International Journal of Geo-Information

MDPI

Article

Similarity Search on Semantic Trajectories Using Text Processing

Damião Ribeiro de Almeida [1,*], Cláudio de Souza Baptista [1] and Fabio Gomes de Andrade [2]

[1] Department of Computer Science, Federal University of Campina Grande, Campina Grande 58429-900, Brazil; baptista@computacao.ufcg.edu.br
[2] Federal Institute of Paraíba, Cajazeiras 58900-000, Brazil; fabio@ifpb.edu.br
* Correspondence: damiao@copin.ufcg.edu.br

Abstract: The use of location-based sensors has increased exponentially. Tracking moving objects has become increasingly common, consolidating a new field of research that focuses on trajectory data management. Such trajectories may be semantically enriched using sensors and social media. This enables a detailed analysis of trajectory behavior patterns. One of the problems in this field is the search for a semantic trajectory database that is flexible and adaptable; flexibility in the sense of retrieving trajectories that are closest to the user's query and not just based on exact matching. Adaptability refers to adjusting to different types of semantic trajectories. This article proposes a new approach for representing and querying semantic trajectories based on text-processing techniques. Furthermore, we describe a framework, called SETHE (SEmantic Trajectory HuntEr), that performs similarity queries on semantically enriched trajectory databases. SETHE can be adapted according to the aspect types posed in user queries. We also presented an evaluation of the proposed framework using a real dataset, and compare our results with those of state-of-the-art approaches.

Keywords: semantic trajectories; textual search; similarity measuring

Citation: Ribeiro de Almeida, D.; de Souza Baptista, C.; de Andrade, F.G. Similarity Search on Semantic Trajectories Using Text Processing. *ISPRS Int. J. Geo-Inf.* **2022**, *11*, 412. https://doi.org/10.3390/ijgi11070412

Academic Editors: Hartwig H. Hochmair and Wolfgang Kainz

Received: 23 May 2022
Accepted: 15 July 2022
Published: 21 July 2022

1. Introduction

The proliferation of smartphones, low-cost sensors, and wireless communication devices has enabled the monitoring in geographic spaces of mobile entities such as people, animals, cars, ships, and natural phenomena.Currently, there are several ways to obtain the location of moving objects, with the Global Positioning System (GPS) being the most straightforward and common way to construct raw trajectories [1]. GPS consists of a sequence of geospatial points (latitude, longitude, and altitude coordinates) ordered by timestamps [2–4]. Trajectory data are important for analyzing and understanding the behavior of moving objects. For example, trajectory analysis may identify traffic jams, people's behavior patterns, navigation routes, fishing areas, animal migration, and hurricane trajectories [5].

Many studies have enriched trajectory data by including context-based information. Emmanouilidis et al. [6] defined context as a synonym for the range of information that may influence service adaptation. Such information may arise from the environment, user, or other systems. Context-based information enables the enrichment of trajectory analysis and improves the understanding of moving-object behavior [7]. The use of context information can provide insights into the behavioral aspects of mobile objects that would not be possible using only raw trajectories, such as which point of interest (POI) was visited, the type of activities performed, and the trajectory purpose.

Ubiquitous computing and Internet of Things help obtain context-based information [8]. Various devices, such as smartwatches, medical sensors, radio frequency identification (RFID) devices, and environmental sensors, can capture context-based information. Another way of implicitly obtaining trajectory data and context information is by volunteered

geographic information (VGI) [9], which consists of geographic data provided by citizens through location-based social networks, such as LinkedGeoData (http://linkedgeodata.org/, accessed on 22 May 2022) and OpenStreetMap (https://www.openstreetmap.org/, accessed on 22 May 2022). In addition, some social media platforms, such as Flickr (https://www.flickr.com/, accessed on 22 May 2022), Twitter (https://twitter.com/, accessed on 22 May 2022), Facebook (https://www.facebook.com/, accessed on 22 May 2022), and Foursquare (https://foursquare.com/, accessed on 22 May 2022), provide geolocation from their posts. Other behavioral information may be extracted from social media, such as the user's activity and POI evaluation.

Adding context information to trajectory data creates a semantically enriched trajectory, or simply a semantic trajectory [10,11]. In a semantic trajectory dataset, trajectories contain annotations. The waypoints are enriched with information regarding either the environmental or mobile object context, such as the POI name or user heartbeat. An aspect is any type of information that can be annotated to the trajectory POIs. Examples of POI aspects include their name, category, weather, means of transport, and rating [12]. Hence, the trajectory becomes a complex object with several contextual data dimensions associated with its movement [13].

To better understand what semantic trajectories are, consider the example represented in Figure 1, which depicts the short trajectory of a tourist in the city of Pisa in Italy. Each stop is semantically enriched with four aspects: POI name, category, means of transport used to reach the POI, and environmental temperature. The route starts at the POI *Cappella dal Pozzo* which belongs to the Chapel category. The tourist is walking, and the local temperature is 22 °C. Then, the tourist moves by bus to the *Museo delle Sinopie,* where the temperature is 21 °C. Finally, the route ends at *Teatro Sant'Andrea,* where the tourist arrives by taxi, and the temperature is 23 °C.

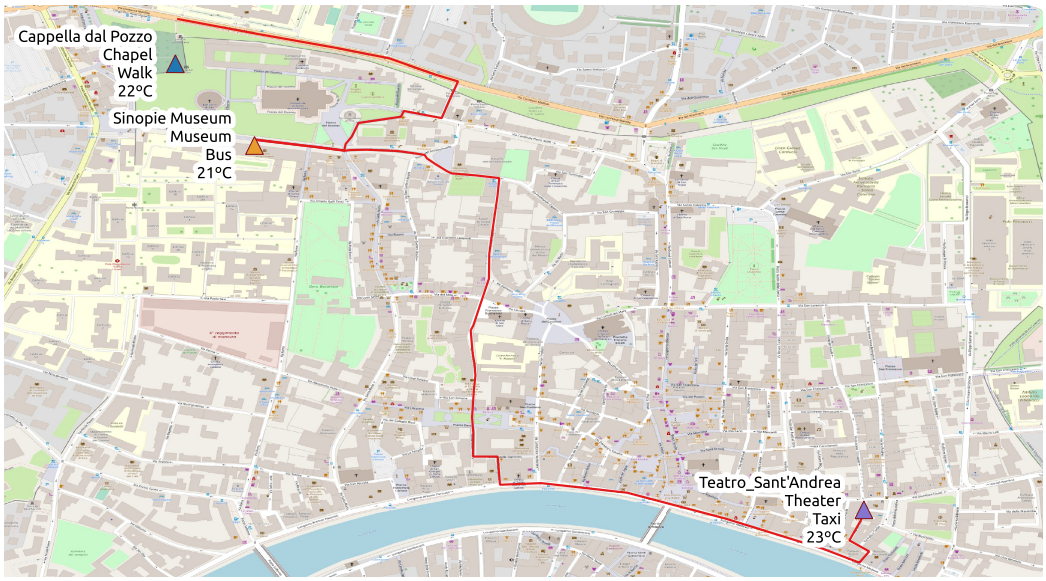

Figure 1. Example of tourist trajectory in the city of Pisa.

When dealing with semantic trajectories, we need to decide how to represent context information. For example, Noel et al. [14] represented trajectories in a multidimensional manner, in which each dimension focused on a single aspect, and each aspect was represented by a trajectory. Table 1 presents a representation of the trajectory in Figure 1. In Table 1, the transition event is the displacement between the stopping points. The first line is the POI name trajectory, which begins at *Cappella dal Pozzo* and ends at *Teatro Sant'Andrea.* We then have the category trajectory following the same direction, starting at a chapel and ending at a theater. Finally, we obtain the means of the transport and temperature trajectories. Hence, it is possible to analyze trajectories from different viewpoints and solve

queries concerning certain aspects. For example, it is possible to search for trajectories in which a person travels using only a bus as the transport mean or trajectories that start at a mall and end up at a theater.

Table 1. Example of user semantic trajectory.

POI Name	**Cappella Dal Pozzo**	**Museo Delle Sinopie**	**Teatro Sant'Andrea**
Category	Chapel	Museum	Theater
Transport means	Walk	Bus	Taxi
Temperature	22 °C	21 °C	23 °C

Current search engines on semantic trajectory datasets retrieve only the set of trajectories that exactly matches each constraint defined in the user query. For example, suppose someone is looking for the trajectories of a person arriving at a given church by bus. In this case, the query result will only contain trajectories with the category attribute equal to *church*, and the transport mean attribute equal to bus. Occasionally, it becomes challenging to find trajectories that perfectly match all query constraints. The more constraints a query has, the more difficult it becomes to find compatible trajectories. For example, a query that looks for people trajectories who went by taxi to a chapel, then by bus to a museum, and walked to a theater would not return the trajectory shown in Figure 1. Among all the constraints defined in the query, only the transport mean aspect of the first and last locations are not satisfied by that trajectory. Hence, even when satisfying almost all query restrictions, the trajectory depicted in Figure 1 cannot be retrieved as a result.

