Research Article **Open Access**

Dong Wang, Lei Zou*, and Dongyan Zhao

g^{st}-store: Querying Large Spatiotemporal RDF Graphs

https://doi.org/10.1515/dim-2017-0008
Received October 19, 2017; accepted November 19, 2017

Abstract: The Simple Protocol and RDF Query Language (SPARQL) query language allows users to issue a structural query over a resource description framework (RDF) graph. However, the lack of a spatiotemporal query language limits the usage of RDF data in spatiotemporal-oriented applications. As the spatiotemporal information continuously increases in RDF data, it is necessary to design an effective and efficient spatiotemporal RDF data management system. In this paper, we formally define the spatiotemporal information-integrated RDF data, introduce a spatiotemporal query language that extends the SPARQL language with spatiotemporal assertions to query spatiotemporal information-integrated RDF data, and design a novel index and the corresponding query algorithm. The experimental results on a large, real RDF graph integrating spatial and temporal information (> 180 million triples) confirm the superiority of our approach. In contrast to its competitors, g^{st}-store outperforms by more than 20%-30% in most cases.

Keywords: spatiotemporal query; RDF graph; tree index

1 Introduction

Nowadays, we can use the Resource Description Framework (RDF) (Klyne, Carroll, & McBride, 2004), which is recommended by the World Wide Web Consortium (W3C) as the foundation of the Semantic Web, to restore the knowledge. An RDF statement is a triple presented as $\langle subject, predicate, object\rangle$, which describes a property value of a subject or the relation between the two entities – the *subject* and the *object*. In practice, a huge amount of entities and statements contains spatial and temporal information, e.g., a city is always located in a specific location, and a transient event happens at a specific time point. Therefore, the schema of the RDF data needs to be extended to express the spatiotemporal semantics. For example, ⟨Ulm Coordinates 48.39841/9.99155⟩ describes the longitude and latitude of a spatial entity "Ulm". ⟨Albert_Einstein WonPrize Nobel_Prize (59.35,18.0667) (1921-##-##,1921-##-##)⟩ denotes the event that Einstein won Nobel Prize in 1921 in a location with the coordinates 59.35°N,18.0667°E.

Based on the spatiotemporal RDF data, users can ask more meaningful queries. For example, it is useful to count the fast food restaurants nearby someone's workplace, or to find the spouses in Hollywood whose age difference is more than 10 years. More practically, for meetings, incentives, conventions, and events (MICE) tourists, it is important to find those newly renovated motels that are near the places of interest. In order to answer these spatiotemporal queries more efficiently and effectively, it is important to build an RDF query engine for the spatiotemporal RDF data.

Although the spatial and temporal data can be managed using traditional spatiotemporal databases, the "pay-as-you-go" nature of RDF proposes new challenges for existing solutions. Firstly, the RDF data have diverse graph structures for different entities, which property does not fit the traditional entity–relationship (ER) model. Though the column-based relational database partially solves the problem, it also suffers due to the multiple values and null values in RDF data. Secondly, the incomplete spatiotemporal information in RDF data makes it inefficient to retrieve the spatiotemporal RDF data using a "join" operator, i.e., too many intermediate results might be generated. The entities and the statements without necessary spatiotemporal information should be more efficiently filtered early. In summary, the traditional spatiotemporal databases are not suitable for spatiotemporal RDF data management.

A spatiotemporal RDF data set can be linked to other RDF repositories to provide structural queries with both semantic and spatiotemporal features. In this case, the spatiotemporal information-integrated RDF data are more suitable for providing location-based and time-based se-

Dong Wang, Peking University, Beijing, China, E-mail: wangd@pku.edu.cn
***Corresponding author: Lei Zou,** Peking University, Beijing, China, E-mail:zoulei@pku.edu.cn
Dongyan Zhao, Peking University, Beijing, China, E-mail: zhaodongyan@pku.edu.cn

mantic search for users. Though queries are often related to spatiotemporal information, it is hard to find a short query that includes all the four spatiotemporal queries . As a result, we artificially build an example: a user wants to find a physicist who was born in a circular area, with the center located at coordinates (49°N,10°E) and having a radius of 300 km (this area is the southern area of Germany), and who won some academic award in some place where the distance between the place and his birth place is <1500 km. Additionally, he was born before the year 1900, and he won the prize before his 50th birthday. The query can be represented as a Simple Protocol and RDF Query Language (SPARQL)-like query as follows. Section 3.2 gives the formalized definition of the query.

Example 1.
```
SELECT ?x WHERE{
       ?x BornIn ?y ?l1 ?t1.
       ?x WonPrize ?z ?l2 ?t2.
       ?y type Populated_place.
       ?z type Academic_Awards.}
FILTER {distance(place(?y),place(49,10))<300
        AND distance(place(?l2),place(?y))<1500
        AND time(?t1)<date(1900.01.01)
        AND time(?t2)<time(?t1)
                    +(50year.01month.01day)}
```

In this paper, we extend the semantics of the SPARQL language by integrating the spatial and temporal feature assertions (the extended SPARQL query is called the S-T query, short for spatial-temporal SPARQL query). The spatial and the temporal constraints assert the location of an entity/event and the event's valid time, for instance, $distance(place(?y), place(49, 10)) < 300$ and $time(?t1) < date(1900.01.01)$ in Example 1.

In order to answer S-T queries in a uniform manner, we propose a tree-style index structure (called the ST-tree). The ST-tree index is a height balanced n-ary tree. The semantic features and the spatiotemporal features are integrated within the ST-tree, and the ST-tree combines the advantages of the advantages of the R-tree (Guttman, 1984) and the VS-tree (Zou, Mo, Chen, Özsu, & Zhao, 2011) in two steps. First, we encode the entities and RDF triples into bit strings (called "signatures") to form a signature graph. The ST-tree is constructed over the signature graph, and a list of pruning rules that consider both spatiotemporal and semantic constraints in the query is proposed to reduce the search space during S-T query processing. Second, we introduce a cost model to guide the ST-tree construction.

To summarize, we make the following contributions in this paper.

1. We formalize the spatiotemporal queries by extending the semantics of SPARQL queries, and the spatiotemporal queries are used to retrieve information over the RDF data integrating the spatial and temporal information. Flexible spatiotemporal and semantic constraints are used in the spatiotemporal queries.
2. We build a novel tree-style index integrating the spatiotemporal features and the semantic features, and we design a cost model-based approach to build ST-tree.
3. Based on the ST-tree, we design a novel S-T query-processing algorithm that includes both semantic and spatiotemporal pruning rules to reduce the search space.
4. We evaluate our approach on a large real-world data set, and the result shows that our approach outperforms the baselines.

The remainder of this paper is organized as follows. Section 2 reviews the existing spatiotemporal RDF data management systems and some related works. Section 3 describes the basic idea of our work and gives a list of formal definitions. Section 4 gives the whole framework of our gst-store. Then, Section 5 and 6 show the technical details of our index and the query algorithm. The experimental results are shown in Section 7. Finally, we conclude this paper in Section 8.

2 Related Work

Recently, researchers have begun to pay attention to the spatiotemporal RDF data. There are some available real-world RDF data sets that integrate spatial and temporal information, such as YAGO2[1] (Hoffart, Suchanek, Berberich, & Weikum, 2013), OpenStreetMap[2] (Haklay & Weber, 2008), GovTrack[3] and so on, are available. YAGO2 (Hoffart, Suchanek, Berberich, & Weikum, 2013) is an RDF data set based on Wikipidea and WordNet. Additionally, YAGO2 integrates GeoNames[4], which is a geographical database that contains more than 10 million geographical names, to express the spatial information of the entities. At the same time, some statements have temporal information, e.g., the objects of the predicates "bornOnDate", "wasCreatedOnDate" and so on denote the time that the were born in or created. Based on the spatiotemporal information and some simple inference rules, YAGO2 generates a list of spatial entities and a list of spatiotemporal statements (Hoffart, Suchanek, Berberich, & Weikum, 2013).

Many RDF management systems (Abadi, Marcus, Madden, & Hollenbach, 2009, 2007; Broekstra, Kamp-

1 http://www.mpi-inf.mpg.de/YAGO-naga/YAGO/downloads.html
2 http://planet.openstreetmap.org/
3 http://www.govtrack.us/data/rdf/
4 http://www.geonames.org/about.html

man, & Van Harmelen, 2002; Neumann & Weikum, 2009; Weiss, Karras, & Bernstein, 2008; Broekstra, Kampman, & Van Harmelen, 2002; Wilkinson, 2009; Wilkinson, 2009; Wilkinson, Sayers, Kuno, & Reynolds, 2003) have been proposed in the past years. RDF-3x (Neumann & Weikum, 2010), Hexastore (Weiss, Karras, & Bernstein, 2008) and gStore (Zou, Mo, Chen, Özsu, & Zhao, 2011) are the state-of-the-art RDF management systems. In these management systems, the RDF data are well organized and indexes are used to efficiently and effectively answer the RDF queries. Unfortunately, since the indexes are well-designed and none of the systems takes spatial or temporal features into consideration, all the systems are unsuitable for spatiotemporal RDF data management without great modification.

To the best of our knowledge, few SPARQL query engines consider spatial and temporal queries over RDF data in a uniform manner except for YAGO2 Demo (Hoffart et al., 2011) and SPARQL-ST (Perry, Jain, & Sheth, 2011). However, YAGO2 Demo (Hoffart et al., 2011) uses hard-coded spatial/temporal predicates to define the spatiotemporal queries. Six (hard-coded) spatial predicates ("northOf", "eastOf", "southOf", "westOf", "nearby", and "locatedIn") and four (hard-coded) temporal predicates ("before", "after", "during", and "overlaps") over statements are employed used in the YAGO2 Demo. Users can construct queries as a list of triple patterns with the spatial and temporal predicates. Other spatiotemporal queries are not supported. Since all spatiotemporal predicates are determinative, YAGO2 Demo does not allow flexible spatiotemporal range queries or join queries. The spatiotemporal semantics for the statements is limited, and the spatial semantics of the entities is missing.

Perry et al. (Perry, Jain, & Sheth, 2011) propose SPARQL-ST based on the work of Gutierrez et al. (Gutierrez, Hurtado, & Vaisman, 2007) for integrating the spatial information of entities and the temporal information of statements. In their study, Perry et al. (Perry, Jain, & Sheth, 2011) formalize the storage schema for the spatial entities and the temporal statements, in addition to formalizing the spatiotemporal graph pattern to construct SPARQL-ST. Their work implements a query engine by extending a commercial relational database that supports spatial objects, i.e., by dividing the spatiotemporal RDF data into three main tables (namely, triple table, spatial table, and temporal table) to restore the data and by utilizing the literal and the spatiotemporal indexes of the relational database to evaluate the SPARQL-ST queries. In contrast to our framework, the spatial semantics of statements is missing, and the storage schema is not suitable for real, big RDF data, e.g., >100 million statements; moreover, only parts of them have spatiotemporal information. The time cost on self-joins is unacceptable.

Furthermore, Batsakis et al. (Batsakis & Petrakis, 2010) and Lyell et al. (Lyell, Voyadgis, Song, Ketha, & Dibner, 2011) try to build spatiotemporal ontology to organize the spatiotemporal RDF data. The corresponding ontology-based query languages are introduced to retrieve the spatiotemporal RDF data. These works introduce well-designed ontology, and the query capability has been widely extended. However, these two reports have little discussion on how to answer the spatiotemporal RDF queries efficiently, and the query performance is not evaluated. Additionally, since the statements can not be seen as vertices, the ontology-based model is not suitable for organizing the spatiotemporal information of the statements.

