g^{st}-Store: An Engine for Large RDF Graph Integrating Spatiotemporal Information

Dong Wang, Lei Zou[*], Dongyan Zhao
Institute of Computer Science & Technology, Peking University
Beijing, China
{wangd,zoulei,zhaodongyan}@pku.edu.cn

ABSTRACT

In this paper, we present a spatiotemporal information integrated RDF data management system, called gst-Store. In gst-Store, some entities have spatiotemporal features, and some statements have valid time intervals and occurring locations. We introduce some spatiotemporal assertions into the SPARQL query language to answer the spatiotemporal range queries and join queries. Some examples are listed to demonstrate our demo.

Keywords

RDF query, Spatiotemporal information, Range query, Join query

1. INTRODUCTION

In recent years, the success of the knowledge bases such as DBpedia[2], YAGO[10], Freebase[3] etc. greatly enhances the research of Semantic Web. In fact, a substantial part of knowledge is relevant to spatiotemporal information. For example, the events "Einstein won Nobel Prize" and "the birth of Lincoln" are relevant to some specific places and time, i.e., Einstein won Nobel Prize at Stockholm in 1921 and Lincoln was born in 1809 at Kentucky. With the advance of the spatiotemporal RDF data research, there have been some RDF data sets with either spatial or temporal information. For example, OpenStreetMap[6] is a spatial RDF data set, and the GovTrack[1] data set have temporal information. In contrast, YAGO2 is the first data set that harmoniously combines the spatiotemporal features and the semantic features.

Though researchers have made some efforts on spatiotemporal RDF data management and proposed some corresponding data models and query languages, the existing mature RDF management systems, such as RDF-3x[9], Hexastore[12], Jena2[13], SW-store[1] etc., only consider the semantic(structure) features of RDF data except for a few systems, such as YAGO2 Demo[8] and Virtuoso[4]. However, YAGO2 Demo only supports some hard coded spatiotemporal assertions on statements and Virtuoso implements a post-process way for spatial RDF queries. It is not friendly or flexible enough to express users' query requirements in YAGO2 Demo system, and the post-process way in Virtuoso is inefficient for complex spatiotemporal SPARQL queries.

In this demo, we present gst-Store, a graph-based spatiotemporal RDF data management system. Similar to YAGO2[7], we represent each statement as a five-tuple $\langle s, p, o, l, t\rangle$, where s, p, o, l, t represent for the features *subject*, *predicate*, *object*, *location* and *time interval* respectively. Note that l and t can be null if there is no spatial nor temporal information.

We introduce a variety of spatiotemporal assertions to express users' query requirements into traditional SPARQL queries. The spatial range assertions restrict the area of an entity that locates in or an event (an event is denoted as a triple statement) that happens in. The spatial join assertions require that the distance between the locations of two entities/events is no more than a given distance threshold r. The temporal range assertions restrict the time interval of an entity or an event. The temporal join assertions express the relative position of the time intervals of two entities/events.

The rest of this paper is organized as follow. Firstly, the details about the data and query model are discussed in Section 2. Then, we briefly describe the query format and the architecture of g^{st}-Store in Section 3. At the end, we demonstrate gst-Store by some real examples in Section 4.

2. DATA MODEL

Both the entity model and the statement model are modified in our spatiotemporal RDF model. Therefore, in this section, we first separately introduce the entity model and the statement model, and then describe our spatiotemporal query model.