A query on a semantic trajectory database expresses the disposition of stop points along the trajectory [15]. Examples include searching for trajectories that start at the *Leaning Tower of Pisa*, trajectories that end at a museum, or trajectories that visit a church and then a theater.

Aiming to solve the aforementioned limitations, this study proposes a semantic trajectory framework that represents multi-aspect trajectories and can search for the most similar trajectories according to the aspect values contained in the query through a ranking approach. The framework queries a semantic trajectory database using text processing techniques. Hence, the trajectory is represented as a string vector. Each string may represent the POI's name, category, or other aspects. A query is also represented by the vectors. The distance between the query and trajectory vectors determines the matching and ranking of the result set. As a baseline, we used the semantic trajectory search framework developed by Izquierdo et al. [15], which describes a formal framework for semantic trajectories using description logic (DL) and SPARQL.

Thus, the main contributions of this article are as follows:

- The proposition of a new approach to represent trajectory data based on text.
- The development of a search engine for querying semantic trajectories taking into account not only the POIs name and categories, but also the semantic trajectory aspects.
- The specification of a new ranking algorithm that enables searching for trajectory similarity.
- The implementation of a simple and efficient approach—execution time and storage requirements—to perform queries on semantic trajectories, when compared to the SPARQL-based approach.

To validate our approach, we implemented a case study using TripBuilder [16], a trajectory dataset built from Flickr data, combined with Wikipedia data.

The remainder of the article is structured as follows. Section 2 discusses related work. Section 3 presents the fundamental concepts and the formal definition of the semantic trajectory query framework. Section 4 presents a running example to instantiate the SETHE framework. Section 5 describes the experiments performed. Finally, Section 6 concludes the paper and discusses further work to be undertaken.

2. Related Work

Usually, raw trajectory data are captured and stored in a spatial database known as a moving object database (MOD) [17]. However, once these data are captured and analyzed, it is necessary to enrich them with context-based information to increase analytics processing and to enable users to perform tasks such as identifying traffic jams, finding people's behavior patterns, observing navigation routes, identifying fishing areas, studying animal migration, understanding hurricane trajectories, and so on [5]. SeMiTri [18] is a trajectory enrichment system that uses semantic annotation to identify trajectory stops and moves [19]. SeMiTri semantically describes trajectories with information about the POI, means of transport, and type of geographic region (residential, business, market, etc.).

CONSTAnT is a conceptual data model that represents the main aspects of a semantic trajectory [20]. The model is divided into two parts. The first part describes simple entities, providing information about the mobile object, trajectory, sub-trajectory, semantic points, environment, places, and events. The second part describes complex entities in which data mining techniques are utilized to identify information such as the purpose of movement, means of transport used, and behavior of the moving object.

Nöel et al. [14] proposed a semantic trajectory model composed of multiple aspects, where each aspect has a group of related attributes. The authors argue that a semantic trajectory can be analyzed from different points of view, such as residential and professional. The city name where the user stayed, the type of place (house or apartment), and rent value are attributes when looking at the trajectory from the residential point of view. Work, occupation, and salary are semantic attributes when examining trajectories from a professional perspective.

RDF graphs and ontologies have also emerged as solutions to enrich semantic trajectories [13,21]. The representation of semantic trajectory data in RDF enables the inference of new knowledge and the publication of data as linked open data (LoD). The CRISIS system is an example of an application that deals with trajectory data streams and uses an RDF graph that semantically represents the marine data received from several sensors [22]. Baquara[2] is another example of a conceptual framework that analyzes and semantically enriches trajectories by using a customizable process [7]. The MASTER project models the trajectory and its context using RDF and uses the rendezvous database to store the RDF data [13].

Alvares et al. [23] proposed a model that represents essential parts of a trajectory (stop, move, and semantics) and uses the SQL language to perform queries on places visited, types of places, and other analyses. Izquierdo et al. [15] addressed the problem of queries on semantic trajectories using a stop-and-move representation. The authors described a formal framework using DL to formally introduce the syntax and semantics of trajectories and the mechanisms needed to express queries in their database. As a proof of concept, the authors used the TripBuilder [16] dataset with georeferenced photos captured by Flickr users. The stop points were enriched with the POI name, category, and movements with the means of transport. The concepts described in DL were implemented in RDF, and the queries were expressed using SPARQL.

The aforementioned studies used complex data models to represent the semantic trajectories. These models contain many entities and relationships that make the entire scenario challenging to understand. These become more complex as new aspects are added to the data model. Consequently, queries become more difficult to express and are based on exact matching without ranking. This study proposes a text representation of semantic trajectories, resulting in a simpler data model that optimizes memory size requirements and query performance. Consequently, a similarity query returns more results, reducing the frustration of the user due to empty answers.

3. SETHE: A Semantic Trajectory Retrieval Approach

In this section we present the SETHE (SEmantic Trajectory HuntEr) framework, a new approach for representing semantic trajectories and their aspects using text processing. In

SETHE, POI names, categories, and other aspects are represented as a sequence of terms. We then perform queries using text processing and rank the result set according to how close the trajectory result set was to the query. SETHE searches for trajectories containing at least one sub-sequence corresponding to the query, and whose semantic values are closest to the aspects specified in a user query. In this section, we formalize the SETHE underpinning trajectory model.

3.1. Basic Concepts

There are several similar definitions of trajectories [18,20,23]. However, we present a new approach for representing and dealing with trajectory data, in which a semantic trajectory is represented as a text vector. Hence, a query may be expressed using regular expressions, and a ranking approach is used to return not only exact matches, but also similar results. Considering that a POI is a specific location in which someone may be interested [24], we present a new aspect-based semantic trajectory definition.

Definition 1. *An aspect-based semantic trajectory is a sequence of POIs* $T = \langle p_1, p_2, \ldots, p_n \rangle$ *ordered chronologically and represented by a set of tuples* $ST = \{\langle name, st_1 \rangle, \langle category, st_2 \rangle, \langle aspectName_3, st_3 \rangle, \ldots, \langle aspectName_a, st_a \rangle\}$, *where* $a \geq 2$. *Every* st_i *is a sequence of n text values, one for each point* $p_i \in T$. *A POI of ST can be simple with only name and category, so* $ST = \{\langle name, st_1 \rangle, \langle category, st_2 \rangle\}$, *or more complex, with semantic aspects other than the POI name and category. Hence, names and categories are mandatory, and other aspects are optional.*

Applying Definition 1, we can represent the Figure 1 trajectory as:

$$\begin{aligned} ST_{f1} = \{&\langle name, \langle Cappella\ dal\ Pozzo, Sinopie\ Museum, Teatro\ Sant'Andrea \rangle\rangle, \\ &\langle category, \langle Chapel, Museum, Theater \rangle\rangle, \\ &\langle transport\ mean, \langle Walk, Bus, Taxi \rangle\rangle \\ &\langle temperature, \langle 22, 21, 23 \rangle\rangle\} \end{aligned}$$

A sub-sequence is another important concept for understanding semantic trajectory query processing. According to Gusfield [25], while a sub-trajectory represents consecutive points of a trajectory T, the POIs do not need to be consecutive in a sub-sequence of T. Sub-sequences are used during query processing to retrieve POI sequences from T that match the user's query.

Definition 2. *An aspect-based semantic sub-sequence trajectory* $SST = \{\langle name, sst_1 \rangle, \langle category, sst_2 \rangle, \langle aspectName_3, sst_3 \rangle, \ldots, \langle aspectName_a, sst_a \rangle\}$ *represents a sub-sequence of POIs of T, where* $a \geq 2$.

According to Definition 2, the sub-sequence SST_{f1} represents a sub-sequence of ST_{f1}. We can see that the SST_{f1} POIs are sequential points of ST_{f1}, but they are not necessarily consecutive.

$$\begin{aligned} SST_{f1} = \{&\langle name, \langle Cappella\ dal\ Pozzo,\ Teatro\ Sant'Andrea \rangle\rangle, \\ &\langle category, \langle Chapel, Theater \rangle\rangle, \\ &\langle transport\ mean, \langle Walk, Taxi \rangle\rangle \\ &\langle temperature, \langle 22, 23 \rangle\rangle\} \end{aligned}$$

3.2. Query Processing

To perform a search in a semantic trajectory dataset, users must provide some information, such as POIs names, categories, and/or other aspects. The stops can be identified based on the POI name or category. Figure 2 shows a graphical example of a query in which a user is searching for trajectories that pass through a museum and end up at the *Leaning Tower of Pisa*.

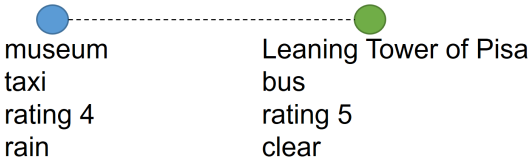

Figure 2. Example of a trajectory query with information on stops and aspects.