Besides, several other proposals take either spatial features or temporal features of RDF data into consideration. Brodt et al. (Brodt, Nicklas, & Mitschang, 2010) and Erling and Mikhailov (Erling & Mikhailov, 2009) utilize RDF query engines and spatial index to manage spatial RDF data. Brodt et al. (Brodt, Nicklas, & Mitschang, 2010) uses RDF-3x as the base RDF query engine, and adds a spatial index for filtering entities before or after RDF-3x join operations. These two approaches only support range query (and spatial join (Erling & Mikhailov, 2009)) on entities, and the spatial entities follow the GeoRSS GML (Singh, Turner, Maron, & Doyle, 2008) model. Our early work on S-store (Wang et al., 2013) integrates spatial information into the RDF data. In S-store, a tree index SS-tree is used. First, an R-tree based on the spatial entities and a VS-tree based on the nonspatial entities are built separately, and then the two trees are combined to form the SS-tree. The R-tree and the VS-tree pruning rules are used to generate the candidates for the queries. The brute force combining method disregards that an entity integrates the spatial features and the semantic features at the same time. In contrast, we propose a cost model-based method to take both spatiotemporal features and semantic features into consideration while constructing the tree index.

Gutierrez et al. (Gutierrez, Hurtado, & Vaisman, 2007; Gutiérrez, Hurtado, & Vaisman, 2005) give formally definitions of the temporal RDF graph, and prefer to use time interval labeling on an RDF graph to integrate temporal information into RDF data. Furthermore, their first work (Gutierrez, Hurtado, & Vaisman, 2007) introduces a simple query language for temporal RDF data. Based on the work of Gutierrez et al., several query languages have been proposed, such as T-SPARQL (Grandi, 2010), SPARQL-ST (Perry, Jain, & Sheth, 2011) and τ-SPARQL (Tappolet & Bernstein, 2009). Tappolet et al. (Tap-

polet & Bernstein, 2009) propose a temporal RDF data management framework. The named graph is used to manage the statements with different time intervals, and a tree-style index keyTree is introduced to efficiently retrieve the valid time interval and the involved triples at a certain time point. In contrast, Pugliese et al. (Pugliese, Udrea, &Subrahmanian, 2008) extend the work of (Gutierrez, Hurtado, & Vaisman, 2007), and introduce a novel tree-style index to efficiently and effectively answer the temporal RDF queries. Firstly, they combine the graph distance metric and the temporal distance metric to build a metric called tGRIN distance metric. Then, based on the tGRIN distance metric, the entries are clustered. The clusters with different granularity constitute the tGRIN tree-style index. Based on the tGRIN index, two pruning rules are introduced to efficiently answer the temporal queries. However, most of the statements in real data sets (e.g., YAGO2), most of the statements lack temporal information. Therefore, the tGRIN metric fails since it is hard to compute the temporal distance between temporal statements and non-temporal statements[5]. Thus, the pruning rules are inefficient. Besides, the pruning rules are based on mapping the constant in the query to the data set. If the constants in the query are high-degree nodes (e.g., the type "city") or if there is no constant in the query, the pruning rules are also inefficient.

3 Problem Definition

3.1 SPARQL vs. Subgraph Match

An RDF data set is a list of RDF triples. Here, we have a sample RDF data set (shown in Figure 1(a)), which that consists of 25 triples. We call each triple a statement. The answer of a SPARQL query is a list of statements that satisfy the SPARQL constraints. We also regard an RDF data set as a graph (called RDF graph G). Figure 1(b) shows the corresponding RDF graph of the sample data set. Furthermore, a SPARQL query can be also modeled as a graph structure Q. Therefore, answering a SPARQL query is equivalent to finding subgraph matches of query graph Q over RDF graph G. The formal definitions are given as follows.

Definition 1. *A statement is a triple $\langle s, p, o\rangle$, where s, p, and o represent subject, predicate, and object, respectively.*

Definition 2. *The RDF data graph is denoted as $G = \langle V, E, L_V, L_E\rangle$, where*

(1)$V = V_l \cup V_e \cup V_c \cup V_b$ denotes all RDF vertices where V_l, V_e, V_c and V_b are the sets of literal vertices, entity vertices, class vertices and blank nodes respectively.

(2) E is a collection of the edges between vertices.

(3) $L_V = \{URI\} \cup \{Literal\ Value\} \cup \{null\}$ is the collection of all vertex labels (i.e., label(v)), where $v \in \{V_e \cup V_c\} \Leftrightarrow label(v) \in \{URI\}$ and $v \in V_l \Leftrightarrow label(v) \in \{Literal\ Value\}$. For $v \in V_b$, $label(v)$ is null.

(4)L_E is the collection of edge labels, i.e., all possible predicates.

Definition 3. *The SPARQL query graph is denoted as $Q = \langle V, E, L_V, L_E\rangle$, where*

(1)$V = V_l \cup V_e \cup V_c \cup V_b \cup V_p$, where V_p denotes the parameter vertices, and V_l, V_e, V_c, and V_b are the same as in Definition 2.

(2)E and L_E are the same as in Definition 2.

(3) L_V is the same as in Definition 2, expect for $v \in V_p \Leftrightarrow label(v)$ is null.

Definition 4. *Consider an RDF graph G and a query graph Q with n vertices $\{v_1, \ldots, v_n\}$. A list of n corresponding vertices $\{u_1, \ldots, u_n\}$ in G is said to be a match of Q if and only if the following conditions hold:*

(1) If $v_i \in \{V_l \cup V_c \cup V_e\}$, $u_i \in \{V_l \cup V_c \cup V_e\}$ and $label(v_i) = label(u_i)$;

(2) If $v_i \in \{V_b \cup V_p\}$, $lable(u_i)$ is unrestricted;

(3). If there is an edge $\overrightarrow{v_i v_j}$ from v_i to v_j in Q , there is also an edge $\overrightarrow{u_i u_j}$ from u_i to u_j in G. If $\overrightarrow{v_i v_j}$ has predicate p, $\overrightarrow{u_i u_j}$ must have the same predicate p.

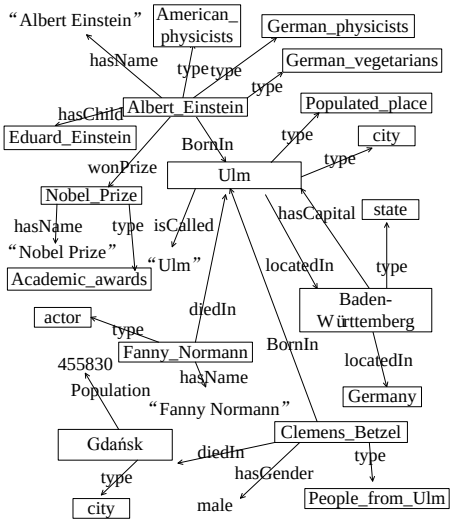
Here, an RDF data set is seen as a list of **statements**. A statement is also regarded as an edge in the RDF graph connecting the subject vertex and the object vertex with the edge label (the predicate). The subjects and the objects contain the vertex set of the RDF graph. If a vertex is an entity or a class, the vertex label is a uniform resource identifier (URI). If the subject or the object is a string, the vertex label is the corresponding literal value. Note that the label of a vertex can be null, i.e., the vertex is a blank node.

A SPARQL query is a small graph similar to the RDF graph. In contrast to the RDF graph, the SPARQL query graph contains a special case of vertices, i.e., the parameter vertices. The identifier of a parameter vertex is started with a "?", and the label of the parameter vertex is seen as

5 Actually, the nontemporal statements should be filtered out early since they dissatisfy the temporal constraint, and they are filtered out early in gst-store.

	Subject	Predict	Object
#1	Albert_Einstein	BornIn	Ulm
#2	Albert_Einstein	hasChild	Eduard_Einstein
#3	Albert_Einstein	type	American_physicists
#4	Albert_Einstein	type	German_physicists
#5	Albert_Einstein	type	German_vegetarians
#6	Albert_Einstein	WonPrize	Nobel_Prize
#7	Albert_Einstein	hasName	"Albert Einstein"
#8	Ulm	type	city
#9	Ulm	isCalled	"Ulm"
#10	Ulm	LocatedIn	Baden-Württemberg
#11	Ulm	type	Populated_place
#12	Baden-Württemberg	hasCapital	Ulm
#13	Baden-Württemberg	type	state
#14	Baden-Württemberg	LocatedIn	Germany
#15	Nobel_Prize	type	Academic_awards
#16	Nobel_Prize	hasName	"Nobel Prize"
#17	Fanny_Normann	diedIn	Ulm
#18	Fanny_Normann	hasName	"Fanny Normann"
#19	Fanny_Normann	type	actor
#20	Clemens_Betzel	diedIn	Gdańsk
#21	Clemens_Betzel	hasGender	male
#22	Clemens_Betzel	BornIn	Ulm
#23	Clemens_Betzel	type	People_from_Ulm
#24	Gdańsk	type	city
#25	Gdańsk	Population	455830

(a)



(b)

Fig. 1: Sample RDF Data Set

null. In the RDF graph, the graph matches of the SPARQL query graph are the result of the SPARQL query.

3.2 Spatiotemporal RDF

In this section, we formally define the spatiotemporal RDF data and the spatiotemporal SPARQL query as follows.

Definition 5. *An entity e is called a* ***spatial entity*** *if it has an explicit location labeled with the coordinates x and y (for the two-dimensional situation). The other entities are called* ***nonspatial entities****.*

Definition 6. *An* ***S-T statement*** *is a five-tuple* $\langle s, p, o, L, T\rangle$*, where* s*,* p*,* o*,* L *and* T *represent* $subject$*,* $predicate$*,* $object$*,* $location$*, and* $time\ interval$*, respectively. The S-T statement is an extension of the original RDF statement, where* s*,* p*, and* o *are the original elements.* L *denotes the spatial feature (the coordinates) of a statement, and* T *has the start time* T_s*, and the end time* T_e *to denote the valid time interval of a statement, i.e., the statement is considered to be credible in this time interval: specifically,* $T_s = T_e$ *if and only if the statement happens at a time point.*

Definition 7. *If the* L *in a statement is not null, the statement is called a* ***spatial statement****. Otherwise, it is called a* ***non-spatial statement****.*

Definition 8. *If the* T *in a statement is not null, the statement is called a* ***temporal statement****. Otherwise, it is called a* ***non-temporal statement****.*

Definition 9. *An* ***S-T triple pattern*** *is a five-tuple* $\langle s, p, o, L, T\rangle$*, where* s*,* p*,* o*,* L*, and* T *represent* $subject$*,* $predicate$*,* $object$*,* $location$*, and* $time\ interval$ *respectively. In contrast to the S-T statement, each item of an S-T triple pattern can be replaced by a variable. An S-T statement S is called a match of an S-T triple pattern P if the nonvariable items are the same in S and P. The variable items in P are mapped to the corresponding items in S.*

Definition 10. *An* ***S-T query*** *is a list of S-T triple patterns with some spatial and temporal filtering conditions. If there is neither spatial nor temporal filtering condition, the S-T query is degraded to a traditional SPARQL query.*

Definition 11. *The spatiotemporal filtering conditions are represented as spatiotemporal assertions in this paper.*

Given an S-T query Q, the spatial assertions are expressed as an expression that $distance(place(a), place(b)) < d$*, where "distance" and "place" are reserved words in* gst*-store,* a *and* b *are variables in Q or a specific geometry point, and* d *is a constant given by the user.*

The temporal assertions are expressed as $time(a) < time(b) \pm Xyear\text{-}Ymonth\text{-}Zday$ *or* $time(c) < y\text{-}m\text{-}d$*. Here,* X*,* Y*,* Z *and* y*,* m*,* d *are parameters given by the user to de-*

	Subject	Predict	Object	Location(x,y)	Start Time	End Time
#1	Albert_Einstein	BornIn	Ulm	(48.39841,9.99155)	1879-03-14	1879-03-14
#2	Albert_Einstein	hasChild	Eduard_Einstein			
#3	Albert_Einstein	type	American_physicists			
#4	Albert_Einstein	type	German_physicists			
#5	Albert_Einstein	type	German_vegetarians			
#6	Albert_Einstein	WonPrize	Nobel_Prize	(59.35,18.0667)	1921-##-##	1921-##-##
#7	Albert_Einstein	hasName	"Albert Einstein"		1879-03-14	1955-04-18
#8	Ulm	type	city	(48.39841,9.99155)		
#9	Ulm	isCalled	"Ulm"	(48.39841,9.99155)		
#10	Ulm	LocatedIn	Baden-Württemberg		1181-##-##	1803-##-##
#11	Ulm	type	Populated_place	(48.39841,9.99155)		
#12	Baden-Württemberg	hasCapital	Ulm			
#13	Baden-Württemberg	type	state	(48.5,9.0)		
#14	Baden-Württemberg	LocatedIn	Germany			
#15	Nobel_Prize	type	Academic_awards			
#16	Nobel_Prize	hasName	"Nobel Prize"			
#17	Fanny_Normann	diedIn	Ulm	(48.39841,9.99155)	1951-##-##	1951-##-##
#18	Fanny_Normann	hasName	"Fanny Normann"			
#19	Fanny_Normann	type	actor			
#20	Clemens_Betzel	diedIn	Gdańsk	(54.35,18.66667)	1945-03-27	1945-03-27
#21	Clemens_Betzel	hasGender	male		1895-06-09	1945-03-27
#22	Clemens_Betzel	BornIn	Ulm	(48.39841,9.99155)	1895-06-09	1895-06-09
#23	Clemens_Betzel	type	People_from_Ulm			
#24	Gdańsk	type	city	(54.35,18.66667)		
#25	Gdańsk	Population	455830			

Fig. 2: Spatiotemporal RDF Data

note the values of year, month and day; a, b, c are variables in Q, and "time" and "year-month-day" are reserved words in gst-store. Note that the compare symbol "<" can be replaced by "=" or ">".