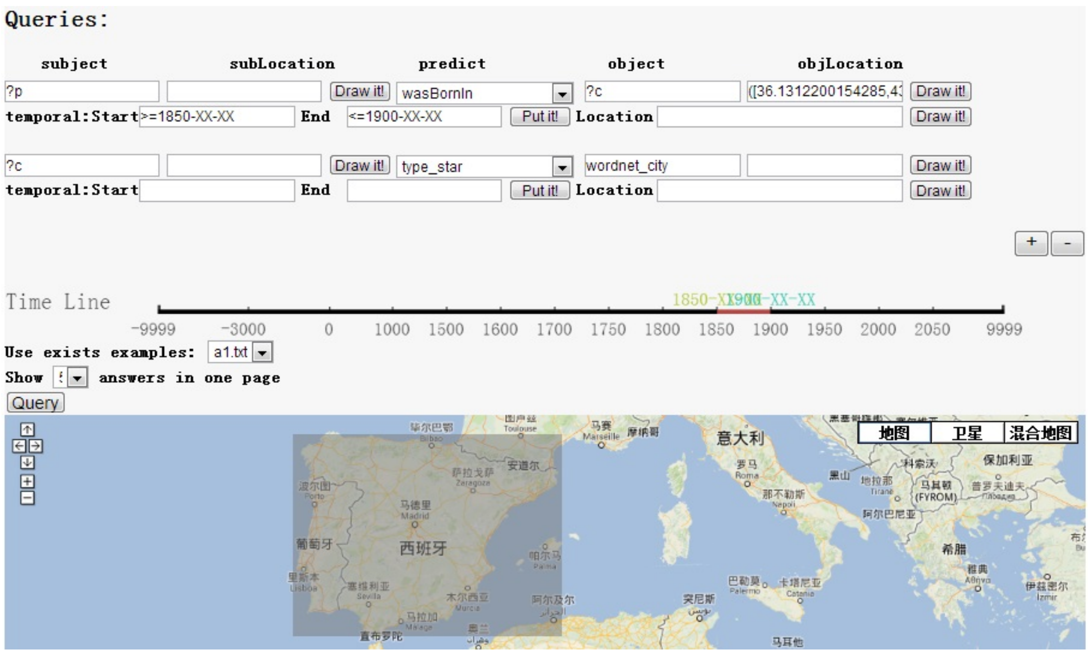
2.1 Entity Model

In spatiotemporal RDF data, we suppose that entities may have their own spatial features. A spatial feature represents the specific location of an entity. Thus, an entity can be denoted as a tuple $\langle s, locatedIn, l\rangle$, where l is represented as longitude and latitude coordinates. Obviously, only part of the entities have spatial features. For example, the White House has a specific location on the map. In contrast, people (such as Abraham Lincoln) and virtual concepts (such as Physics) have no spatial information. So far, we only model the spatial information by longitude and latitude coordinates. Actually, our framework can also solve more complex ways, e.g., to use a polygon to denote the spatial features of an entity.

A large amount of entities have inherent temporal features. For example, a person has his/her birthday and deathday, and an organization is active during its registration date and its dissolution date. These temporal features are used to build up the life cycle of

[*]corresponding author, zoulei@pku.edu.cn. This work was supported by NSFC under grant No.61370055 and 61272344 and CCF-Tencent Open Research Fund.

[1]http://www.govtrack.us/data/rdf/


Queries:

subject	subLocation	predict	object	objLocation
?p		wasBornIn	?c	([36.1312200154285,4
temporal:Start >=1850-XX-XX		End <=1900-XX-XX	Location	
?c		type_star	wordnet_city	
temporal:Start		End	Location	

Use exists examples: a1.txt

Show answers in one page

Figure 1: The Input Interface & Example 1

a person, a building or an organization etc.. Thus, we use a time interval to express the life cycle of an entity, i.e., an entity is active in the time interval. Since the users are always interested in the active entities during a specific temporal interval, the perpetual entities like virtual concepts or the celestial bodies would not be considered if the user gives a temporal assertion over the variables, and only the active entities satisfying the given temporal assertion are considered as the result.

