MAT-Index: An Index for Fast Multiple Aspect Trajectory Similarity Measuring

Ana Paula Ramos de Souza[1], Chiara Renso[2], Raffaele Perego[3], and Vania Bogorny[4]

[1,4]PPGCC, Federal University of Santa Catarina, Brazil
[2,3]ISTI-CNR, Pisa, Italy
[1]ana.ramos@posgrad.ufsc.br, [2,3]{chiara.renso,raffaele.perego}@isti.cnr.it,
[4]vania@inf.ufsc.br

February 15, 2022

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 like querying, clustering, similarity, classification, etc. Multiple aspect trajectory similarity measuring 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 propose a fast point-to-point comparison. In this paper we propose the Multiple Aspect Trajectory Index (MAT-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 up to 98.1%.

Keywords: Spatio-temporal indexing. Similarity Measures. Multiple Aspect Trajectory similarity indexing.

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, etc. The trajectory definition has evolved from *raw*, before 2007, to *semantic* in 2008 [28] and then to the very recent concept of *multiple aspect trajectory* in 2016 [23, 8].

Figure 1 shows an example of multiple aspect trajectory where each trajectory point holds 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.

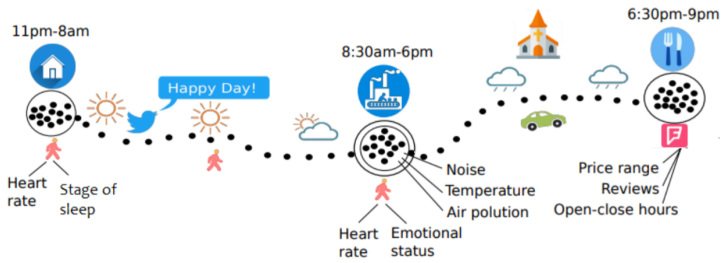

Figure 1: Example of a Multiple Aspect Trajectory Melo et al. (2019) [23]

We can notice from the figure that the multiple aspect trajectory (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 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 like querying, clustering, similarity, and classification, all research topics that have received large interest in recent years. Trajectory similarity measuring is the basics for querying and clustering moving objects with similar characteristics. There are several similarity/distance measures in the literature as DTW (Dynamic Time Warping), MDTW (Modified Dynamic Time Warping), LCSS (Longest Common Subsequence), EDR (Edit Distance for Real Sequences), Fréchet Distance, etc, but most of them were either developed for time series or do not support all three dimensions of mobility data that are space, time, and semantics.

Similarity measures that were specifically developed for trajectories include UMS (Uncertain Movement Similarity) [10], MSM (Multidimensional Similarity Measure) [12] and MUITAS (MUltIple aspect TrAjectory Similarity) [26]. UMS is very robust for spatial similarity, but it does not consider time and semantic dimensions, what is fundamental for mobility data. On the other hand, MSM and MUITAS have outperformed the well known older measures DTW [1], LCSS [29], and EDR [3]. EDR and LCSS force 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 using a different distance function 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. In the example, a trajectory A has three points (a_1, a_2, a_3) and trajectory B has five points $(b_1, b_2, b_3, b_4, b_5)$. 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, so far they are the most robust for measuring the trajectory similarity, independently of the dimensions present in the dataset.

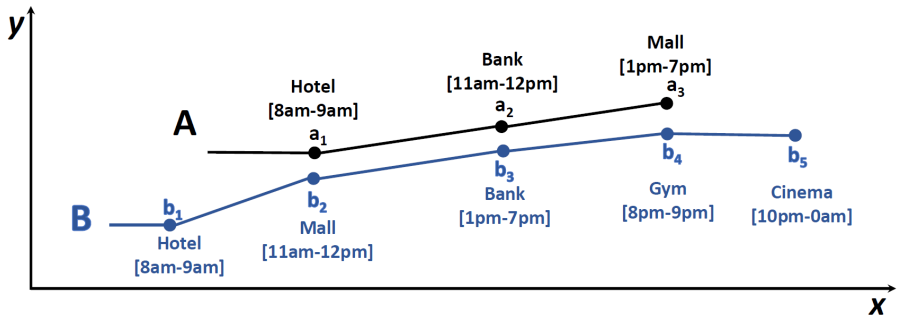

Figure 2: Two Semantic Trajectories Furtado et al. (2016) [12]

