INTRODUCTION

The popularization of mobile devices, social networks, and the Internet of Things enables a vast collection of mobility data represented by trajectories. Trajectories are sequences of points located in space and time that can describe any movement behavior of people, animals, vehicles, ships, hurricanes, and so on. The trajectory definition has evolved from raw, before 2007, to semantic in 2008 (Spaccapietra et al., 2008) and then to the very recent concept of multiple aspect trajectory (MAT) in 2016 (Ferrero, Alvares, & Bogorny, 2016; Mello et al., 2019).

Abstract

The semantic enrichment of mobility data with several information sources has led to a new type of movement data, the so-called multiple aspect trajectories. Comparing multiple aspect trajectories is crucial for several analysis tasks such as querying, clustering, similarity, and classification. Multiple aspect trajectory similarity measurement is more complex and computationally expensive, because of the large number and heterogeneous aspects of space, time, and semantics that require a different treatment. Only a few works in the literature focus on optimizing all these dimensions in a single solution, and, to the best of our knowledge, none of them proposes a fast point-to-point comparison. In this article we propose the Multiple Aspect Trajectory Index, an index data structure for optimizing the point-to-point comparison of multiple aspect trajectories, considering its three basic dimensions of space, time, and semantics. Quantitative and qualitative evaluations show a processing time reduction of up to 98.1%.

1 | INTRODUCTION

The popularization of mobile devices, social networks, and the Internet of Things enables a vast collection of mobility data represented by trajectories. Trajectories are sequences of points located in space and time that can describe any movement behavior of people, animals, vehicles, ships, hurricanes, and so on. The trajectory definition has evolved from raw, before 2007, to semantic in 2008 (Spaccapietra et al., 2008) and then to the very recent concept of multiple aspect trajectory (MAT) in 2016 (Ferrero, Alvares, & Bogorny, 2016; Mello et al., 2019).
Figure 1 shows an example of a multiple aspect trajectory where each trajectory point possesses several semantic attributes considering multiple points of view (personal, environmental, transportation means, social media posts, etc.). The example shows the movement of an object who wears a smartwatch and works at a smart office equipped with numerous sensors and microphones.

We observe from the figure that the MAT is a very complex data type. The richer a trajectory is in terms of semantic aspects, the more knowledge about the moving object can be extracted, but also the more costly it becomes to process the data. Trajectory data are complex by nature, being composed of a sequence of spatio-temporal points, each one having the dimensions of space, time, and a set of semantics. By semantic aspect or dimension, we mean any type of information that can be added to trajectories that is neither spatial nor temporal.

In trajectory data analysis, comparing trajectories is crucial for several analysis tasks such as querying, clustering, similarity, and classification, all research topics that have attracted great interest in recent years. Trajectory similarity measurement is fundamental to querying and clustering moving objects with similar characteristics. There are several similarity/distance measures in the literature such as dynamic time warping (DTW), modified dynamic time warping, longest common subsequence (LCSS), edit distance for real sequences (EDR), and Fréchet distance, but most of them were either developed for time series or do not support all three dimensions of mobility data, that is, space, time, and semantics.

Similarity measures that were specifically developed for trajectories include uncertain movement similarity (UMS; Furtado, Alvares, Pelekis, Theodoridis, & Bogorny, 2018), multidimensional similarity measure (MSM; Furtado, Kopanaki, Alvares, & Bogorny, 2016) and multiple aspect trajectory similarity (MUITAS; Petry, Ferrero, Alvares, Renso, & Bogorny, 2019). UMS is very robust for spatial similarity, but it does not consider time and semantic dimensions, which are fundamental to mobility data. On the other hand, MSM and MUITAS have outperformed the well-known older measures DTW (Berndt & Clifford, 1994), LCSS (Vlachos, Kollios, & Gunopulos, 2002), and EDR (Chen, Özsu, & Oria, 2005). EDR and LCSS require a matching in all dimensions of two points to consider them as similar, while MSM and MUITAS do not. MSM and MUITAS are flexible measures that consider similar two trajectories that do similar things but not necessarily in the same order, thus not forcing a match in all dimensions. This flexibility is reasonable since it is rare that two moving objects do precisely the same things, at the same place and time, in the same sequence. MUITAS is also flexible in considering the dimensions as independent or dependent in the matching process, covering MSM and part of LCSS and EDR. Indeed, MSM and MUITAS allow a different distance function to be used for measuring the similarity of each dimension, apart from defining weights that give more or less importance to each dimension.

To better understand the similarity problem addressed in this work, let us consider the simple example of trajectories A and B in Figure 2. Trajectory A has three points (a₁, a₂, a₃) and trajectory B has five points (b₁, b₂, b₃, b₄, b₅). Both trajectories visit the same places (same semantics) but in a different order. A and B visit hotel, bank, and
mall, but not necessarily in this order. As MUITAS and MSM consider any type of trajectory dimension, they have hitherto been the most robust methods of measuring the trajectory similarity, independently of the dimensions present in the dataset.

In the example of Figure 2, MSM and MUITAS need to compare each point of trajectory A to all points of the trajectory B, and for all dimensions, to discover that semantically trajectory A is totally contained in B, that is, they share the semantic similarity of three points. In other words, MSM compares each dimension of point $a_1$ to all points of B, in order to discover that A and B visit hotel, bank and mall and that they move at similar times. Because of the point-to-point comparison, MSM and MUITAS have a high complexity, and require a smart indexing data structure that supports all three dimensions for fast similarity search in real datasets.

In 2018, Furtado proposed fast trajectory similarity measuring (FTSM), an index for fast similarity measurement of UMS and MSM (Furtado et al., 2018). However, it indexes only the spatial dimension, which is not sufficient to support large trajectory datasets that have many semantic aspects. A survey of general indexing data structures is presented in Mahmood, Punni, and Aref (2018) and shows that only a limited number of works propose indexes considering all three dimensions (space, time, and semantics). To the best of our knowledge, none of these indexes was developed for trajectory similarity purposes. In general, they focus on indexing only space, or space and time for range and top-K queries. Another common limitation of the index data structures is the considerable storage cost due to the redundant data structures when indexing the three dimensions.

In this work we aim to answer the following question: can we build an efficient index for point-to-point multiple aspect trajectory similarity measuring that takes into account all dimensions of space, time and semantics? In this article we propose an index data structure for historical data called the MAT-Index that significantly reduces the processing time for measuring the multidimensional similarity of trajectories with MSM and MUITAS. The MAT-Index construction avoids the need for point-to-point comparison of MSM and MUITAS, and its main advantage and difference from the state-of-the-art is that, apart from being able to consider all different dimensions in a single data structure, the final index contains the matching scores. Our proposal is for an index support for similarity measures of multiple aspect trajectories, and how the similarity algorithm measures the similarity among different aspects depends on the definition of the measure itself. A qualitative evaluation shows how the MAT-Index can drastically reduce the number of comparisons, and a quantitative evaluation shows a gain for all tested scenarios of up to 98.1% processing time reduction and up to 87.2% in scalability evaluations.

The remainder of the article is structured as follows. Section 2 describes the concepts required to understand this work. Section 3 discusses related works, highlighting their limitations. Section 4 introduces the proposed index for multiple aspect trajectories. Section 5 discusses the evaluation results that assess work efficiency. Finally, Section 6 concludes the article and suggests directions for future work.
2 | BASIC CONCEPTS

This section presents the main concepts needed to understand this work. Section 2.1 defines multiple aspect trajectories, Section 2.2 presents the basics about the similarity measures developed for this type of data, focusing on MSM (Furtado et al., 2016) and MUITAS (Petry et al., 2019), and Section 2.3 presents some basic definitions.

2.1 | Multiple aspect trajectories

A multiple aspect trajectory is a sequence of points $T = (p_1, p_2, ..., p_n)$, such that $p_i = (x, y, t, A)$ is its $i$th point composed of a location $(x, y)$, also called the spatial dimension, a time-stamp $t$, also called the temporal dimension, and a non-empty set of aspects $A = \{a_1, a_2, ..., a_m\}$, representing the semantic dimension.

An aspect $a = (\text{desc}, \text{ATV})$ is a relevant real-world fact for mobility data analysis. It is composed of a description $(\text{desc})$ and an instance of its corresponding aspect type $(a_{\text{type}})$, that is represented as a set of attribute-value pairs $\text{ATV} = \{\text{att}_1:v_1, \text{att}_2:v_2, ..., \text{att}_z:v_z\}$.

