
  (?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (<http://lsdis.cs.uga.edu/military/Platoon_12918>
  <http://lsdis.cs.uga.edu/military#platoon_of>
  <http://lsdis.cs.uga.edu/military/Battalion_12814>)'', 'l' ,
  SDO_RDF_Models('syn_hist'),
  'd5_military_mod_geo',
  'id' ));
A.1. QUERIES USED IN EVALUATION

July 10, 2008

SDO_RDF_Rulebases('RDFS'),
'd5_military_mod.geo',
'id')));

5 hop

select count(*) from table (spatial_extent('(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
'http://lsdis.cs.uga.edu/military/Platoon_12918>')
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
('<http://lsdis.cs.uga.edu/military/Platoon_12918>
'http://lsdis.cs.uga.edu/military#platoon_of'
'http://lsdis.cs.uga.edu/military/Battalion_12814>')
(?z <http://lsdis.cs.uga.edu/military#leader_of>
'http://lsdis.cs.uga.edu/military/Platoon_12918>')', 'l',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod.geo',
'id')));

6 hop

select count(*) from table (spatial_extent('(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
'http://lsdis.cs.uga.edu/military/Platoon_12918>')
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
('<http://lsdis.cs.uga.edu/military/Platoon_12918>
'http://lsdis.cs.uga.edu/military#platoon_of'
'http://lsdis.cs.uga.edu/military/Battalion_12814>')
(?z <http://lsdis.cs.uga.edu/military#leader_of>
'http://lsdis.cs.uga.edu/military/Platoon_12918>')
('<http://lsdis.cs.uga.edu/military/Battalion_12814>
'http://lsdis.cs.uga.edu/military#trains_at> ?t')', 'l',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod.geo',
'id')));

7 hop

select count(*) from table (spatial_extent('(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
A.1. QUERIES USED IN EVALUATION

July 10, 2008

<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918)
<http://lsdis.cs.uga.edu/military#platoon_of>
(http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?m <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(http://lsdis.cs.uga.edu/military/Battalion_12814>
<http://lsdis.cs.uga.edu/military#trains_at> ?t), 'l',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id'));

8 hop

select count(*) from table (spatial_extent(''
(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#platoon_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?m <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(http://lsdis.cs.uga.edu/military/Battalion_12814>
<http://lsdis.cs.uga.edu/military#trains_at> ?t)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#trains_at> ?n), 'l',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id'));

9 hop

select count(*) from table (spatial_extent(''
(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
A.1. QUERIES USED IN EVALUATION

July 10, 2008

(?
y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(<http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#platoon_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?
z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?
m <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(<http://lsdis.cs.uga.edu/military/Battalion_12814>
<http://lsdis.cs.uga.edu/military#trains_at> ?t)
(<http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#trains_at> ?n)
(?t <http://lsdis.cs.uga.edu/STT#located_at> ?q)'', 'l',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id')));

10 hop

select count(*) from table (spatial_extent('
  (?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
  (?x <http://lsdis.cs.uga.edu/military#assigned_to>
  <http://lsdis.cs.uga.edu/military/Platoon_12918>)
  (?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
  (<http://lsdis.cs.uga.edu/military/Platoon_12918>
  <http://lsdis.cs.uga.edu/military#platoon_of>
  <http://lsdis.cs.uga.edu/military/Battalion_12814>)
  (?z <http://lsdis.cs.uga.edu/military#leader_of>
  <http://lsdis.cs.uga.edu/military/Platoon_12918>)
  (?m <http://lsdis.cs.uga.edu/military#leader_of>
  <http://lsdis.cs.uga.edu/military/Battalion_12814>)
  (<http://lsdis.cs.uga.edu/military/Battalion_12814>
  <http://lsdis.cs.uga.edu/military#trains_at> ?t)
  (<http://lsdis.cs.uga.edu/military/Platoon_12918>
  <http://lsdis.cs.uga.edu/military#trains_at> ?n)
  (?t <http://lsdis.cs.uga.edu/STT#located_at> ?q)
  (?n <http://lsdis.cs.uga.edu/STT#located_at> ?r)'', 'l',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id')));

Gov Track Dataset

166
A.1. QUERIES USED IN EVALUATION

spatial extent
2 hop
select count(*) from table (spatial_extent('(?z <http://.../usgovt/congress> "106")
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'),
'1',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'gov_all_geo_mod',
'id'));