In addition to specifying the stops, the user may also specify what aspects are associated with each specific point. In Figure 2, the user wants to search for trajectories of people who use a taxi to go to a museum with a rating score of four, and rainy weather. Finally, the person traveled by bus to the *Leaning Tower of Pisa*, given a rating score of five, and the weather was clear.

The use of exact matching for the query depicted in Figure 2 may result in few or no results. To solve this limitation, SETHE searches for trajectories that most closely match the query using a ranking algorithm. For this, the user must also provide a distance function and weight for each aspect type. The distance function calculates the distance of the query aspect from a given trajectory. In addition to the distance function, the weight represents the degree of importance of each aspect of the user query. The distance and weight influence the final result rank. Table 2 lists some examples of these functions, where random values are compared to the aspect values shown in Figure 2. In this example, we used a *word2vec* function for the means of transport, the equal function for the weather aspect, and the Euclidean function for the rating aspect. The *word2vec* function calculates the semantic distance between the terms. The equal function returns only one of these two values: 0 (zero) when the terms are different and 1 (one) when the terms are equal. The Euclidean function is computed as $Euclidean(a, b) = |(a - b)|$.

Table 2. Example of distance functions for aspects.

***word2vec* Funtion**	***Equals* Function**	***Euclidean* Function**
word2vec(taxi, bus) = 0.74	*equals*(rain, fog) = 0	*Euclidean*(4.0, 5.0) = 1.0
word2vec(bus, walk) = 0.67	*equals*(clear, clear) = 1	*Euclidean*(5.0, 1.0) = 4.0

The following subsections detail the SETHE querying process. This process is accomplished in several steps: building a query to be interpreted by the framework, building a vector representation for the query, retrieving the sub-sequence with the same stop points specified in the query, building a vector representation for each retrieved sub-sequence and its aspects, and calculating the similarity between the query and sub-sequence vectors.

3.2.1. Query Building

A query is a sequence of expressions that may contain a POIs name, categories, and other aspects that indicate the semantic trajectory in which the user is interested. For example, using the categories sequence (*museum*; *tower*) and the sequence of transport mean aspects (*Bus*; *Taxi*), SETHE looks for routes that use a bus to arrive at a museum and a taxi to arrive at a tower.

During the searching process, SETHE considers either the name or category of the POIs at any position in the trajectory. However, it is possible to use features of regular expressions to inform the position of the POI in the textual path. For example, when using the symbol ^, we indicate that the POI must be at the beginning of a trajectory, and with the symbol $, we say that the POI should be at the end of a trajectory. Therefore, when performing a query using the sequence (^*museum*; *tower*$), SETHE looks for trajectories that start at a museum and end at a tower.

Table 3 shows five regular expression symbols used in the query process and two new symbols ((?-) and ∼) that help building query expressions.

Table 3. List of regular expressions for the semantic path query filter.

Expression	Explanation	Query Example	Result Example
^	begins with	(^museum; .*)	museum, chapel, tower
$	ends with	(.*; museum$)	chapel, tower, museum
|	"or" operator	((museum|chapel); tower)	chapel, square, tower
.*	any text	(.*; chapel$)	square, tower, chapel
(exp)*	the expression may be repeated zero or more times	((tower)*; museum)	tower, tower, museum
(?-)	The value ahead is repeated in the previous positions until the last expression of the trajectory begins	(?-) (Taxi; (?-) Bus)	(?-) Taxi, Bus, Bus, Bus
∼	proximity aspect	(museum; tower) (.*; ∼)	museum, tower, square

Inspired by [12], we define the query as follows.

Definition 3. *Query Q is a tuple $Q = (E, A, W, D, L)$, where:*

- *E represents the POI sequence of a nonempty set of tuples $E = \{\langle name, e_1\rangle, \langle category, e_2\rangle\}$. Each e_i is a sequence of m regular expressions, with one for each POI.*
- *A is another representation of the POI sequence represented by a set of tuples $A = \{\langle aspectName_1, b\alpha_1\rangle, \ldots, \langle aspectName_b, \alpha_b\rangle\}$, where $b \geq 0$ is the number of optional aspects. Each α_i is a sequence of m regular expressions, with one for each POI.*
- *$W = \{w_1, w_2, \ldots, w_b\}$ is a set of weights, where each weight is associated with an optional aspect, and $\sum_{i=1}^{b} w_i = 1$. If $A = \{\varnothing\}$, then $W = \{\varnothing\}$.*
- *$D = \{d_1, d_2, \ldots, d_b\}$ denotes a set of distance functions. A distance function exists for each optional aspect. If $A = \{\varnothing\}$, then $D = \{\varnothing\}$.*
- *$L = \{thr_1, thr_2, \ldots, thr_b\}$ is the threshold set that each distance function may achieve. There is a threshold for each function, and if $A = \{\varnothing\}$, $L = \{\varnothing\}$.*

To facilitate query comparison we use only the distance function in the optional aspects. Following Definition 3, we can use the tuple $Q_{f3} = (E_{f3}, A_{f3}, W_{f3}, D_{f3}, L_{f3})$ to express the query in Figure 2. This query specifies two points. The first is a category (*museum*), and the second is a POI name (*Torre Pendente di Pisa*). Therefore, sequence E must contain two tuples (*name* and *category*), and sequence e_i must have two points, where

$$E_{f3} = \{\langle name, \langle .*; \textit{Tower Pendente di Pisa}\rangle\rangle, \langle category, \langle museum; .*\rangle\rangle\}$$

The regular expression $.*$ is used when POI names and categories are unknown. The optional aspects in Figure 2 are means of transport, weather, and rating. The weight of each aspect depended on the user's choice. In this example, we set the highest priority to the means of transport and the weather aspect as the lowest priority. Therefore, we assigned the following weight sequence: $W_{f3} = \{0.5, 0.2, 0.3\}$. To calculate the distance between two aspect values, we use the *word2vec* function for the means of transport, the *equals* function for the weather aspect, and the *Euclidean* function for the rating aspect; therefore, $D_{f3} = \{word2vec, equals, Euclidean\}$. We adopted the values $L_{f3} = \{1, 1, 5\}$ for the threshold sequence. The value of 1 (one) is the highest possible value for the *word2vec* function. The maximum *equals* function value is 1, and 5 is the maximum value for the *Euclidean* function of the rating aspect, which varies between 1 and 5. Therefore, query Q_{f3} is expressed as

$$
\begin{aligned}
Q_{f3} = (&\{\langle name, \langle .*; Torre\ Pendente\ di\ Pisa\rangle\rangle,\\
&\langle category,\ \langle museum;\ .*\rangle\rangle\},\\
&\{\langle transport\ mean,\ \langle Bus,\ Taxi\rangle\rangle,\\
&\langle weather,\ \langle rain,\ clear\rangle\rangle,\\
&\langle rating,\ \langle 4,5\rangle\rangle\},\\
&\{0.5,\ 0.2,\ 0.3\},\\
&\{word2vec,\ equals,\ Euclidean\},\\
&\{1,\ 1,\ 5\})
\end{aligned}
$$

SETHE transforms a query into text to compare it with the textual trajectory database. This process is divided into four main steps.

1. Using the regex function to obtain the trajectories that pass through the POIs with the names and categories of the expressions.
2. Extracting the sub-sequences of T trajectory, in which both the name and category of the POIs match E regular expressions.
3. Using distance functions and aspect weights to calculate the query coefficient similarity with the sub-sequences.
4. Ranking the result according to the coefficient in descending order.

The function $regex(text, pattern)$ was used to determine the trajectories. The $text$ parameter can take one of two sentences: either POI name sequence st_1 or POI category sequence st_2. The $pattern$ parameter is a regular expression composed of the concatenation of e_i elements. The regular expression $(.*)$ was used to merge the e_i expressions. The $regex$ function informs if the st_i has at last one sub-sequence that matches the $pattern$. Using the Q_{f3} example, the $pattern$ value is $(.*)(.*)(Torre\ Pendente\ di\ Pisa)$ for $text$ equal to st_1 and $(museum)(.*)(.*)$ for $text$ equal to st_2.

Finally, SETHE uses two $regex$ functions over the database to look for the semantic trajectories that passed through the museum and the *Leaning Tower of Pisa*. The final expression is as follows:

$$regex(st_1,\ \text{“}(.*)(.*)(Torre\ Pendente\ di\ Pisa)\text{”})\ and\ regex(st_2,\ \text{“}(museum)(.*)(.*)\text{”})$$

Before proceeding, it is necessary to transform a query into a vector representation. A query Q is represented by sentence δ_q and vector $\overrightarrow{v_q}$. The sentence $\delta_q = (y_1^1, y_1^2, \ldots, y_1^b, y_2^1, y_2^2, \ldots, y_2^b, y_m^1, y_m^2, \ldots, y_m^b)$ is the interleaved concatenation for each POI of the regular expressions α_i of the optional aspects. The coordinates of the vector $\overrightarrow{v_q} = (v_1, v_2, v_3, \ldots, v_z)$ are the interleaved weights associated with each POI aspect, where Equation (1) identifies the aspect weight in W.

$$v_i = (w_k \mid 1 \leq i \leq z),\ where\ k = ((i-1) mod\ b) + 1 \tag{1}$$

Using the example Q_{f3} and applying Equation (1) to the $\delta_q = (taxi\ rain\ 4\ bus\ clear\ 5)$, we obtain the vector $\overrightarrow{v_q} = (0.5,\ 0.2,\ 0.3,\ 0.5,\ 0.2,\ 0.3)$.