An aspect type $a_{\text{type}} = \{\text{att}_1, \text{att}_2, ..., \text{att}_z\}$ is a categorization of a real-world fact composed of a set of attributes $(\text{att})$. In other words, an aspect type and its attributes act as a metadata definition for an aspect.

As an aid to understanding, consider the following example adapted from Mello et al. (2019) where an aspect type hotel is defined by the following attributes: geographic coordinates, address, and stars. A possible aspect related to this type could be Il Campanario Resort with the following attribute values: geographic coordinates, $-27.439771, -48.500802$; address, Buzios Ave., Florianopolis; stars, 5.

2.2 | Similarity measures

Similarity measures express on a numerical scale how similar two points are. Several similarity measures have been developed for sequential data, such as LCSS (Vlachos et al., 2002), EDR (Chen et al., 2005), and $w$-constrained Discrete Fréchet distance (Ding, Trajcevski, & Scheuermann, 2008), or for trajectories, such as common visit time interval (Kang, Kim, & Li, 2009), maximal semantic trajectory pattern similarity (Ying, Lu, Lee, Weng, & Tseng, 2010), maximum travel match (Xiao, Zheng, Luo, & Xie, 2012), MSM (Furtado et al., 2016), UMS (Furtado et al., 2018), stops and moves similarity measure (SMSM) (Lehmann, Alvares, & Bogorny, 2019), and multiple aspect trajectory similarity measure (MUITAS) (Petry et al., 2019). To the best of our knowledge, only MSM, SMSM, and MUITAS were specifically developed for semantic or multiple aspect trajectories, supporting all their three dimensions: space, time, and semantics. Only MSM and MUITAS deal with independent attributes, and MUITAS is the only one that also considers semantically related attributes. MSM and MUITAS compute the similarity of two trajectories considering a point-to-point analysis, and are currently the most robust for similarity, therefore we will focus on these measures in the indexing proposal.

2.2.1 | Multidimensional similarity measure

MSM (Furtado et al., 2016) computes the similarity between two trajectories $P = (p_1, ..., p_n)$ and $Q = (q_1, ..., q_m)$, comparing each point $p \in P$ to all points $q \in Q$. It computes the parity of two trajectories, $(P, Q)$ and $(Q, P)$, using the maximum matching score given by:

$$parity(P, Q) = \sum_{i=1}^{\|P\|} \max_{q \in Q} \text{score} (p_i, Q) \quad \forall q \in Q.$$  (1)
The score is computed as:

$$\text{score} (p, q) = \sum_{k=1}^{n} \text{match}_k (p, q) \times \omega_k$$

and provides a matching score of two points $p$ and $q$ for each dimension. It involves pairwise comparison of the attributes $A$ of $p$ with $q$, considering the user-defined thresholds $\text{maxDist}_k$, and scores the matches according to their respective weights ($\omega$):

$$\text{match}_k (p, q) = \begin{cases} 1, & \text{if } \text{dist}_k (p, q) \leq \tau_k, \\ 0, & \text{otherwise}. \end{cases}$$

The total similarity of two trajectories is given by summing both parities and dividing the result by the sum of both trajectory lengths ($P$ and $Q$):

$$\text{MSM} (P, Q) = \begin{cases} 0, & \text{if } |P| = 0 \text{ or } |Q| = 0, \\ \text{parity} (P, Q) + \text{parity} (Q, P), & \text{otherwise}. \end{cases}$$

### 2.2.2 Multiple aspect trajectory similarity

MUITAS (Petry et al., 2019) is the first similarity measure natively developed to work with multiple aspect trajectories. MUITAS introduced an essential concept of relationship between attributes, being the first to consider trajectory dimensions as totally independent, partially independent, or dependent. When a set of attributes is defined as dependent or partially independent, this set is called a feature. Such features make MUITAS a flexible measure that supports MSM when attributes are independent and is similar to LCSS when the attributes are defined as dependent, forcing a match of all attributes in the feature. It shares with MSM and SMSM the support of different data types and the ability to assign different distance functions to each attribute.

A feature $f = \{a_1, a_2, \ldots, a_z\}$ is a non-empty set of attributes of a multiple aspect trajectory. It is possible to aggregate attributes to work as independent and dependent by using this concept. For instance, a feature $f_i = \{\text{place category, price tier, rating}\}$ represents information about places of interest (POIs) visited. There are three associated attributes, and this analysis unit is dependent. However, the feature $f_j = \{\text{weather condition}\}$ represents an independent analysis unit.

Suppose $P$ and $Q$ are trajectories and $p$ and $q$ are trajectory points, such that $p \in P$ and $q \in Q$. Then the matching score between $p$ and $q$ is computed as:

$$\text{match}_{fi} (p, q) = \begin{cases} 1, & \text{if } \forall a_j \in f_i, \text{dist}_j (p, q) \leq \tau_j, \\ 0, & \text{otherwise}. \end{cases}$$

For each attribute $A$ of a feature $\mathcal{F}$, the points will match if the distance between them is less than a given threshold ($\tau$).

For each feature, the score is computed as the weighted sum ($\omega$) of matching points:

$$\text{score} (p, q) = \sum_{i=1}^{n} \text{match}_{fi} (p, q) \times \omega_i.$$
After comparing all points, MUITAS calculates the \textit{parity}(P, Q) as the sum of the best scores of each attribute of each point \( p \in P \) compared to \( Q \):

\[
\text{parity} (P, Q) = \sum_{k=1}^{|P|} \max \text{ score} (p_k, Q) \quad \forall q \in Q,
\]

and similarly \( \text{parity}(Q, P) \).

The final similarity score between two trajectories is the sum of parities, divided by the sum of the trajectory lengths:

\[
\text{MUITAS} (P, Q) = \begin{cases} 
0, & \text{if } |P| = 0 \text{ or } |Q| = 0, \\
\frac{\text{parity} (P, Q) + \text{parity} (Q, P)}{|P| + |Q|}, & \text{otherwise}.
\end{cases}
\]

Note that the flexibility of both point-to-point approaches outperformed the other previously mentioned methods. However, it also made them very expensive in terms of processing time.

2.3 | Indexes

There are several indexes developed specifically for spatial data. Examples are the Quadtree (Finkel & Bentley, 1974), Z-Ordering Tree (Morton, 1966), and OCTree (Meagher, 1980). In fact, trees are one of the most commonly adopted data structures for indexing purposes, but the previously mentioned indexes do not support multidimensional spatiotemporal similarity.

For semantic contents, an \textit{inverted index} (also known as an \textit{inverted list}) is frequently used. It saves the data in a smaller size and more organized way, like a dictionary composed of two main parts: the search structure (aka keyword or key), and a list of references (aka value) (Bütcher, Clarke, & Cormack, 2016). Figure 3 illustrates an example of an inverted index containing seven documents that have keyword occurrences previously processed (left). The corresponding inverted index (right) stores the distinct keywords \textit{Hotel}, \textit{Cinema}, \textit{Park}, \textit{Home}, and \textit{Work} as keys containing its references corresponding to the table in the left.

A naive strategy for text retrieval compares a list of queried words with all keyword documents. The inverted list provides single access to obtain all references that contain the same keyword, which limits the universe to be compared in a single access.

| DOCUMENTS | INVERTED INDEX |
|------------|----------------|
| **Reference** | **Key** | **List of References** |
| 1 Home, Park | Hotel | 2,4 |
| 2 Cinema, Hotel | Cinema | 2,3,4,5 |
| 3 Cinema, Home, Work | Park | 1,4,6,7 |
| 4 Cinema, Hotel, Park | Home | 1,3 |
| 5 Cinema, Work | Work | 3,5,7 |
| 6 Park | |
| 7 Park, Work | |
Despite the several solutions for different data, indexing space, time, and semantic data together is far more complex, especially for point-to-point comparison purposes. It is necessary to keep fast access to the entire dataset content for a problem that requires quadratic computation and does not allow pruning strategies.

