SPARQL-ST: Extending SPARQL to Support Spatiotemporal Queries

Matthew Perry and Prateek Jain and Amit P.Sheth

Abstract Spatial and temporal data is plentiful on the Web, and Semantic Web technologies have the potential to make this data more accessible and more useful. Semantic Web researchers have consequently made progress towards better handling of spatial and temporal data.SPARQL, the W3C-recommended query language for RDF, does not adequately support complex spatial and temporal queries. In this work, we present the SPARQL-ST query language. SPARQL-ST is an extension of SPARQL for complex spatiotemporal queries. We present a formal syntax and semantics for SPARQL-ST. In addition, we describe a prototype implementation of SPARQL-ST and demonstrate the scalability of this implementation with a performance study using large real-world and synthetic RDF datasets.

1 Introduction

Nearly all human activity is rooted in space and time, and increasing amounts of spatial and temporal data are appearing on the Web. Examples include spatial and temporal data about tracking hurricanes and aquatic animals[1] [2]. We have also seen increasing amounts of user-generated geospatial metadata created with geotagging vocabularies such as GeoRSS. The number of Web mashups created with public map services alone is a testament to the usefulness of maps and spatial data in a variety of applications. These real-world scenarios motivate us to argue that current

Matthew Perry
Oracle,1 Oracle Drive, Nashua, NH 03062, USA e-mail: matthew.perry@oracle.com

Prateek Jain
Kno.e.sis Center, Wright State University, Dayton, OH 45435 USA e-mail: prateek@knoesis.org

Amit P. Sheth
Kno.e.sis Center, Wright State University, Dayton, OH 45435 USA e-mail: amit@knoesis.org

[1] http://weather.unisys.com/hurricane/index.html

[2] http://whale.wheelock.edu/whalenet-stuff/stop_#cover.html

tools for managing Semantic Web data must be extended to better handle spatial and temporal data.

Researchers have made initial progress in this direction. Gutierrez, et al. proposed Temporal RDF Graphs to model temporal aspects of RDF triples [8, 9]. The RDF statement is extended in this model from a triple to a quad where the fourth element is the valid time of the RDF statement. There has also been significant research inspired by the Geospatial Semantic Web vision [6]. An architecture of ontologies for the Geospatial Semantic Web has been proposed [14], and a variety of tools and systems to manage spatial data on the Semantic Web have been introduced [15, 22, 27]. In addition, groups such as the W3C Geospatial Incubator Group [17] have pursued standard ontologies for geospatial data.

Query language support for spatial and temporal RDF data is currently lacking. SPARQL [23] has recently emerged as the W3C-recommended query language for RDF data, but, to date, no extensions of SPARQL to support complex spatial and temporal queries exist. This chapter proposes SPARQL-ST, an extension of SPARQL that supports queries over spatiotemporal RDF graphs (i.e. temporal RDF Graphs that contain spatial objects). Consider the SPARQL-ST query below. This query selects all politicians (and their tenure) that represent a congressional district that is inside a given polygon.

```
SELECT ?p, %g, intersect(#t1, #t2, #t3, #t4)
        WHERE {
        ?p usgov:hasRole ?r #t1 .
        ?r usgov:forOffice ?o #t2 .
        ?o usgov:represents ?c #t3 .
        ?c stt:located_at %g #t4 .
        SPATIAL FILTER (inside(%g, GEOM(POLYGON ((-75.14 40.88, -70.77 40.88,
         -70.77 42.35, -75.14 42.35,-75.14 42.35, -75.14 40.88))))}
```

In addition to normal SPARQL variables (denoted with a ? prefix), SPARQL-ST introduces a spatial variable type (denoted with a % prefix) and a temporal variable type (denoted with a # prefix). Spatial variables represent complex spatial features rather than a single URI, and the concept of a mapping is extended so that spatial variables map to a set of triples that represent a spatial feature. The spatial variable *%g* is used in the query above to represent the spatial extent of a congressional district. Temporal variables map to time intervals rather than a URI and can appear in the quad position of what we term a spatiotemporal triple pattern. Temporal variables are used in the example query to retrieve the valid time of each temporal RDF statement. In addition, SPARQL-ST allows computation of derived time intervals. For example, the query above computes the interval intersection of four time intervals to derive the valid time of the entire triple pattern. SPARQL-ST also introduces *SPATIALFILTER* and *TEMPORALFILTER* expressions to filter results using spatial and temporal conditions. The query above applies a filtering condition to the spatial extent of each congressional district.

With the objective of providing better support for spatiotemporal queries over Semantic Web data, this work makes the following contributions:

1. a formal syntax and semantics for the SPARQL-ST query language,

2. a prototype implementation of SPARQL-ST built on top of a relational database management system, and
3. a performance evaluation of the prototype system using both synthetic and real-world RDF datasets.

The remainder of the chapter is organized as follows. Section 2 presents the RDF data model and approaches for modeling spatial and temporal data in RDF. Section 3 introduces the SPARQL-ST query language by defining its formal syntax and semantics. Section 4 describes our prototype implementation of SPARQL-ST, and Section 5 evaluates the scalability of our prototype using synthetic and real-world RDF datasets. Related work is discussed in Section 6. Finally, Section 7 gives conclusions and discusses directions for future work.

2 Modeling Approach

We give details of our approach for modeling spatial and temporal data using RDF in this section. We incorporate temporal information using Temporal RDF Graphs [8, 9], and we present an ontology based on the Open Geospatial Consortium (OGC) Geographic Modeling Language (GML) specification to model spatial features. Temporal RDF triples are encoded using standard RDF reification. Our formal definition of SPARQL-ST depends on this modeling approach, so we present the specifics of our modeling approach (first described in [21]) as a prerequisite to our SPARQL-ST definition in the next section. We first formally define the RDF model and Temporal RDF graphs and then present our ontology for spatial features. Although SPARQL-ST currently depends on a particular serialization of temporal RDF and spatial ontology, the concepts of SPARQL-ST are equally applicable to other temporal RDF serializations and other spatial ontologies.