Figure 2 shows a subset of a spatiotemporal RDF data set. *Ulm*, *Baden-Württemberg*, and *Gdańsk* are spatial entities. Some statements are spatial statements, such as #1, #2, and #6, and some statements are temporal statements, such as #10, #17, and #20. Besides, there are a lot of nonspatial entities, as well as nonspatial and nontemporal statements. For example, people have no spatial information since we cannot locate a person on the map. Similarly, statements such as $\langle People\ hasName\ Name\rangle$ are nonspatial and non-temporal statements. In gst-store, we use "S-T assertion" to represent the spatiotemporal constraints in S-T queries. For example, in Example 1, the filtering conditions list four kinds of spatiotemporal constraints, where $place(?y), place(49, 10)) < 300$ is a spatial range constraint, $distance(place(?l2), place(?y)) < 1500$ is a spatial join constraint, $time(?t1) < date(1900.01.01)$ is a temporal range constraint, and $time(?t2) < time(?t1) \pm (50year - 01month - 01day)$ is a temporal join constraint. In this stage, we support (i) the spatial range query and the spatial join semantics for spatial entities and statements, and (ii) the temporal range query and the temporal join semantics for temporal statements.

In practice, we use $place(?x)$ to denote the spatial label of variable $?x$. Also, $distance(a, b)$[6]$< r$ denotes that the distance between a and b should be below the threshold r, where a and b should be a specific location or a variable. If either a or b is a constant, the constraint is called a *spatial range assertion*. If both a and b are variables, the constraint is called a *spatial join assertion*. Note that a spatial query can have range assertions and spatial join assertions at the same time.

Similarly, we use $time_{start}(?x)$ and $time_{end}(?x)$ to denote the T_s and T_e features of variable $?x$ respectively. Note that $time(?x)$ denotes that both T_s and T_e should satisfy the constraints. In the temporal assertions, we use "$a < b$", "$a = b$" and "$a > b$" to denote the time order of a and b, where a, and b are either a temporal feature or a time point. If either a or b is a time point, the constraint is called a *temporal range assertion*. If both of a and b are expressions that include variables, the constraint is called a *temporal join assertion*.

For instance, the Example 1 is an S-T query including the spatial range assertion, the spatial join assertion, the temporal range assertion and the temporal join assertion at the same time.

6 In this paper, for the ease of presentation, we adopt the Euclidean distance between two locations. Actually, we can use "the earth's surface distance" to define the distance between two locations based on latitudes and longitudes.

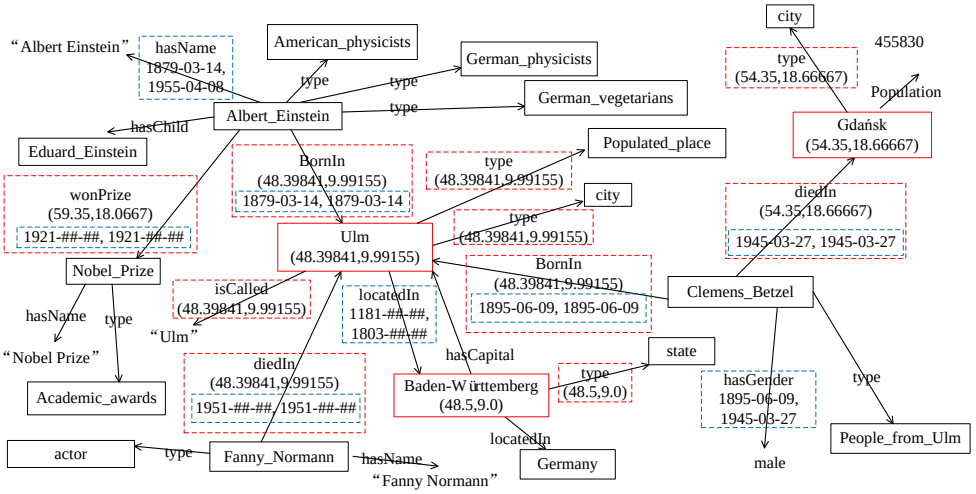


Fig. 3: Spatiotemporal RDF Graph

The S-T RDF data set and the S-T query can be also modeled as graphs (Definitions 12 and 13). The query processing is to find the matches (Definition 14) of an S-T query graph Q in an S-T RDF data graph G. Figure 3 shows the graph corresponding to the S-T RDF data set in Figure 2, where the spatial entities and the spatial statements are all surrounded by red rectangles, and the temporal statements are surrounded by blue rectangles. Note that if a temporal statement is already surrounded by a red rectangle, we only surround the temporal feature of the statement with a blue rectangle.

Definition 12. *The S-T RDF data graph is denoted as* $G = \langle V, E, L_V, L_E, S_V, S_E, T_V, T_E \rangle$, *where*

(1) V, E, L_V, L_E *is the same as in Definition 2.*

(2) S_V *and* S_E *represent the spatial labels of* V *and* E *respectively, where the spatial labels denote the position of the entity (the event), i.e., the latitude and longitude (only valid for spatial entities and spatial statements).*

(3) T_V *and* T_E *represent the temporal labels of* V *and* E *respectively, where the temporal labels denote the time interval when the entity (the event) occurs, i.e., the start time and the end time.*

Definition 13. *The S-T SPARQL query graph is denoted as* $Q = \langle V, E, L_V, L_E, SC_V, SC_E, TC_V, TC_E \rangle$, *where*

(1) V, E, L_V, L_E *is the same as in Definition 3.*

(2) SC_V *and* SC_E *represent the spatial assertions of* V *and* E *respectively, where the spatial assertions can be an absolute area or the relative position for some parameter.*

(3) TC_V *and* TC_E *represent the temporal assertions of* V *and* E *respectively, where the temporal assertions can be an absolute time interval or the relative relation for some parameter expressions, such as ">" "=" and "<".*

Definition 14. *Consider an S-T RDF graph* G *and an S-T query graph* Q *with* n *vertices* $\{v_1, \ldots, v_n\}$. *A list of* n *vertices* $\{u_1, \ldots, u_n\}$ *in* G *is said to be a match of* Q *if and only if the conditions in Definition 4 and the following conditions hold:*

(1) If $v_i \in V_p$, *the spatial label* $S(u_i)$ *must satisfy the spatial assertion* $SC(v_i)$, *and the temporal label* $T(u_i)$ *must satisfy the temporal assertion* $TC(v_i)$;

(2) If there is an edge $\overline{v_i v_j}$ *from* v_i *to* v_j *in* Q , *there is also an edge* $\overline{u_i u_j}$ *from* u_i *to* u_j *in* G. *If* $\overline{v_i v_j}$ *has spatial(temporal) assertions,* $\overline{u_i u_j}$ *must have the corresponding spatial(temporal) label that satisfies the spatial(temporal) assertions.*

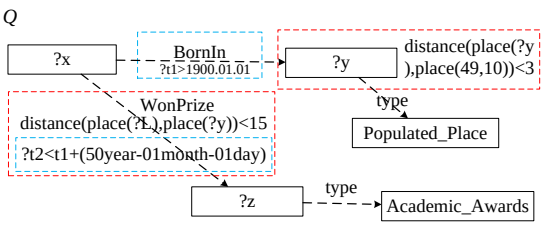


Fig. 4: The Graph View of Q

We show the graph view of Q in Figure 4. We can find that there is a match of Q in the S-T RDF data graph satisfying all the constraints of Q, where the result of $?x$, $?y$ and $?z$ is "Albert_Einstein", "Ulm", and "Nobel_Prize", respectively.

3.3 S-T Signature Graph

In gst-store, we use a bit string[7], a minimum bounding rectangle (MBR), of a spatial feature (the coordinates) and a segment[8] of a temporal feature (the time interval) to denote an entity. The bit string is called a signature. The original S-T RDF graph is converted to an S-T signature graph in gst-store.

The signature sig of each subject s depends on all the edges $\{e_1, e_2, \ldots, e_n\}$ adjacent to s. For each e_i, a list of hash functions are used to generate a signature $sig.e_i$, where the front N bits denote the predicate, and the following M bits denote the object. The valid bits (i.e., the bits with value "1") depend on the hash codes of the corresponding textual information. For instance, suppose that we use two hash functions for the predicates and two hash functions for the URI/literals, and N and M are both set to be 5. Here, for the edge (statement) $Ulm\ isCalled$ "Ulm", the hash codes of the predicate $isCalled$ are 1 and 5 and the hash codes of the literal value "Ulm" are 2 and 4 based on the hash functions. Therefore, the edge is represented as 10001 01010 in Figure 5, where the first 5 bits represent the predicate $isCalled$, and the last 5 bits represent the literal value "Ulm". The signature sig of s is $sig = sig.e_1|sig.e_2|\ldots|sig.e_n$, where $sig.e_1, sig.e_2, \ldots, sig.e_n$ are the out-edges of s.

For example, in Figure 2, there are four edges starting from Ulm (#8, #9, #10, and #11). Suppose that we set the first five bits for the predicate and the following five bits for the object, we can get four signatures 0001101000, 1000101010, 1001000010 and 0001100011 corresponding to the four edges. Thus, Ulm can be represented as 1001101011. Figure 5 shows the encoding processing for "Ulm". Note that only the entity and class vertices in the RDF graph are encoded.

Then, for each vertex (v_i) and each edge e_j, we set the $MBR(v_i)$ and the $MBR(e_j)$ of the entities and the statements, where $MBR(x)$ denotes the MBR of the spatial feature of x. Next, we set the segments $seg(v_i)$ and $seg(e_i)$ in the time axis to denote the time features. Note that the $seg(v)$ of all the entities is $null$ in this stage. Subsequently, for each node v, all the segments $seg(e_i)$ of edge e_i starting from node v are combined as a union segment to denote the temporal feature of the node v, i.e., $seg(v) = \cup seg(e_i)$.

Given an S-T query Q, Q can also be easily transformed into an S-T signature query $Q^\star$ based on the upper conversion method. We define the match of $Q^\star$ in the S-T signature RDF graph as follows. It can be easily derived that each match (Definition 14) of Q in G corresponds to a match (Definition 15) of $Q^\star$ in $G^\star$.

Definition 15. *Given an S-T signature graph $G^\star$ and an S-T signature query graph $Q^\star$ with n signature vertices $\{q_1, \ldots, q_n\}$, a set of distinct signature vertices $\{sig_1, \ldots, sig_n\}$ in $G^\star$ is a match of $Q^\star$ if and only if the following conditions hold:*

1. $\forall q_i$, $sig_i.signature \& q_i.signature = q_i.signature$;

2. $\forall q_i$, the spatiotemporal labels of sig_i must satisfy the spatiotemporal assertions;

3. If there is an edge $\overline{q_iq_j}$ from q_i to q_j in $Q^\star$, there is also an edge $\overline{sig_isig_j}$ from sig_i to sig_j in $G^\star$, and $\overline{q_iq_j}.signature \& \overline{sig_isig_j}.signature = \overline{q_iq_j}.signature$. If $\overline{q_iq_j}$ has spatial(temporal) assertion, $\overline{sig_isig_j}$ must have the spatial(temporal) label that satisfies the assertion.