2.2 Statement Model

We use a five-tuple $\langle s, p, o, l, t\rangle$ to denote a statement, where s, p, o, l, t represent for $subject$, $predicate$, $object$, $location$ and $time\ interval$ respectively. The first three items are the same as the original RDF statements, and $location$ and $time\ interval$ denote the place and the valid time interval of the statement (when it describes an event) respectively. Specifically, we use the longitude and latitude $\langle x, y\rangle$ to denote the happening location of a statement. The "$time\ interval$" feature is a bigram $\langle start, end\rangle$, i.e., it has a start time and an end time. Both the start time and the end time are dates with format "*yyyy-mm-dd*". Similar as the original RDF data model, the spatiotemporal RDF data can be organized as a graph including spatiotemporal features. More detailed discussions can be found in [11].

2.3 Query Model

Our statement query model in g^{st}-Store is also organized as a five-tuple $\langle s, p, o, l, t\rangle$, where s and o can be replaced with variables with spatiotemporal assertions, and l and t specify the spatiotemporal assertions of the statement. Since a query statement is also an edge connecting two entities, a list of statement query patterns form a query graph including a list of spatiotemporal assertions. If two statement query patterns share the same variable, the two corresponding edges are connected via the same variable (i.e., a vertex in the query graph).

Besides, we have specific spatial assertions of s and o to give the spatial constraints of the variables. The detailed assertion formats are given in the following section. In practice, most data sets have incomplete entity temporal information. Therefore, explicit entity temporal assertions usually cause incorrect answers. For example, suppose that a user wants to find a person who was alive during 1800 and 1810. If there're only the birth dates of all the people, every person who was born before 1800 would be returned (if we suppose that every person who was not announced to be dead is alive), or no person is returned (if we only return the confirmed answers). In this case, in our demo, we provide the statement temporal assertions to users, and then use the statement temporal assertions to build the entity temporal assertions in the background.

Generally speaking, we take spatiotemporal assertions on variables and statements into account. A SPARQL query with spatiotemporal assertions can be modeled as a query graph Q. It contains the following elements.

- Vertices in Q which denote the subjects or objects, and edges which indicate the triple patterns in a SPARQL query.
- The textual features of vertices or edges, i.e. the non-variable subjects, predicates and objects.
- A list of spatiotemporal assertions of variables.
- A list of spatiotemporal assertions of statements.

The goal of query processing is to find the subgraph matches in the RDF graph meanwhile satisfying the spatiotemporal assertions. The formally definitions can be found in [11]. The matches of a query is considered as the query results. g^{st}-Store organizes each match as a list of statements corresponding to the query patterns and visualize the matches on the map.

3. GST-STORE SYSTEM

3.1 Form the query

In [11], we have defined a spatial RDF query language as a SPARQL-like language. In this demo paper, we extend the query language with the spatiotemporal semantics. For ease of use, we implement the query interface as a list of statement query patterns in our demo system, as shown in Figure 1. The user can easily input the spatiotemporal assertions of the variables and the statement query patterns just following the corresponding variables or the corresponding statement query patterns.

In Figure 1, each statement query pattern has 8 components, which are designed for the triples and the corresponding spatiotemporal assertions. The "subject" and "object" boxes are used to input the URI/text of an entity or a variable. Note that the input is a variable if and only if the string starts with "?". We also utilize the AJAX technique to facilitate the inputs for "subject" and "object". The "predict" box receives the predicates of the statement query patterns. For ease of use, we implement a drop down box including all the predicates in the RDF data set. When a user needs more or less query patterns, the "+" and "-" buttons are provided to add or to remove a statement query pattern.

The spatial assertions have a uniform input format. Since gst-Store supports both spatial range queries and spatial join queries, we introduce the following input formats to form the query requirement (i.e. spatial assertions).