2.1 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 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. RDF Schema (RDFS) [4] provides a standard vocabulary for describing the classes and relationships used in RDF graphs and consequently provides the capability to define ontologies.

A set of entailment rules are also defined for RDF and RDFS [10]. 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.

2.2 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. [8, 9] introduced the notion of temporal RDF graphs for this purpose.

Temporal RDF graphs model linear, discrete, absolute time and are defined as follows [8]. 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.

Let us consider the politician Bill Clinton who was governor of Arkansas from 11 January 1983 until 12 December 1992 and president of the United States from 20 January 1993 until 20 January 2001. This would yield the following triples: (*Bill_Clinton, holds_ office, AR_Governor*) : [01:11:1983, 12:12:1992], (*Bill_Clinton, holds_office, US_President*) : [01:20:1993, 01:20:2001].

We must also account for the effects of temporal labels on RDFS inferencing rules. 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*) : $[1,4] \wedge$ (*y, rdfs:subClassOf, z*) : $[3,5] \Rightarrow$ (*x, rdfs:subClassOf, z*) : $[3,4]$).

We use RDF reification 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 [12] to model the time intervals, and a new property *temporal* asserts that the reified statement is valid during the given time interval. Figure 1 illustrates this approach.

2.3 Spatial Ontology

Spatial features 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 OGC Semantic Web Interoperability Experiment [18] and the W3C geo incubator group [17]. The existing OGC GML

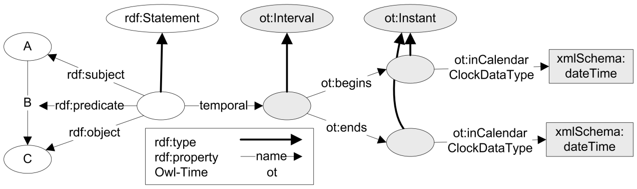


Fig. 1 Temporal reification of the RDF statement (*A B C*). Constructs from the Owl-Time ontology are shown in gray.

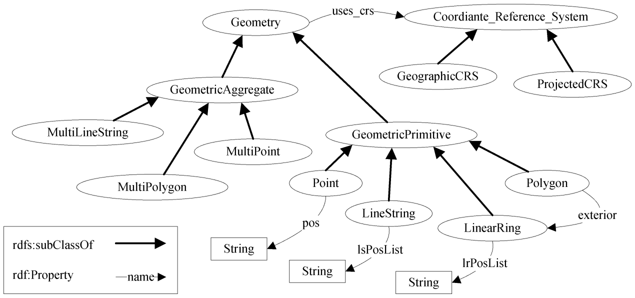


Fig. 2 GeoRSS GML-based ontology modeling basic spatial geometries. Note that Geometric Aggregates contain collections of their respective Geometric Primitives (e.g., MultiPolygon contains a collection of Polygons). These relations and attributes of Coordinate Reference System have been left out of the figure for clarity.

specification serves as an excellent basis for these ontologies as discussed in [1] and [14]. We propose a spatial ontology based on the GeoRSS GML specification [26]. The ontology models 2-dimensional spatial geometries and associated spatial reference system information. Figure 2 illustrates the RDFS representation of this ontology.

3 The SPARQL-ST Query Language

This section presents the SPARQL-ST query language. We first give a formal syntax for SPARQL-ST and present a formal semantics for SPARQL-ST queries. We then illustrate the concrete syntax of SPARQL-ST with a series of examples. At the end of this section, we present motivations for various aspects of the SPARQL-ST design and discuss possible alternatives.

3.1 Formal Syntax for SPARQL-ST

In this section, we give a formalization of the SPARQL-ST syntax that is based on the formalization of the SPARQL syntax given by [19]. 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.

3.1.1 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 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* FILTER *R*) is a spatiotemporal graph pattern
- 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 [19] and remains unchanged. Spatial built-in conditions and temporal built-in conditions are described below.

3.1.2 Spatial Built-in Conditions

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

A *qualitative spatial function* is a Boolean function $qsf : S \times S \rightarrow \mathbb{B}$, where S is the set of all possible spatial geometries as defined by the ontology in Figure 2. Any of the following topological spatial relations identified by Egenhofer and Herring [7] may be used as qualitative spatial functions in our formalization: *disjoint, touch, overlap boundary disjoint, overlap boundary intersect, equal, contains, covers, in-*

side, 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 $msf : S \times S \to \mathbb{R}$. We use one metric spatial function $distance : S \times S \to \mathbb{R}$, 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 $r \in \mathbb{R}$.

$$\langle mse \rangle ::= \langle msf(s_1, s_2) \rangle \ \langle comp \rangle \ r$$
$$\langle comp \rangle ::= < \mid > \mid \leq \mid \geq \mid =$$

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 expressions. A spatial built-in condition takes the following form.

$$\langle sf \rangle ::= \langle mse \rangle \mid \langle qse \rangle \mid \langle sf \rangle \ \textbf{AND} \ \langle sf \rangle \mid \langle sf \rangle \ \textbf{OR} \ \langle sf \rangle \mid \ \textbf{NOT} \ \langle sf \rangle$$

3.1.3 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 expressions. 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 $i \in I$.

$$\langle tp \rangle ::= intersect(V'_T) \mid range(V'_T) \mid vt \mid i$$

A *qualitative temporal function* is a Boolean function $qtf : I \times I \to \mathbb{B}$. Any of the thirteen interval relations identified by Allen [2] 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 $mtf : I \times I \to \mathbb{Z}$. We use one metric temporal function $elapsed\_time : I \times I \to \mathbb{Z}$, 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. We define a *metric temporal expression*, *mte*, as follows, where $z \in \mathbb{Z}$.

$$\langle mte \rangle ::= \langle mtf(\langle tp \rangle, \langle tp \rangle) \rangle \ \langle comp \rangle \ z$$
$$\langle comp \rangle ::= < \mid > \mid \leq \mid \geq \mid =$$

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

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

3.2 Formal Semantics for SPARQL-ST

We first give some initial definitions and then present the formal semantics of SPARQL-ST.