In the example of Figure 2, MSM and MUITAS need to compare each point of the trajectory A to all points of the trajectory B, and for all dimensions, to discover that semantically the trajectory A is totally contained in B, i.e, they share the semantic similarity of three points. In other words, MSM compares each dimension of point of 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 FTSM (Fast Trajectory Similarity Measuring), an index for fast similarity measuring of UMS and MSM [11]. However, it indexes only the spatial dimension, what is not sufficient to support large trajectory datasets that have many semantic aspects. A survey about general indexing data structures is presented in [20], 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 were 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 paper we propose an index data structure for historical data called 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 MAT-Index can drastically reduce the number of comparisons, and a quantitative evaluation shows a gain for all tested scenarios up to 98.1% processing time reduction and up to 87.2% in scalability evaluations.

The rest of the paper 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 paper and suggests directions of future works.

2 Basic Concepts

This section presents the main concepts to understand this work. Section 2.1 defines the multiple aspect trajectories, Section 2.2 presents the basics about the similarity measures developed for this type of data, focusing on the Multidimensional Similarity Measure (MSM) [12] and the MUltIple aspect TrAjectory Similarity [26], and Section 2.3 presents the index basic definitions.

2.1 Multiple Aspect Trajectories

Multiple aspect trajectory is a sequence of points where each point has the dimensions of space, time, and several semantic aspects.

Multiple Aspect Trajectory A Multiple Aspect Trajectory is a sequence of points $T = \langle p_1, p_2, ..., p_n \rangle$, such that $p_i = (x, y, t, A)$ is its i-th point composed of a location (x, y), also called as spatial dimension, a timestamp t, also called temporal dimension, and a non-empty set of *aspects* $A = \{a_1, a_2, ..., a_m\}$, representing the semantic dimension.

Aspect An Aspect $a = (desc, ATV)$ is a relevant real-world fact for mobility data analysis. It is composed of a description ($desc$) and an instance of its corresponding aspect type (a_{type}), that is represented as a set of attribute-value pairs $ATV = \{att_1 : v_1, att_2 : v_2, ..., att_z : v_z\}$.

Aspect Type An aspect type $a_{type} = \{att_1, att_2, ..., att_z\}$ is a categorization of a real-world fact composed of a set of attributes (att). In other words, an aspect type and its attributes act as a metadata definition for an aspect.

For the sake of understanding, consider the following example adapted from [23] 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 either for sequential data as LCSS [29], EDR [3], w-constrained Discrete Frechet distance (wDF) [5], or for trajectories as CVTI [15], MSTP [33], MTM [31], MSM [12], UMS [10], SMSM [16], and MUITAS [26]. 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 on the indexing proposal.

2.2.1 Multidimensional Similarity Measure (MSM)

The Multidimensional Similarity Measure (MSM) [12] 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 $q \in Q$. The method computes the parity of two trajectories, (P, Q) and (Q, P), using the maximum matching *score* Equation 1.

$$parity(P, Q) = \sum_{i=1}^{|P|} max\ score(p_i, Q) | q_i \in Q\} \tag{1}$$

The *score* is computed according to Equation 2, and provides a matching score of two points p and q for each dimension. It consists of pairwise comparing the attributes A of p with q, considering the user defined thresholds ($maxDist_k$), and scores the matches (Equation 3) according to their respective weights (ω).

$$score(p, q) = \sum_{k=1}^{|A|} (match_k(p, q) * \omega_k) \tag{2}$$

$$match_k(p, q) = \begin{cases} 1, & \text{if} \quad dist_k(p, q) \leq maxDist_k \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

The total similarity of two trajectories is given in Equation 4, by summing both parities and dividing the result by the sum of both trajectory lengths (P and Q).

$$MSM(P,Q) = \begin{cases} 0, & \text{if } |P| = 0 \text{ or } |Q| = 0 \\ \frac{parity(P,Q)+parity(Q,P)}{|P|+|Q|}, & \text{otherwise} \end{cases} \quad (4)$$

2.2.2 MUltIple aspect TrAjectory Similarity (MUITAS)

MUITAS [26] 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 named feature. This property of feature makes MUITAS a flexible measure that supports both 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 capability to assign different distance functions to each attribute.

A *feature* $f = \{a_1, a_2, ..., a_z\}$ is a nonempty 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 $f_i = \{place\_category, price\_tier, rating\}$ represents information about visited *POIs*. There are three associated attributes, this analysis unit is dependent. However, the feature $f_j = \{weather\_condition\}$ represents an independent analysis unit.