3.2.2. Discovering Sub-Sequences

After retrieving the semantic trajectories using the regex query, SETHE calculates a vector representation for each ST sub-sequence that matches all regular expressions defined in e_1 and e_2. We describe this process with four algorithms. Algorithm 1 (the main function) is responsible for invoking the functions described in the other algorithms. Algorithm 1 demonstrates how to extract sub-sequences from an aspect-based semantic trajectory. The *calcSubsequences* function receives parameters st_1, st_2, e_1, and e_2. We used a tree as the data structure that will help to determine the trajectory sub-sequences. The tree starts with an

empty child, which will be the tree root, and its children will be the start sub-sequence POI. According to Definition 3, regular expressions e_1 and e_2 must have the same size. Algorithm 1 initially verifies if e_1 is equal to ".*", then it takes e_2. The regex function looks for all matches in the text for each e_i regular expression. Each item in the matches variable has a POI text (name or category) and POI position at the trajectory. A node is created for each match and added to the tree. The new node receives the text value of the match, the text position in the st_i, and the index of the regular expression in e_i. If it is the first regular expression, the node is added as a child of the root of the tree. Otherwise, the recursive function *addNewNode* will look for the correct position of the node in the tree. The $fixTree$ function removes all tree branches whose heights are less than the size of e_i. Thus, only the ST sub-sequences that match e_1 and e_2 remain in the tree. The *extractSubsequence* function is responsible for traversing the tree nodes to extract the trajectory sub-sequences and store them in the *listSubs* variable. In the final of the *calcSubsequences* function, the *listSubs* variable has all the sub-sequences of ST.

Algorithm 1 Extract Sub-sequence from ST

```
1:  function CALCSUBSEQUENCES (st1, st2, e1, e2)
2:      tree ← empty tree
3:      root ← tree.root
4:      listSubs ← empty list
5:      for index = 1 to e1.size do
6:          text ← st1
7:          exp ← e1 [index]
8:          if exp == ".*" then
9:              text ← st2
10:             exp ← e2 [index]
11:         end if
12:         matches ← regex (text, exp)
13:         for all ma in matches do
14:             node ← new node
15:             node.text ← ma.text
16:             node.textPosition ← ma.position
17:             node.expIndex ← index
18:             if index == 1 then
19:                 root.addChildren (node)
20:             else
21:                 for all nodeChild in root.children do
22:                     addNewNode (nodeChild, node)
23:                 end for
24:             end if
25:         end for
26:     end for
27:     fixTree (tree, root, e1.size)
28:     for all child in root.children do
29:         extractSubsequence (child, {}, listSubs)
30:     end for
31:     return listSubs
32: end function
```

Algorithm 2 describes the *addNewNode* recursive function. This function receives two parameters: the father and the child nodes created by Algorithm 1. To add the new node as a child of the father node, there are two constraints: first, the node position must be greater than the father node position; second, the index of the new node must be one unit above the index of the parent node.

Algorithm 2 Insert a Node in the Tree

```
1:  function ADDNEWNODE (father, newNode)
2:      if newNode.textPosition > father.textPosition then
3:          if newNode.expIndex == father.expIndex + 1 then
4:              father.addChildren (newNode)
5:          end if
6:      else
7:          for all childNode in father.children do
8:              addNewNode (childNode, newNode)
9:          end for
10:     end if
11: end function
```

Algorithm 3 describes the recursive function $fixTree$. This function receives three parameters: the tree, the node to be checked, and the leaf node height. Each node of the tree is visited recursively until reaching the leaf nodes. If the index of the leaf node is different from the height (e_i size), the node is removed from the tree. This process is repeated until the end to remove all the nodes with no child and index less then height.

Algorithm 3 Remove Incomplete Sub-sequence from the Tree

```
1:  function FIXTREE (tree, node, height)
2:      for all childNode in node.children do
3:          fixTree (tree, childNode, height)
4:      end for
5:      if node.children is empty then
6:          if node.expIndex ! = height then
7:              tree.removeNode (node)
8:          end if
9:      end if
10: end function
```

Algorithm 4 describes the function *extractSubsequence* behavior. This recursive function receives three parameters: the tree node, the sub-sequence currently being processed, and the sub-sequence list, which is the variable that stores the final result. The algorithm iterates through all child nodes and adds the value to the *subsequence* variable. If a node has more than one child, it means that more sub-sequences contain that node. Therefore, the *subsequence* is cloned, and the *extractSubsequence* function is invoked again with the following parameters: child node, clone, and list of sub-sequences *listSub*. When the node parameter is empty, there are no more children to be processed; hence, the *subsequence* value will be an ST sub-sequence. Then, the value of the *subsequence* variable is added to *listSub*. At the end of the function, variable *listSub* will have a list of all ST sub-sequences that satisfy both e_1 and e_2.

Suppose a trajectory ST has five POIs, and for the sake of simplicity, we highlighted only the trajectory st_2. Following Algorithm 1, two sub-sequences are extracted from st_2, which we call SST_1 and SST_2, containing the category sub-sequences $(c_2\ c_5)$ and $(c_4\ c_5)$.

	c_1	c_2	c_3	c_4	c_5	
st_2=(	shop	museum	church	museum	tower	)

Algorithm 4 Extract Sub-sequence from Tree Algorithm

```
1: function EXTRACTSUBSEQUENCE (node, subsequence, listSubs)
2:     numChildren ← node.children.size
3:     subsequenceSize ← subsequence.size
4:     for all child in node.children do
5:         if numChildren > 1 then
6:             clone ← copy of subsequence
7:             clone [subsequenceSize + 1] ← node.value
8:             extractSubsequence (child, clone, listSubs)
9:         else
10:            subsequence [subsequenceSize + 1] ← node.value
11:            extractSubsequence (child, subsequence, listSub)
12:        end if
13:    end for
14:    if numChildren == 0 then
15:        subsequence [subsequenceSize + 1] ← node.value
16:        listSubs.add (subsequence)
17:    end if
18: end function
```

3.2.3. Transforming a Sub-Sequence into a Vector

After identifying SST_1 and SST_2 sub-sequences, SETHE creates a sentence δ for each one of them, similar to what was performed for query Q_{f3}. A new sentence δ_{sst} and a vector $\overrightarrow{v}_{sst}$ are created for each sub-sequence, where $\delta_{sst} = (r_1^1, r_1^2, \ldots, r_1^b, r_2^1, r_2^2, \ldots, r_2^b, r_n^1, r_n^2, \ldots, r_n^b)$ and $\overrightarrow{v}_{sst} = (v_1^1, v_1^2, \ldots, v_1^b, v_2^1, v_2^2, \ldots, v_2^b, v_n^1, v_n^2, \ldots, v_n^b)$, such that r corresponds to the SST optional aspects and $v_i^j = score(y_i^j, r_i^j)$.

The score is calculated for each term of δ_{sst}, and its value is vector $\overrightarrow{v}_{sst}$. The score function uses a distance function to calculate the closeness of a term δ_{sst} to the same index term of δ_q.

The smaller the distance, the higher the score for the term of δ_{sst}. If there are two terms, one belonging to query Q and the other belonging to a sub-sequence of ST, such that $y_i^j \, \epsilon \, \delta_q$ and $r_i^j \, \epsilon \, \delta sst$, the equation to calculate the score between the two terms is

$$score(y_i^j, r_i^j) = \begin{cases} 0, & if \quad d_j(y_i^j, r_i^j) > thr_j, \\ w_j, & if \quad y_i^j = (.*), \\ \left| \left| \dfrac{(w_j * d_j(y_i^j, r_i^j))}{thr_j} \right| - w_j \right|, & otherwise \end{cases} \tag{2}$$

where $w_j \in W$, $d_j \in D$ and $thr_j \in L$

The trajectory coefficient of ST consists of the highest similarity value between the query vector of Q and the vectors of ST. The similarity function must return a value between zero and one. Examples of such similarity functions include Jaccard and cosine functions. Let Vts be the set of all vectors created by the ST sub-sequences. The coefficient is calculated as

$$coef(ST) = max(similarity(\vec{v}_q, \vec{x})) \mid \vec{v}_q \in V_q, \ \vec{x} \ in \ V_{ts} \tag{3}$$

where, $0 \ll similarity(.,.) \ll 1$

A composite query $CQ = (Q_1, Q_2, \ldots, Q_n)$ consists of multiple queries gathered into a single query, in which SETHE executes each query separately and merges the results of each query into a single result set.