4 Overview of gst-store

gst-store uses a hybrid index that integrates both R-tree (Guttman, 1984) and VS-tree (Zou, Mo, Chen, Özsu, & Zhao, 2011). Therefore, the pruning strategies of R-tree and VS-tree are also integrated as the searching strategy of gst-store. Our framework consists of the preprocessing, the index construction and the query processing stages.

In the preprocessing stage, we first encode each vertex and edge as a bit string (we call it a *signature*). Subsequently, we build the *S-T signature graph* $G^\star$. Figure 6 shows a running example. In Figure 6, the entities or the statements surrounded by the dotted rectangles have the spatial feature or the temporal feature. The spatial features are represented as red "$MBR(\cdot,\cdot)$", and the temporal feature are represented as blue "####-##-##". Since the nodes that have no out-edges are not encoded, they are not taken into consideration in the S-T signature graph.

In the index construction stage, we construct a tree-style index based on the S-T signature graph to effectively reduce the search space. The index is called *ST-tree*. Figure 7 shows an running example. The nodes on the same level of the ST-tree form an S-T signature graph. If there's a match of a query Q in a lower S-T signature graph, there

7 the bit string is also used in our previous work gStore (Zou, Mo, Chen, Özsu, & Zhao, 2011).

8 A segment is a 1-dimensional MBR.

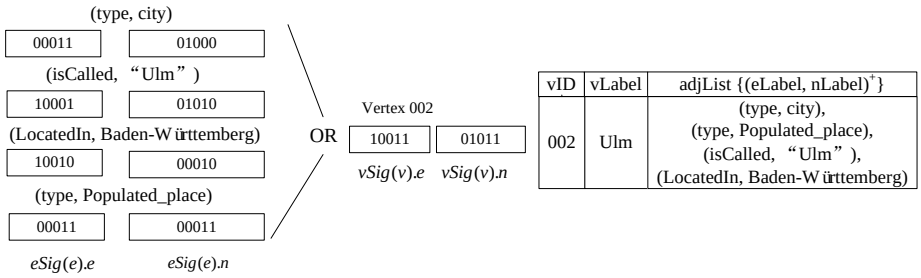


Fig. 5: Encoding Technique

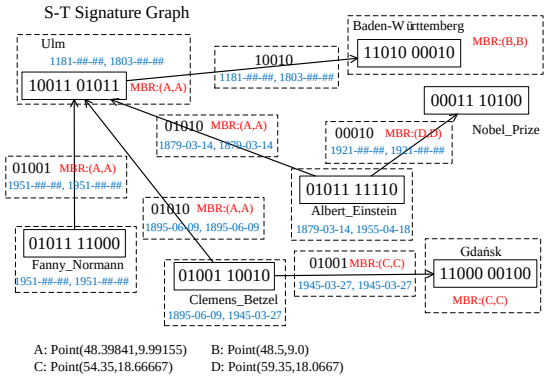


Fig. 6: S-T Signature Graph

must be a corresponding match in each higher S-T signature graph. Therefore, we need to guarantee that ST-tree is a height-balanced tree.

In the query processing stage, given a query graph Q, we first convert Q into the *S-T signature query graph* $Q^\star$. Figure 8 shows the S-T signature query graphs of the example Q in Section 3.2. In Figure 8, the edges and the nodes are encoded, and the spatiotemporal constraints are added to the edges and nodes. Note that if there is a set of vertices in G that matches a query graph Q, there must be a corresponding match in $G^\star$ of $Q^\star$. Subsequently, we implement a top down searching algorithm over the ST-tree to find the matches of $Q^\star$ in $G^\star$. Finally, we retrieve the corresponding textual result and return it to the user.

5 Index Construction

In this section, we introduce our S-T RDF index ST-tree. The index is presented in a tree style. Generally speaking, we build the ST-tree based on the VS-tree and the R-tree. The ST-tree is used to generate the candidates for the variables.

5.1 The ST-tree Structure

The ST-tree is a hybrid tree style index combining the VS-tree (Zou, Mo, Chen, Özsu, & Zhao, 2011) and the R-tree (Guttman, 1984). The VS-tree is an extension of the S-tree (Deppisch, 1986). As shown in Figure 7, the ST-tree is a height balanced n-ary tree, and each level of the ST-tree comprises an S-T signature graph. The leaves of the ST-tree and the corresponding edges between the leaves comprise the S-T signature RDF data graph, and the inner nodes of the ST-tree obey the ST-tree rule.

- *ST-tree Rule:* Consider two S-T signature nodes v_1 and v_2 and their father nodes n_1 and n_2. The following conditions hold:
  (1) $n_1.sig \& v_1.sig = v_1.sig$, $n_2.sig \& v_2.sig = v_2.sig$;
  (2) $v_1.MBR \subseteq n_1.MBR$, $v_2.MBR \subseteq n_2.MBR$;
  (3) $v_1.seg \subseteq n_1.seg$, $v_2.seg \subseteq n_2.seg$;
  (4) If there is an edge $\overline{v_1v_2}$ between v_1 and v_2, there must be an edge $\overline{n_1n_2}$ between n_1 and n_2, where $\overline{n_1n_2}.sig \& \overline{v_1v_2}.sig = \overline{v_1v_2}.sig$, $\overline{v_1v_2}.MBR \subseteq \overline{n_1n_2}.MBR$ and $\overline{v_1v_2}.seg \subseteq \overline{n_1n_2}.seg$, even if $n_1 = n_2$.

The ST-tree rule ensures that the upper-level S-T signature graph is a summary graph of the lower-level S-T signature graph, i.e., each node/edge in the upper level is the union of its descendants. For example, the node d_3^3 is the father of the nodes *Ulm* and *Baden-Württemberg*. Thus, the signature of d_3^3 is 1101101011, which is the union of the signatures of *Ulm* and *Baden-Württemberg*, 1001101011 and 1101000010. The spatial MBR and the temporal interval of d_3^3 are also the union of the corresponding features of the nodes *Ulm* and *Baden-*

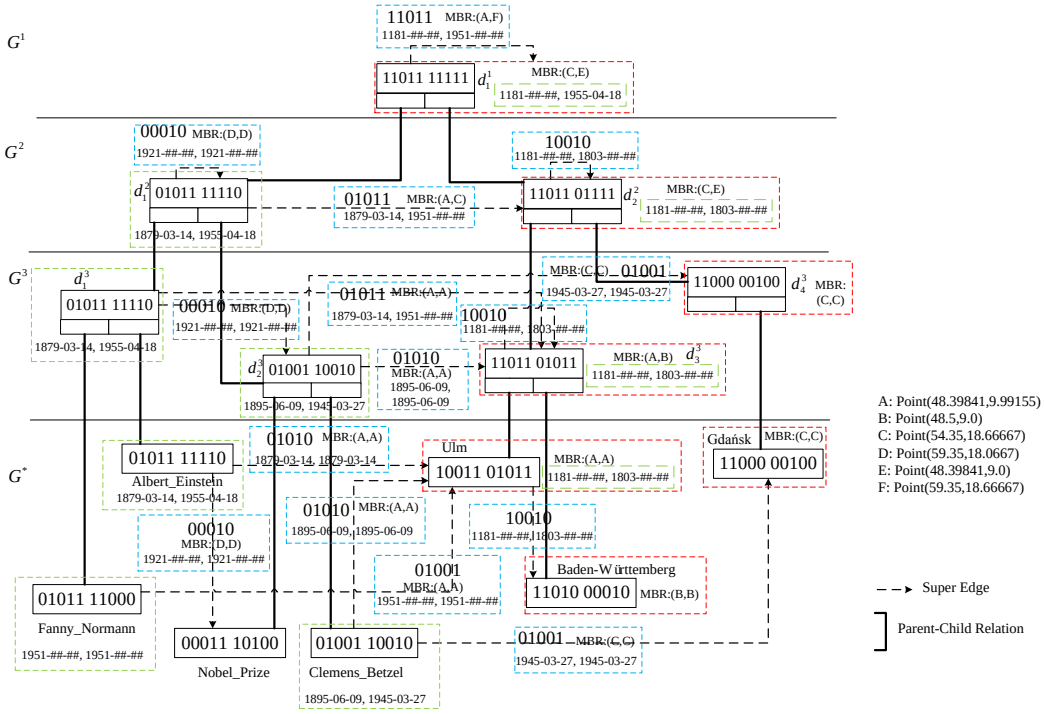


Fig. 7: ST-tree

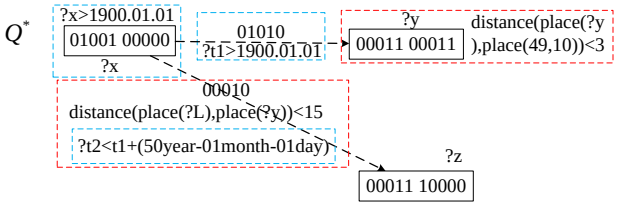


Fig. 8: The S-T Signature Graph Q^* of Q

Württemberg. In the ST-tree, given levels i and $i \pm 1$[9], we call the S-T signature graph $G_i^\star$ in level i as the summary graph of the S-T signature graph $G_{i+1}^\star$ in level $i \pm 1$, and $G_{i+1}^\star$ is the expanded graph of $G_i^\star$.

Theorem 1. *Given an S-T signature query $Q^\star$ and level i, if there is a match of $Q^\star$ in the S-T signature graph $G_{i+1}^\star$ in level $i \pm 1$, there should be a corresponding match in $G_{i+1}^\star$'s summary graph $G_i^\star$.*

Proof. Suppose the match of $Q^\star$ $\{q_1, \ldots, q_n\}$ in $G_{i+1}^\star$ is $\{v_1^{i+1}, \ldots, v_n^{i+1}\}$. Let $v_1^i, \ldots, v_n^i$ denotes the corresponding father node set of $\{v_1^{i+1}, \ldots, v_n^{i+1}\}$, i.e., v_j^i is the father node of v_j^{i+1}. Based on Definition 15 and the ST-tree Rule, $v_j^i.sig \& q_j.sig = v_j^i.sig \& v_j^{i+1}.sig \& q_j.sig = v_j^{i+1}.sig \& q_j.sig = q_j.sig$. Since $v_j^{i+1}.MBR \subseteq v_j^i.MBR$ and $v_j^{i+1}.seg \subseteq v_j^i.seg$, if v_j^{i+1} satisfy the spatiotemporal assertions, v_j^i also satisfies the spatiotemporal assertions. Similarly, if there is an edge $\overline{q_x q_y} \in Q^\star$, the corresponding edge $\overline{v_x^{i+1} v_y^{i+1}} \in G_{i+1}^\star$ is a sub edge of $\overline{v_x^i v_y^i} \in G_i^\star$. All constraints are satisfied, i.e., $\{v_1^i, \ldots, v_n^i\}$ is a match of $Q^\star$ in $G_i^\star$. □

Theorem 1 gives the correctness guarantee. If the ST-tree can be separated into several layers, i.e., the ST-tree is a height-balanced tree, the tree nodes in the upper layer can be safely pruned if the signature, the MBR, or the segment is unsatisfied.