Suppose P and Q as two trajectories and p and q trajectory points, such that $p \in P$ and $q \in Q$, the Equation 5 computes the matching score between p and q. For each attribute A of a feature F, the points will match if the distance between them is lower than a given threshold (δ).

$$match_{f_i}(p,q) = \begin{cases} 1, & \text{if } \forall a_j \in f_i, dist_j(p,q) \leq \delta_j \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

For each feature, Equation 6 computes the score as the weighted sum (ω) of matching points.

$$score(p,q) = \sum_{i=1}^{|\mathcal{F}|} (match_{f_i}(p,q)) * \omega_i \quad (6)$$

After comparing all points, MUITAS calculates the $parity(P,Q)$ as the sum of the best scores of each attribute of each point p in P comparing to Q (Equation 7), and the $parity(Q,P)$.

$$parity(P,Q) = \sum_{k=1}^{|P|} max\ score(p_k, Q) | \forall q \in Q\} \quad (7)$$

The final similarity score between two trajectories is the sum of parities, divided by the sum of the trajectory lengths (Equation 8).

$$MUITAS(P,Q) = \begin{cases} 0, & \text{if } |P| = 0 \text{ or } |Q| = 0 \\ \frac{parity(P,Q)+parity(Q,P)}{|P|+|Q|}, & \text{otherwise} \end{cases} \quad (8)$$

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

2.3 Indexes

There are several indexes developed specifically for spatial data. Examples are the Quadtree [9], Z-Ordering Tree [24], and OCTree [21]. In fact ***Trees*** are one of the most commonly adopted data structures for indexing purposes, but the previously mentioned indexes do not support multidimensional spatio-temporal similarity.

Concerning semantic contents, an *inverted index* (also known as an *inverted list*) is frequently used. It saves the data into a smaller and more organized way, like an dictionary composed of two main parts: the search structure (aka *keyword* or *key*), and a list of references (aka *value*) [2]. Figure 3 illustrates an example of inverted index containing seven documents that have keyword occurrences previously processed (on the left side table). The corresponding inverted index (table on the right) stores the distinct keywords *Hotel*, *Cinema*, *Park*, *Home*, and *Work* as keys containing its corresponding references to the table in the left.

DOCUMENTS	
Reference	**Document Keywords**
1	Home,Park
2	Cinema,Hotel
3	Cinema,Home,Work
4	Cinema,Hotel,Park
5	Cinema,Work
6	Park
7	Park,Work

INVERTED INDEX	
Key	**List of References**
Hotel	2,4
Cinema	2,3,4,5
Park	1,4,6,7
Home	1,3
Work	3,5,7

Figure 3: Example of Inverted Index

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

Despite of the several solutions for different data, indexing space, time, and semantic data together is far more complex, especially concerning to 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 [20] 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, like in [35, 34, 14, 4] that only focuses on spatial and semantic dimensions disregarding the temporal dimension, and [14] that is also limited to a single semantic keyword. Still among these works, some approaches are limited to ranges like in [27, 22, 14, 13] focusing on indexing semantic keywords according to both spatial and temporal range/granularity, and [18, 19, 30] that index spatial and keywords limited only to a time interval/range.

Concerning the applicability, most of these mentioned works focus on accelerating range queries [30, 27, 19, 22, 14, 13, 18], while [30, 27, 35, 19, 34, 18] intends to efficiently answer TOP-K searches. Both mentioned approaches are intrinsically pruning strategies, limiting access to indexed data to a restricted portion of the universe analyzed, facilitating processing – the same problem of works limited to exact matches [27, 13]. However, before making conclusions, it is crucial to comprehend the whole picture by comparing trajectories, not only isolated points. This way, we can more assertively extract information based on similar behaviors, not in circumstantial cases. Thus these approaches are incompatible with our aim to compare all dataset, since it is not feasible to prune neither of multiple aspect trajectories dimensions and still to establish a precise comparison.