4. Running Example

This section presents an example of how SETHE works using a query and a database composed of three trajectories. Let us suppose a Q query that looks for trajectories in which a mobile object initially goes to the *Leaning Tower of Pisa,* then visits a chapel or a church, and later on, stops at a museum.

Let the query $Q = (E, A, W, D, L)$, where

- $E = \{\langle name, \langle Leaning\_Tower\_of\_Pisa;\ .*;\ .*\rangle\rangle,$ $\langle category, \langle\ .*;\ (church|chapel); museum\ \rangle\rangle\}$. As we define the first point as the *Leaning Tower of Pisa*, we do not need to specify the category of the first point.
- $A = \{\langle transport\ mean, \langle taxi, bus, walk\rangle\rangle, \langle rating, \langle 5, 5, 4\rangle\rangle\}$.
- $W = \{0.7,\ 0.3\}$.
- $D = \{equals,\ Euclidean\}$.
- $L = \{1,\ 5\}$.

In this example, we can see that the transport mean has a higher weight than rating. Transforming the query into sentence δ_q and applying Equation (1), we obtain the query vector

$$\vec{v}_q = (0.7,\ 0.3,\ 0.7,\ 0.3,\ 0.7,\ 0.3)$$

Suppose that we have a semantic trajectory database ST, as described in Section 3.2.1. SETHE performs a search using the *regex* function, which receives as parameters the textual trajectory and the regular expression formed by the elements of e_1 and e_2. In this case, two *regex* functions were used: one for st_1 and the other for st_2.

$$regex(st_1, \text{“}Leaning\_Tower\_of\_Pisa\text{”})\ and\ regex(st_2, \text{“}(church|chapel)(.*)(museum)\text{”})$$

After searching the collection of semantic trajectories using regex, SETHE retrieves three trajectories: $STF = \{<name, st_{F1}>, <category, st_{F2}>, <trasportmeans, st_{F3}>, <rating, st_{F4}>\}$, $STG = \{<name, st_{G1}>, <category, st_{G2}>, <trasportmeans, st_{G3}>, <rating, st_{G4}>\}$, and $STH = \{<name, st_{H1}>, <category, st_{H2}>, <trasportmeans, st_{H3}>, <rating, st_{H4}>\}$. For simplicity, we highlighted only the trajectory category and aspects, shown below. In addition, for ease of explanation, we placed an index on each term in the trajectory:

$$\underset{\quad f_1 \quad f_2 \quad f_3 \quad f_4 \quad f_5 \quad f_6 \quad f_7}{st_{F2} = \langle tower,\ gate,\ tower,\ chapel,\ church,\ campanile,\ museum\rangle}$$

$$\underset{\quad g_1 \quad g_2 \quad g_3 \quad g_4 \quad g_5}{st_{G2} = \langle tower,\ chapel,\ campanile,\ chapel,\ museum\rangle}$$

$$\underset{\quad h_1 \quad h_2 \quad h_3 \quad h_4 \quad h_5 \quad h_6}{st_{H2} = \langle tower,\ piazza,\ chapel,\ museum,\ chapel,\ museum\rangle}$$

The trajectories of the means of transport and rating aspects relate to ST_F, ST_G, and ST_H trajectories are, respectively,

$st_{F3} = \langle Subway, Subway, Bus, Walk, Subway, Subway, Walk\rangle$
$st_{F4} = \langle 4, 4, 4, 5, 4, 3, 3\rangle$
$st_{G3} = \langle Walk, Walk, Subway, Bus, Subway\rangle$
$st_{G4} = \langle 5, 3, 4, 3, 3\rangle$
$st_{H3} = \langle Taxi, Taxi, Bus, Subway, Bus, Walk\rangle$
$st_{H4} = \langle 5, 2, 3, 4, 3, 3\rangle$

The second step is to identify the sub-sequences that satisfy the e_1 and e_2 expressions. Following Algorithm 1, a tree of paths is constructed, as depicted in Figure 3. Each level of the tree, except the root, represents an expression in either e_1 or e_2. Each POI of ST that satisfies $regex(st_1, e_1)$ or $regex(st_2, e_2)$ is added to the tree as a child of lower index expressions. For example, using the function $regex(f_4, \text{“}(church|chapel)\text{”})$, the category f_4 matches the regular expression $(church|chapel)$, so the algorithm adds f_4 as an f_1 and f_3 child in the tree. Category h_4, for example, is added only as a child of h_3 and not of h_5, as h_4 occurs before h_5 in the trajectory.

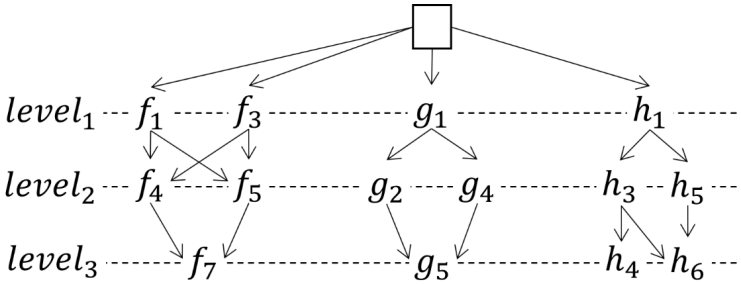

Figure 3. POI tree.

The sub-sequences extracted from the POIs tree are:

$$\begin{array}{lll} \sigma_{Fa} = \langle f_1, f_4, f_7 \rangle & & \\ \sigma_{Fb} = \langle f_1, f_5, f_7 \rangle & \sigma_{Ga} = \langle g1, g2, g5 \rangle & \sigma_{Ha} = \langle h1, h3, h4 \rangle \\ \sigma_{Fc} = \langle f_3, f_4, f_7 \rangle & \sigma_{Gb} = \langle g1, g4, g5 \rangle & \sigma_{Hb} = \langle h1, h3, h6 \rangle \\ \sigma_{Fd} = \langle f_3, f_5, f_7 \rangle & & \sigma_{Hc} = \langle h1, h5, h6 \rangle \end{array}$$

After identifying the trajectory sub-sequences, the next step is to calculate the score of each optional aspect to compose the vector that will serve as a similarity comparison with the $\vec{v}_q$ vector.

Tables 4–7 present the sentences of each ST_F sub-sequence that were created by interleaving the optional aspects. Each term corresponds to a sub-sequence optional aspect, starting with the transport mean and rating aspects related to each POI sub-sequence. As specified in the query Q, the *equals* function is used to calculate the similarity between two aspects of the transport mean type. Therefore, there are only two possible values: 1 (one), if the values are the same, and 0 (zero), if they are different. For the rating aspect, we use the Euclidean distance function. Applying Equation (2) to each word, we find the score for each aspect. For example, the rating score of value 4 (four) is calculated as follows:

$$score_{rating}(5,4) = \left| \left| \frac{0.3(4-5)}{5} \right| - 0.3 \right| = 0.24$$

Table 4. Scores for the sub-sequence σ_{Fa}.

σ_{Fa}	f_1		f_4		f_7	
δ_{Fa}	Subway	4	Walk	5	Walk	3
score δ_{Fa}	0	0.24	0	0.3	0.7	0.24

Table 5. Scores for the sub-sequence σ_{Fb}.

σ_{Fb}	f_1		f_5		f_7	
δ_{Fb}	Subway	4	Subway	4	Walk	3
score δ_{Fb}	0	0.24	0	0.24	0.7	0.24

Table 6. Scores for the sub-sequence σ_{Fc}.

σ_{Fc}	f_3		f_4		f_7	
δ_{Fc}	Bus	4	Walk	5	Walk	3
score δ_{Fc}	0	0.24	0	0.3	0.7	0.24

Table 7. Scores for the sub-sequence σ_{Fd}.

σ_{Fd}	f_3		f_5		f_7	
δ_{Fd}	Bus	4	Subway	4	Walk	3
score δ_{Fd}	0	0.24	0	0.24	0.7	0.24

Tables 8 and 9 present the score for the ST_G, and Tables 10–12 contain the scores for ST_H sub-sequences.

Table 8. Scores for the sub-sequence σ_{Ga}.

σ_{Ga}	g_1		g_2		g_5	
δ_{Ga}	Walk	5	Walk	3	Subway	3
score δ_{Ga}	0	0.3	0	0.18	0	0.24

Table 9. Scores for the sub-sequence σ_{Gb}.

σ_{Gb}	g_1		g_4		g_5	
δ_{Gb}	Walk	5	Bus	3	Subway	3
score δ_{Gb}	0	0.3	0.7	0.18	0	0.24

Table 10. Scores for the sub-sequence σ_{Ha}.

σ_{Ha}	h_1		h_3		h_4	
δ_{Ha}	Taxi	5	Bus	3	Subway	4
score δ_{Ha}	0.7	0.3	0.7	0.18	0	0.3

Table 11. Scores for the sub-sequence σ_{Hb}.

σ_{Hb}	h_1		h_3		h_6	
δ_{Hb}	Taxi	5	Bus	3	Walk	3
score δ_{Hb}	0.7	0.3	0.7	0.18	0.7	0.24

Table 12. Scores for the sub-sequence σ_{Hc}.