3.2.1 Initial Definitions

Let T be a set of totally ordered time points. Let G_t be a temporal RDF graph defined over T. $TRIPLES(G_t)$ denotes the set $\{(s,p,o) \mid \exists\, t \in T$ with $(s,p,o) : [t] \in G_t\}$. For each statement $e = (s,p,o) \in TRIPLES(G_t)$, let $temporal(e) = \{t \mid (s,p,o) : [t] \in G_t\}$. For a set of time points $T' \subseteq T$, let $contig\_intervals(T') = \{[t_i,t_j] \mid \forall\, t \in T$: (if $t_i \leq t$ and $t \leq t_j$ then $t \in T'$) and $t_{i-1} \notin T'$ and $t_{j+1} \notin T'\}$. Consider the following example: suppose $T = \{1,2,3,4,5,6,7,8,9,10\}$ and $T' = \{2,3,4,7,8\}$, then $contig\_intervals(T') = \{[2,4],[7,8]\}$.

Given a set of time intervals $I = \{(s_1,t_1),\ (s_2,t_2),\ ...,\ (s_n,t_n)\}$ defined over T, let $s_{min} = \min_{1 \leq i \leq n} s_i$, $s_{max} = \max_{1 \leq i \leq n} s_i$, $t_{min} = \min_{1 \leq i \leq n} t_i$, and $t_{max} = \max_{1 \leq i \leq n} t_i$. We define two values, *intersect* and *range*, as follows: $intersect(I) = [s_{max}, t_{min}]$ if $s_{max} \leq t_{min}$, else *null*, $range(I) = [s_{min}, t_{max}]$ if $s_{min} \leq t_{max}$, else *null*. Conceptually, $intersect(I)$ is the largest time interval that intersects each interval in I, and $range(I)$ is the smallest interval that contains each interval in I.

3.2.2 SPARQL-ST Semantics

The semantics of a SPARQL-ST spatiotemporal graph pattern query are based on the concept of a mapping introduced by Perez, et al. in [19] to provide a formal semantics for SPARQL. Here, we extend this mapping concept 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. Recall that for a set A, 2^A denotes the powerset of A. A mapping μ 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 Geometry instance
- if $vt \in V_T$ then $\mu(vt) = i \in I$

For a mapping μ, the subset of $(V_N \cup V_S \cup V_T)$ where it is defined is called its domain $dom(\mu)$. Two mappings μ_1 and μ_2 are compatible if, for all $x \in dom(\mu_1) \cap dom(\mu_2)$, 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:

$$\begin{aligned} M_1 \bowtie M_2 = \{ & \mu_1 \cup \mu_2 \mid \mu_1 \in M_1 \text{ and } \mu_2 \in M_2 \\ & \text{and } \mu_1 \text{ and } \mu_2 \text{ are compatible mappings} \} \end{aligned}$$

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 *t_triple*, 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 $head : (RT \cup 2^{((U \cup B) \times U \times RT)}) \to RT$. This function is defined as follows:

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

Conceptually, if t is a single URI, $head(t)$ returns this single URI, and if t is a collection of triples representing a Geometry instance, $head(t)$ returns the top level URI of the Geometry instance. For the example in Figure 3, the top level URI is $geo : polygon\_123$.

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix geo: <http://knoesis.org/geo#> .

geo:polygon_123 rdf:type geo:Geometry .
geo:polygon_123 rdf:type geo:GeometricPrimitive .
geo:polygon_123 rdf:type geo:Polygon .
geo:polygon_123 geo:exterior geo:LinearRing_123 .
geo:LinearRing_123 geo:lrPosList
                              “-122.84501 42.240328, -122.8075 42.240328,
                                  -122.8075 42.3764, -122.84501 42.3764,
                                  -122.84501 42.240328” .
geo:polygon_123 geo:uses_crs geo:CRS_NAD83 .
geo:CRS_NAD83 geo:srsName “urn:ogc:def:crs:EPSG:6.6:4269” .
```

Fig. 3 Set of triples representing a polygon.

For a mapping μ and a spatial triple pattern sp, we denote the triple obtained by replacing the variables v in sp with the value $head(\mu(v))$ as $triple(\mu, sp)$. For a mapping μ 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 $head(\mu(v))$ and the variables $t \in V_T$ in stp with the value $\mu(t)$ as $t\_triple(\mu, stp)$.

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:

- $[[sp]]_{G_t} = \{\mu \mid dom(\mu) = var(sp)$ and $triple(\mu, sp) \in TRIPLES(G_t)\}$
- $[[stp]]_{G_t} = \{\mu \mid dom(\mu) = var(sp)$ and for $(s, p, o) : [t_1, t_2] = t\_triple(\mu, stp)$ it is the case that $(s, p, o) \in TRIPLES(G_t)$ and $[t_1, t_2] \in contig\_intervals(temporal((s, p, o)))\}$
- $[[SP_1$ 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 μ satisfies a spatial built-in condition sf written $\mu \models sf$ if $var(sf) \subseteq dom(\mu)$ and sf evaluates to true when each variable $vs \in V_S$ in sf is replaced with $geom(\mu(vs))$. Note that the function $geom : 2^{((U \cup B) \times U \times RT)} \rightarrow \mathbb{R}^2$ maps the RDF serialization of a Geometry to an actual point, line or polygon. A mapping μ satisfies a temporal built-in condition tf written $\mu \models tf$ if $var(tf) \subseteq dom(\mu)$ 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,

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

3.3 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 Section 3.2.2.

(Temporal Filter Query) Find all house members who sponsored a bill after April 2, 2008. This query returns each representative and the *intersect* interval representing the time the bill was sponsored. This query uses the $TEMPORALFILTER$ construct to ensure that the bill was sponsored after April 2, 2008.