Besides, all the indexes mentioned above have different data structures connected, demanding a high storage cost due to the redundant data kept to optimize the access. This situation could force the Operating System to excessively transfer data between memory levels, delaying access in a known problem named *thrashing*. Thus, some solutions design hybrid (i.e., part memory, part disk) [35, 34, 13, 17] or disk [14] allocation strategies to avoid system collapse. However, the disk and hybrid allocation solutions require transferring data, thus multiple accesses. The secondary memory is a very slow resource, therefore inefficient for processing large and complex trajectories datasets. Still, some works deal with the performance by adopting approximate solutions in [27, 34], 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 measuring. Indeed, existing indexing works do neither provide 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.

4 Multiple Aspect Trajectory Index

In this section we propose MAT-Index (Multiple Aspect Trajectory 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.

MAT-Index is divided in 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 *Index Integration* step integrates the three resulting data structures built in the previous steps. The following sections detail each step of MAT-index.

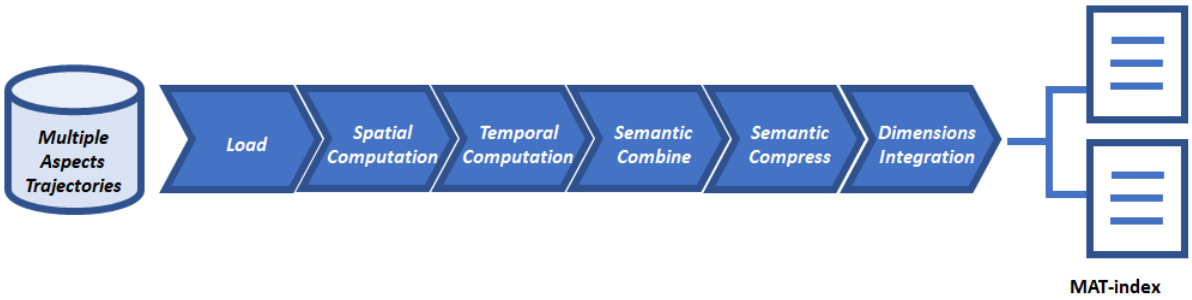

Figure 4: MAT-Index Flow

4.1 Load

The *Load* stage saves the spatial, temporal, and semantic dimensions into separated data structures to be merged in the last index step. Figure 5 shows the data structures saved in the load step. Figure 5 (a) 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, eleven 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. For the sake of understanding, in this example we consider that each feature (as defined in Section 2.2.2) contains only one attribute, thus $\mathcal{F} = \{\{price\}, \{poi\}, \{weather\}\}$. 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), that is the