σ_{Hc}	h_1		h_5		h_6	
δ_{Hc}	Taxi	5	Bus	3	Walk	3
score δ_{Hc}	0.7	0.3	0.7	0.18	0.7	0.24

The coordinates of each trajectory vector contain the score calculated for each sub-sequence. Thus, we have the following vectors for the ST_F sub-sequences:

$$\vec{v}_{Fa}(0, 0.24, 0, 0.3, 0.7, 0.24),\ \vec{v}_{Fb}(0, 0.24, 0, 0.24, 0.7, 0.24),$$
$$\vec{v}_{Fc}(0, 0.24, 0, 0.3, 0.7, 0.24),\ \vec{v}_{Fd}(0, 0.24, 0, 0.24, 0.7, 0.24)$$

In this example, we use the Jaccard index to calculate the similarity between the vectors of the trajectories and the query vector $\vec{v}_q$. Equation (4) calculates the Jaccard index:

$$J(\vec{v}_i, \vec{v}_q) = \frac{\sum_k min(\vec{v}_i[k], \vec{v}_q[k])}{\sum_k max(\vec{v}_i[k], \vec{v}_q[k])} \tag{4}$$

Applying Equation (4) for ST_F and $\vec{v}_q$ vectors, we have the following values:

$$J(\vec{\sigma}_{Fa}, \vec{v}_q) = 0.493,\ J(\vec{\sigma}_{Fb}, \vec{v}_q) = 0.473$$

$$J(\vec{\sigma}_{Fc}, \vec{v}_q) = 0.493,\ J(\vec{\sigma}_{Fd}, \vec{v}_q) = 0.473$$

Applying Equation (3), the coefficient for the ST_F trajectory is 0.493. After performing the same process for the ST_G and ST_H trajectories, we have the final result:

$$coef(ST_H) = 0.94,\ coef(ST_F) = 0.493,\ coef(ST_G) = 0.47$$

Therefore, the ST_H trajectory has the highest coefficient, which is closest to the user query. Second is the ST_F path, and ST_G is the path with the least similarity to the Q query.

5. Experiments and Results

We used the *TripBuilder* dataset [16] to evaluate the performance of our solution. We evaluated the performance of SETHE based on the framework described in [15], which presents a set of 10 queries for the city of Pisa, Italy.

5.1. Dataset

The *TripBuilder* RDF dataset contains 1,617,582 triples and 55,474 trajectories, modeled into *Trajectory*, *Stop*, *Move*, *Transportation*, and *POI* classes. Figure 4 shows the UML representation of *TripBuilder*. A trajectory can have several stop and moves, as represented by the * symbol in the relationship. Each trajectory has start and end points, and each point represents a POI. The *Move* class represents the transition between two stops, and is semantically enriched by the *Transportation* class.

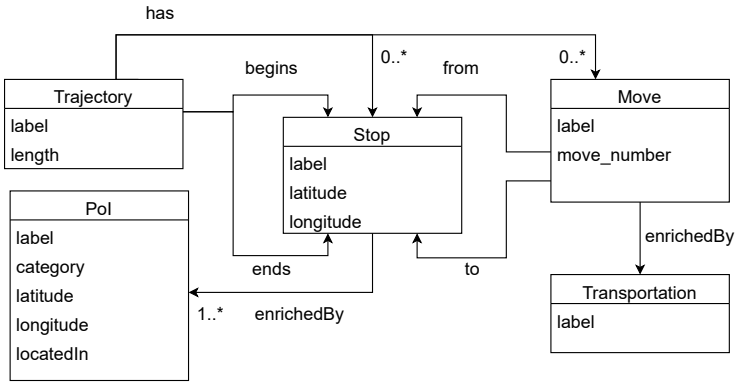

Figure 4. Representation of the *TripBuilder* dataset.

To conduct the experiments with SETHE, we transformed the RDF TripBuilder dataset into a text dataset. We modeled a database to store the textual trajectories, as depicted in Figure 5. The *Trajectory* entity has a one-to-many relationship to each entity representing a different trajectory type. The *POI* entity represents the textual trajectory where each point contains the name of the place where the moving object stopped. The *Category* entity contains the category textual trajectories. The *Move* entity contains the textual trajectories of the transport mean utilized to reach each stop. The *LocatedIn* entity contains the trajectories of the regions where the moving object has stopped.

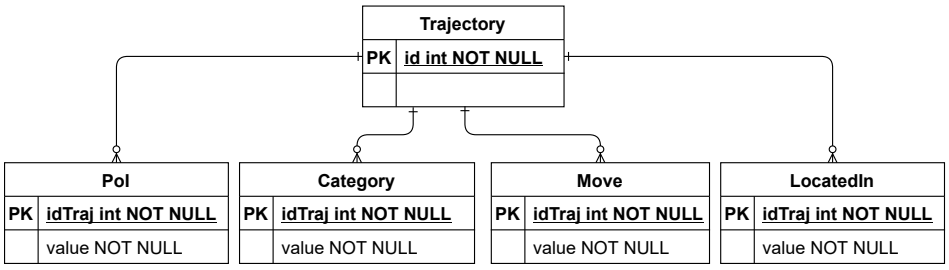

Figure 5. ER diagram of the textual trajectories database.

Table 13 presents a sample of data of the POI name trajectory stored in the *Value* column. Table 14 shows category trajectories. Table 15 presents some examples of transport mean trajectories. The first move does not have an associated transport mean; therefore, it receives the value N/A. Indeed, SETHE enables trajectories with missing information for a given aspect. When this happens, i.e., a given POI without aspect, we use the special value N/A. For example, consider a trajectory in which the first and third POIs do not have a particular aspect, say transportation. Then the transportation aspect for that trajectory would be represented in the following way: $\langle N/A, Subway, N/A, Taxi, Bus, Subway \rangle$.

Table 13. Example of POIs name trajectory.

IdTraj	Value
TF10018	Statua_equestre_di_Cosimo_I_de_Medici, Loggia_della_Signoria, Castello_dAltafronte, Torre_della_Pagliazza, Palazzo_Bartolini-Torrigiani
TF10019	Palazzo_dei_Vescovi_a_San_Miniato_al_Monte, Basilica_di_San_Miniato_al_Monte
TF10027	Palazzo_Roffia, Porta_della_Mandorla, Campanile_di_Giotto, Battistero_di_San_Giovanni_(Firenze), Porta_della_Mandorla, Campanile_di_Giotto, Torre_dei_Caponsacchi, Palazzo_dei_Vescovi_a_San_Miniato_al_Monte, Basilica_di_Santa_Croce, Torre_dei_Caponsacchi

Table 14. Example of category trajectory.

IdTraj	Value
TF10018	scultureafirenze, loggedifirenze, castellidifirenze, torridifirenze,palazzidifirenze
TF10019	palazzidifirenze, basilichedifirenze
TF10027	palazzidifirenze, cattedralidellaprovinciadifirenze, campanili, battisteridellatoscana, cattedralidellaprovinciadifirenze,campanili, torridifirenze, palazzidifirenze, basilichedifirenze, torridifirenze

Table 15. Example of transport mean trajectory.

IdTraj	Value
TF10018	N/A, Subway, Taxi, Subway, Bus, Subway
TF10019	N/A, Subway
TF10027	N/A, Walk, Bus, Subway, Subway, Bus, Taxi, Taxi, Taxi, Taxi

5.2. Results and Discussion

The experiments were carried out on a computer with a Core i7-7700 3.60 GHz processor, 32 GB of RAM, and 500 GB HD, with a GNU/Linux Ubuntu 18.04 operating system. We installed the RDF dataset in the tuple database (TDB) of the Apache Jena Fuseki 4.3.2 server running on the Java platform jdk-16.0.2. We used PostgreSQL 13.2 to store the textual trajectory database. We converted the RDF triples to CSV spreadsheets, removed accents and special characters, and then loaded the data into the text database. Izquierdo et al. [15] described a semantic trajectory search framework and specified ten queries for city of Pisa to evaluate their framework performance. The same queries were used to evaluate the SETHE framework. The queries are listed in Table 16.

Table 17 shows how to use the SETHE framework to answer the aforementioned queries. Some queries require POIs visited consecutively. Thus, the proximity between the stopping points was also used as a trajectory aspect. In this case, proximity refers to the number of stops between two POIs. In SETHE, when a proximity attribute is set to a tilde symbol (∼), the closer two POIs are, the higher their score. Izquierdo et al. [15] use the *equals* operator to compare means of transport; therefore, we used the distance function

equals for the transport mean aspect. In queries that use both aspects (e.g., transport mean and proximity), we adopted the exact weight of 0.5 in these examples.

Table 16. Semantic trajectories queries for the city of Pisa.