```
SELECT ?p, intersect(#t1, #t2, #t3, #t4)
     WHERE {
     ?p usgov:hasRole ?r #t1 .
     ?r usgov:forOffice ?o #t2 .
     ?o usgov:isPartOf usgov:congress/house #t3 .
     ?p usgov:sponsor ?b #t4 .
     TEMPORAL FILTER
     (
      after(intersect(#t1, #t2, #t3, #t4),
                interval(04:02:2008, 04:02:2008,
                MM:DD:YYYY)))}}
```

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

```
SELECT ?p, %g
    WHERE {
         ?v usgov:hasBallot ?b .
         ?v usgov:billNo "88" .
         ?b usgov:voter ?p .
         ?b usgov:hasOption "Aye" .
         ?p usgov:hasRole ?r .
         ?r usgov:forOffice ?o .
         ?o usgov:represents ?c .
         ?c stt:located_at %g }}
```

:Filtered Spatiotemporal Query) Find all politicians representing congressional districts within a given bounding box and return the times that those politicians represented those areas. This query uses a *SPATIALFILTER* involving the *inside* function to ensure each returned congressional district falls within the given geographical area.The intersect interval of several temporal variables is used to select the desired temporal intervals.

```
SELECT ?p, intersect(#t1, #t2, #t3, #t4)
    WHERE {
         ?p usgov:hasRole ?r #t1 .
         ?r usgov:forOffice ?o #t2 .
         ?o usgov:represents ?c #t3 .
         ?c stt:located_at %g #t4 .
         SPATIAL FILTER (inside(%g, GEOM(POLYGON ((
         -75.14 40.88, -70.77 40.88, -70.77 42.35,
         -70.77 42.35, -75.14 42.35,
         -75.14 42.35, -75.14 40.88)) ))}}
```

3.4 Design Decisions

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.

The example below illustrates the first alternative where the relevant parts of a spatial object are specified in a graph pattern.

```
SELECT ?positions
WHERE {
<http://house/106/nh> usgov:represents ?x .
?x stt:located\_at ?sr .
?sr geo:exterior ?lr .
?lr geo:lrPosList ?positions }
```

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*. A query using this approach is shown in the example below. This query returns all triples making up each named graph (geometry) in the result.

```
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 approach makes the semantics of our STT modeling approach less clear because it hides the fact that the query is dealing with spatial objects. 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 geometry.

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. This approach is problematic for the following reasons. First, it is extremely verbose, as it would take eight triple patterns to retrieve the valid times for each statement. Second, the semantics of temporal RDF are lost because the query will simply match triples in the RDF dataset, and the concepts of temporal RDFS inferencing (see Section 4.1) are lost. In addition, special temporal variables make it clear that one is querying a temporal RDF graph rather than a plain RDF graph.

4 Implementation Framework

We have implemented SPARQL-ST by extending a commercial relational database that supports spatial objects [3]. 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. Our prototype implementation supports qualitative spatial and temporal relationships and spatial and temporal filter expressions involving conjunctions of filtering conditions.

[3] License restrictions related to publication of timing results prevent us from disclosing the name of the database vendor.

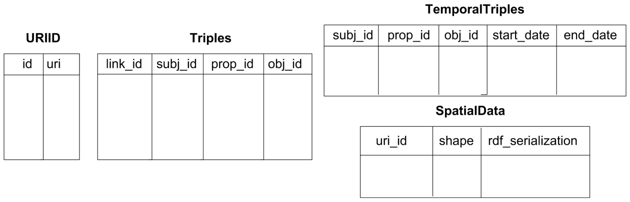


Fig. 4 Table structures used for our SPARQL-ST implementation.

4.1 Storage and Indexing Scheme

Our storage scheme for spatiotemporal RDF is shown in Figure 4. RDF triples are stored using the schema-oblivious storage scheme [28]. A $URIID$ table maps full URIs to numeric ids, and a $Triples$ table stores subject, predicate and object ids. This basic scheme is augmented with additonal structures for efficient processing of spatial and temporal data. A $TemporalTriples$ table stores subject, predicate and object ids along with two datetime columns that represent the start and end of the triple's valid time interval. A $SpatialData$ table maps Geometry URIs with their representation in the native spatial object type of the database. This table also stores the RDF/XML serialization of the Geometry (e.g., the triples in Figure 3) to allow for efficient retrieval of spatial variable mappings. The $TemporalTriples$ and $SpatialData$ tables are constructed during a post processing step after all asserted triples are loaded into the $URIID$ and $Triples$ tables.

The complete set of asserted and inferred temporal triples is stored in the $TemporalTriples$ table. A post processing step performs RDF/S inferencing and computes the valid time intervals for inferred statements. For example, given the asserted temporal triples $(x,\ p,\ y) : [1,5], (p,\ \textit{rdfs:domain},\ a) : [0,10]$, we would infer $(x,\ \textit{rdf:type},\ a) : [1,5]$ through rule *rdfs2* (refer to [10] for the complete set of RDFS inferencing rules). In each case, the computed valid time interval is the intersection of the valid time intervals of the set of triples used to make the inference. Temporal evolution of ontology schemas is beyond the scope of this work, so we therefore limit temporal RDFS inferencing to instance-level statements. That is, we assume schema-level statements are valid during the interval $[0,\infty]$, and we compute valid times for all *rdf:type* statements inferred through rules *rdfs2*, *rdfs3* and *rdfs9* and all instance-level statements inferred from *rdfs:subPropertyOf* (i.e. rule *rdfs7*). We ensure that the final valid times recorded for each statement are stored as the minimal set of contiguous intervals as described in Section 3.2.1. The algorithm for this post processing step is given in our earlier work [22].