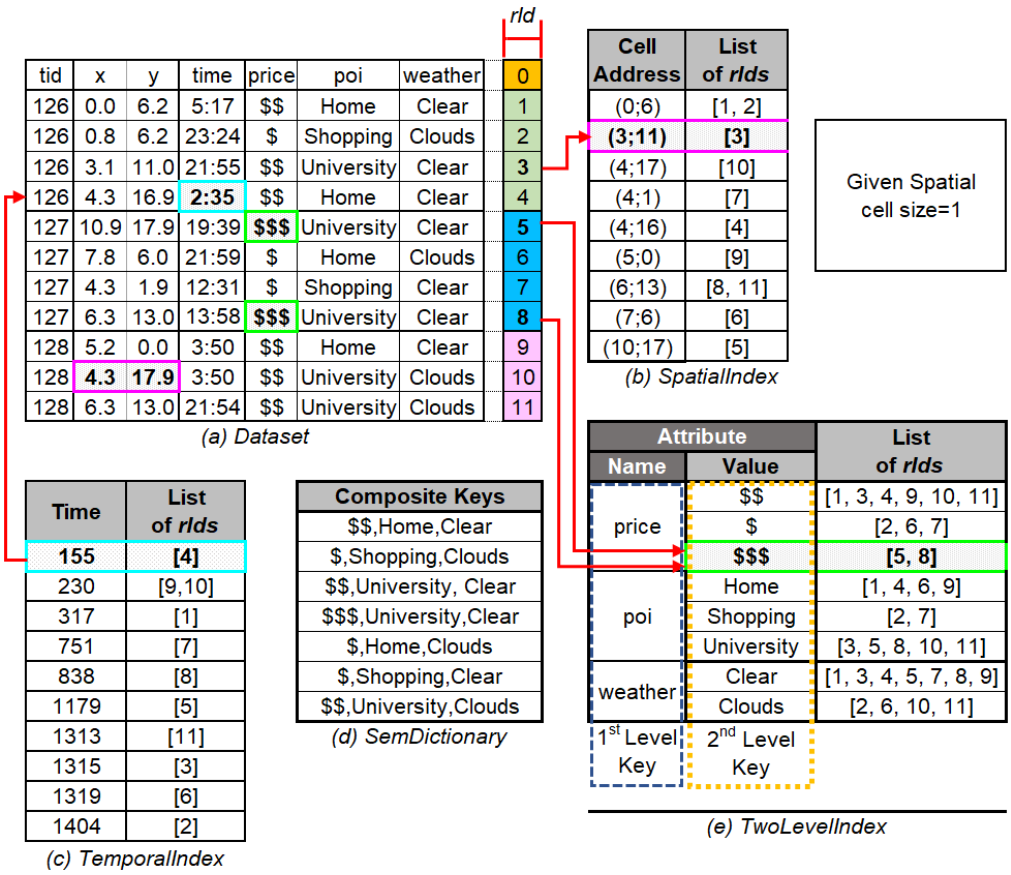

Figure 5: Running example with the spatial (b), temporal (c), and semantic (d and e) data as preliminarily indexed

header of the dataset, is processed by the method *addAttributeToTwoLevelIndex* (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 5 (e) in the delimited dashed area on the left, named as 1st level key.

From rId=1 to rId=11, i.e., for all trajectory points, the process is repeated, sequentially reading the dataset. For each row, it stores the trajectory dimensions as follows: the method *addToSpatialIndex* saves the *SpatialIndex* in the format shown in Figure 5 (b); the method *addToTemporalIndex* saves the *TemporalIndex* in the format shown in Figure 5 (c) and the methods *addToSemDictionary* and *addValueToTwoLevelIndex* save the semantic structures *SemDictionary* in Figure 5 (d) and *TwoLevelIndex* in Figure 5 (e). Theses methods are explained in the sequence.

The ***addToSpatialIndex*** (line 3) saves the spatial content of each rId in the *SpatialIndex*

Algorithm 1: Load

input : Dataset // Figure 5(a)
output: SpatialIndex, TemporalIndex, SemDictionary, TwoLevelIndex

```
addAttributeToTwoLevelIndex(header) // Saves header (rId=0) -- Fig 5(e) 1st lvl
foreach point ∈ Dataset do          // From rId=1 until the last dataset row
    addToSpatialIndex(point.coordinates)                    // Figure 5(b)
    addToTemporalIndex(point.time)                          // Figure 5(c)
    addToSemDictionary(point.semCompositeKey)               // Figure 5(d)
    addValueToTwoLevelIndex(point.attributeValues)  // Figure 5(e) 2nd level
end
return SpatialIndex, TemporalIndex, SemDictionary, TwoLevelIndex
```

shown in Figure 8 (b). To generate the *SpatialIndex*, MAT-index simulates a grid data structure as shown in Figure 6 (left), without allocating a matrix that brings the sparsity issue, and the difficulty to balance 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 τ 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 as τ.

Thus, all the points in the same cell (as the example [1,2] in Figure 7 left) do automatically match, since they are below the threshold, thus not requiring the spatial comparison in these cases. The *SpatialIndex* is than 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.

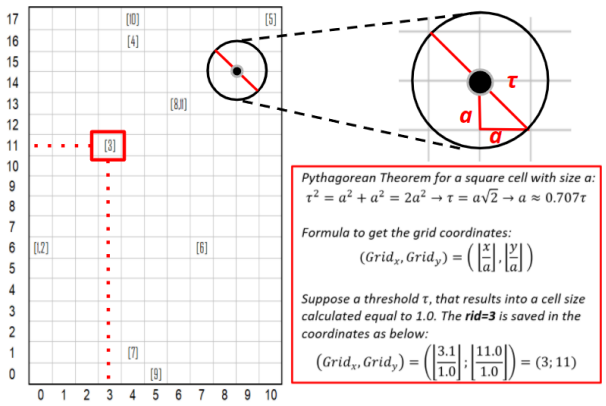

Figure 6: Spatial Allocation Method

Regarding the temporal content, MAT-Index allocates only actual occurrences in an inverted list, similarly to the spatial indexing process. However, it is worth noticing that a day

time unit can be expressed as 24 hours, 1,440 minutes, or still 86,400 seconds, 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 uni-dimensional value that can be segmented based on the **unit** of the temporal threshold (τ) used by the similarity measures and aggregated later into temporal intervals according to the threshold **value**. The method ***addToTemporalIndex*** (line 4) saves the temporal dimension in the *TemporalIndex* inverted list, as shown in Figure 5 (c). Each entry $< key, value >$ in the $TemporalIndex$ holds a *time* and a list of $rIds$ with the same temporal information. The running example of Figure 5 (a) has a temporal threshold of five minutes, which means that the key *time* of the $TemporalIndex$ must be saved in minutes. For instance, the rId=4 in Figure 5 (a) has a time in hours (2:35), which is then saved in minutes (2x60+35=155 minutes), as shown in the first entry in Figure 5 (c).