- **Spatial range assertion:** We provide two range shapes, a rectangle or a circle in g^{st}-Store. Specifically, we use the range of the longitude and latitude to denote a rectangle, and use the longitude and latitude of the center O as well as the radius r to denote a circle. We implement an interactive way on the google map to support the input.
  - *rectangle:* We use "$([x_1, x_2], [y_1, y_2])$" to denote a rectangle. The outermost "(" and ")" means it's a spatial range assertion, and the two "$[x_1, x_2]$" and "$[y_1, y_2]$" denote the ranges of the longitude x and latitude y.
  - *circle:* We use "$(O = (x, y), r = R)$" to denote a circle. "$O = (x, y)$" denotes the longitude x and latitude y of the center of a circle, and "$r = R$" denotes the radius R of the circle.
- **Spatial join assertion:** The spatial join assertions restrict the maximum distance between a variable/statement and another variable/statement. We use expressions like "$[expression, r = D]$" to denote a spatial join assertion. The outermost "[" and "]" means that it is a spatial join assertion, and the $expression$ refers to the corresponding variable/statement. The $r = D$ expression denotes the maximum distance D between the matches of the corresponding variable/statements.
  - If the $expression$ is the variable name (i.e. $expression$ starts with "?"), the bound item is the corresponding variable.
  - If the $expression$ is an integer, the bound item is the corresponding statement query pattern. The first input statement query pattern has serial number "1", and the second has "2" and so on.

The temporal assertions also have range and join semantics. In gst-Store, the time intervals of the corresponding statements are restricted via setting the temporal assertions of the time interval boundaries, i.e., the start time and the end time. The valid input formats are shown as follow.

- **Temporal range assertion:** In temporal range assertions, users can use comparison symbols or a temporal range to limit the boundary of the both ends of the time interval.
  - *comparison symbols:* A user can use $year\text{-}month\text{-}day$ to denote a time point, and use $>$, $<$, $=$, $>=$ or $<=$ to denote the range of the assertion of the start/end time. Note that if a user cannot determine the exact date, the last bits can be replaced by "#" as a wildcard. For instance, "19##-##-##" means any day in the 20th century.
  - *temporal range:* We use the "$(year\text{-}month\text{-}day, year\text{-}month\text{-}day)$" format to describe a time interval as a temporal range for the start/end time of the constrained statement query pattern.
- **Temporal join assertion:** The temporal join assertions are like "$[symbol, id, year\text{-}month\text{-}day]$", where "$year\text{-}month\text{-}day$" denotes the interval length from the constrained time point. Note that if the last characters of id are "$.s$" or "$.e$", it means that the start time or the end time of the corresponding statement query pattern. For example, "[<,1.e,10-##-##]" means that the time point is earlier than the first statement's end time for at least 10 years.
  - $symbol$ is a comparison symbol to point the direction of the assertion. "$>$"("$<$") means the assertion is after(before) a time point(or the end time of a time interval), and "$<>$" means this statement happens during a time interval that around the corresponding statements within a given length.
  - id is the input serial number (it is same as in spatial join queries) to indicate the corresponding statement query pattern. In order to point out the start time or the end time or the idth statement query pattern, we allow users to add "$.s$"(start time) or "$.e$"(end time) following the id.
  - $year\text{-}month\text{-}day$ denotes the time length from the constrained time point. Note that $year$ can be positive or negative.

Furthermore, we implement a convenient way to input the spatiotemporal range assertions. For each spatial assertion input box, users can click the nearby "Draw it!" button to draw a rectangle on the Google Map tool. Firstly, users should click the button, and then click on the map to determine one corner of the rectangle, and finally click the opposite corner of the rectangle on the map. Similarly, to input a temporal range assertion, users can first click the "Put it!" button, and then choose the start time point on the time line, and finally choose the end time point to denote the temporal range assertion.

3.2 Query Processing

When a user issues a query, the query processing program turns active to respond the query. Our query processing stage is based on a tree-style index and a top-down search algorithm. The solution follows the ideal of gStore[14], and we design a novel data model to integrate the spatiotemporal features and a query model to efficiently and effectively answer the spatiotemporal RDF queries.