4.2 Query Evaluation Procedure

The evaluation of a SPARQL-ST query proceeds in two basic steps. First, the SPARQL-ST query is translated into a SQL query against the table structures described in Section 4.1. This initial query is referred to as the base query. Second, further processing of the results of the base query is done on a row-by-row basis, and the appropriate result set is constructed and returned.

The first step in our query evaluation procedure is construction of the base SQL query for a given SPARQL-ST query. We first translate the graph pattern into a multi-way join over the $TemporalTriples$, $URIID$, and $SpatialData$ tables. If an appropriate SPATIAL FILTER or TEMPORAL FILTER condition is present (i.e., a condition involving a variable and a constant spatial geometry or temporal interval), we augment this multi-way join query with a spatial or temporal predicate that utilizes the built-in spatial and temporal indexes of the DBMS. We only push down a single filter condition to the base query, and spatial conditions are given preference over temporal conditions. Spatial conditions are favored due to their better performance in our previous experiments [22].

The second step in our query evaluation procedure performs additional processing on the results of the base query on a row-by-row basis. In this step, we evaluate any filter conditions that were not pushed down to the base query, and we construct any *intersect* or *range* intervals that need to be returned from the table function. We also construct and return a result row of the table function in this step.

5 Performance Evaluation

The experimental evaluation of our implementation is described in this section. All experiments were run 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 sort area 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 5 consecutive executions. We report the mean execution time over these 5 consecutive executions.

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.

Testing details (e.g., queries used and datasets) are available at [4].

[4] http://knoesis.wright.edu/students/mperry/sparql-st.html

5.1 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 1 shows the characteristics of these datasets.

Table 1 Characteristics of GovTrack and SynHist datasets

Dataset	Num Triples (in thousands)			Num Spatial Features	Avg Num Points per Polygon
	Asserted	Inferred	Total		
SH1	71	50	121	3,470	98
SH2	980	643	1,623	28,488	63
SH3	4,295	2,708	7,002	77,440	67
SH4	11,593	7,559	19,152	169,722	56
SH5	17,616	11,290	28,906	244,653	61
GT1	2,959	3,036	5,995	3,433	4
GT2	5,245	5,226	10,471	3,433	4
GT3	12,820	13,099	25,919	3,433	4

Table 2 Scalability with respect to datset size for GovTrack dataset.Legends: # T = Number of Triples, # V = Number of variables, # R = Result Size.

Query	Description	# T	# V	# R	Execution Time (sec)		
					GT1	GT2	GT3
G1	t-select	5	5	94	0.14	0.136	0.137
G2	t-filter – int / after	5	6	483	0.614	0.609	0.565
G3	t-join – int / during	3/3	3/3	90	0.821	0.817	0.838
G4	t-join – int / before	3/3	3/3	120	0.376	0.376	0.375
G5	s-select	5	5	428	2.663	2.658	2.660
G6	s-filter – anyinteract	5	6	562	3.340	3.360	3.345
G7	s-join – overlap	4/1	4/2	144	0.99	0.995	0.981
G8	s-filter – anyinteract + t-filter – int / during	5	6	397	3.444	3.438	3.463

5.1.1 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

Table 3 Scalability with respect to dataset size for SynHist dataset. Legends: # T = Number of Triples, # V = Number of variables, # R = Result Size.

Query	Description	# T	# V	# R	Execution Time (sec)				
					SH1	SH2	SH3	SH4	SH5
H1	t-select	5	5	178	0.290	0.291	0.290	0.290	0.292
H2	t-filter – int / overlap	5	6	128	0.178	0.321	0.572	1.179	2.238
H3	t-join – int / overlap	3/3	3/3	184	0.808	0.838	0.896	1.020	1.108
H4	t-join – int / anyinteract	3/3	3/3	42	0.360	0.361	0.374	0.389	0.392
H5	s-select	5	5	224	3.267	3.266	3.258	3.256	3.275
H6	s-filter – inside	5	6	303	3.999	3.996	3.984	3.989	3.981
H7	s-join – equal	4/2	4/2	48	0.296	0.296	0.298	0.297	0.295
H8	s-filter – inside + t-filter – int / overlap	5	6	13	3.772	3.770	3.779	3.768	3.781

temporal labels for the statements in the populated ontology. To populate the thematic portion of the battlefield analysis ontology, we used the ontology population tool described in [20]. 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.

To add spatial aspects to this dataset, we randomly assigned a spatial geometry to each instance of Geometry in the ontology. We used year 2000 census block group boundary polygons from the US Census Bureau [5] 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.

5.1.2 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. The GovTrack data contained a significant amount of temporal information. However, this information was encoded using separate properties rather than as temporal RDF. A preprocessing step was therefore needed to transform the dataset into a temporal RDF graph. To enhance the dataset with spatial data, we linked *Congressional_District*

[5] http://www.census.gov/geo/www/cob/bdy_files.html

instances with a bounding box representation of their corresponding boundary polygons available from the US Census Bureau [**?**]. 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.