Concerning the semantic dimension, MAT-Index preliminary saves the semantics into two structures. The method ***addToSemDictionary*** *(row 5) saves* a semantic dictionary with the distinct semantic composite keys in Figure 5 (d). The method ***addValueToTwoLevelIndex*** *(line 6)* stores in the *TwoLevelIndex* the second level key (dashed area named as 2nd level key) in Figure 5 (e), 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 5 (e). 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 appeared in the dataset of Figure 5. For instance, value 1 could represent a price, a rating, an age, or others. 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 demand a pairwise comparison of two trajectory points to check if the distance between them does not exceed the similarity threshold (τ). Therefore, MAT-index uses the auxiliar data structure *SpatialIndex* generated in the Load step to avoid the comparison among all trajectory points. The *SpatialIndex* presented in Figure 5 (b) is an inverted list where each entry is a pair $< key, value >$, being each key 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 $< cellAddress, rIds >$. For each rId (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 method *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 5(b)
   output: spatialMatches updated
 1 foreach < cellAddress, rIds > ∈ SpatialIndex do // Entry <Key, Value>
 2 |  foreach rId in rIds do
 3 |  |  Points.get(rId).spatialMatches = Points.get(rId).spatialMatches OR
   |  |   SpatialIndex.get(cellAddress)
 4 |  |  foreach (pCandidate ∈ getCandidateCells(cellAddress)) do
 5 |  |  |  if ( distance(Points.get(rId), Points.get(pCandidate) ≤ spatialThreshold) )
   |  |  |   then
 6 |  |  |  |  Points.get(rId).spatialMatches.set(pCandidate)
 7 |  |  |  |  Points.get(pCandidate).spatialMatches.set(rId)
 8 |  |  |  end
 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 5 (b), the rIds {8,11} automatically match as well as {1,2} because they are in the same cell. The pairs of rIds {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:

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

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

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, as shown in the example of Figure 5 (c), a list with all the matching rIds, that are the trajectory points belonging to the cells in the interval admitted by the temporal threshold τ.

Algorithm 3 shows the pseudo-code that receives as input the TemporalIndex generated in the Load step and provides as output. It starts by sequentially reading the entries in the *TemporalIndex* (line 1), and, for each entry composed of $< time,\ rIds >$, 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 (line 3), i.e., $rIds$, the $listOfMatches$ is associated to the $temporalMatches$ of the processed rId (line 4).

Algorithm 3: computeTemporalMatches

input : *TemporalIndex* `// Figure 7(a)`
output: $temporalMatches$ updated

1 **foreach** ¡*time,rIds*¿ ∈ *TemporalIndex* **do** `// Entry <key, value> to be analyzed`
  `// Union of all` $rIds$ `associated to entries in the interval admitted`
2 listOfMatches = $\bigcup_{i=time-\tau}^{time+\tau} TemporalIndex.get(i).rIds$ `//` τ`: temporal threshold`
3 **foreach** $rId \in rIds$ **do** `// Row Ids from the entry analyzed`
4 Points.get(rId).temporalMatches = listOfMatches
5 **end**
6 **end**
7 **return** $temporalMatches$ `// For all trajectory points they are updated`

Figure 7 (a) shows the temporal data as stored in the Load phase, and (b) the corresponding list of matches. All $rIds$ in Figure 7 (a) shared 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 timestamps and a distance threshold of 5 minutes. Figure 7 (b) shows the corresponding Matching $rIds$ where, for instance, the cells 1313 and 1315 are within the threshold, so the list of $rIds$ of both cells, i.e., rId [3] and rId [11] are added to both corresponding Matching $rIds$ entries in Figure 7 (b). Additionally, 1315 and 1319 are within the threshold of 5 minutes, thus the process is repeated and the $rIds$ [3] and [6] are added to 1315 and 1319 matching index cells in Figure 7 (b). The result of a non-transitive property here is clear, since we can notice that cell 1315 shares 1313 and 1319 as match ($rIds$ 3,6, and 11), with both —1315-1313— and —1315-1319— less or equal than the threshold of 5 minutes, while 1313 and 1319 do not match at all.