5.2 ST-tree Construction

The ST-tree is constructed over the S-T signature graph. In Section 3.3, we have described the generation of the S-T signature graph. Each node in the S-T signature graph has three features: the signature, the spatial MBR and the temporal segment. Based on these three features, we can build an S-tree, an R-tree of spatial information and

9 The root node is on level 0, and level i denotes that the depth is i.

Algorithm 1 ST-tree Construction

Require: ST-tree T, max node size N, root r of T, entity set $E = \{e_1, e_2, \ldots, e_n\}$.
Ensure: ST-tree T.
1: $T.clear()$
2: **for** $e_i \in E$ **do**
3: Set entry $e = null$
4: $e.id = e_i.name$
5: $e.sig = e_i.signature$
6: $e.MBR = MBR(Point(e_i.x, e_i.y), Point(e_i.x, e_i.y))$
7: $e.seg = segment(T_s, T_e)$
8: $insert(T, N, r, e)$
9: **end for**

Algorithm 2 $insert(T, N, r, e)$

Require: ST-tree T, max node size N, root r of T, entry e.
Ensure: ST-tree T.
1: Set node $n = r$
2: **while** $n.isLeaf() = false$ **do**
3: $n.sig = n.sig|e.sig$
4: $n.MBR = n.MBR \odot e.MBR$
5: $n.seg = n.seg \otimes e.seg$
6: Set node $next = null$
7: Set $minCost = \infty$
8: **for** each child n_i of n **do**
9: **if** $cost(n_i, e) < minCost$ **then**
10: $minCost = cost(n_i, e)$
11: $next = n_i$
12: **end if**
13: **end for**
14: $n = next$
15: **end while**
16: $n.sig = n.sig|e.sig$
17: $n.MBR = n.MBR \odot e.MBR$
18: $n.seg = n.seg \otimes e.seg$
19: **if** $n.isFull()$ **then**
20: $split(T, N, r, n)$
21: **end if**

an R-tree of temporal information respectively. In the ST-tree, we integrate the three trees with different features.

We use the "insert" operation to build the ST-tree. Given a list of S-T signature nodes, we insert the nodes one by one into the ST-tree. Since the ST-tree is a height-balanced n-ary tree, we implement a similar "insert and split" strategy as for other height-balanced n-ary trees, such as B$^+$-tree, R-tree, S-tree, and so on. When a node n comes, the strategy works as follows.

1. Iteratively choose the node from top-down manner with the lowest cost when inserting n into it. If the chosen node v is a leaf, insert n into v.
2. If v is full, split v into two separate nodes v_1 and v_2, where the costs of v_1 and v_2 are minimized. If a splitting operation makes the father of the split node become full, split the father node iteratively.
3. If the root is split to r_1 and r_2, set a new root r and make r to be the father of r_1 and r_2.

Since the R-tree and the VS-tree have similar cost model, we can adopt a cost model while constructing the ST-tree by integrating the independent cost models of the R-tree and the VS-tree.

The first cost in our model is the signature (bit string) cost. The signature cost represents the dissimilarity of two signatures. Equation 1 shows how to compute the signature cost when given signatures of tree nodes sig_1 and sig_2, where $Cost_{sig}$ denotes the signature cost, $bit_{count}(sig)$ counts the number of the valid bits, and $\oplus$ means the xor operation.

$$Cost_{sig} = bit_{count}(sig_1 \oplus sig_2) \tag{1}$$

The second cost in our model is the spatial cost. While two entries are combined, the spatial cost is the increasing area of the MBR. In order to avoid ineffective insertion or splitting, we use the area of the rectangle's circum circle[10] instead of the area of the original rectangle. Equation 2 shows how to compute the spatial cost while combining the tree nodes n_1 and n_2, where $Cost_{spa}$ denotes the spatial cost, $Area(S_i)$ means the area of the the rectangle's circum circle, and $d(MBR_i)$ denotes the diameter of the tree node n_i's MBR_i . Note that $R_i \odot R_j$ denotes a binary operator to generate a rectangle surrounding the rectangles R_i and R_j.

$$\begin{aligned} Cost_{spa} &= Area(R_1 \odot R_2) \\ &\quad - Area(R_1) - Area(R_2) \\ &= \frac{\pi}{4}(d(MBR_1 \odot MBR_2)^2 \\ &\quad - d(MBR_1)^2 - d(MBR_2)^2) \end{aligned} \tag{2}$$

The third cost in our model is the temporal cost. The temporal cost of two entities e_1 and e_2 is the increased length while combining e_1 and e_2. Equation 3 shows how to compute the temporal cost while combining the tree nodes n_1 and n_2, where $seg_1 \otimes seg_2$ denotes the time interval surrounding seg_1 and seg_2.

$$\begin{aligned} Cost_{tem} &= 2 \times length(seg_1 \otimes seg_2) \\ &\quad - length(seg_1) - length(seg_2) \end{aligned} \tag{3}$$

Since each tree node owns both spatiotemporal and signature features, we take both the spatiotemporal cost

10 For the two-dimensional situation.

Algorithm 3 $split(T, N, r, n)$

Require: ST-tree T, max node size N, root r of T, full node n.
Ensure: ST-tree T.
1: Set $S = n$'s children set.
2: Find $n_1, n_2 \in S$ s.t. $\forall n_i, n_j \in S, cost(n_i, n_j) < cost(n_1, n_2)$.
3: Set $seed_1 = n_1$, $seed_2 = n_2$ and $S_1 = \emptyset$, $S_2 = \emptyset$.
4: **while** $S_1.size < N/2$ and $S_2.size < N/2$ **do**
5: Let $diff(n_i)$ denotes $cost(n_i, n_1) - cost(n_i, n_2)$.
6: Find $n_i \in S$ s.t. $\forall n_j \in S, |diff(n_j)| < |diff(n_i)|$.
7: Move n_i to S_1 if $diff(n_i) < 0$, and vice versa.
8: **end while**
9: **if** $S.size > 0$ **then**
10: Put all $n_i \in S$ into the smaller S_i, $S.clear()$.
11: **end if**
12: Allocate new node n'_1, n'_2.
13: Set n'_1's children is S_i, $i = 1, 2$.
14: **if** $n == r$ **then**
15: Allocate new node n_r as $root$.
16: Set n'_1, n'_2 and n_r's children, Delete n.
17: **else**
18: Set $n_p = n.parent$ and delete n.
19: $n_p.setAsChild(n'_1), n_p.setAsChild(n'_2)$
20: **if** $n_p.isFull()$ **then**
21: $split(T, N, r, n_p)$
22: **end if**
23: **end if**

and the signature cost into account when inserting an entity or splitting a full node. The cost of combining two nodes is shown in Equation 4, where $0 < \alpha < 1$, $0 < \beta < 1$, and $0 < \alpha + \beta < 1$. Note that $Z_{spa} = \sum Cost_{spa}$, $Z_{tem} = \sum Cost_{tem}$, and $Z_{sig} = \sum Cost_{sig}$ are the normalized parameters to balance the scale of the spatial cost, the temporal cost and the signature cost respectively. In the section of the experiments, we design a specific experiment to determine the values of the parameters α and β.

$$Cost = \alpha \frac{Cost_{spa}}{Z_{spa}} + \beta \frac{Cost_{tem}}{Z_{tem}} + (1 - \alpha - \beta) \frac{Cost_{sig}}{Z_{sig}} \quad (4)$$

Based on the cost model, we propose a method to construct an ST-tree. Algorithm 1 shows the procedure to build an ST-tree given a set of entities. In the very beginning, the ST-tree only has a empty root, and we set the maximum node size of the ST-tree. Given a set of entities, we iteratively insert the entities one by one into the ST-tree. While inserting an entity, we generate an entry representing the entity and insert the entry into the node with the lowest cost in a top-down manner. If the insertion produces a full node, we split the full node into two half full nodes. Note that the splitting operation may produce a new full node. If the root needs to be split, we generate a new node, and then we set the new node as the root of the ST-tree and set the two split nodes as the new node's children.

5.3 Analysis of the ST-tree

5.3.0.1 Guarantee of Balance

Based on Theorem 1, the ST-tree should be a height-balanced tree. Since the ST-tree is built based on the "insert" and "split" operations, it can be proven that the ST-tree is a height-balanced tree.

Lemma 1. *Given a height-balanced tree T, T is also balanced after splitting a node n to n_1 and n_2 using the "split" operation.*

Proof. (Sketch) Since T is a height-balanced tree, the subtree T_n rooted at n is also a balanced tree, and the subtrees rooted at n's children are balanced tree too. Based on the definition of the height-balanced tree, the new trees T_1 and T_2 rooted at n_1 and n_2, respectively are both height balanced trees since the children of $n_1(n_2)$ are the subsets of n's children.

Clearly, the height of T is the bigger one of T_1's height and T_2's height, and the difference between T_1's height and T_2's height is at the most 1. Therefore, the new tree T'_f rooted at n's father after splitting remains to be a height balanced tree, and the height of T'_f is unchanged.

Since the remaining part of T is unchanged, if n is not the root of T, T remains height balanced after the splitting.

If n is the root of T, a new root r is set to be the father of n_1 and n_2. Since T_1 and T_2 are height balanced and the height difference between T_1 and T_2 is no more than 1, the new tree is also height balanced. □

Lemma 2. *Given a height-balanced tree T and a new node n, T is also a height-balanced tree after inserting n into T.*

Proof. (Sketch) n is added to the lowest layer of T via the "insert" operation. Then,

(1) the father node of n is not full, i.e., the insertion does not cause splitting procedure. Clearly, T is also a height-balanced tree since the height of T is unchanged and the depth of each node in T is unchanged except for n.

(2) the father node of n is not full, i.e., a splitting procedure is triggered. Based on Lemma 1, T is also a height-balanced tree after the necessary splitting steps.

In summary, T remains height balanced after inserting n into T. □

Theorem 2. *An ST-tree T is a height-balanced tree.*

Proof. (1) Clearly, an empty ST-tree is a height-balanced tree.

(2) Based on Lemma 2, if an ST-tree is height balanced, it remains height balanced after an insertion.

In summary, T is a height balanced tree since the construction of T is a series of insertions. □ □

In this stage, we ignore the "update" and the "delete" operations of the ST-tree because (1) in contrast to occasionally removing/editing statements, the real RDF data sets prefer to increase their scales, i.e., insertion is more important, and (2) we can use time stamps to manage the changed statements. In fact, it is easy to design the "update" and the "delete" operations of the ST-tree by referring to the same operations of the R-tree (or B$^+$-tree, S-tree, and so on).

5.3.0.2 Time Complexity

The ST-tree construction is a series of "insert" and "split" operations. Therefore, the time complexity of the tree construction depends on the time complexity of the two operations and the number of times that the operations are triggered. Suppose that the node capacity is set to be k, i.e., the tree nodes in the ST-tree have no more than k children without splitting. In other words, if a tree node has $k + 1$ children after an "insert" or a "split" operation, the node should be split.

Based on Algorithm 2, given an ST-tree T with height h, the time cost of an "insert" operation is $O(h \times k)$, where an insertion needs h times of comparison to find the lowest cost path, and $O(k)$ times of cost computation with all the children of the chosen node in each comparison. Suppose that an RDF data set has n entities, the time cost of the insertion is $O(n) \times O(h \times k) = O(n) \times O(\lg n \times k) = O(nk \lg n)$. Note that an insertion introduces a new node to the ST-tree.

Based on Algorithm 3, it takes $O(\frac{k\times(k-1)}{2} + (k-1)(k-2)+k) = O(k^2)$, where $O(\frac{k\times(k-1)}{2})$ is the time cost to find the two seeds, $O((k-1)(k-2))$ is the time cost to separate the children of the split node, and $O(k)$ is the time cost to allocate the children to the two new nodes. Obviously, a new node is added to the ST-tree if a "split" operation is executed[11]. Therefore, the number of executions of the "split" operation is equal to $N - n - (h - 2)$, where N is the node count of the ST-tree, n is the entity node count and h is the height of the ST-tree. Since n is fixed and h is far less than N (the maximum h is $\log_2 N$ when the ST-tree degenerates to a binary tree), the total time cost of the "split" operation is $O(N \times k^2)$. In the best case, the tree node are all full, i.e., $N = \frac{n}{k} + \frac{n}{k^2} + \ldots + 1 = O(\frac{n}{k-1})$. In the worst-case scenarios, the tree nodes are all half-full and the roots have only two children, i.e, all the insertions are focused on the same path. Thus, $N = O(2 \times \frac{n}{k-2} + 1)$. In summary, $N = O(\frac{n}{k})$, and the time complexity of the splits is $O(N \times k^2) = O(nk)$.