3 hop
select count(*) from table (spatial_extent('(?y <http://.../politico/represents> ?z)
  (?y <http://.../usgovt/congress> "106")
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'),
'1',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'gov_all_geo_mod',
'id'));

4 hop
select count(*) from table (spatial_extent('(?y <http://.../politico/represents> ?z)
  (?y <http://purl.org/dc/terms/isPartOf> ?a)
  (?z <http://.../usgovt/congress> "106")
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'),
'1',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'gov_all_geo_mod',
'id'));

5 hop
select count(*) from table (spatial_extent('(?y <http://.../politico/represents> ?z)
  (?y <http://purl.org/dc/terms/isPartOf> ?a)
  (?z <http://.../usgovt/congress> "106")
  (?z <http://.../usgovt/number> ?b)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'),
'1',
'id'));
6 hop

```
select count(*) from table (spatial_extent('(?y <http://.../politico/represents> ?z)
  (?y <http://purl.org/dc/terms/isPartOf> ?a)
  (?z <http://.../usgovt/congress> "106")
  (?z <http://.../usgovt/number> ?b)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
  (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)'
, 'l',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_all_geo_mod',
  'id')));
```

7 hop

```
select count(*) from table (spatial_extent('(?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  (?y <http://purl.org/dc/terms/isPartOf> ?a)
  (?z <http://.../usgovt/congress> "106")
  (?z <http://.../usgovt/number> ?b)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
  (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)'
, 'l',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_all_geo_mod',
  'id')));
```

8 hop

```
select count(*) from table (spatial_extent('(?p <http://.../politico/hasRole> ?x)
  (?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  (?y <http://purl.org/dc/terms/isPartOf> ?a)
  (?z <http://.../usgovt/congress> "106")
  (?z <http://.../usgovt/number> ?b)
```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

9 hop
select count(*) from table (spatial_extent(' 
  (?p <http://.../politico/hasRole> ?x) 
  (?x <http://.../politico/forOffice> ?y) 
  (?y <http://.../politico/represents> ?z) 
  (?y <http://purl.org/dc/terms/isPartOf> ?a) 
  (?z <http://.../usgovt/congress> "106") 
  (?z <http://.../usgovt/number> ?b) 
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l) 
  (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m) 
  (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)') 
', 
'1', 
SDO_RDF_Models('gov_track'), 
SDO_RDF_Rulebases('RDFS'), 
'gov_all_geo_mod', 
'id')));

10 hop
select count(*) from table (spatial_extent(' 
  (?p <http://.../politico/hasRole> ?x) 
  (?x <http://.../politico/forOffice> ?y) 
  (?y <http://.../politico/represents> ?z) 
  (?y <http://purl.org/dc/terms/isPartOf> ?a) 
  (?z <http://.../usgovt/congress> "106") 
  (?z <http://.../usgovt/number> ?b) 
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l) 
  (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m) 
  (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n) 
  (?n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f)') 
', 
'1', 
SDO_RDF_Models('gov_track'), 
SDO_RDF_Rulebases('RDFS'), 
'gov_all_geo_mod', 
'id')));
A.1.7 Queries used for Table 7.5

GovTrack Dataset
spatial filter plus temporal filter

3 hop
select count(*) from
  table (spatial_find(''
    (?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'
    , 'l',
    'POLYGON ((-75.14734 40.884813, -70.77847 40.884813,
    -70.77847 42.3525606, -75.14734 42.3525606,
    -75.14734 40.884813))', 8265,
    'GEO_RELATE(mask=inside)'',
    SDO_RDF_Models('gov_track'),
    SDO_RDF_Rulebases('RDFS'),
    'gov_small_geo_mod',
    'id',
    'shape', '/** LEADING(S) FIRST_ROWS */')) s,
  table (temporal_restrict(''
    (?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'
    , to_date('1993-01-01', 'yyyy-mm-dd'),
    to_date('2000-09-21', 'yyyy-mm-dd'),
    'DURING',
    SDO_RDF_Models('gov_track'),
    SDO_RDF_Rulebases('RDFS'),
    'INTERSECT', '/** FIRST_ROWS */')) t
where s.x = t.x and s.y = t.y and s.z = t.z and s.l = t.l;