3 | RELATED WORKS

In the state of the art there is an extensive list of access methods for trajectories organized in Mahmood et al. (2018) according to the temporal context of the data (past, present, and future; we focus on the past), with different indexing arrangements. We observe that only a few of these works index textual with space and/or temporal data. Regarding these works, some papers do not deal with all three dimensions together; for example, Chen et al. (2019), Issa & Damiani (2016), Zheng et al. (2015), and Zheng, Shang, Yuan, & Yang (2013) only focus on spatial and semantic dimensions, disregarding the temporal dimension, and Issa & Damiani (2016) is also limited to a single semantic keyword. Also among these works, some approaches are limited to ranges; for example, Han, Wang, Zhang, Zhang, & Xuemin Lin (2015), Issa & Damiani (2016), Mehta, Skoutas, & Voisard (2015), and Skovsgaard, Sidlauskas, & Jensen (2014) focus on indexing semantic keywords according to both spatial and temporal range/granularity, and Magdy et al. (2016), Magdy, Mokbel, Elnikety, Nath, & He (2014), and Wang et al. (2017) index spatial and keywords limited only to a time interval/range.

Concerning applicability, most of these mentioned works focus on accelerating range queries (Han et al., 2015; Issa & Damiani, 2016; Magdy et al., 2014, 2016; Mehta et al., 2015; Skovsgaard et al., 2014; Wang et al., 2017), while Magdy et al. (2014, 2016), Skovsgaard et al. (2014), Wang et al. (2017), and Zheng et al. (2013, 2015) aim to efficiently answer top-k searches. The approaches for solving range queries and top-k problems lead to intrinsic pruning strategies that, in general, restrict the indexed access to a small portion of the entire dataset, facilitating processing. Works limited to exact matches (Han et al., 2015; Skovsgaard et al., 2014) have the same problem since each queried term can restrict the content where the next term must occur, otherwise, it does not attend the search. However, before drawing conclusions, it is crucial to comprehend the whole picture by comparing trajectories, not just isolated points. In this way, we can more assertively extract information based on similar behaviors, not in circumstantial cases. Thus, these approaches are incompatible with our aim of comparing all datasets, since it is not feasible to prune neither of multiple aspect trajectories dimensions and still to establish a precise comparison.

All indexes mentioned in this section have hybrid data structures that are connected. It demands a high storage cost to link them and keeping redundant data to optimize the access. The high demand for memory allocation could force the operating system to excessively transfer data between memory levels, delaying access; this is a known problem referred to as thrashing. Thus, some solutions design hybrid (i.e., part memory, part disk; Han et al., 2015; Liu et al., 2017; Zheng et al., 2013, 2015) or disk (Issa & Damiani, 2016) allocation strategies to avoid system collapse. However, the disk and hybrid allocation solutions require transferring data, thus multiple access. The secondary memory is a very slow resource, and is therefore inefficient for processing large and complex trajectory datasets. Still, some works deal with the performance by adopting approximate solutions (Skovsgaard et al., 2014; Zheng et al., 2015), which for similarity search purposes, would propagate a possible error to all other related comparisons, affecting the reliability of the score.

To the best of our knowledge, there are no works in the literature that fully index the three multiple aspect trajectory dimensions for similarity measurement. Indeed, existing indexing works provide neither a data structure to avoid the point-to-point matching nor the number of matches between points for each dimension, including the partial matches. Therefore, a novel data structure is needed to accurately process an entire trajectory dataset and return the top matching scores, preventing redundant comparisons.
MULTIPLE ASPECT TRAJECTORY INDEX

In this section we propose the MAT-Index, a novel access method designed to speed up the similarity analysis of multiple aspect trajectories. It focuses on indexing all three dimensions of space, time, and semantics in a single data structure, in order to facilitate the comparison between trajectories, eliminating redundant operations.

Calculation of the MAT-Index is divided into six steps, as presented in Figure 4. The load step stores each dimension in a separated data structure that is processed as follows: the spatial indexing and the temporal indexing treat the spatial and temporal data, respectively; the semantic combine and the semantic compress steps process the semantic content. The Dimensions Integration step integrates the three resulting data structures built in the previous steps. The following sections detail each step of the MAT-Index.

4.1 Load

The load stage saves the spatial, temporal, and semantic dimensions into separate data structures to be merged in the last index step. Figure 5 shows the data structures saved in the load step. Figure 5a shows an example of the dataset, where each row contains the information associated to a trajectory point. Each row contains the trajectory identifier (tid), followed by the spatial coordinates (x, y), the time (time), and the semantic attributes price, poi, and weather. The tid is repeated according to the number of points the trajectory has. In the example, 11 points belong to three trajectories: trajectory 126 has 4 points, trajectory 127 has 4 points, and trajectory 128 has 3 points.

Figure 5 shows the spatial (b), temporal (c), and semantic (d and e) intermediate data structures as loaded. As an aid to understanding, in this example we consider that each feature (as defined in Section 2.2.2) contains only one attribute, thus \( \mathcal{F} = \{\text{price}\}, \{\text{poi}\}, \{\text{weather}\} \). This scenario is applicable to both MSM and MUITAS similarity measures, as discussed in Section 2.2.

Algorithm 1 presents the load pseudo-code. It consists of reading the dataset and saving each trajectory dimension in each proper data structure. The first row (rId = 0), the header of the dataset, is processed by the addAttributeToTwoLevelIndex function (line 1). It saves the names of the semantic attributes contained in the header, as they come right after the spatial coordinates and the temporal dimension. The aim is to group the trajectory points with the same semantic attribute values and considering their contexts. In the example, the attribute names price, poi, and weather are the first-level keys of the TwoLevelIndex data structure, as presented in Figure 5e in the delimited dashed area on the left, called the first-level key.

![MAT-Index flow](image)
From \( rld = 1 \) to \( rld = 11 \), that is, for all trajectory points, the process is repeated, sequentially reading the dataset. For each row, it stores the trajectory dimensions as follows: the command `addToSpatialIndex` saves the `SpatialIndex` in the format shown in Figure 5b; the command `addToTemporalIndex` saves the `TemporalIndex` in the format shown in Figure 5c; and functions `addToSemDictionary` and `addValueToTwoLevelIndex` save the semantic data as—preliminarily indexed.

![Figure 5](image-url)
structures SemDictionary in Figure 5d and TwoLevelIndex in Figure 5e. These commands are explained in the sequence.

The addToSpatialIndex method (line 3) saves the spatial data in an inverted index (Figure 5b) simulating each entry as a coordinate of the corresponding non-empty cell in the grid data structure depicted in Figure 6 (left). The MAT-Index spatial logical grid does not demand allocating a matrix that gives rise to the sparsity issue (note that only 9 of the 198 cells in the corresponding grid are non-empty), and the difficulty of balancing the size of the interval and the memory required to process it. The cell size is calculated based on the threshold defined by the similarity measure, as MUITAS and MSM define a threshold $\tau$ that specifies the maximum spatial distance between two points to consider them as similar. Therefore, MAT-Index builds the SpatialIndex as squared cells such that the maximum distance between two points in the same cell never exceeds this threshold. Since the maximum distance in a square is its diagonal, we assume this diagonal to be $\tau$.

Thus, all the points in the same cell of Figure 6 (left)—as the pairs of trajectory points [1,2] and [8,11]—automatically match, since they are below the threshold, thus do not require spatial comparison in these cases. The SpatialIndex is then created as an inverted list that allocates as keys the position/address of the cell and as the values the list of rIds that represent the points inside the cell.

Regarding the temporal content, MAT-Index allocates only actual occurrences in an inverted list, similar to the spatial indexing process. However, it is worth observing that a daytime unit can be expressed as 24 h, 1440 min, or even 86,400 s, even if this last option is less likely to be used. It is a small number of possibilities if compared with the considerable amount of data to be processed. Therefore, the time is a unidimensional value that can be segmented based on the unit of the temporal threshold ($\tau$) used by the similarity measures and aggregated later into temporal intervals according to the threshold value. The command addToTemporalIndex (line 4) saves the temporal dimension in the TemporalIndex inverted list, as shown in Figure 5c. Each $\langle$key, value$\rangle$ entry in the TemporalIndex holds a time and a list of rIds with the same temporal information. The running example of Figure 5a has a temporal
threshold of 5 min, which means that the key time of the TemporalIndex must be saved in minutes. For instance, the rld = 4 in Figure 5a has a time in hours (2:35), which is then saved in minutes (2 \times 60 + 35 = 155 min), as shown in the first entry in Figure 5c.