Qid	Free Text Query
Q1	Trajectories that stop at a museum and then at a chapel.
Q2	Trajectories that stop at a tower, then stop at a chapel or church, then stop at a chapel or church again, and then at a museum.
Q3	Trajectories that stop at least once in a tower, and then at a museum.
Q4	Trajectories that stop at the Lion Tower and then at the Leaning Tower, or stop at the Leaning Tower and then at the Lion Tower.
Q5	Trajectories that begin at a museum and then end at a chapel.
Q6	Trajectories that stop at a museum and, later on, end at a chapel or a church optionally.
Q7	Trajectories that begin at a chapel, stop at zero or more chapels, and end at a chapel.
Q8	Trajectories that stop at a museum and then take a bus to a chapel.
Q9	Trajectories that begin at a chapel or a church, always move by bus between stops, and end at the Leaning Tower.
Q10	Trajectories that begin at a tower, then walk to take a bus to a church, and then, using any transportation means, end at a palace.

Table 17. Expressing queries using SETHE.

Qid	SETHE Query
Q1	$e_2 = \langle museidipisa; cappelledipisa \rangle$ $\alpha_1 = \langle .*;\ \sim \rangle$
Q2	$e_2 = \langle torridipisa; (cappelledipisa \vert chiesedipisa);$ $(cappelledipisa \vert chiesedipisa); museidipisa \rangle$ $\alpha_1 = \langle .*;\ \sim; .*; \sim \rangle$
Q3	$e_2 = \langle torridipisa; museidipisa \rangle$ $\alpha_1 = \langle .*;\ \sim \rangle$
Q4	$Q4 = \{Q4_1 = \{\{\langle name, \langle Torre\_del\_Leone; Torre\_pendente\_di\_Pisa \rangle\rangle,$ $\{\langle proximity, \langle .*; \sim \rangle\rangle\}\},$ $Q4_2 = \{\{\langle name, \langle Torre\_pendente\_di\_Pisa; Torre\_del\_Leone \rangle\rangle,$ $\{\langle proximity, \langle .*;\ \sim \rangle\rangle\}\}\}$
Q5	$e_2 = \langle\ \hat{}\ (museidipisa); (cappelledipisa)\$ \rangle$ $\alpha_1 = \langle .*;\ \sim \rangle$
Q6	$e_2 = \langle museidipisa; (cappelledipisa \vert chiesedipisa) * \$ \rangle$ $\alpha_1 = \langle .*;\ .* \rangle$
Q7	$e2 = \langle \hat{}(cappelledipisa); (cappelledipisa)\$ \rangle$ $\alpha_1 = \langle .*;\ .* \rangle$
Q8	$e_2 = \langle museidipisa; cappelledipisa \rangle$ $\alpha_1 = \langle .*;\ Bus \rangle$ $\alpha_2 = \langle .*;\ \sim \rangle$
Q9	$e_1 = \langle .*; (Torre\_pendente\_di\_Pisa)\$ \rangle$ $e_2 = \langle \hat{}(cappelledipisa \vert chiesedipisa); .* \rangle$ $\alpha_1 = \langle .*;\ (?-)Bus \rangle$
Q10	$e_2 = \langle \hat{}(torridipisa); .*; chiesedipisa; (palazzidipisa)\$ \rangle$ $\alpha_1 = \langle .*;\ Walk;\ Bus;\ .* \rangle$

As shown in Table 17, when the query does not have a value of e_1, it is assumed that this value is empty. All queries are simple, except for query Q4, which is a composite query.

We compared the performance of the SETHE PostgreSQL queries to that of the SPARQL queries. Regular expressions were extended with $\sim$ and $(?-)$ operators. Each query was executed ten times for both SPARQL and SETHE. Figure 6 shows the average execution time for each query on a logarithmic scale. We observed that SETHE has a better response time in most cases than SPARQL queries [15]. The *Q*10 SPARQL query was not charted because it took approximately one hour to run.

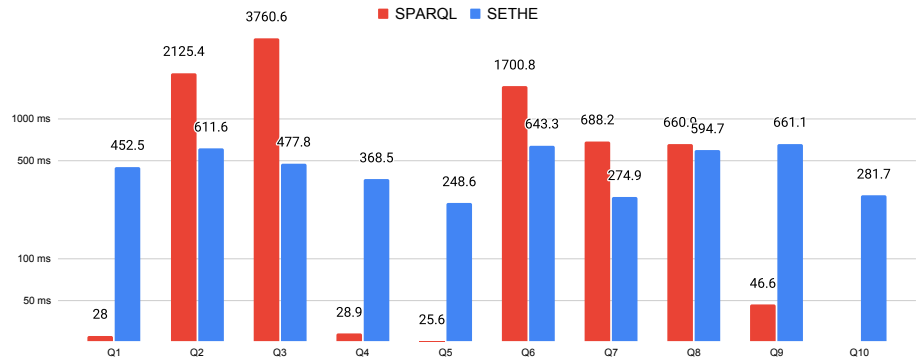

Figure 6. Performance graph.

Owing to the ranking algorithm, which implies better recall, as shown in Figure 7. The first SETHE query results are the same as the SPARQL query; they fit perfectly with the user query. The blue bars in the graph in Figure 7 represent the result set returned by SETHE that was not retrieved by SPARQL. The trajectories in the blue area are similar to the user query specifications but do not fit perfectly with the SPARQL queries.

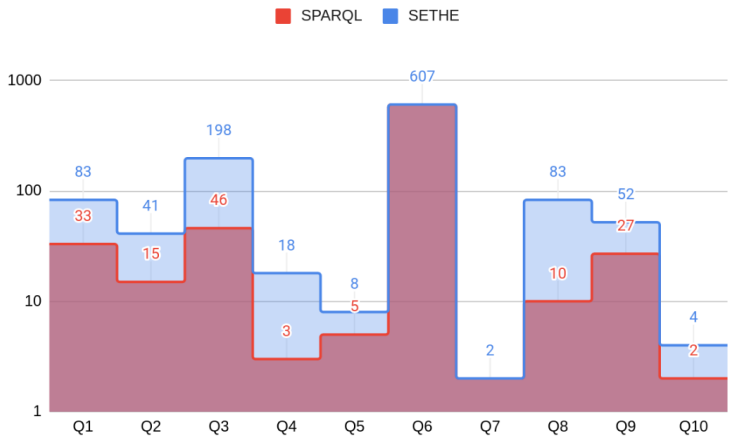

Figure 7. Number of results per query.

Another important issue to be analyzed is the storage space of each investigated approach. The Apache Jena server uses TDB to store the RDF graph. Figure 8 shows the storage space between TDB and PostgreSQL. It was observed that the TDB demanded more than five times the memory size demanded by our textual approach.

The *TripBuilder* database was obtained from the repository https://figshare.com/articles/online_resource/Trajectories_RDF_Dataset_From_TripBuilder/11559090 (accessed on 22 May 2022). The source code for our project can be downloaded from https://github.com/DamiaoRA/SETHE (accessed on 22 May 2022).

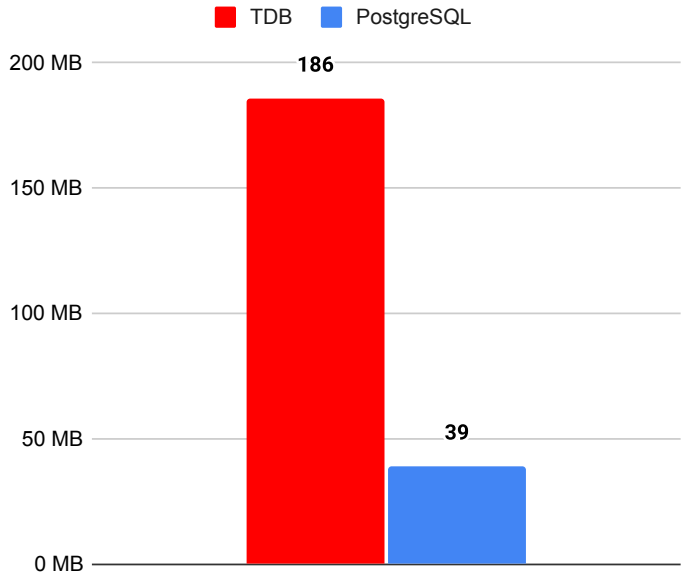

Figure 8. Memory space consumption.

6. Conclusions

The insertion of context-based information into trajectory data results in semantically enriched trajectories. Thus, trajectories may be analyzed from different perspectives, also known as aspects. Each perspective enables spatiotemporal context-based information analytics. In this study, these trajectories were called aspect-based semantic trajectories. Depending on the application, the trajectory aspects may vary significantly in terms of quantity and type. Some related approaches represent semantic trajectories using RDF graphs, ontologies, or conceptual models in which the search process is based only on an exact match. Depending on the complexity of the query, exact matches may yield few or no results.

This article proposes the SETHE framework, a search engine for querying aspect-based semantic trajectory datasets using text processing. The SETHE implements partial matching using a similarity coefficient between the aspect-based semantic trajectory and the user's query to rank the result set. In traditional semantic trajectory search tools, there is no weight related to a given aspect; hence, all aspects have the same priority as the user. Our approach uses a distance function and a weight assigned to each aspect that impacts the ranking algorithm. The result set contains trajectories ranked by their coefficients calculated from the distance functions and weights. Using a ranking approach, the trajectories closest to the user query may be returned.