5 hop
select count(*) from
  table (spatial_find(''
    (?p <http://.../politico/hasRole> ?x)
    (?p <http://.../usgovt/party> ?a)
    (?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'
    , ?a),
    'POLYGON ((-75.14734 40.884813, -70.77847 40.884813,
    -70.77847 42.3525606, -75.14734 42.3525606,
    -75.14734 40.884813))', 170,
    'GEO_RELATE(mask=inside)'',
    SDO_RDF_Models('gov_track'),
    SDO_RDF_Rulebases('RDFS'),
    'gov_small_geo_mod',
    'id',
    'shape', '/** LEADING(S) FIRST_ROWS */')) s,
A.1. QUERIES USED IN EVALUATION

July 10, 2008

A.1.8 Queries used for Table 7.6

SynHist Dataset

spatial filter plus temporal filter

select count(*) from table (spatial_find(''
  (p <http://lsdis.cs.uga.edu/military#assigned_to> ?z)
  (?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
  (?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)''
  'l',
  'POLYGON((-122.84501 42.240328, -122.8075 42.240328,
    -122.8075 42.3764, -122.84501 42.3764,
    -122.84501 42.240328))'', 8265,
  'GEO_RELATE(mask=overlapbdyintersect)',
  SDO_RDF_Models('syn_hist'),
  SDO_RDF_Rulebases('RDFS'),
  'd1_military_mod_geo',
  'id',
  '/** LEADING(S) FIRST_ROWS */')) s,
'shape', '/*+ LEADING(s) FIRST_ROWS */)) s,

5 hop

where s.x = t.x and s.z = t.z and s.y = t.y and s.l = t.l;
A.1.9 Queries used for Figure 7.5

GovTrack Dataset

2 hop

```sql
select count(*) from table (spatial_extent(' (?z <http://.../usgovt/congress> "106") (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'
, 'l',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'gov_all_geo_mod',
'id')) s,
table (temporal_extent(' (?z <http://.../usgovt/congress> "106") (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'
, 'l',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT')) t
where s.z = t.z and s.l = t.l;
```

3 hop

```sql
select count(*) from table (spatial_extent(' (?y <http://.../politico/represents> ?z) (?z <http://.../usgovt/congress> "106") (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'
, 'l',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'gov_all_geo_mod',
'id')) s,
table (temporal_extent(' (?y <http://.../politico/represents> ?z) (?z <http://.../usgovt/congress> "106") (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'
, 'l',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT')) t
where s.y = t.y and s.z = t.z and s.l = t.l;
```

4 hop

```sql
173
```
select count(*) from table (spatial_extent('(?y <http://.../politico/represents> ?z)
  (?y <http://purl.org/dc/terms/isPartOf> ?a)
  (?z <http://.../usgovt/congress> "106")
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'),
'1',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'gov_all_geo_mod',
'id')) s,
  table (temporal_extent('(?y <http://.../politico/represents> ?z)
  (?y <http://purl.org/dc/terms/isPartOf> ?a)
  (?z <http://.../usgovt/congress> "106")
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'),
'1',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT')) t
where s.y = t.y and s.a = t.a and s.z = t.z and s.l = t.l;

5 hop

select count(*) from table (spatial_extent('(?y <http://.../politico/represents> ?z)
  (?y <http://purl.org/dc/terms/isPartOf> ?a)
  (?z <http://.../usgovt/congress> "106")
  (?z <http://.../usgovt/number> ?b)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'),
'1',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'gov_all_geo_mod',
'id')) s,
  table (temporal_extent('(?y <http://.../politico/represents> ?z)
  (?y <http://purl.org/dc/terms/isPartOf> ?a)
  (?z <http://.../usgovt/congress> "106")
  (?z <http://.../usgovt/number> ?b)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)'),
'1',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT')) t
where s.y = t.y and s.z = t.z and s.a = t.a and s.b = t.b and s.l = t.l;
6 hop

```sql
select count(*) from table (spatial_extent('(?y <http://.../politico/represents> ?z)
  (?y <http://purl.org/dc/terms/isPartOf> ?a)
  (?z <http://.../usgovt/congress> "106")
  (?z <http://.../usgovt/number> ?b)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
  (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)'
  'l',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_all_geo_mod',
  'id')) s,
  table (temporal_extent('(?y <http://.../politico/represents> ?z)
    (?y <http://purl.org/dc/terms/isPartOf> ?a)
    (?z <http://.../usgovt/congress> "106")
    (?z <http://.../usgovt/number> ?b)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
    (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)'
  'l',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT')) t
where s.y = t.y and s.z = t.z and s.a = t.a and s.b = t.b and
s.l = t.l and s.m = t.m;
```