Concerning the semantic dimension, MAT-Index preliminary saves the semantics into two structures. The command addToSemDictionary (row 5) saves a semantic dictionary with the distinct semantic composite keys in Figure 5d. The command addValueToTwoLevelIndex (line 6) stores in the TwoLevelIndex the second-level key (dashed area) in Figure 5e, associated to the trajectory point references with the same attribute name and value in the corresponding list of rIds, as shown on the right of Figure 5e. It is worth mentioning that the two-level model preserves the context of the values, since each attribute name will contain its particular distinct values as appear in the data-set of Figure 5. For instance, the value 1 could represent a price, a rating, or an age. As previously mentioned, line 1 of Algorithm 1 saves the first level of the two-level inverted index, while line 6 populates its second level. The combine step (Section 4.4) merges both data structures, and the algorithm finishes by returning all intermediate files.

4.2 | Spatial computation

For the spatial dimension, MAT-Index uses a logical grid, storing only the cell addresses that contain at least one trajectory point. The spatial dimension demands a pairwise comparison of two trajectory points to check if the distance between them does not exceed the similarity threshold (\( \tau \)). Therefore, MAT-Index uses the auxiliary SpatialIndex data structure generated in the load step to avoid the comparison among all trajectory points. The SpatialIndex presented in Figure 5b is an inverted list where each entry is a pair \( \langle \text{key}, \text{value} \rangle \), each key being a cell address and the value a list of rIds (trajectory points) belonging to the same key (cell address).

Algorithm 2 explains how the spatial computation step works. First, it sequentially reads each entry of the SpatialIndex (line 1) composed of the pair \( \langle \text{cellAddress}, \text{rIds} \rangle \). For each rld (trajectory point) in the list of rIds (line 2), the OR operator in line 3 updates the list of spatial matches with the rIds in the same cell address. In line 4, the command getCandidateCells returns a list of point candidates (pCandidate), testing if the distance between the points rId and pCandidate does not exceed the spatialThreshold. If not, the lists of spatialMatches of both points are updated (lines 6 and 7). After the cell address processing, the entry is removed from the SpatialIndex (line 11) to prevent the points from being double-checked, which is unnecessary due to the symmetry of the spatial distance.

```
Algorithm 2: computeSpatialMatches

input : SpatialIndex // Figure 5b
output: spatialMatches updated
1 foreach () cellAddress, rIds () \in SpatialIndex do // Entry ()Key, Value()
2     foreach rId in rIds do
3         Points.get(rId).spatialMatches = Points.get(rId).spatialMatches OR
4             SpatialIndex.get(cellAddress)
5             foreach (pCandidate \in getCandidateCells(cellAddress)) do
6                 if (distance(Points.get(rId), Points.get(pCandidate) \leq spatialThreshold) ) then
7                     Points.get(rId).spatialMatches.set(pCandidate)
8                     Points.get(pCandidate).spatialMatches.set(rId)
9                 end
10         end
11     SpatialIndex.remove(cellAddress)
12 end
13 return spatialMatches
```
Going back to the running example, for the sake of understanding, suppose the spatial threshold is 1.42, resulting in a cell size equal to 1. We use the Euclidean distance to compute the spatial distance, but any other distance measure could be used. In Figure 5b, the rlds [8,11] automatically match as well as [1,2] because they are in the same cell. The pairs of rlds [4,10] and [7,9] are in adjacent cells. Thus, it is necessary to check in both cases if the Euclidean distance does not exceed the threshold, as presented below:

\[ d(4,10) = \sqrt{(4.3-4.3)^2 + (17.9-16.9)^2} = 1 \leq 1.42? \text{ TRUE} \]

\[ d(7,9) = \sqrt{(4.3-5.2)^2 + (1.9-0)^2} = 4.42 \leq 1.42? \text{ FALSE} \]

Therefore, concerning the spatial indexing, only the pairs [1,2], [8,11], and [4,10] match. These pairs of matches will be used to update the final MAT-Index score, in the final step, the index integration step (Section 4.6).

### 4.3 Temporal computation

For the time indexing, the strategy is to create, for each temporal index entry, a list with all the matching rlds, which are the \( t < r \) trajectory points belonging to the cells in the interval admitted by the temporal threshold \( \tau \).

Algorithm 3 shows the pseudo-code that receives as input the TemporalIndex generated in the load step and provides updated TemporalMatches as output. It starts by sequentially reading the entries in the TemporalIndex (line 1), and, for each entry composed of \( \langle \text{time}, \text{rIds} \rangle \), it aggregates to the listOfMatches all the rIds belonging to the groups in the interval \( \pm \tau \) (line 2). For every rId in the entry (i.e., rIds, line 3), the listOfMatches is associated to the temporalMatches of the processed rId (line 4).

```
Algorithm 3: computeTemporalMatches

input : TemporalIndex                      // Figure 7a
output: temporalMatches updated
1 foreach \( \langle \text{time}, \text{rIds} \rangle \in \text{TemporalIndex} \) do       // Entry \( \langle \text{key}, \text{value} \rangle \) to be analyzed
   // Union of all rIds associated to entries in the interval admitted
2    listOfMatches = \( \bigcup_{i=\text{time}+\tau}^{\text{time}+\tau} \text{TemporalIndex.get}(i).rIds \)    // \( \tau \): temporal threshold
3    foreach rId \( \in \text{rIds} \) do // Row ids from the entry analyzed
4       \text{Points.get(rId).temporalMatches} = \text{listOfMatches}
5 end
6 return temporalMatches                   // For all trajectory points they are updated
```

Figure 7a shows the temporal data as stored in the load phase, and Figure 7b the corresponding list of matches. All rlds in Figure 7a share the same list of matches. In the running example, let us consider the absolute time difference as distance function to measure the similarity of two time-stamps and a distance threshold of 5 min. Figure 7b shows the corresponding matching rlds where, for instance, the cells 1313 and 1315 are within the threshold, so the list of rlds of both cells (i.e., rld 3 and rld 11) are added to both corresponding matching rlds entries in Figure 7b. Additionally, 1315 and 1319 are within the threshold of 5 min, thus the process is repeated and rlds 3 and 6 are added to 1315 and 1319 matching index cells in Figure 7b. The result of a non-transitive property here is clear, since we observe that cell
1315 shares 1313 and 1319 as a match \((rIds\ 3,\ 6,\ and\ 11)\), with both \((1315–1313\ and\ 1315–1319)\) less than or equal to the threshold of 5 min, while 1313 and 1319 do not match at all.

The temporal approach brings two benefits: first, the allocation by unit allows us to get the matches by aggregating the rIds belonging to the groups in the same interval. This way, we prevent comparing the temporal content among all trajectories; and second, the number of index entries is limited to the threshold unit (i.e., if expressed in minutes, 1440 possibilities), thus it tends to require fewer iterations to process the temporal dimension. It is worth noticing that real datasets are bigger than our running example. Thus, the list of rIds tends to contain, on average, more points, since we have a very limited number of possible cells allocated (1440 in this case). It saves memory and especially processing time because the idea is to process the matches in groups (by cell), not by rId.

### 4.4 | Semantic combine

The combine step treats the semantic dimension. It creates the composite index by taking the composite keys in the semantic dictionary from Figure 5d and associating each one to a match counter, created using the occurrences in the two-level inverted index from Figure 5e. The match counter represents the number of matches by trajectory point if compared to the composite key itself. It avoids the pairwise comparison of the attribute values of each pair of points by exploring the transitive property (e.g., for all \(P, Q, R\), if \(P \sim Q\) and \(Q \sim R\), then \(P \sim R\)) of this kind of data.

Algorithm 4 illustrates the combine pseudo-code, where each semantic composite key in the dictionary (line 1) provides one valid combination of attribute values. The individual attribute values of each combination are used to build its corresponding match counter (line 2, in the getMatchCounter function). The command SemCompositeIndex.put (line 2) saves the semCompositeKey and its corresponding MatchCounter, returned by the getMatchCounter function. After, the SemCompositeIndex is completed, then returned in line 4.