3.2.1 Index Structure

The index of gst-Store is called ST-tree. Firstly, we convert a data graph into a signature graph [11], i.e., we use a bit string to denote each entity in the RDF data set. Specifically, the bit string of an entity is computed based on the neighborhood of the entity. And then, we use an MBR and a segment to denote the spatial and the temporal features of the entity respectively. The ST-tree is built based on the "$insert$" and the "$split$" operations similar as the R-tree[5] and the VS-tree[14]. We integrate the signature cost, the spatial cost and the temporal cost into a uniform cost model. The "$insert$" and the "$split$" operations are based on the cost model. In the $insert$ process, we insert the entity into the leaf node of the ST-tree with the lowest cost. In the $split$ process, the full tree node is separated to two half size nodes with the lowest cost. It can be proved that the ST-tree is a balance tree, and each layer of the ST-tree forms a signature graph with spatiotemporal features.

3.2.2 *Query Evaluation*

Based on the ST-tree, we propose a top-down search algorithm to find the query results. Similar with VS-tree [14], it can be proved that each match of the query corresponds to a match of the signature query graph in arbitrary signature graph lies on any layer of the ST-tree. Therefore, we propose a series of pruning rules for efficiently retrieve the matches of the query by considering both semantic and spatiotemporal constraints. Specifically, the processing is divided into two stages. At the first stage, we obtain the candidate sets of the variables in the query. Then, we verify the validity of each combination of the candidates. The verification guarantees the correctness of each result. As soon as we obtain the candidates, we generate each match of the query in the data graph, and show the results in the interface. In Section 4, some examples are listed to present the outputs of g^{st}-Store.

4. DEMONSTRATIONS

In this section, we demonstrate g^{st}*-Store* by two examples. We use Yago2[2] as the underlying data set in our demo which is built automatically from Wikipedia, GeoNames, and WordNet. It contains 447 million facts about 9.8 million entities [7]. Our demo system can support users to precisely issue their query intentions (structural queries) to find the entities in Wikipedia, which is not supported by existing fulltext search function in Wikipedia.

EXAMPLE 1. Find all the people who was born between 1850 and 1900 in a city which is located in Iberian Peninsula.

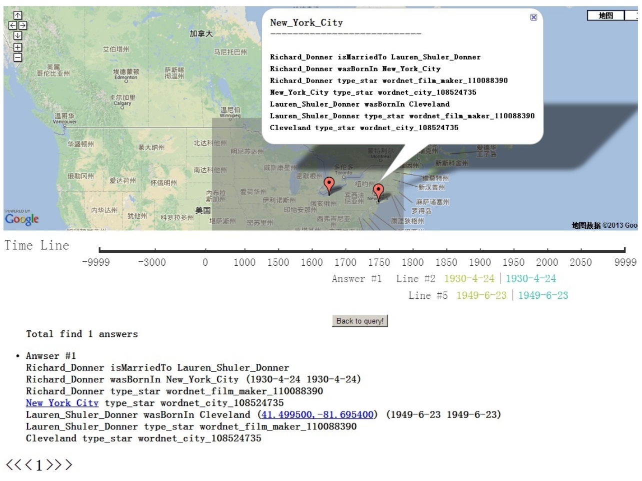
The query input of Example 1 is shown in Figure 1. The triple patterns means that "$?p$" is a person who was born in a city "$?c$". The rectangle surrounds Iberian Peninsula and the segment on the time line represents the time interval of 1850-1900. Users can use the google map tool to draw a rectangle to input a spatial range assertion, and the time line is given to directly specify the temporal assertions.

EXAMPLE 2. Find all married couples p_1 and p_2 satisfying the following conditions:
1. p_1 was born in city c_1 during 1900th and 1950th.
2. p_2 was born in city c_2 after p_1'th birth at least 10 years.
3. c_1 locates in east America.
4. The distance between c_1 and c_2 is no more than 2000km.
5. Both p_1 and p_2 are film makers.

Example 2 has complex inputs. In Example 2, we have spatial range, spatial join, temporal range and temporal join assertions.