7 hop

```sql
select count(*) from table (spatial_extent('(?x <http://.../politico/forOffice> ?y)
  (?y <http://.../politico/represents> ?z)
  (?y <http://purl.org/dc/terms/isPartOf> ?a)
  (?z <http://.../usgovt/congress> "106")
  (?z <http://.../usgovt/number> ?b)
  (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
  (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)'
  'l',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'gov_all_geo_mod',
  'id')) s,
  table (temporal_extent('(?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?y <http://purl.org/dc/terms/isPartOf> ?a)
    (?z <http://.../usgovt/congress> "106")
    (?z <http://.../usgovt/number> ?b)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
    (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)'
  'l',
  SDO_RDF_Models('gov_track'),
  SDO_RDF_Rulebases('RDFS'),
  'INTERSECT')) t
where s.x = t.x and s.y = t.y and s.z = t.z and s.a = t.a and s.b = t.b and
s.l = t.l and s.m = t.m;
```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

(?y <http://purl.org/dc/terms/isPartOf> ?a)
(?z <http://.../usgovt/congress> "106")
(?z <http://.../usgovt/number> ?b)
(?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
(?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)

SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT')) t
where s.x = t.x and s.y = t.y and s.z = t.z and s.a = t.a and
s.b = t.b and s.l = t.l and s.m = t.m;

8 hop

select count(*) from table (spatial_extent(''
    (?p <http://.../politico/hasRole> ?x)
    (?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?y <http://purl.org/dc/terms/isPartOf> ?a)
    (?z <http://.../usgovt/congress> "106")
    (?z <http://.../usgovt/number> ?b)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
    (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)
',
'1',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'gov_all_geo_mod',
'id')) s,
table (temporal_extent(''
    (?p <http://.../politico/hasRole> ?x)
    (?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?y <http://purl.org/dc/terms/isPartOf> ?a)
    (?z <http://.../usgovt/congress> "106")
    (?z <http://.../usgovt/number> ?b)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
    (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)
',
SDO_RDF_Models('gov_track'),
SDO_RDF_Rulebases('RDFS'),
'INTERSECT')) t
where s.p = t.p and s.x = t.x and s.y = t.y and s.z = t.z and
s.a = t.a and s.b = t.b and s.l = t.l and s.m = t.m;

9 hop

select count(*) from table (spatial_extent(''

176
A.1. QUERIES USED IN EVALUATION

10 hop

select count(*) from table (spatialExtent(''
    (?p <http://.../politico/hasRole> ?x)
    (?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?y <http://purl.org/dc/terms/isPartOf> ?a)
    (?z <http://.../usgovt/congress> "106")
    (?z <http://.../usgovt/number> ?b)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
    (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)
    (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)'
    ,
    'l',
    SDO_RDF_Models('gov_track'),
    SDO_RDF_Rulebases('RDFS'),
    'gov_all_geo_mod',
    'id')) s,

table (temporalExtent(''
    (?p <http://.../politico/hasRole> ?x)
    (?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?y <http://purl.org/dc/terms/isPartOf> ?a)
    (?z <http://.../usgovt/congress> "106")
    (?z <http://.../usgovt/number> ?b)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
    (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)
    (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)'
    ,
    SDO_RDF_Models('gov_track'),
    SDO_RDF_Rulebases('RDFS'),
    'INTERSECT')) t

where s.p = t.p and s.x = t.x and s.y = t.y and s.z = t.z and s.a = t.a and s.b = t.b and s.l = t.l and s.m = t.m and s.n = t.n;