5.2 *Experiments*

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

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:

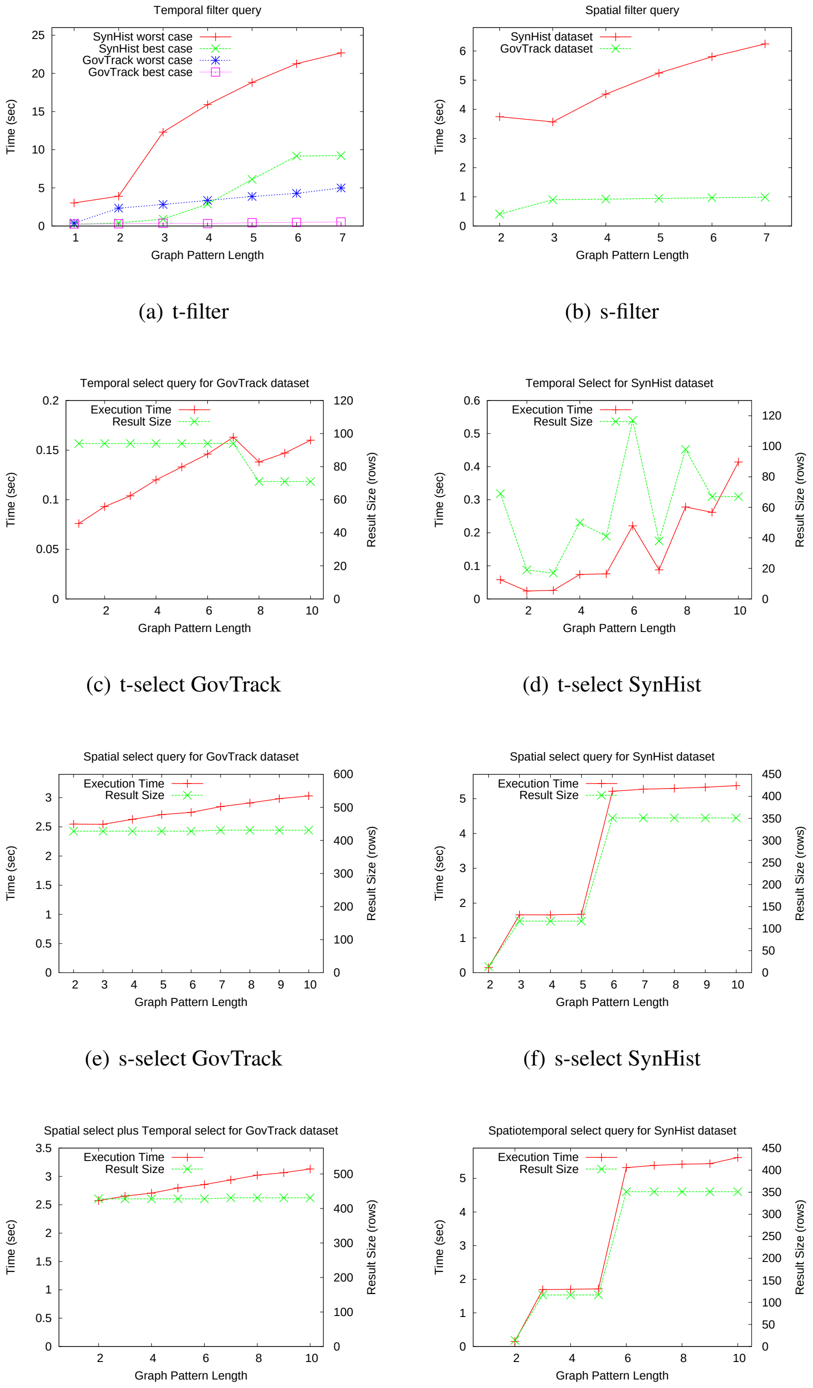
```
?p usgov:hasRole ?y .
?y usgov:forOffice usgov:congress/senate/va
```

5.2.1 Scalability w.r.t. Dataset Size

Tables 2 and 3 summarize the results of our experiments with respect to dataset size. These experiments were designed to test the scalability of our implementation for a basic set of SPARQL-ST queries.

Temporal Selection: Queries G1 and H1 select the intersect interval of the triples making up 5 hop selective graph patterns. The results show that 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.

Temporal Filter: Queries G2 and H2 test the scalability of our implementation for a SPARQL-ST query involving a TEMPORAL FILTER condition between a derived intersect interval and a constant time interval. 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



(a) t-filter

(b) s-filter

(c) t-select GovTrack

(d) t-select SynHist

(e) s-select GovTrack

(f) s-select SynHist

(g) st-select GovTrack

(h) st-select SynHist

Fig. 5 Results of scalability experiments for graph pattern complexity for SynHist (SH5) and GovTrack (GT3) datasets.

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 our implementation. Because the INTERSECT / RANGE interval derived for a graph pattern instance is constructed dynamically from the temporal labels of edges 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 individual triple patterns 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 5.2.2.

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 more densely packed valid time intervals with a higher degree of overlap than do the triples in 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.

Temporal Join: Queries G3, G4 and H3, H4 tested the scalability of our implementation for SPARQL-ST queries involving a TEMPORAL FILTER condition between two derived time intervals. The filter condition acts as a join between two disjoint graph patterns. The execution times for queries G3, G4 and H4 are relatively constant as the dataset size grows, but query H3 shows a slight growth in execution time. The growth for this query results from a combination of the particular temporal relation used and the denser set of time intervals in the SynHist dataset.

Spatial Selection: Queries G5 and H5 select a spatial variable. These queries use a selective graph pattern involving a single spatial variable. As a result of the index-based join strategy used by the DMBS, query execution time is near constant as dataset size increases. These queries have a significantly longer execution time than the corresponding temporal selection queries. The longer time is a result of the overhead of populating the result set of the query with RDF/XML serialization (stored as a CLOB) of each spatial feature in the result.

Spatial Filter: Queries G6 and H6 use an unselective graph pattern in combination with a SPATIAL FILTER expression over a spatial variable and a constant spatial feature (a rectangle in each case). The execution times for each query are near constant as the dataset size increases due to the index-based join strategy used by the DBMS to evaluate the graph pattern. The execution times for the GovTrack dataset are a bit faster because the spatial features in this dataset are simpler. Again, extra time is needed for spatial queries to populate the result set.

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.

Spatial Join: Queries G7 and H7 involved a graph pattern with two disjoint components and a SPATIAL FILTER condition over two spatial variables that acts as a spatial join for the two components of the graph pattern. Again, the execution times for each query are near constant as the dataset size increases as a result of the index-based join strategy used to evaluate the graph pattern. The times are a bit faster than other spatial queries because of the smaller result set sizes for these queries, which limits the overhead of populating the result set.