The getMatchCounter function shows how the match counter is obtained. The semCompositeKey provides the valid combination of attribute values to be processed. For each attribute value in the combination, the function retrieves the list of rIds of the corresponding attribute values, as in the example shown in Figure 5e. The feature \(f\) in \(F\) matches (line 6) if all its corresponding attribute values match at the same rId position. Thus, the function executes an AND operator to get a unique list of rIds where all attributes match for \(f\) (line 7). Then, only the rIds in

\[
\begin{array}{|c|c|}
\hline
Cell & List of rIds \\
\hline
155 & [4] \\
230 & [9, 10] \\
317 & [1] \\
751 & [7] \\
838 & [8] \\
1179 & [5] \\
1313 & [11] \\
1315 & [3] \\
1319 & [6] \\
1404 & [2] \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
Matching rIds & \\
\hline
[4] & [9, 10] \\
[1] & [7] \\
[8] & [5] \\
[3, 11] & [3, 6, 11] \\
[3, 6] & [2] \\
\hline
\end{array}
\]

\textbf{FIGURE 7}  Result after the temporal computation: (a) the temporal data structure saved on load; (b) the list of matches associated with each entry.
common are updated in the MatchCounter. After processing the combination for all sets of features, the function returns the MatchCounter in line 11.

Figure 8 presents an example of how to obtain a match counter for the composite key \( \langle $$, Home, Clear \rangle \), with \( \mathcal{F} = \{ \{ \text{price} \}, \{ \text{poi} \}, \{ \text{weather} \} \} \) the features. The result is an array that holds the number of matching features comparing the key content to each trajectory point. For a composite key with \( N \) features, the maximum score obtained is \( N \). Thus, note that rIds 1,4 and 9 are composed of the same set of attribute values, so for three attributes values the maximum score is 3.

Algorithm 4: Semantic Combine

```plaintext
input : SemDictionary
output: SemCompositeIndex // Figure 5d
1 foreach semCompositeKey \in\ SemDictionary do
2     SemCompositeIndex.put(semCompositeKey, getMatchCounter(semCompositeKey))
3 end
4 return SemCompositeIndex
```

```plaintext
input : semCompositeKey, TwoLevelIndex // Figures 5 d,e
output: MatchCounter
5 Function getMatchCounter (semCompositeKey):
6     foreach \( f \in\ \mathcal{F} \) do // Each feature \( f \) is a set of attributes
7         // rIds where all feature attributes values in \( f \) match (Equation 5)
8         foreach \( rId \in\ \cap_{i=1}^{\lfloor f \rfloor} \text{TwoLevelIndex(attributeValue}_{i}\right) rIds \) do
9             \( \text{MatchCounter}[rId] \) += \( \text{MatchCounter}[rId] \)
10 end
11 return MatchCounter
```

Figure 9 shows the entire composite index after the combine step execution, including that previously mentioned in Figure 8. Here we can observe one of the main advantages of our proposal: once the composite index is ready, a single direct access may retrieve the number of semantic features that match between trajectory points, although the points were never pairwise compared.

The match counter of a composite key shows the number of matches for all trajectory points. For instance, every trajectory point that has the semantic combination \( \langle $$, Home, Clear \rangle \) (see the first row of the tables in Figure 9) entirely matches with the rIds \{1,4,9\}. It also matches with \( rId = 3 \) in two of the three semantic attribute values, only once with rIds \{5,6,7,8,10,11\}, but not with \( rId = 2 \). Thus, if we need to know how many semantic attributes \( rId = 1 \) has in common with \( rId = 10 \), we just need to look at the corresponding semantic combination of \( rId = 1 \) in \( \langle $$, Home, Clear \rangle \) at position \( rId = 10 \).
4.5 Semantic compress

The similarity algorithms for multiple aspect trajectories MSM (Furtado et al., 2016) and MUITAS (Petry et al., 2019) retrieve the best semantic match of a point when compared to a trajectory. In this case, we can further compress the index by keeping the maximum score. Therefore, the compress step stores only the top scores by trajectory, saving memory and avoiding redundant comparisons that would degrade the performance.

Algorithm 5 shows the pseudo-code for the compress step. For each `semCompositeKey` in `SemCompositeIndex` (Figure 9) (Algorithm 5, line 1), the combine gets the top number of semantic matches by trajectory using the function `compressMatchCounter` in line 2. The function return is associated to the `semCompositeKey` computed, saving the result as an entry in `CSemCompositeIndex` (line 2, `CSemCompositeIndex.put`).

**Algorithm 5: Semantic compress**

```
input : SemCompositeIndex
output: CSemCompositeIndex   // Figure 10a
1 foreach (semCompositeKey, matchCounter) ∈ SemCompositeIndex do
2       CSemCompositeIndex.put(semCompositeKey, compressMatchCounter(MatchCounter))
3 end
4 return CSemCompositeIndex
```  

The function `compressMatchCounter` in line 5 presents how to compress a match counter with the trajectory top scores. For each trajectory point `p` (line 6), the compressed match counter `CMatchCounter` at `p.cId` position (i.e., the corresponding trajectory id position of `p`) is updated if the score of the trajectory point `p` is greater than the current score of its trajectory `p.cId` (line 7).
Figure 10 presents the scores of the running example before and after the compression, by trajectory point (Figure 10a) and by trajectory (Figure 10b). We observe that the 11 trajectory points turned into only three points, corresponding to the number of trajectories, since we only keep the maximum score for each trajectory and not all the points. The first column of the MatchCounter in Figure 10a corresponds to the header position in the dataset (see Figure 5a at \( rId = 0 \)), that is, where the names of attributes are placed. The header does not contain trajectory content, being maintained at first to avoid testing the position multiple times during the load and the combine processing. Therefore, the \( rId = 0 \) is discarded in this step.

4.6 Dimensions integration

Indexing space, time, and several semantic dimensions in a single data structure avoids redundant comparisons, speeding up the trajectory similarity analysis. Therefore, the index integration phase consolidates the matching scores into a single data structure, finalizing the MAT-Index construction.

The pseudo-code for this step is presented in Algorithm 6. For each point \( p \) (line 1), the semantic composite key of \( p \) is used to retrieve its corresponding compressed match counter \( auxCMatchCounter \) and store it in an auxiliary variable (line 2). The compressed match counter holds the top scores by trajectory for a valid combination of semantic attribute values. In line 3, the function gets all the points that match both in space and time with \( p \), using an AND operator to obtain the list of \( rIds \). Thus, using the auxiliary compressed match counter (\( auxCMatchCounter \)) of \( p \) for each \( rId \) in \( bothMatch \), the \( auxCMatchCounter \) is updated at the compressed id position. The compressed id (\( cId \)) is the corresponding trajectory id to which the \( rId \) belongs. The \( CMatchCounter \) is then updated with the maximum value between its current value at \( cId \) position, as shown in Figure 11a, and the value of the MatchCounter at \( rId \) position is increased by 2, as shown in Figure 11b. By checking the maximum value, the method avoids repeating the compression step for each trajectory point. After updating the cases that match in both space and time, the algorithm treats the cases that exclusively match either for space or time. The list is obtained by using an exclusive OR operator (XOR) (line 7). Thus, for each \( rId \) in \( oneMatch \) (line 8) that exclusively matches with point \( p \) either in space or time, the pseudo-code saves the maximum score between the corresponding compressed match counter \( CMatchCounter \) of \( p \) at the \( cId \) trajectory position for the \( rId \) and the MatchCounter at \( rId \) position is increased by 1. The advantage of separating the process of updating in \( oneMatch \) and \( bothMatch \) is to avoid redundant processing for the cases where space and time match simultaneously. In line 11, the point \( p \) with its corresponding scores in
auxCMatchCounter is added to the MAT-Index. After all points $p$ are processed, the compressed composite index in Figure 11a and the composite index in Figure 11b data structures are no longer required, so they are discarded (lines 13 and 14). The MAT-Index (illustrated in Figure 11c) is ready to use.