10 hop

select count(*) from table (spatialExtent(''
    (?p <http://.../politico/hasRole> ?x)
    (?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?y <http://purl.org/dc/terms/isPartOf> ?a)
    (?z <http://.../usgovt/congress> "106")
    (?z <http://.../usgovt/number> ?b)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
    (?z <http://www.w3.org/2000/01/rdf-schema#label> ?m)
    (?p <http://www.w3.org/2001/vcard-rdf/3.0#N> ?n)
    (?n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f)'
    ,
    'l',
    SDO_RDF_Models('gov_track'),
    SDO_RDF_Rulebases('RDFS'),
    'gov_all_geo_mod',
    'id')) s,
A.1. QUERIES USED IN EVALUATION

July 10, 2008

SDO_RDF_Models('gov.track'),
SDO_RDF_Rulebases('RDFS'),
'gov.all.geo_mod',
'id')) s,

```
table (temporal_extent(''
    (?p <http://.../politico/hasRole> ?x)
    (?x <http://.../politico/forOffice> ?y)
    (?y <http://.../politico/represents> ?z)
    (?y <http://purl.org/dc/terms/isPartOf> ?a)
    (?z <http://.../usgovt/congress> "106")
    (?z <http://.../usgovt/number> ?b)
    (?z <http://lsdis.cs.uga.edu/STT#located_at> ?l)
    (?p <http://www.w3.org/2000/01/rdf-schema#label> ?m)
    (?n <http://www.w3.org/2001/vcard-rdf/3.0#Family> ?f),'
    SDO_RDF_Models('gov.track'),
    SDO_RDF_Rulebases('RDFS'),
    'INTERSECT')) t
```

where s.p = t.p and s.x = t.x and s.y = t.y and s.z = t.z and
s.a = t.a and s.b = t.b and s.l = t.l and s.m = t.m and s.n =
t.n and s.f = t.f;

SynHist Dataset

2 hop

```
select count(*) from table (spatial_extent(''
    (<http://lsdis.cs.uga.edu/military/Soldier_3013>
     <http://lsdis.cs.uga.edu/military#participates_in> ?y)
    (?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l'), 'l',
    SDO_RDF_Models('syn_hist'),
    SDO_RDF_Rulebases('RDFS'),
    'd5_military_mod_geo',
    'id')) s,
```

```
table (temporal_extent(''
    (<http://lsdis.cs.uga.edu/military/Soldier_3013>
     <http://lsdis.cs.uga.edu/military#participates_in> ?y)
    (?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l'), 'l',
    SDO_RDF_Models('syn_hist'),
    SDO_RDF_Rulebases('RDFS'),
    'RANGE')) t
```

where s.y = t.y and s.l = t.l;

3 hop

```
select count(*) from table (spatial_extent(''
```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
 <http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)'', 'l',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id')) s,
table (temporal_extent('(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
 <http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)'',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'RANGE')) t
where s.x = t.x and s.y = t.y and s.l = t.l;

4 hop
select count(*) from table (spatial_extent('(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
 <http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
 http://lsdis.cs.uga.edu/military#platoon_of>
 <http://lsdis.cs.uga.edu/military/Battalion_12814>)', 'l',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id')) s,
table (temporal_extent('(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
 <http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
 http://lsdis.cs.uga.edu/military#platoon_of>
 <http://lsdis.cs.uga.edu/military/Battalion_12814>)',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'RANGE')) t
where s.x = t.x and s.y = t.y and s.l = t.l;
5 hop

```sql
select count(*) from table (spatial_extent('(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
<http://lsdis.cs.uga.edu/military/Platoon_12918>))
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#platoon_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>'), 'l',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id')) s,
table (temporal_extent('(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
<http://lsdis.cs.uga.edu/military/Platoon_12918>))
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#platoon_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>'),
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'RANGE')) t
where s.x = t.x and s.y = t.y and s.l = t.l and s.z = t.z;
```

6 hop

```sql
select count(*) from table (spatial_extent('(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
<http://lsdis.cs.uga.edu/military/Platoon_12918>))
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#platoon_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>))
(http://lsdis.cs.uga.edu/military/Battalion_12814>
<http://lsdis.cs.uga.edu/military#trains_at> ?t'), 'l',
```
A.1. QUERIES USED IN EVALUATION

July 10, 2008

SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id')) s,
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'RANGE')) t where s.x = t.x and s.y = t.y and s.l = t.l and s.z = t.z and s.t = t.t;

7 hop

SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id')) s,
A.1. QUERIES USED IN EVALUATION