We also present a new approach to representing aspect-based semantic trajectory data, where each trajectory is represented only by text. The experiments using this approach demonstrated that the memory consumption for storing trajectories and their aspects is lower than that of an approach using an RDF graph, one of the main semantic trajectory representations used.

To assess the relevance of our work, we compared the results with those of one of the most recent studies in the field of semantic trajectory search. The results demonstrated that the SETHE had a better average response time. Furthermore, a SETHE query usually returns more results as we use a partial-match ranked approach. In future work, we intend to use the normalized discounted cumulative gain (NDCG) to measure ranking quality. We plan to extend our SETHE framework to encompass multidimensional modeling so that users can run rollup and drill down operators over trajectory aspects. Finally, we will work on implementing a graphical user interface (GUI) and perform a user assessment using the ISO 9241 standard—parts 14, 16, and 17.

Author Contributions: Conceptualization, Damião Ribeiro de Almeida and Cláudio de Souza Baptista; Methodology, Damião Ribeiro de Almeida, Fabio Gomes de Andrade and Cláudio de Souza Baptista; Software, Damião Ribeiro de Almeida; Writing—original draft, Damião Ribeiro de Almeida; Writing—review and editing, Cláudio de Souza Baptista and Fabio Gomes de Andrade; Supervision, Cláudio de Souza Baptista. All authors have read and agreed to the published version of the manuscript.

Funding: This research received external funding from the Computing Department of the Federal University of Campina Grande (UFCG).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data used in this research can be extracted from: https://figshare.com/articles/online_resource/Trajectories_RDF_Dataset_From_TripBuilder/11559090.

Acknowledgments: The second author would like to thank the National Council for Scientific and Technological Development (CNPQ), Brazil, for partially funding this research.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kong, X.; Li, M.; Ma, K.; Tian, K.; Wang, M.; Ning, Z.; Xia, F. Big trajectory data: A survey of applications and services. *IEEE Access* **2018**, *6*, 58295–58306. [CrossRef]
2. Fileto, R.; Raffaetà, A.; Roncato, A.; Sacenti, J.A.; May, C.; Klein, D. A semantic model for movement data warehouses. In Proceedings of the 17th International Workshop on Data Warehousing and OLAP, Shanghai, China, 3–7 November 2014; pp. 47–56.
3. Nardini, F.M.; Orlando, S.; Perego, R.; Raffaetà, A.; Renso, C.; Silvestri, C. Analysing trajectories of mobile users: From data warehouses to recommender systems. In *A Comprehensive Guide through the Italian Database Research over the Last 25 Years*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 407–421.
4. Wagner, R.; Macedo, J.A.F.d.; Raffaetà, A.; Renso, C.; Roncato, A.; Trasarti, R. Mob-warehouse: A semantic approach for mobility analysis with a trajectory data warehouse. In Proceedings of the International Conference on Conceptual Modeling, Hong Kong, China, 11–13 November 2013; Springer: Berlin/Heidelberg, Germany, 2013; pp. 127–136.
5. Alsahfi, T.; Almotairi, M.; Elmasri, R. A survey on trajectory data warehouse. *Spat. Inf. Res.* **2020**, *28*, 53–66. [CrossRef]
6. Emmanouilidis, C.; Koutsiamanis, R.A.; Tasidou, A. Mobile guides: Taxonomy of architectures, context awareness, technologies and applications. *J. Netw. Comput. Appl.* **2013**, *36*, 103–125. [CrossRef]
7. Fileto, R.; May, C.; Renso, C.; Pelekis, N.; Klein, D.; Theodoridis, Y. The Baquara2 Knowledge-Based Framework for Semantic Enrichment and Analysis of Movement Data. *Data Knowl. Eng.* **2015**, *98*, 104–122. [CrossRef]
8. Qin, Y.; Sheng, Q.Z.; Falkner, N.J.; Dustdar, S.; Wang, H.; Vasilakos, A.V. When things matter: A survey on data-centric internet of things. *J. Netw. Comput. Appl.* **2016**, *64*, 137–153. [CrossRef]
9. Goodchild, M.F. Citizens as sensors: The world of volunteered geography. *GeoJournal* **2007**, *69*, 211–221. [CrossRef]
10. Parent, C.; Spaccapietra, S.; Renso, C.; Andrienko, G.; Andrienko, N.; Bogorny, V.; Damiani, M.L.; Gkoulalas-Divanis, A.; Macedo, J.; Pelekis, N.; et al. Semantic trajectories modeling and analysis. *ACM Comput. Surv. CSUR* **2013**, *45*, 42. [CrossRef]
11. Almeida, D.R.d.; Baptista, C.d.S.; Andrade, F.G.d.; Soares, A. A Survey on Big Data for Trajectory Analytics. *ISPRS Int. J. Geo-Inf.* **2020**, *9*, 88. [CrossRef]
12. Petry, L.M.; Ferrero, C.A.; Alvares, L.O.; Renso, C.; Bogorny, V. Towards semantic-aware multiple-aspect trajectory similarity measuring. *Trans. GIS* **2019**, *23*, 960–975. [CrossRef]
13. Mello, R.d.S.; Bogorny, V.; Alvares, L.O.; Santana, L.H.Z.; Ferrero, C.A.; Frozza, A.A.; Schreiner, G.A.; Renso, C. MASTER: A multiple aspect view on trajectories. *Trans. GIS* **2019**, *23*, 805–822. [CrossRef]
14. Noël, D.; Villanova-Oliver, M.; Gensel, J.; Le Quéau, P. Modeling semantic trajectories including multiple viewpoints and explanatory factors: Application to life trajectories. In Proceedings of the 1st International ACM SIGSPATIAL Workshop on Smart Cities and Urban Analytics, Bellevue, WA, USA, 3–6 November 2015; pp. 107–113.
15. Izquierdo, Y.T.; Monteagudo Garcia, G.; Casanova, M.A.; Paes Leme, L.A.P.; Sardianos, C.; Tserpes, K.; Varlamis, I.; Ruback Rodrigues, L.C. Stop-and-move sequence expressions over semantic trajectories. *Int. J. Geogr. Inf. Sci.* **2021**, *35*, 793–818. [CrossRef]
16. Brilhante, I.; Macedo, J.A.; Nardini, F.M.; Perego, R.; Renso, C. Tripbuilder: A tool for recommending sightseeing tours. In Proceedings of the European Conference on Information Retrieval, Amsterdam, The Netherlands, 13–16 April 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 771–774.
17. Güting, R.H.; Schneider, M. *Moving Objects Databases*; Elsevier: Amsterdam, The Netherlands, 2005.
18. Yan, Z.; Chakraborty, D.; Parent, C.; Spaccapietra, S.; Aberer, K. SeMiTri: A framework for semantic annotation of heterogeneous trajectories. In Proceedings of the 14th International Conference on Extending Database Technology, Uppsala, Sweden, 21–24 March 2011; ACM: New York, NY, USA, 2011; pp. 259–270.

19. Spaccapietra, S.; Parent, C.; Damiani, M.L.; de Macedo, J.A.; Porto, F.; Vangenot, C. A conceptual view on trajectories. *Data Knowl. Eng.* **2008**, *65*, 126–146. [CrossRef]
20. Bogorny, V.; Renso, C.; de Aquino, A.R.; de Lucca Siqueira, F.; Alvares, L.O. Constant—A Conceptual Data Model for Semantic Trajectories of Moving Objects. *Trans. GIS* **2014**, *18*, 66–88. [CrossRef]
21. Nikitopoulos, P.; Vlachou, A.; Doulkeridis, C.; Vouros, G.A. DiStRDF: Distributed Spatio-temporal RDF Queries on Spark. In Proceedings of the EDBT/ICDT Workshops, Vienna, Austria, 26 March 2018; pp. 125–132.
22. Dividino, R.; Soares, A.; Matwin, S.; Isenor, A.W.; Webb, S.; Brousseau, M. Semantic Integration of Real-Time Heterogeneous Data Streams for Ocean-Related Decision Making. In Proceedings of the Big Data and Artificial Intelligence for Military Decision Making, Bordeaux, France, 30 May–1 June 2018. [CrossRef]
23. Alvares, L.O.; Bogorny, V.; Kuijpers, B.; de Macedo, J.A.F.; Moelans, B.; Vaisman, A. A model for enriching trajectories with semantic geographical information. In Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems, Seattle, WA, USA, 7–9 November 2007; pp. 1–8.
24. Chang, B.; Park, Y.; Kim, S.; Kang, J. DeepPIM: A deep neural point-of-interest imputation model. *Inf. Sci.* **2018**, *465*, 61–71. [CrossRef]
25. Gusfield, D. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*; Cambridge University Press: New York, NY, USA, 1997.