**Algorithm 6: Dimensions Integration**

```plaintext
input : SemCompositeIndex, CSemCompositeIndex  // Figure 10
output: MATIndex  // Figure 11c

1 foreach $p$ in Points do
2     auxCMatchCounter = CSemCompositeIndex.get($p$.semCompositeKey)
3     bothMatch = $p$.spatialMatches AND $p$.temporalMatches
4     foreach $rId$ in bothMatch do
5         auxCMatchCounter[Points.get($rId$).cId] =
6             max(auxCMatchCounter[Points.get($rId$).cId],
7                 SemCompositeIndex.get($p$.semCompositeKey)[rId] + 2)
8     end
9     oneMatch = $p$.spatialMatches XOR $p$.temporalMatches
10    foreach $rId$ in oneMatch do
11        auxCMatchCounter[Points.get($rId$).cId] =
12            max(auxCMatchCounter[Points.get($rId$).cId],
13                 SemCompositeIndex.get($p$.semCompositeKey)[rId] + 1)
14    end
15    MATIndex.put($p$.rId, auxCMatchCounter)
16 end
17 CSemCompositeIndex.clear();
18 SemCompositeIndex.clear();
19 return MATIndex
```

Returning to our example in Figure 5, each trajectory point now has a final score for all dimensions. The points themselves are updated by 2, since now both spatial and temporal dimensions are taken into account, resulting in the value 5 (Figure 11c). By updating the spatial and temporal matches, the pairs of trajectory points $\{1, 2\}$, $\{8, 11\}$, and $\{4, 10\}$ match spatially. The pairs of trajectory points $\{9, 10\}$, $\{3, 11\}$, and $\{3, 6\}$ match temporally. All matches are updated in the final MAT-Index in Figure 11c.

Note that the final score in Figure 11c is the maximum of the result incremented in Figure 11b and the auxiliary top score preserved in Figure 11a, as explained in Algorithm 6. Take the example of updating the match between $rIds$ 3 and 11, starting with $rId = 11 (University, Clouds)$. The trajectory point $rId = 3$ corresponds to the trajectory $cId = 0$. In Figure 11b, the composite key $(University, Clouds)$ at position $rId = 3$ has score of 3. This is higher than the old top score 2 in Figure 11a. So, the method updates the score of trajectory point $rId = 11$ at $cId = 0$ to 3 in the MAT-Index (Figure 11c). The method repeats this process for all matches. The updates are highlighted in Figure 11c.

The final MAT-Index data structure, depicted in Figure 11c, considers the spatial, temporal, and semantic dimensions for each trajectory point. Therefore, it is sufficient to access the trajectory point using the row id to directly access the score at the corresponding trajectory position (the compressed row id, $cId$).
4.7 Complexity analysis

The MAT-Index algorithm sequentially executes its six steps. All data are stored in hash maps using $O(1)$ get/put operations.

The first step (load, Algorithm 1) performs a linear scan over the trajectory datasets and populates the data structures. It has complexity $O(N)$, where $N$ is the total number of trajectory points.

The second step (spatial computation, Algorithm 2) retrieves each $rId$ distributed among the cell addresses. The $rIds$ belonging to the same cell automatically match; thus, only the $rIds$ placed in adjacent cells require further computation. Here again we have $O(N)$ complexity.

The third step (temporal computation, Algorithm 3) reads all the hash map entries sequentially. Let $k$ be the number of distinct groups and let us assume we have $N/k$ points in each group. Temporal computation performs $2 \tau N/k$ operations for each group, resulting in $2 \tau N$ operations plus $N$ operations to assign the result to each trajectory point. We thus have $O(2 \tau N)$ complexity.

The fourth step (semantic combine, Algorithm 4) processes each semantic composite key in the semantic dictionary. By taking advantage of the BitSet data structure used in the TwoLevelIndex, we simply obtain the $rIds$ that must be updated. We thus sum the occurrences in the match counter data structure. After computing all match counters, we have added each attribute in $A$ precisely $N$ times (i.e., $A \times N$ times). Considering that $N$ tends to be much bigger than $A$ in order of magnitude, the process has linear complexity $O(AN)$.

The fifth step (semantic compress, Algorithm 5) gets the maximum scores of each trajectory by composite key. The best case is when all the composite keys are equal, resulting in a linear cost of $O(N)$. In the worst case, all the composite keys are distinct, resulting in complexity $O(N^2)$. Both extremes are unlikely. For instance, considering the assessed datasets, Foursquare has 7107 keys. BerlinMOD has only 67 distinct semantic combinations, a very tiny number compared to the number of trajectory points (227,403 and 346,657, respectively). That is the reason why the behavior of MAT-Index tended toward linear, as discussed in Section 5.

The final step (dimensions integration, Algorithm 6) updates only the cases that match. Since it maintains BitSets for spatial and temporal matches, the logical operations prevent double retrieval of the same content, which is done in linear time.
After analyzing all MAT-Index processes, we can conclude that the MAT-Index bottleneck lies in the compression step, the complexity of which can vary from linear to quadratic depending on the number of semantic composite keys. Again, both situations are unlikely, as demonstrated in all the state-of-the-art assessed datasets.

Considering that MAT-Index processes the dimensions independently before integrating the data, in cases where the matching criteria are changed, only the affected dimension must be reprocessed and then reintegrated. Considering the building costs for each dimension, the update requires minimal time compared to the original similarity measures.

5 | EVALUATING MAT-INDEX FOR TRAJECTORY SIMILARITY MEASUREMENT

In this section we evaluate the performance of MSM and MUITAS using MAT-Index. We also compare MAT-Index with FTSM (Furtado et al., 2018), a spatial index proposed to accelerate the comparison of the spatial dimension of MSM. MAT-Index is general and can potentially be applied to other point-to-point similarity measures for multiple aspect trajectories. We implemented both MSM and MUITAS with and without using the proposed index. The source code for MAT-Index and the datasets used in the experiments are available in a public GitHub repository, MasterDegree, at https://github.com/anapbr/branch PaperVersion.

Before we detail the experimental evaluation, it is important to remember that MSM computes the similarity by pairwise comparing a set of attributes, while MUITAS compares features, which are a non-empty set $\mathcal{F}$ of attributes. MUITAS can also compute MSM similarity by considering the particular case where each feature is composed of only one attribute. For instance, suppose we have a dataset composed of three attributes $\text{att}_1$, $\text{att}_2$, and $\text{att}_3$. MSM can only pairwise compare all three attributes separated, while MUITAS can process different feature configurations such as $\mathcal{F} = \{ att_1 \}, \{ att_2 \}, \{ att_3 \}$ (where each feature has only one attribute like in MSM), $\mathcal{F} = \{ att_1, att_2 \}$, $\{ att_2, att_3 \}$, or even all attributes together in the same feature, $\mathcal{F} = \{ att_1, att_2, att_3 \}$.

We evaluate MAT-Index considering both qualitative and quantitative aspects. Section 5.1 concerns the qualitative perspective where we show via an example how using our index simplifies the similarity computation, thus drastically reducing the number of comparisons needed to compute the similarity for both evaluated algorithms. Section 5.2 presents a quantitative evaluation of MAT-Index, focusing on time and scalability metrics.

5.1 | Qualitative evaluation

We present a scenario in which both MUITAS and MSM employ MAT-Index to obtain the similarity score of two trajectories. By exploiting MAT-Index, the similarity for both MSM and MUITAS no longer requires comparing each pair of attributes/features to compute the point-to-point score as the original methods do. It means that both similarity measures can start computing the parity (MSM Equation 1, MUITAS Equation 7) when using MAT-Index, thus skipping Equations (2) and (3) (MSM), and Equations (5) and (6) (MUITAS). Accessing the MAT-Index data structure (Figure 11c), we compute the final score in linear time of $|P| + |Q|$ accesses, corresponding to the number of retrievals to get the top score of each point of $P$ and $Q$. The retrieved scores are added. The original computation requires quadratic cost, since it compares all points of $P$ with all points of $Q$ and vice-versa, for each dimension. In other words, $2 \times |P| \times |Q| \times |A|$ comparisons, being $A$ the number of attributes compared (space, time, and semantic attributes).