Spatiotemporal Filter: Queries G8 and H8 involve unselective graph patterns and both SPATIAL FILTER and TEMPORAL FILTER conditions. In each case, query execution time is near constant as the dataset size increases. Queries over the SynHist dataset are slower relative to their result set size because of the less efficient temporal processing and more complicated spatial features in this dataset.

5.2.2 Scalability w.r.t. Graph Pattern Complexity

Our next experiments are designed to test the scalability of our implementation with respect to query complexity: that is, the size of the graph pattern used. All experiments used the GT3 and SH5 datasets.

Temporal Filter: Our first experiment tested TEMPORAL FILTER queries involving unselective graph patterns and selective temporal filtering conditions. The key to the performance of these queries is to reduce the amount of search space by placing partial temporal constraints on individual triple patterns 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 temporal filter 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 5(a) 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 for the worst-case scenario. The performance is better for the GovTrack dataset because of the nature of the temporal intervals in each dataset as we discussed in Section 5.2.1. 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 after executing the base query.

Spatial Filter: Our next experiment tested SPATIAL FILTER queries involving unselective graph patterns and selective spatial filtering conditions. Figure 5(b) shows the execution times of these queries. As the graph pattern size grows, the query execution times show linear scalability on both datasets. These spatial queries are initially slower than the temporal filter queries but become faster for the larger graph patterns because the time for temporal filtering outweighs the time needed to populate spatial features in the result set. The faster execution times result from the more effective spatial indexing. The spatial index is initially used to select the URIs satisfying the spatial filtering condition, which reduces the search space for evaluating the rest of the graph pattern. The queries over the SynHist dataset have slower execution times because spatial computations are more expensive for the more complex spatial geometries in the SynHist dataset.

Basic Selection Queries: Our final experiments tested the scalability of our implementation for spatial, temporal and spatiotemporal selection queries using selective graph patterns ranging in size from 1 to 10 triples. The results of these experiments are shown in Figure 5(c) – Figure 5(h). 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. The spatial and spatiotemporal queries are slower than the temporal queries due to the overhead of populating spatial features in the result set.

6 Related Work

Extensions of SPARQL are abundant in the literature. These range from extensions for handling spatio-temporal data [16] to computing semantic associations [3, 13] to extensions for enabling skyline queries [25].

In a recent work [16] the authors have extended RDF (known as stRDF) to represent spatial and temporal data. stRDF is a constraint data model which extends RDF with represent spatial and temporal data. This is done primarily by using the main ideas from spatial and temporal constraint databases and by representing spatial objects using quantifier-free formulas in a first-order logic of linear constraints. Further, stSPARQL extends SPARQL (stSPARQL) so that spatial and temporal data can be queried using a declarative and user-friendly language. Although the objective of the two works are identical, we do so without any extensions of modification to existing RDF Models, thus making our approach useful for querying existing real world spatial data resources such as GovTrack, Geonames. For modeling time, both stSPARQL and our work rely on temporal RDF graphs presented in [9] to represent the valid time of a triple. The spatial modeling aspects of our work is significantly different from stSPARQL. Geometries in stRDF and stSPARQL are based on the mathematical concept of semi-linear subsets of Q^k , using notions of linear algebra. Further, in our work we have presented an evaluation of our approach on real world datasets and thus proved its performance and usefulness. We could not find a similar kind of evaluation in the work related to stSPARQL. Thus to summarize, the works follow different approaches for reaching the same objective of supporting modeling and querying of spatio-temporal- data using Semantic Web technologies.

Another discussion of querying spatial data using SPARQL appears in a paper by Kolas and Self [15] 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 [5] 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 .
?x georss:where ?y .
?y rcc:part ?p .
?y rcc:part ?p .
?p rdf:type gml:Buffer .
?p gml:radius "1" .
?p gml:bufferGeometry ?g .
?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 $SPATIALFILTER$ clause instead of encoding the spatial constraint within the graph pattern. Without introducing spatial variables this approach would suffer from the shortcomings described in Section 3.4. In addition, their implementation only supported the relations *connected* and *part*, and no performance results were presented.

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 [11]. For example, code implementing property functions that extend ARQ for geospatial relations appears at [6].

The following example query uses a $nearby()$ property function to select hotels near a certain point.