<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#platoon_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?m <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(http://lsdis.cs.uga.edu/military/Battalion_12814>
<http://lsdis.cs.uga.edu/military#trains_at> ?t)'

8 hop

select count(*) from table (spatial_extent('?
(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#platoon_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?m <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(http://lsdis.cs.uga.edu/military/Battalion_12814>
<http://lsdis.cs.uga.edu/military#trains_at> ?t)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#trains_at> ?n)', 'l'

SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'RANGE')) t
where s.x = t.x and s.y = t.y and s.l = t.l and s.z = t.z and
s.t = t.t and s.m = t.m;

8 hop

select count(*) from table (spatial_extent('?
(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#platoon_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?m <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(http://lsdis.cs.uga.edu/military/Battalion_12814>
<http://lsdis.cs.uga.edu/military#trains_at> ?t)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#trains_at> ?n)', 'l'

SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id')) s,
table (temporal_extent('?
(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
A.1. QUERIES USED IN EVALUATION

July 10, 2008

(<http://lsdis.cs.uga.edu/military/Platoon_12918> <http://lsdis.cs.uga.edu/military#trains_at> ?n),
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'RANGE')) t
where s.x = t.x and s.y = t.y and s.l = t.l and s.z = t.z and
s.t = t.t and s.m = t.m and s.n = t.n;

9 hop

select count(*) from table (spatial_extent('
(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to> <http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(<http://lsdis.cs.uga.edu/military/Platoon_12918> <http://lsdis.cs.uga.edu/military#trains_at> ?n)
(?t <http://lsdis.cs.uga.edu/STT#located_at> ?q)', 'l',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id')) s,
table (temporal_extent('
(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to> <http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(<http://lsdis.cs.uga.edu/military/Platoon_12918>
A.1. QUERIES USED IN EVALUATION

July 10, 2008

<http://lsdis.cs.uga.edu/military#platoon_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?m <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(http://lsdis.cs.uga.edu/military/Battalion_12814>
 http://lsdis.cs.uga.edu/military#trains_at> ?t)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
 http://lsdis.cs.uga.edu/military#trains_at> ?n)
(?t <http://lsdis.cs.uga.edu/military/STT#located_at> ?q)

SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'RANGE')) t
where s.x = t.x and s.y = t.y and s.l = t.l and s.z = t.z and
s.t = t.t and s.m = t.m and s.n = t.n and s.q = t.q;

10 hop
select count(*) from table (spatial
extent(''
(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
 http://lsdis.cs.uga.edu/military#platoon_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?m <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(http://lsdis.cs.uga.edu/military/Battalion_12814>
 http://lsdis.cs.uga.edu/military#trains_at> ?t)
(http://lsdis.cs.uga.edu/military/Platoon_12918>
 http://lsdis.cs.uga.edu/military#trains_at> ?n)
(?t <http://lsdis.cs.uga.edu/military/STT#located_at> ?q)
(?n <http://lsdis.cs.uga.edu/STT#located_at> ?r)'', 'l',
SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'd5_military_mod_geo',
'id')) s,
table (temporal
extent(''
(?x <http://lsdis.cs.uga.edu/military#participates_in> ?y)
(?x <http://lsdis.cs.uga.edu/military#assigned_to>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)

184
A.1. QUERIES USED IN EVALUATION

(?y <http://lsdis.cs.uga.edu/STT#occurred_at> ?l)
(<http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#platoon_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(?z <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Platoon_12918>)
(?m <http://lsdis.cs.uga.edu/military#leader_of>
<http://lsdis.cs.uga.edu/military/Battalion_12814>)
(<http://lsdis.cs.uga.edu/military/Battalion_12814>
<http://lsdis.cs.uga.edu/military#trains_at> ?t)
(<http://lsdis.cs.uga.edu/military/Platoon_12918>
<http://lsdis.cs.uga.edu/military#trains_at> ?n)
(?t <http://lsdis.cs.uga.edu/STT#located_at> ?q)
(?n <http://lsdis.cs.uga.edu/STT#located_at> ?r)

SDO_RDF_Models('syn_hist'),
SDO_RDF_Rulebases('RDFS'),
'RANGE')) t
where s.x = t.x and s.y = t.y and s.l = t.l and s.z = t.z and
s.t = t.t and s.m = t.m and s.n = t.n and s.q = t.q and s.r =
t.r;