As a result, the time complexity of the tree construction is $O(nk \lg n) + O(nk) = O(nk)$, where n is the entity number in the data set, and k is the node capacity of the ST-tree.

6 Query Processing

Given an S-T query Q, we first convert the Q to an S-T signature graph $Q^\star$. The conversion process consists of three steps.

(1) Encode the triple patterns as described in Section 3.3.
(2) For each spatiotemporal range assertion, we add the corresponding absolute MBR or segment on the specific variables.
(3) For each spatiotemporal join assertion, we add the relevant MBRs or segments on the variables.

The $Q^\star$ corresponding to Q is shown in Figure 8. The signatures are generated as G to $G^\star$, where the variables contribute no valid bit. The range assertions of Q are converted to the absolute MBRs binding $?y$ in $Q^\star$, and the join assertions of Q are converted to the relevant MBRs in $Q^\star$. Specifically, if an out-edge from node n has temporal range assertions, we add the assertions on n, which is called "infection".

After the corresponding $Q^\star$ is generated, we next search the matches of $Q^\star$ in $G^\star$ exploiting the ST-tree. Considering an S-T signature query graph $Q^\star = \{q_1, \ldots, q_n\}$, we first generate the node candidate set $NodeSet_i$ for each variable q_i, and then verify each candidate in the query candidate set $QSet = \{NodeSet_1 \times \ldots \times NodeSet_n\}$ to generate the matches of Q in G.

6.1 Pruning Rules

For efficiently generating the node candidate sets, we have the following five pruning rules. Pruning rules 1 and 2 are based on the spatial range and spatial join assertions respectively. Pruning rule 3 is based on the temporal range assertions. Pruning rule 4 is based on the signature, and Pruning rule 5 considers the edge features. Based on Theorem 1 (in Section 5), when node n is unsatisfied, the subtree rooted at n can be safely pruned.

11 If the split node is the root of the ST-tree, the number of added node is 2.

Algorithm 4 Query Processing

Require: $Q^\star = \langle v_1, \ldots, v_n \rangle$, ST-tree T, root r of T, signature data graph $G^\star$.
Ensure: The node candidate sets $\{NodeSet\}$ of nodes of $Q^\star$ in $G^\star$.

```
1: Set each NodeSet_i = r //initialize the node candidate set.
2: while true do
3:   if ∀NodeSet_i ∈ G* then
4:     return {NodeSet} //the sets contains real data points.
5:   end if
6:   for all NodeSet_i do
7:     NodeSet_i =the children of each node n_i ∈ NodeSet_i
8:     Set MBR_i = ⋃{n|n ∈ NodeSet_i}
9:   end for
10:  for all node n_i ∈ NodeSet_i do
11:    if v_i.sig&n_i.sig = n_i.sig then
12:      remove n from tempNodeSet //pruning rule 4.
13:    end if
14:    if v_i is bound by range assertions then
15:      if intersection(v_i.mbr, n_i.mbr) = ϕ then
16:        remove n from tempNodeSet //pruning rule 1.
17:      end if
18:    end if
19:    if T_v is not null then
20:      if intersection(v_i.seg, n_i.seg) = ϕ then
21:        remove n from tempNodeSet //pruning rule 3.
22:      end if
23:    end if
24:    if ∃e = v_iv_j then
25:      if n_i.neighbour ∩ NodeSet_j = ϕ then
26:        remove n from tempNodeSet //pruning rule 5.
27:      end if
28:    end if
29:    if dist(v_i, v_j) <= l then
30:      if dist(n_i, MBR_j) > l then
31:        remove n from tempNodeSet //pruning rule 2.
32:      end if
33:    end if
34:  end for
35: end while
```

6.1.0.1 Pruning Rule 1

Consider a variable v bound with a range assertion. If there is a tree node n where $v.mbr$ has no intersection with $n.mbr$, the subtree rooted on n can be pruned safely.

For example, $?y$ in $Q^\star$ has a range assertion. Thus, the subtrees rooted at d_1^2 and d_4^3 can be safely pruned, because the spatial features are unsatisfied.

6.1.0.2 Pruning Rule 2

Consider two variables v_i and v_j bound by a spatial join assertion, and $NodeSet_i$ is the candidate set of v_i and $NodeSet_j$ is the candidate set of v_j. Suppose the max distance is set to be $MaxDist$. Let $n_i \in NodeSet_i$; if the distance from MBR of n_i to any node $n_j \in NodeSet_j$ is larger than $MaxDist$, n_i can be safely pruned.

In practice, we combine all the MBRs of the candidates of one variable into one MBR, and the minimal distance between two combined MBRs is considered as the lower bound of each candidate pair. Thus, the time complexity is reduced from $O(m \times n)$ to $O(m + n)$, where m and n are the sizes of two candidate sets respectively.

For example, if the distance between $?x$ and $?y$ is set to be less than 50km, when only node d_3^3 is considered as a candidate of $?x$, d_4^3 can be safely pruned for $?y$, since the distance lower bound from d_4^3 to d_3^3 is much more than 50km.

6.1.0.3 Pruning Rule 3

Consider a variable v, if the temporal assertion is not null and there is a tree node n where $v.seg \cap n.seg = \phi$, the subtree rooted on n can be pruned safely.

For example, $?x$ in $Q^\star$ has a temporal assertion. Thus, the subtree rooted at d_2^2 can be safely pruned for $?x$.

6.1.0.4 Pruning Rule 4

Consider a variable v, if there is a tree node n where $v.sig\&n.sig! = n.sig$, the subtree rooted on n can be pruned safely.

In $Q^\star$, d_4^3 for $?x$, d_1^2 for $?y$ and d_2^3 for $?z$ etc. can be safely pruned.

6.1.0.5 Pruning Rule 5

Consider two linked variables v_i and v_j with an edge $e = \overrightarrow{v_iv_j}$ from v_i to v_j, and $NodeSet_i$ is the candidate set of v_i and $NodeSet_j$ is the candidate set of v_j in the same S-T signature graph. Let $n_i \in NodeSet_i$; if there is no edge from n_i to any node $n_j \in NodeSet_j$, n_i can be safely pruned. Additionally, if there is an S-T assertion on e, the unsatisfied edges are considered nonexistent.

The pruning rule is based on the fact that if there is no satisfied edge from n_i to any node $n_j \in NodeSet_j$, there is no satisfied edge from the descendants of n_i to any descendants of the $n_j \in NodeSet_j$. In practice, given a node n, all the features of the edges starting from n are integrated into one signature, one MBR, and one segment to reduce the time complexity.

Algorithm 4 describes the generation process for the top-down node candidate sets generating process. The use of the pruning rules is shown in Lines 9-21.

6.2 Verification

Algorithm 5 Verification

Require: node candidates $\{NodeSet\}$, $Q^\star = \langle v_1, \ldots, v_n \rangle$, Q, G.
Ensure: the matches $\{M\}$ of Q.

```
Set the match candidate list of Q* L = φ.
for each connected region Q_i ⊆ Q* do
    Select the NodeSet_j with the smallest size in Q_i.
    Set the Q_i's match candidate set M_c^i = φ. //Initialize the match candidate sets.
    for each node n_k ∈ NodeSet_j do
        Run the BFS process from n_k.
        if ∃match candidate m_c^i of Q_i then
            M_c^i.add(m_c^i). //If all edges are valid, it is a match candidate.
        end if
    end for
end for
Set M_c^* = M_c^1 × ... × M_c^k. //The match candidates of Q*.
Set M* = φ. //The matches of Q*.
for each m_c^* ∈ M_c^* do
    if all S-T assertions are valid on m_c^* then
        M*.add(m_c^*).
    end if
end for
Set M = φ. //The matches of Q.
for each m* ∈ M* do
    Get the subgraph m ⊆ G corresponding to m*.
    if all literal constraints are valid on m then
        M.add(m).
    end if
end for
return M.
```

For the node candidate set $\{NodeSet\}$, we generate a list of nodes $\langle v_1, \ldots, v_n \rangle$ from each item of $\{NodeSet\}$, respectively, and verify if $\langle v_1, \ldots, v_n \rangle$ forms the connected regions that correspond to the connected regions in $Q^\star$. If $\langle v_1, \ldots, v_n \rangle$ can form, we consider it a match candidate of $Q^\star$, or we discard it otherwise. The generating process can be accomplished by using a breadth first search (BFS) algorithm starting from the smallest node candidate sets in each connected region. If there is an edge $e = \overline{v_k v_l}$ in $Q^\star$, there must be an corresponding edge.

Given a match candidate $Q_c^\star$ of $Q^\star$, we verify whether all the spatiotemporal assertions are satisfied. The satisfied match candidates are the matches of $Q^\star$. Subsequently, since the encoding technique may bring false-positive error, we verify whether all S-T triple patterns in Q are satisfied given a match of $Q^\star$. The valid candidates are the matches of Q. Then, the matches of Q are returned to users.

7 Experiments

To the best of our knowledge, only YAGO2 Demo (Hoffart et al., 2011) and SPARQL-ST (Perry, Jain, & Sheth, 2011) are available spatiotemporal RDF data management systems. Since the technical details of YAGO2 Demo are not reported, SPARQL-ST is chosen to make a comparison with gst-store. In addition, we also make a comparison between gst-store, a post-processing method, S-store (Wang et al., 2013) and an enterprise system Virtuoso.

Our demo is available at http://59.108.48.17:8080/GStoreWangDong/query.jsp.

7.1 Data Set & Setup

7.1.0.1 Data Set

YAGO2 is a real data set based on Wikipedia, WordNet, and GeoNames. The latest version of YAGO2 has >10 million entities and 440 million statements. We obtain a spatiotemporal RDF data set from YAGO2 by removing some statements that describe the date when another statement is extracted or the uniform resource locator (URL) where another statement is extracted from. The condensed data set has >10 million entities/classes and >180 million statements. More than 7 million entities are spatial entities, >90 million statements are spatial statements, and >28 million statements are temporal statements. Based on YAGO2, we generate 10,557,223 S-T signature nodes, wherein 7,394,075 of them have not null spatial features, and 1,266,865 of them have not null temporal features.

7.1.0.2 Queries and Setup

In order to evaluate our approach, we manually generate 20 sample S-T SPARQL queries that have different features. The sample queries are divided into 10 classes, i.e., Ssimple, SRE, SJE, SS, SC, Tsimple, TR, TJ, TC and ST. We run all queries on a personal computer (PC) server with an Intel Xeon CPU E5645 running at 2.40 GHz and 16 GB main memory. The node capacity is set to be 100, i.e., a node in the ST-tree should have no more than 100 children. Our previous work (Wang et al., 2013) shows that different node capabilities affect the performance little.

- Ssimple: **Simple** queries with **S**patial range assertions of entities.
- SRE: Queries with **S**patial **R**ange assertions of **E**ntities.
- SJE: Queries with **S**patial **J**oin assertions of **E**ntities.
- SS: Queries with **S**patial assertions of **S**tatements.

Tab. 1: The Result Set Size of Queries

	Ssimple1	Ssimple2	SRE1	SRE2	SJE1
S-T Queries	3	1,177	1	10	18
SPARQL Queries ignoring S-T assertions	10,137,491	8,567	36	50	36
	SJE2	SS1	SS2	SC1	SC2
S-T Queries	25	2	23	7	12
SPARQL Queries ignoring S-T assertions	50	36	50	40	50
	Tsimple1	Tsimple2	TR1	TR2	TJ1
S-T Queries	5,513	37	13	4	25
SPARQL Queries ignoring S-T assertions	90,647	8,567	36	50	36
	TJ2	TC1	TC2	ST1	ST2
S-T Queries	7	25	23	8	3
SPARQL Queries ignoring S-T assertions	50	36	50	36	50

- SC: **C**omplex queries with all kinds of **S**patial assertions.
- Tsimple: **Simple** queries with **T**emporal range assertions.
- TR: Queries with **T**emporal **R**ange assertions.
- TJ: Queries with **T**emporal **J**oin assertions.
- TC: **C**omplex queries with all kinds of **T**emporal assertions.
- ST: Queries with all kinds of **S**patio**T**emporal assertions.