[6] http://geospatialweb.googlecode.com/svn/trunk/jenaext/src/org/geospatialweb/arqext/

```
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. In addition, 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 [8, 9] and [24] discuss aspects of querying temporal RDF graphs. [8, 9] 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 *TEMPORALFILTER* clause provides the needed temporal reasoning capabilities, and the need for maximal intervals is taken care of during our temporal RDFS inferencing procedure. Pugliese, et al. formally define a temporal RDF query [24]. 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, neither of these works discuss extensions of SPARQL needed to support their proposed querying approaches.

7 Conclusions

This work presented SPARQL-ST, an extension of SPARQL for spatiotemporal queries. SPARQL-ST adds spatial variables and constructs for manipulating temporal triples. We gave a formal syntax and semantics for SPARQL-ST and presented a prototype implementation built on top of a commercial DBMS. We demonstrated the scalability of our prototype implementation with an experimental evaluation using both real-world and synthetic RDF datasets of over 25 million triples. In the future, we plan to investigate standardization issues with respect to our spatiotemporal extensions to SPARQL. We also plan to do a comparative study of the RDF/SPARQL approach for spatiotemporal querying presented in this work and an approach using OWL-DL and specialized spatial and temporal reasoners. Such a study would help determine the pros and cons of each method.

Acknowledgements We thank Professor T. K. Prasad for his helpful comments on our formalization of SPARQL-ST, and Cory Henson for his comments on a draft of this work. This work is partially funded by NSF-ITRIDM Award #0714441 (SemDIS: Discovering Complex Relationships in the Semantic Web) and by NSF Award #IIS-0842129, titled ”III-SGER: Spatio-Temporal-

Thematic Queries of Semantic Web Data: a Study of Expressivity and Efficiency (09/01/2008-08/31/2010)".

References

1. Alia I. Abdelmonty, Philip D. Smart, Christopher B. Jones, Gaihua Fu, and David Finch. A critical evaluation of ontology languages for geographic information retrieval on the internet. *Journal of Visual Languages and Computing*, 16(4):331–358, 2005.
2. James F Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
3. Kemafor Anyanwu, Angela Maduko, and Amit P. Sheth. SPARQ2L: Towards support for subgraph extraction queries in RDF databases. In *16th International World Wide Web Conference*, pages 797–806, Banff, Alberta, Canada, 2007.
4. Dan Brickley and Ramanathan V. Guha. RDF vocabulary description language 1.0: RDF schema. W3C recommendation. http://www.w3.org/tr/rdf-schema/.
5. Anthony G Cohn, Brandon Bennett, John Gooday, and Nicholas Mark Gotts. Oualitative spatial representation and reasoning with the region connection calculus. *GeoInformatica*, 1(3):275–316, 1997.
6. Max J Egenhofer. Toward the semantic geospatial web. In *10th ACM International Symposium on Advances in Geographic Information Systems*, pages 1–4, McLean, VA, USA, 2002.
7. Max J Egenhofer and John R Herring. Categorizing binary topological relations between regions, lines, and points in geographic databases. Technical Report 94-1, University of Maine, National Center for Geographic Information and Analysis, 1994.
8. Claudio Gutierrez, Carlos Hurtado, and Alejandro Vaisman. Temporal RDF. In *2nd European Semantic Web Conference*, pages 93–107, Heraklion, Crete, Greece, 2005.
9. Claudio Gutierrez, Carlos Hurtado, and Alejandro Vaisman. Introducing time into RDF. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):207–218, February 2007.
10. Patrick Hayes. RDF semantics. http://www.w3.org/tr/rdf-mt/.
11. Hewlett-Packard Development Company. ARQ - a SPARQL processor for jena. http://jena.sourceforge.net/arq/.
12. Jerry Hobbs and Feng Pan. An ontology of time for the semantic web. *ACM Transactions on Asian Language Processing (TALIP): Special issue on Temporal Information Processing*, 3(1):66–85, 2004.
13. Krys Kochut and Maciej Janik. SPARQLeR: Extended SPARQL for semantic association discovery. In *4th European Semantic Web Conference*, pages 145–159, Innsbruck, Austria, 2007.
14. David Kolas, John Hebeler, and Mike Dean. Geospatial semantic web: Architecture of ontologies. In *1st International Conference on GeoSpatial Semantics*, pages 183–194, Mexico City, Mexico, 2005.
15. David Kolas and Troy Self. Spatially-augmented knowledgebase. In *6th International Semantic Web Conference*, pages 792–801, Busan, South Korea, 2007.
16. Manolis Koubarakis and Kostis Kyzirakos. Modeling and Querying Metadata in the Semantic Sensor Web: the model strdf and the query language stsparql. In Lora Aroyo, Grigoris Antoniou, Eero Hyvönen, Annette ten Teije, Heiner Stuckenschmidt, Liliana Cabral, and Tania Tudorache, editors, *Proceedings of the 7th Extended Semantic Web Conference (ESWC2010), Heraklion, Crete, Greece, May 30 - June 3, 2010, Proceedings, Part I*, volume 6088 of *Lecture Notes in Computer Science*. Springer, June 2010.
17. Joshua Lieberman. W3C geospatial incubator group. http://www.w3.org/2005/incubator/geo/.
18. Open Geospatial Consortium. Open geospatial consortium geospatial semantic web interoperability experiment. http://www.opengeospatial.org/projects/initiatives/gswie.
19. Jorge Perez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. In *5th International Semantic Web Conference*, pages 30–43, Athens, GA, USA, 2006.

20. Matthew Perry. Tontogen: A synthetic data set generator for semantic web applications. *AIS SIGSEMIS Bulletin*, 2(2):46–48, 2005.
21. Matthew Perry, Farshad Hakimpour, and Amit Sheth. Analyzing theme, space and time: An ontology-based approach. In *14th ACM International Symposium on Geographic Information Systems*, pages 147–154, Arlington, VA, USA, 2006.
22. Matthew Perry, Amit P. Sheth, Farshad Hakimpour, and Prateek Jain. Supporting complex thematic, spatial and temporal queries over semantic web data. In *2nd International Conference on Geospatial Semantics*, pages 228–246, Mexico City, Mexico, 2007.
23. Eric Prud'hommeaux and Andy Seaborne. SPARQL query language for RDF, W3C recommendation. http://www.w3.org/tr/rdf-sparql-query/.
24. Andrea Pugliese, Octavian Udrea, and V S Subrahmanian. Scaling RDF with time. In *17th International World Wide Web Conference*, pages 605–614, Beijing, China, 2008.
25. Wolf Siberski, Jeff Z. Pan, and Uwe Thaden. Querying the semantic web with preferences. In *5th International Semantic Web Conference*, pages 612–624, Athens, GA, USA, 2006.
26. Raj Singh, Andrew Turner, Mikel Maron, and Allan Doyle. GeoRSS: Geographically encoded objects for RSS feeds. http://georss.org/gml.
27. Philip D. Smart, Alia I. Abdelmonty, Baher A. El-Geresy, and Christopher B. Jones. A framework for combining rules and geo-ontologies. In *1st International Conference on Web Reasoning and Rule Systems*, pages 133–147, Innsbruck, Austria, 2007.
28. Yannis Theoharis, Vassilis Christophides, and Gregory Karvounarakis. Benchmarking database representations of RDF/S stores. In *5th International Semantic Web Conference*, pages 685–701, Galway, Ireland, 2005.