Returning to the running example in Figure 5 and considering the features $\mathcal{F} = \{ \text{price}, \text{poi}, \text{weather} \}$, suppose we want to measure the similarity between trajectory ids 126 and 128. For reference, the first column (tid) of the dataset in Figure 5a identifies the trajectory ids and the trajectory points. The first step in finding the similarity score $\text{Sim}(126, 128)$ is to compute the parities by querying the top scores of the points of both trajectories in MAT-Index (Figure 11c). Starting from the computation of $\text{parity}(126, 128)$, we recall that the trajectory 126 ($cld = 0$)
is composed of four points with \( rld \)s \( \{1, 2, 3, 4\} \). It is necessary to retrieve the value in \( key = rld \) at position \( cld = 2 \) (128) to get their top scores. Once MAT-Index indexes five features (space, time, and the semantics \( price, poi, \) and \( weather \)), the sum of scores must be divided by 5. Therefore,

\[
parity(126, 128) = \frac{\sum_{rld=1}^{4} \max (rld, cld)}{|P|} = \frac{3 + 1 + 3 + 3}{5} = 2.
\]

The process must be repeated by inverting the references for all points of 128 to get \( \text{parity}(128, 126) \), thus summing the retrieved scores at position \( cld = 0 \):

\[
parity(128, 126) = \frac{\sum_{rld=9}^{11} \max (rld, cld)}{|P|} = \frac{3 + 2 + 3}{5} = 1.6.
\]

After calculating both parities, the final similarity score, generically indicated by \( Sim(126, 128) \), is the result for both \( MSM(126, 128) \) and \( MUITAS(126, 128) \) computations are the sum of parities divided by the sum of each trajectory length (number of points), such that:

\[
Sim(126, 128) = \frac{parity(126, 128) + parity(128, 126)}{|126| + |128|} = \frac{2 + 1.6}{4 + 3} \approx 0.51.
\]

This process can be repeated for any pair of trajectories, using their \( rld \)s as \( key \) to find the top scores of all trajectory points in the dataset.

5.2 Quantitative evaluation

The quantitative evaluation is organized into two parts. In the first part we employ two publicly available datasets composed of spatial, temporal, and multiple semantic dimensions to compare the running times of the similarity measures \( MSM \) and \( MUITAS \) with and without MAT-Index support. In the second part we employ a synthetic dataset to evaluate the index scalability to state the impact of the trajectory size on processing times.

All quantitative experiments were performed on an Intel® Core™ i7-9750H Coffee Lake CPU @ 2.60 GHz (12 MB cache), 32 GB Crucial Dual-Channel @ 1.330 GHz (19-19-19-43), 500 GB Samsung SSD 970 EVO Plus, and 4 GB NVIDIA GeForce GTX 1650 using Windows 10 Education 64-bit, with a command prompt boot option without graphical and network support to avoid overlapping second plan processes.

5.2.1 Evaluating MAT-Index processing time for similarity measuring

We split this experiment into two parts, where we are interested in how much faster both similarity measures process the entire dataset when employing MAT-Index and FTSM access methods, and how the number of attributes and the distinct semantic combinations affect the performance. We used two publicly available datasets that contain spatial, temporal, and multiple semantic dimensions. One is the Foursquare NYC (Yang, Zhang, Zheng, & Yu, 2015) dataset containing real data, while the other is a benchmark dataset generated by BerlinMOD (Düntgen, Behr, & Güting, 2009). These datasets have different characteristics, including the number of points and average trajectory sizes, making them suitable for evaluating the MAT-Index.

The Foursquare NYC dataset contains semantically enriched check-ins of users collected from April 2008 to October 2010. The Foursquare API (https://developer.foursquare.com/) provided the semantic information
related to the POI: category (\textit{root-type}), subcategory (\textit{type}), and \textit{price}, the latter a numeric classification; the Weather Wunderground API (https://www.wunderground.com/weather/api/) provided the \textit{weather} conditions. The Berlin Moving Object Dataset (BerlinMOD) is a public spatiotemporal dataset containing moving point data, simulating workers commuting between their homes and workplaces on the real street network of the German capital Berlin over a period of two days. It holds two semantic attributes regarding the \textit{type} of user and the \textit{transportation mode}. In both datasets, the time-stamp provided the weekday (\textit{day}).

Table 1 summarizes the main characteristics of the datasets. \textit{Traj Size} refers to the average trajectory length and its standard deviation; the \# of \textit{Users} attribute indicates the number of distinct anonymous users tracked; thus, the attribute \# of \textit{Traj} means that there are one or more trajectories of each user. Finally, \textit{Attributes} refers to the spatial, temporal, and semantic attributes processed in the experiment.

Figure 12 compares the running times of MSM and MUITAS similarity measures execution for each dataset considering the following scenarios: (i) without index support; (ii) with FTSM index support for the spatial dimension; and (iii) with MAT-Index support for all dimensions. To properly evaluate both MSM and MUITAS, fully exploiting their characteristics, the Foursquare dataset was evaluated considering two different sets of features given as follows:

- \textbf{NYC1}. This scenario considers each semantic attribute as individual features, thus, it can be applied to both measures. In other words, we have a set of five features \(\mathcal{F} = \{f_1, \ldots, f_5\}\), where
  \(f_1 = \{\text{day}\}\), \(f_2 = \{\text{price}\}\), \(f_3 = \{\text{weather}\}\), \(f_4 = \{\text{root-type}\}\), and \(f_5 = \{\text{type}\}\).

- \textbf{NYC2}. This scenario considers the \textit{root-type} and the \textit{type} attributes as semantically related, that is, they are

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Dataset} & \textbf{Description} \\
\hline
Foursquare NYC & Traj size: 209, 97 ± 188, 36 \\
(Yang et al., 2015) & \# of traj.: 3079 \\
& \# of points: 227,403 \\
& \# of users: 1083 \\
& Attributes: Latitude, longitude, time, weekday, price, weather, POI type and root-type \\
\hline
BerlinMOD (Düntgen et al., 2009) & Traj size: 417, 24 ± 279, 68 \\
& \# of traj.: 1797 \\
& \# of points: 346,657 \\
& \# of users: 141 \\
& Attributes: Latitude, longitude, start time, weekday, type of user, and transportation mode \\
\hline
\end{tabular}
\caption{Summary of the trajectory datasets used in the experiments}
\end{table}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure12.png}
\caption{Similarity computation elapsed time by dataset}
\end{figure}
processed together in the same feature. In other words, we have a set of four features $\mathcal{F} = \{f_1, \ldots, f_4\}$ covering five semantic attributes, where $f_1 = \{\text{day}\}$, $f_2 = \{\text{price}\}$, $f_3 = \{\text{weather}\}$, $f_4 = \{\text{root-type, type}\}$. Consequently, only the performance of MUITAS is plotted for this case.

We observe from Figure 12 that, in the original implementations, MUITAS performed worse than MSM in all datasets due to the computation overhead for checking the semantic-related attributes. We also observe that, although FTSM efficiently indexes the spatial dimension in average cases, its integration into both MSM and MUITAS similarity measures for space, time and semantics overloaded the processing time in all datasets. However, MAT-Index treats all dimensions in an integrated data structure; thus, the results are much better, reducing execution times by between 93.6 and 98.1%. The semantic relationships among the attributes did not affect the performance, keeping very close results in both similarity measures. In the larger dataset, BerlinMOD, both MSM and MUITAS performed even better than when processing Foursquare, suggesting that the larger the dataset, the better the results with MAT-Index. We observe that since grouping data is the basic idea of MAT-Index, BerlinMOD performance better due to the low variability of values, including time and space, which reduced the computation cost. This indicates that the variability of values influences performance, where the larger a dataset is, the more its data tend to repeat.

It is worth recalling that each set of attributes leads to a different number of distinct semantic composite keys. Therefore, we evaluate how the number of semantic composite keys and attributes impacts the processing times in each dataset. Considering that the semantic related attributes did not affect the processing time in the first evaluation (Figure 12), both datasets were executed in all possible ways, excluding the semantically related attributes: the five semantic attributes from the Foursquare dataset (Table 1) were distributed in scenarios combining one to five attributes, resulting in 31 distinct possibilities. The three semantic attributes from the BerlinMOD dataset were analogously distributed, resulting in seven other processing possibilities, totaling 38 scenarios. For instance, a possible scenario with one attribute for the Foursquare dataset can be only the price $\mathcal{F} = \{\{\text{price}\}\}$, only the weather $\mathcal{F} = \{\{\text{weather}\}\}$, and so on; with two attributes, we can process the price and weather $\mathcal{F} = \{\{\text{price}\}, \{\text{weather}\}\}$ or other pairs of semantic attributes, and so on.

Figure 13 shows, for each dataset, the elapsed time variability in terms of the number of processed attributes. We observe that MAT-Index is stable and faster in all tested scenarios. The elapsed time for the BerlinMOD dataset varies between 63 and 67 s, and for Foursquare NYC between 76 and 105 s. The original implementation gave elapsed times between 1,200 and 3,516 s for BerlinMOD and between 519 and 2,380 s for Foursquare NYC.
increasing with the number of attributes. As expected, the results with MAT-Index tend to improve as the number of trajectory aspects/attributes increases. This characteristic is particularly important for multiple aspect trajectories since they tend to have a large number of attributes.