We show the query results of Example 2 in Figure 2. We lie the satisfied spatial features on the map and the satisfied temporal features on the time line. In this example, we hit only one couple satisfying all the assertions, Mr. and Mrs. Donner. Mr. Donner was born in New York on April 24th, 1930 and Mrs. Donner was born in Cleveland when Mr. Donner was 19 years old. If a given query have lots of matches, the user can set how many results are returned in each page. The matched spatial entities and the spatial

[2]http://www.mpi-inf.mpg.de/yago-naga/yago/


```
New_York_City
-----------------------------

Richard_Donner isMarriedTo Lauren_Shuler_Donner
Richard_Donner wasBornIn New_York_City
Richard_Donner type_star wordnet_film_maker_110088390
New_York_City type_star wordnet_city_108524735
Lauren_Shuler_Donner wasBornIn Cleveland
Lauren_Shuler_Donner type_star wordnet_film_maker_110088390
Cleveland type_star wordnet_city_108524735
```

```
Time Line
-9999  -3000  0  1000  1500  1600  1700  1750  1800  1850  1900  1950  2000  2050  9999
                              Answer #1   Line #2  1930-4-24 | 1930-4-24
                                          Line #5  1949-6-23 | 1949-6-23

Total find 1 answers

• Anwser #1
  Richard_Donner isMarriedTo Lauren_Shuler_Donner
  Richard_Donner wasBornIn New_York_City (1930-4-24 1930-4-24)
  Richard_Donner type_star wordnet_film_maker_110088390
  New York City type_star wordnet_city_108524735
  Lauren_Shuler_Donner wasBornIn Cleveland (41.499500,-81.695400) (1949-6-23 1949-6-23)
  Lauren_Shuler_Donner type_star wordnet_film_maker_110088390
  Cleveland type_star wordnet_city_108524735

<<<1>>>
```

Figure 2: The Result of Example 2

statements are anchored on the map, and the users can click the hyperlink to find the corresponding location on the Google Map tool.

5. REFERENCES

[1] D. J. Abadi, A. M. 0002, S. Madden, and K. Hollenbach. Sw-store: a vertically partitioned dbms for semantic web data management. *VLDB J.*, 18(2), 2009.

[2] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. Dbpedia - a crystallization point for the web of data. *J. Web Sem.*, 7(3), 2009.

[3] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. ACM, 2008.

[4] O. Erling and I. Mikhailov. Rdf support in the virtuoso dbms. In *Networked Knowledge - Networked Media*.

[5] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.

[6] M. Haklay and P. Weber. Openstreetmap: User-generated street maps. *Pervasive Computing, IEEE*, 7(4):12–18, 2008.

[7] J. Hoffart, F. Suchanek, K. Berberich, and G. Weikum. Yago2: a spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 2012.

[8] J. Hoffart, F. M. Suchanek, K. Berberich, E. Lewis-Kelham, G. de Melo, and G. Weikum. Yago2: exploring and querying world knowledge in time, space, context, and many languages. In *WWW (Companion Volume)*, pages 229–232, 2011.

[9] T. Neumann and G. Weikum. Rdf-3x: a risc-style engine for rdf. *PVLDB*, 1(1), 2008.

[10] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: a core of semantic knowledge. In *WWW*, 2007.

[11] D. Wang, L. Zou, Y. Feng, X. Shen, J. Tian, and D. Zhao. S-store: An engine for large rdf graph integrating spatial information. In *DASFAA (2)*, pages 31–47, 2013.

[12] C. Weiss, P. Karras, and A. Bernstein. Hexastore: sextuple indexing for semantic web data management. *PVLDB*, 1(1), 2008.

[13] K. Wilkinson, C. Sayers, H. A. Kuno, and D. Reynolds. Efficient rdf storage and retrieval in jena2. In *SWDB*, 2003.

[14] L. Zou, J. Mo, L. Chen, M. Özsu, and D. Zhao. gstore: answering sparql queries via subgraph matching. *Proceedings of the VLDB Endowment*, 4(8):482–493, 2011.