Table 1 shows the result set size of each query. In order to illustrate the reason why the postprocessing method (i.e., finding SPARQL query results by ignoring the spatiotemporal assertions and then verifying the candidates by the spatiotemporal assertions) is not efficient, we report the result sizes of all queries discarding the spatiotemporal assertions and the final S-T query result sizes. From Table 1, we observe that the result sizes discarding the spatiotemporal assertions are very large even though the final S-T queries have <10 results, such as Ssimple1. It means that the postprocessing method needs a lot of effort during the verification process.

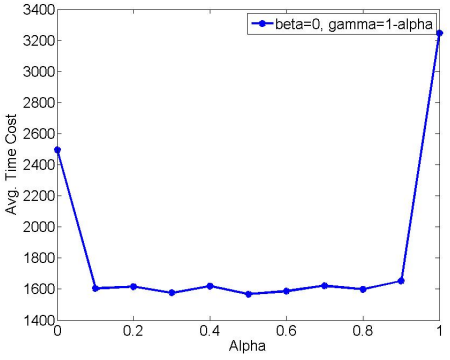


Fig. 9: Average Time Cost of Different Values of α

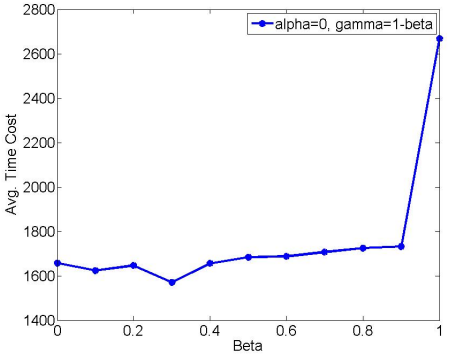


Fig. 10: Average Time Cost of Different Values of β

7.2 Evaluating the Parameters of Cost Model

In this section, we evaluate how does the variations of α and β affects the query performance. Since the combination of α and β can be huge, we just adjust α and β separately, and combine the respective optimal ratios to build the ST-tree. For convenience, we use y to denote $1 - \alpha - \beta$.

First of all, we set β to be zero, i.e., we only focus on spatial information. In order to obtain optimal α, we vary α from 0 to 1 with step size 0.1. The query sets Ssimple, SRE, SJE, SS, and SC are used for adjusting α. We report the average time cost of different α in Figure 9. Based on the performance curve, we set the ratio of y and α to be $5:5$. Second, we set α values to be zero and vary β to choose the best ratio of y and β. The performance of different β values is shown in Figure 10. Note that in this experiment, we use query sets Tsimple, TR, TJ, TC, and ST. Based on the result, the ratio of y and β is set to be $7:3$. Therefore, we use $\alpha = 0.41$, $\beta = 0.18$ and $y = 0.41$ as the optimal cost ratio to build the ST-tree.

7.3 Evaluating Entity Organization

In this section, we evaluate whether different entity organization styles affect the offline and online performances. There are four different tree construction methods, which are ST-tree, VS-tree (Zou, Mo, Chen, Özsu, & Zhao, 2011), R-tree based on the spatial MBR, and R-tree based on the temporal segment. After adding the necessary features and building the S-T signature graphs, all the four kinds of trees can answer S-T queries. In the following, we use VS-tree$^+$ to denote the tree based on VS-tree, R-treeS$^+$ to denote the tree based on spatial R-tree, and R-treeT$^+$ to denote the tree based on temporal R-tree.

Tab. 2: Offline Cost of Tree Construction

Index Style	ST-tree	VS-tree$^+$	R-treeS$^+$	R-treeT$^+$
Index Size(MB)	1,971	2,066	1,975	2,263
Node Count	177,882	186,388	178,178	204,107
Time Cost(s)	1,726	855	773	947

Table 2 shows the offline cost. The ST-tree demands lower storage space than the other three tree styles. Since all the four kinds of tree construction methods can be modeled as cost model-based methods, the result shows that the cost model of the ST-tree is more effective than the others. The last row of Table 2 shows the time cost of the tree construction. Clearly, the more complex the cost model is, the more time cost is incurred to build the tree structure. The ST-tree requires the most time cost. However, only less than half an hour is needed to build the ST-tree. Note that we only consider the tree construction and ignore the S-T signature graph construction.[12]

Tab. 3: On-line Cost of Different Tree styles

Index Style	ST-tree	VS-tree$^+$	R-treeS$^+$	R-treeT$^+$
Average Time(ms)	1,439	2,158	3,503	3,039

Table 3 shows the on-line time cost of the queries based on different tree style indexes. Obviously, the ST-tree based on our cost model outperforms the other tree styles. In other words, the cost model of the ST-tree takes both semantic feature and spatiotemporal feature into consideration, which improves the performance.

12 The time cost of building the whole index is about 18 hours, regardless of the tree style we choose.

7.4 Evaluating Performance

To evaluate the efficiency of our approach, we choose four baseline approaches, which are denoted as gStore$^+$, Virtuoso, S-store$^+$ and SPARQL-ST respectively.

The gStore$^+$ method adopts the postprocessing solution, which runs the SPARQL queries on an RDF query engine by ignoring the spatiotemporal assertions and then refining the candidates by considering the spatiotemporal assertions. In practice, this approach exploits gStore (Zou, Mo, Chen, Özsu, & Zhao, 2011) as the RDF management system, and the node capacity is set to be 100. Besides, MySQL is used to retrieve the spatiotemporal information of the entities and the statements.

The Virtuoso approach is an enterprise system Virtuoso, which declares that the spatial RDF data can be organized using Virtuoso.

The S-store is our early work based on spatial RDF data. The index SS-tree of S-store can be separated into an R-tree based on the spatial entities and a VS-tree based on the nonspatial entities. After simply adding the temporal features in the nodes and edges of the SS-tree, the S-store can be extended to answer S-T queries. Here, we use S-store$^+$ to denote this baseline method.

We implement the method of SPARQL-ST (Perry, Jain, & Sheth, 2011) as the fourth baseline. MySQL is used as the data management system. The B$^+$-tree index and the spatial index of MySQL are used. In this section, we make a comparison between gst-store and the four baselines. The query response times are shown in Table 4.

Since query sets Ssimple and Tsimple have many candidate results (Table 1), the time cost of gStore$^+$ is unacceptable. gStore$^+$ cannot get the results of Ssimple1 or Tsimple1 in reasonable time (half an hour), and the time costs for Ssimple2 and Tsimple2 are >100 seconds. However, our approach (gst-store) can answer these queries efficiently. Although the other queries have just a few candidate results without spatiotemporal assertions, gst-store still outperforms the BASE1 approach.

Actually, only several of the queries(the query sets Ssimple, SRE, and SJE) can be answered using Virtuoso. gst-store outperforms Virtuoso with several orders of magnitude. Here, the mark "-" denote that the query can not be answered.

Compared to the S-store$^+$, gst-store outperforms in all queries except for Ssimple1. In Table 1, we can find that Ssimple1 has low selectivity on the semantic constraint (>90% entities are selected) and high selectivity on the spatial constraint (only three entities are considered as the result). As a result, the S-store$^+$ performs like an R-tree based on the spatial entities (because of the composing

Tab. 4: The Performance Comparison

	Time Cost(s)				
	gst-store	*gStore*$^{+}$	Virtuoso	S-store^{+}	SPARQL-ST
Ssimple1	1.15	>30min	2.79	0.19	0.53
Ssimple2	0.11	112.40	19.84	0.16	>30min
SRE1	0.41	5.89	128.03	0.45	109.33
SRE2	0.69	9.56	>30min	0.78	113.38
SJE1	2.09	4.48	178.31	2.70	157.44
SJE2	2.14	4.13	1727.65	2.74	152.47
SS1	1.98	3.62	-	2.60	-
SS2	2.11	6.75	-	2.70	-
SC1	0.84	5.84	-	1.16	-
SC2	1.39	3,78	-	1.73	-
Tsimple1	1.56	>30min	-	1.92	204.11
Tsimple2	0.07	130.12	-	0.10	156.77
TR1	1.96	4.08	-	2.61	65.29
TR2	2.08	5.14	-	2.70	55.18
TJ1	1.96	4.07	-	2.61	150.44
TJ2	2.11	3.16	-	2.73	160.47
TC1	1.96	4.08	-	2.60	185.97
TC2	2.10	3.46	-	2.69	107.88
ST1	1.16	4.57	-	1.43	-
ST2	1.41	3.70	-	1.75	-

construction method), where gst-store is confused due to the cost model. In other cases, the cost model performs well, and gst-store defeats the S-store^{+}.

The SPARQL-ST approach can answer most of the queries, except query sets SS, SC and ST. This is because these three query sets involve the spatial features of the statements, which are out of the SPARQL-ST data model. For the same reason as the competition with S-store^{+}, the SPARQL-ST approach performs better than gst-store for Ssimple1. In other cases, SPARQL-ST costs several minutes to answer the queries, much slower than gst-store. In summary, gst-store outperforms its competitors in most scenarios.

8 Conclusions

In this paper, we introduce S-T queries, a variant of SPARQL language, to query RDF data with spatiotemporal features. In order to answer S-T queries efficiently, we build a hybrid index, called ST-tree, in our gst-store system, an engine for large RDF graphs integrating spatial and temporal information. Several pruning rules are introduced in the query algorithm to reduce the search space. The experiment results on a real large RDF graph show the effectiveness and the efficiency of our approach.

A The Sample Queries

In this appendix, the 20 queries used in the experiments are shown in Figure 11.

References

Abadi, D. J., Marcus, A., Madden, S. R., & Hollenbach, K. (2007, September). *Scalable semantic web data management using vertical partitioning*. Paper presented at the Proceedings of the 33rd international conference on Very large data bases (pp. 411-422). VLDB Endowment.

Abadi, D. J., Marcus, A., Madden, S. R., & Hollenbach, K. (2009). SW-Store: a vertically partitioned DBMS for Semantic Web data management. *The VLDB Journal, 18(2)*, 385-406. http://dx.doi.org/10.1007/s00778-008-0125-y

Batsakis, S., & Petrakis, E. G. (2010, September). SOWL: spatio-temporal representation, reasoning and querying over the semantic web. *In Proceedings of the 6th International Conference on Semantic Systems* (p. 15). ACM. http://dx.doi.org/10.1145/1839707.1839726

Brodt, A., Nicklas, D., & Mitschang, B. (2010, Novenmber). Deep integration of spatial query processing into native RDF triple stores. *In Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems* (pp. 33-42). ACM. http://dx.doi.org/10.1145/1869790.1869799

Broekstra, J., Kampman, A., & Van Harmelen, F. (2002, June). *Sesame: A generic architecture for storing and querying rdf*