In order to better understand how the variability of values affects performance, Figure 14 presents the same elapsed times of the previous Figure 13, now grouped by the number of semantic composite keys processed. For instance, in the Foursquare dataset, the scenario $\mathcal{F} = \{\text{price}\}$ produces only five distinct price possibilities (–1, 1, 2, 3, and 4); the weather $\mathcal{F} = \{\text{weather}\}$ has six distinct possibilities (clear, clouds, fog, unknown, rain, and snow). However, $\mathcal{F} = \{\text{price}, \text{weather}\}$ has 29 distinct possibilities in the dataset (one fewer than if we individually combine price and weather values). Figure 14 shows the number of composite keys sorted in ascending order, grouped by the number of processed attributes. We segregate both results by data set (indicated at the top) to compare how the length influences performance.

We observe how the dataset size and the number of attributes generate an explosion in the elapsed times for the original implementations. However, we also observe that the MAT-Index performance is more associated with the number of processed semantic composite keys, and this is the reason why the BerlinMOD scenarios are more rapidly processed than the Foursquare dataset, even if they have the same number of attributes. In conclusion, MAT-Index successfully reduces the execution time by between 84.5 and 98.1%.

5.2.2 Evaluating the MAT-Index scalability performance

In this experiment we evaluate the scalability of MAT-Index by studying the impact of the trajectory size in the processing times. For this reason, we use a synthetic dataset designed by Ferrero et al. (2020) with 200,000 points, having as attributes latitude, longitude, time, weekday, price, weather, and poi-category. The points are distributed into six versions of the dataset with increasing trajectory lengths of 10, 20, 50, 100, 200, and 500 points. The computation time required by each similarity measure with and without MAT-Index support is reported in Figure 15.

It is worth observing that, as expected, the shorter a trajectory is, the more the problem tends to a linear comparison. The two public datasets (Table 1) reported in Section 5.2.1 contain trajectories longer than all the datasets used in this scalability experiment. Therefore, here we cover situations not present in the previous
experiments. Indeed, the scalability results show a considerable improvement, with consistent elapsed times for both methods where MAT-Index is used, in all tested scenarios, reducing the running times by between 75.5 and 87.2%.

Considering all the results presented in this evaluation, we can conclude that MAT-Index performs consistently faster. Its efficiency is more associated with the number of semantic composite keys than with the dataset size. For this reason, our algorithm performs much faster in the larger BerlinMOD dataset performed than the Foursquare dataset. The larger a dataset is, the more it tends to exhibit spatial, temporal, and semantic data repetition. For instance, most attributes such as weekdays, price categories, ratings, and even POIs are finite sets. Besides, according to the Pareto principle (Pareto, 1906), 20% of the causes encompass 80% of the problems, reinforcing the repetition tendency. Thus, MAT-Index perfectly fits the aims of this work, namely, to speed up the processing of similarity for larger datasets.

6 | CONCLUSIONS AND FUTURE WORK

Conducting similarity measurement of multidimensional data such as trajectories is a very costly task for clustering, nearest-neighbor queries, etc. Existing similarity measures cannot deal with the high volume of real datasets. Current indexes cannot efficiently manage all trajectory dimensions for similarity analysis. The proposed MAT-Index aims to fill this gap by indexing semantic content with multiple attributes, having spatial and temporal dimensions, into a compact data structure, ensuring efficiency in similarity computation while reducing data redundancy. The index is a combination of a dictionary and inverted indexes. The MAT-Index evaluation used the state-of-the-art trajectory similarity algorithms MSM and MUITAS. We show qualitatively how MAT-Index support for MSM and MUITAS drastically reduces the required comparisons. We also compute the index performance in terms of running time using two public datasets and scalability using one synthetic dataset. Experiments show an improvement of 98.1% in running time and 87.2% in scalability.

In future works we aim to test MAT-Index with additional semantically enriched trajectories datasets and to develop an efficient way to update the index without the need for full reprocessing when new data are added to the dataset.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES) Finance Code 001 and through the research project Big Data Analytics: Lançando Luz dos Genes ao
Cosmos (CAPES/PRINT process number 88887.310782/201800). This work was also supported by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Fundação de Amparo a Pesquisa e Inovação do Estado de Santa Catarina (FAPESC) Project Match co-financing of H2020 Projects Grant 2018TR 1266, and the European Union's Horizon 2020 research and innovation program under Grant Agreement 777695 (MASTER³). The views and opinions expressed in this article are the sole responsibility of the author and do not necessarily reflect the views of the European Commission.

CONFLICT OF INTEREST

Authors declare they have no conflict of interest.

DATA AVAILABILITY STATEMENT

All data used in the article can be made available.

ORCID

Ana Paula Ramos de Souza https://orcid.org/0000-0001-6457-3922
Chiara Renso https://orcid.org/0000-0002-1763-2966
Raffaele Perego https://orcid.org/0000-0001-7189-4724
Vania Bogorny https://orcid.org/0000-0002-0159-4643

REFERENCES

Berndt, D. J., & Clifford, J. (1994). Using dynamic time warping to find patterns in time series. In Proceedings of the 1994 Workshop on Knowledge Discovery in Databases, Seattle, WA (pp. 359–370). Menlo Park, CA: AAAI.

Büttcher, S., Clarke, C. L. A., & Cormack, G. V. (2016). Information retrieval: Implementing and evaluating search engines. Cambridge, MA: MIT Press.

Chen, L., Özsu, M. T., & Oria, V. (2005). Robust and fast similarity search for moving object trajectories. In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, Baltimore, MD (pp. 491–502). New York, NY: ACM.

Chen, X., Xu, J., Zhou, R., Zhao, P., Liu, C., Fang, J., & Zhao, L. (2019). S²R-tree: A pivot-based indexing structure for semantic-aware spatial keyword search. GeoInformatica, 24, 3–25. https://doi.org/10.1007/s10707-019-00372-z

Ding, H., Trajcevski, G., & Scheuermann, P. (2008). Efficient similarity join of large sets of moving object trajectories. In Proceedings of the 15th International Symposium on Temporal Representation and Reasoning, Montreal, QC, Canada (pp. 79–87). Piscataway, NJ: IEEE.

Düntgen, C., Behr, T., & Güting, R. (2009). BerlinMOD: A benchmark for moving object databases. VLDB Journal, 18, 1335–1368. https://doi.org/10.1007/s00778-009-0142-5

Ferrero, C. A., Alvares, L. O., & Bogorny, V. (2016). Multiple aspect trajectory data analysis: Research challenges and opportunities. In Proceedings of the 17th Brazilian Symposium on Geoinformatics, Campos do Jordão, SP, Brazil (pp. 56–67).

Ferrero, C., Petry, L. M., Alvares, L. O., da Silva, C. L., Zalewski, W., & Bogorny, V. (2020). Mastermovelets: Discovering heterogeneous movelets for multiple aspect trajectory classification. Data Mining and Knowledge Discovery, 34, 652–680. https://doi.org/10.1007/s10618-020-00676-x

Finkel, R., & Bentley, J. (1974). Quad trees: A data structure for retrieval on composite keys. Acta Informatica, 4, 1–9. https://doi.org/10.1007/BF00288933

Furtado, A. S., Alvares, L. O., Pelekis, N., Theodoridis, Y., & Bogorny, V. (2018). Unveiling movement uncertainty for robust trajectory similarity analysis. International Journal of Geographical Information Science, 32(1), 140–168. https://doi.org/10.1080/13658816.2017.1372763

Furtado, A. S., Kopanaki, D., Alvares, L. O., & Bogorny, V. (2016). Multidimensional similarity measuring for semantic trajectories. Transactions in GIS, 20(2), 280–298. https://doi.org/10.1111/tgis.12156

Han, Y., Wang, L., Zhang, Y., Zhang, W., & Xuem Lin, X. (2015). Spatial keyword range search on trajectories. In M. Renz, C. Shahabi, X. Zhou, & M. A. Cheema (Eds.), Database systems for advanced applications (pp. 223–240). Cham, Switzerland: Springer.

Issa, H., & Damiani, M. L. (2016). Efficient access to temporally overlaying spatial and textual trajectories. In Proceedings of the 17th IEEE International Conference on Mobile Data Management, Porto, Portugal (pp. 262–271). Piscataway, NJ: IEEE.