Ssimple1	select ?e where{ ?e type_star wordnet_entity_100001740. }Filter(dist(?e,(48.39,10))≤0.01}	Ssimple2	select ?c where{ ?c type_star wordnet_city_108524735. ?c isLocatedIn_transitive Europe. }Filter(dist(?e,(40,5))≤15}
SRE1	select ?x,?c1,?c2 where{ ?x wasBornIn ?c1. ?x hasWonPrize Nobel_Prize_in_Physics. ?x diedIn ?c2. ?c1 type_star wordnet_city_108524735. ?c2 type_star wordnet_city_108524735. }Filter(dist(?c1,(48,10))≤5 and dist(?c2,(40,-95))≤20}	SRE2	select ?p1,?p2,?c1,?c2 where{ ?p1 isMarriedTo ?p2. ?p1 wasBornIn ?c1. ?p1 type_star wordnet_film_maker_110088390. ?c1 type_star wordnet_city_108524735. ?p2 wasBornIn ?c2. ?p2 type_star wordnet_film_maker_110088390. ?c2 type_star wordnet_city_108524735. }Filter(dist(?c1,(40,-95))≤20 and dist(?c2,(40,-95))≤20}
SJE1	select ?x,?c1,?c2 where{ ?x wasBornIn ?c1. ?x hasWonPrize Nobel_Prize_in_Physics. ?x diedIn ?c2. ?c1 type_star wordnet_city_108524735. ?c2 type_star wordnet_city_108524735. }Filter(dist(?c1,?c2)≤3}	SJE2	select ?p1,?p2,?c1,?c2 where{ ?p1 isMarriedTo ?p2. ?p1 wasBornIn ?c1. ?p1 type_star wordnet_film_maker_110088390. ?c1 type_star wordnet_city_108524735. ?p2 wasBornIn ?c2. ?p2 type_star wordnet_film_maker_110088390. ?c2 type_star wordnet_city_108524735. }Filter(dist(?c1,?c2)≤10}
SS1	select ?x,?c1,?c2 where{ ?x wasBornIn ?c1 ?l1. ?x hasWonPrize Nobel_Prize_in_Physics. ?x diedIn ?c2 ?l3. ?c1 type_star wordnet_city_108524735. ?c2 type_star wordnet_city_108524735. }Filter(dist(?l1,(48,10)) ≤ 10 and dist(?l3,(40,-95)) ≤ 20}	SS2	select ?p1,?p2,?c1,?c2 where{ ?p1 isMarriedTo ?p2. ?p1 wasBornIn ?c1. ?p1 type_star wordnet_film_maker_110088390. ?c1 type_star wordnet_city_108524735. ?p2 wasBornIn ?c2 ?l5. ?p2 type_star wordnet_film_maker_110088390. ?c2 type_star wordnet_city_108524735. }Filter(dist(?l5,?c1) ≤ 10}
SC1	select ?x,?c1,?c2 where{ ?x wasBornIn ?c1. ?x hasWonPrize Nobel_Prize_in_Physics. ?x worksAt ?u ?l3. ?c1 type_star wordnet_city_108524735. ?u type_star wordnet_educational_institution_108276342. }Filter(dist(?c1,(54,5)) ≤ 10 and dist(?l3,?c1) ≤ 5}	SC2	select ?p1,?p2,?c1,?c2 where{ ?p1 isMarriedTo ?p2. ?p1 wasBornIn ?c1. ?p1 type_star wordnet_film_maker_110088390. ?c1 type_star wordnet_city_108524735. ?p2 wasBornIn ?c2 ?l5. ?p2 type_star wordnet_film_maker_110088390. ?c2 type_star wordnet_city_108524735. }Filter(dist(?c1,(40,-95)) ≤ 20 and dist(?l5,?c1) ≤ 10}
Tsimple1	select ?x where{ ?x wasBornIn ?y. ?t ?y type_star wordnet_city_108524735. }Filter{?t.s ≥ 183#-##-## and ?t.e ≤ 187#-##-## }	Tsimple2	select ?c where{ ?c type_star wordnet_location_100027167. ?c isLocatedIn_transitive Europe. ?c wasCreatedOnDate ?x ?t }Filter{?t.s ≤ 0-##-##}
TR1	select ?x,?c1,?c2 where{ ?x wasBornIn ?c1. ?t1 ?x hasWonPrize Nobel_Prize_in_Physics. ?x diedIn ?c2. ?t3 ?c1 type_star wordnet_city_108524735. ?c2 type_star wordnet_city_108524735. }Filter{?t1.s > 1800-##-## and ?t1.e < 19##-##-## and ?t3.s > 1940-##-## and ?t3.e < 2000-##-##}	TR2	select ?p1,?p2,?c1,?c2 where{ ?p1 isMarriedTo ?p2. ?p1 wasBornIn ?c1. ?t2 ?p1 type_star wordnet_film_maker_110088390. ?c1 type_star wordnet_city_108524735. ?p2 wasBornIn ?c2. ?t5 ?p2 type_star wordnet_film_maker_110088390. ?c2 type_star wordnet_city_108524735. }Filter{?t2.s > 1930-##-## and ?t2.e < 1939-##-## and ?t5.s ≥ 1940-##-## and ?t5.e ≤ 1950-##-##}
TJ1	select ?x,?c1,?c2 where{ ?x wasBornIn ?c1. ?t1 ?x hasWonPrize Nobel_Prize_in_Physics. ?x diedIn ?c2. ?t3 ?c1 type_star wordnet_city_108524735. ?c2 type_star wordnet_city_108524735. }Filter{?t3.s > ?t1 + 70-##-##}	TJ2	select ?p1,?p2,?c1,?c2 where{ ?p1 isMarriedTo ?p2. ?p1 wasBornIn ?c1. ?t2 ?p1 type_star wordnet_film_maker_110088390. ?c1 type_star wordnet_city_108524735. ?p2 wasBornIn ?c2. ?t5 ?p2 type_star wordnet_film_maker_110088390. ?c2 type_star wordnet_city_108524735. }Filter{?t5.s < ?t2 - 15-##-##}
TC1	select ?x,?c1,?c2 where{ ?x wasBornIn ?c1 ?t1. ?x hasWonPrize Nobel_Prize_in_Physics. ?x diedIn ?c2 ?t3. ?c1 type_star wordnet_city_108524735. ?c2 type_star wordnet_city_108524735. }Filter{?t1.s ≥ 18##-##-## and ?t3.s > ?t1 + 70-##-##}	TC2	select ?p1,?p2,?c1,?c2 where{ ?p1 isMarriedTo ?p2. ?p1 wasBornIn ?c1. ?t2 ?p1 type_star wordnet_film_maker_110088390. ?c1 type_star wordnet_city_108524735. ?p2 wasBornIn ?c2. ?t5 ?p2 type_star wordnet_film_maker_110088390. ?c2 type_star wordnet_city_108524735. }Filter{?t2.s > 1920-##-## and ?t2.e < 1950-01-01 and ?t5.s > ?t2 + (-10-##-##)}
ST1	select ?x,?c1,?c2 where{ ?x wasBornIn ?c1. ?t1 ?x hasWonPrize Nobel_Prize_in_Physics. ?x worksAt ?u ?l3. ?t3 ?c1 type_star wordnet_city_108524735. ?u type_star wordnet_educational_institution_108276342. }Filter{dist(?c1,(54,5)) ≤ 10 and dist(?l3,?c1) ≤ 10 and ?t3.e < 2000-##-## and t1.s < t3 - 50-##-##}	ST2	select ?p1,?p2,?c1,?c2 where{ ?p1 isMarriedTo ?p2. ?p1 wasBornIn ?c1. ?t2 ?p1 type_star wordnet_film_maker_110088390. ?c1 type_star wordnet_city_108524735. ?p2 wasBornIn ?c2 ?t5. ?p2 type_star wordnet_film_maker_110088390. ?c2 type_star wordnet_city_108524735. }Filter{dist(?c1,(40,-95)) ≤ 20 and dist(?c2,c1) ≤ 20 and ?t2.s > 1900-##-## and t2.e < 1950-01-01} and ?t5.s > ?t2 + 10-##-##

Fig. 11: The 20 Sample Queries

and rdf schema. Paper presented at the International semantic web conference (pp. 54-68). Springer, Berlin, Heidelberg.

Deppisch, U. (1986, September). S-tree: a dynamic balanced signature index for office retrieval. *In Proceedings of the 9th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 77-87). ACM. http://dx.doi.org/10.1145/253168.253189

Erling, O., & Mikhailov, I. (2009). RDF Support in the Virtuoso DBMS *Networked Knowledge-Networked Media* (pp. 7-24): Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-02184-8_2

Grandi, F. (2010, Septermber). *T-SPARQL: A TSQL2-like Temporal Query Language for RDF*. Paper presented at the ADBIS (Local Proceedings).

Gutierrez, C., Hurtado, C. A., & Vaisman, A. (2007). Introducing Time into RDF. *IEEE Transactions on Knowledge and Data Engineering*, 19(2), 207-218. http://dx.doi.org/10.1109/TKDE.2007.34

Gutiérrez, C., Hurtado, C. A., & Vaisman, A. A. (2005). *Temporal RDF*. Paper presented at the The Semantic Web: Research and

Applications, Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece, May 29 - June 1, 2005, Proceedings.

Guttman, A. (1984). R-trees: a dynamic index structure for spatial searching. *In ACM SIGMOD International Conference on Management of Data* (Vol.14, pp.47-57). ACM. http://dx.doi.org/10.1145/602259.602266

Haklay, M., & Weber, P. (2008). Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4), 12-18. http://dx.doi.org/10.1109/MPRV.2008.80

Hoffart, J., Suchanek, F. M., Berberich, K., Lewis-Kelham, E., De Melo, G., & Weikum, G. (2011, March). YAGO2: exploring and querying world knowledge in time, space, context, and many languages. *In Proceedings of the 20th international conference companion on World wide web* (pp. 229-232). ACM. http://dx.doi.org/10.1145/1963192.1963296

Hoffart, J., Suchanek, F. M., Berberich, K., & Weikum, G. (2013). YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia. *Artificial Intelligence*, 194, 28-61. http://dx.doi.org/10.1016/j.artint.2012.06.001

Klyne, G., Carroll, J. J., & McBride, B. (2004, February). Resource description framework (RDF): Concepts and abstract syntax. *World Wide Web Consortium Recommendation*.

Lyell, M., Voyadgis, D., Song, M., Ketha, P., & Dibner, P. (2011, May). An ontology-based spatio-temporal data model and query language for use in gis-type applications. *In Proceedings of the 2nd International Conference on Computing for Geospatial Research & Applications* (p. 15). ACM. http://dx.doi.org/10.1145/1999320.1999335

Neumann, T., & Weikum, G. (2009). *RDF-3X: a RISC-style engine for RDF*. Proceedings of the VLDB Endowment, 1(1), 647-659. http://dx.doi.org/10.14778/1453856.1453927

Perry, M., Jain, P., & Sheth, A. P. (2011). Sparql-st: Extending sparql to support spatiotemporal queries. *Semantic Web & Beyond*, 12, 61-86. http://dx.doi.org/10.1007/978-1-4419-9446-2_3

Neumann, T., & Weikum, G. (2010). x-RDF-3X: fast querying, high update rates, and consistency for RDF databases. *Proceedings of the VLDB Endowment*, 3(1-2), 256-263. http://dx.doi.org/10.14778/1920841.1920877

Pugliese, A., Udrea, O., & Subrahmanian, V. (2008, April). Scaling RDF with time. *In Proceedings of the 17th international conference on World Wide Web* (pp. 605-614). ACM. http://dx.doi.org/10.1145/1367497.1367579

Singh, R., Turner, A., Maron, M., & Doyle, A. (2008). GeoRSS: Geographically encoded objects for RSS feeds: http://georss.org/gml.

Tappolet, J., & Bernstein, A. (2009, May). Applied temporal RDF: Efficient temporal querying of RDF data with SPARQL. *In European Semantic Web Conference* (pp. 308-322). Springer, Berlin, Heidelberg. http://dx.doi.org/10.1007/978-3-642-02121-3_25

Wang, D., Zou, L., Feng, Y., Shen, X., Tian, J., & Zhao, D. (2013, April). S-store: An engine for large rdf graph integrating spatial information. *In International Conference on Database Systems for Advanced Applications* (pp. 31-47). Springer, Berlin, Heidelberg. http://dx.doi.org/10.1007/978-3-642-37450-0_3

Weiss, C., Karras, P., & Bernstein, A. (2008). Hexastore: sextuple indexing for semantic web data management. *Proceedings of the VLDB Endowment*, 1(1), 1008-1019. http://dx.doi.org/10.14778/1453856.1453965

Wilkinson, K. (2009). Jena property table implementation. *Ssws*.

Wilkinson, K., Sayers, C., Kuno, H., & Reynolds, D. (2003, September). *Efficient RDF storage and retrieval in Jena2*. Paper presented at the International Conference on Semantic Web and Databases (pp. 120-139). CEUR-WS. org.

Zou, L., Mo, J., Chen, L., Özsu, M. T., & Zhao, D. (2011). gStore: answering SPARQL queries via subgraph matching. *Proceedings of the VLDB Endowment*, 4(8), 482-493. http://dx.doi.org/10.14778/2002974.2002976