The temporal approach brings two benefits: *(i)* 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; *(ii)* the number of index entries is limited to the threshold unit, i.e., if expressed in minutes, 1,440 possibilities, thus it tends to require less 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, in average, more points, since we have a very limited number of possible cells allocated (1,440 in this case). It saves memory and mainly processing time because the idea is to process the matches in groups (by cell), not by rId.

Cell	*List of rIds*
155	[4]
230	[9,10]
317	[1]
751	[7]
838	[8]
1179	[5]
1313	[11]
1315	[3]
1319	[6]
1404	[2]

(a) *TemporalIndex*

Matching rIds
[4]
[9,10]
[1]
[7]
[8]
[5]
[3,11]
[3,6,11]
[3,6]
[2]

(b) *List of Matches*

Figure 7: Result after the Temporal Computation

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 5 (d) and associating each one to a *match counter*, that is created using the occurrences in the Two-Level Inverted Index from Figure 5 (e). 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., $\forall P, Q, R$, 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 is used to build its corresponding match counter (line 2, in Function *getMatchCounter*). 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.

Function $getMatchCounter$ (Algorithm 4) 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 method retrieves the *List of rIds* of the corresponding attribute values, as the example shown in Figure 5 (e). The feature f in $\mathcal{F}$ matches (row 6) if all its corresponding attribute values match at the same rId position. Thus, the method executes an AND operator to get a unique list of rIds where all attributes match for f (row 7). Then, only the rIds in common are updated in the $MatchCounter$. After processing the combination for all sets of features, the method returns the $MatchCounter$ in line 11.

Figure 8 presents an example of how to obtain a match counter for the composite key $< \$\$, Home, Clear >$, with $\mathcal{F} = \{\{price\}, \{poi\}, \{weather\}\}$ 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 *rid*s 1,4 and 9 are composed of the same set of attribute values, so for three attribute values the maximum score is 3.

Algorithm 4: Semantic Combine

input : *SemDictionary* // Figure 5 (d)
output: *SemCompositeIndex* // Figure 9

1 **foreach** *semCompositeKey* $\in$ *SemDictionary* **do**
2 | SemCompositeIndex.put(semCompositeKey, *getMatchCounter(semCompositeKey)*)
3 **end**
4 **return** SemCompositeIndex

input : *semCompositeKey*, *TwoLevelIndex* // Figures 5 (d) and (e)
output: *MatchCounter*

5 **Function** *getMatchCounter*(*semCompositeKey*):
6 **foreach** $f \in \mathcal{F}$ **do** // Being each feature f a set of attributes
| // rIds where all feature attributes values in f match (Eq.5)
7 | **foreach** $rId \in \bigcap_{i=1}^{|f|} TwoLevelIndex(attributeValue)_i.rIds$ **do**
8 | | ++MatchCounter[rId]
9 | **end**
10 **end**
11 **return** MatchCounter

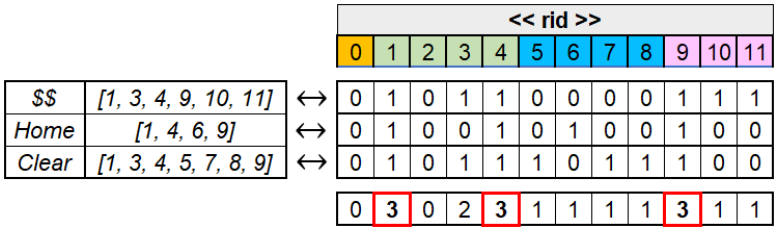

Figure 8: Example of Match Counter computation

Figure 9 shows the entire composite index after the Combine step execution, including the 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 $< \$\$, Home, Clear >$ (see the first row of the tables in Figure 9) entirely matches with the $rIds$ {1,4,9}. It also matches with the rId=3 in two of the three semantic attribute values, only once with the $rIds$={5,6,7,8,10,11}, or yet do not match 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 $< \$\$, Home, Clear >$ at position rId=10.

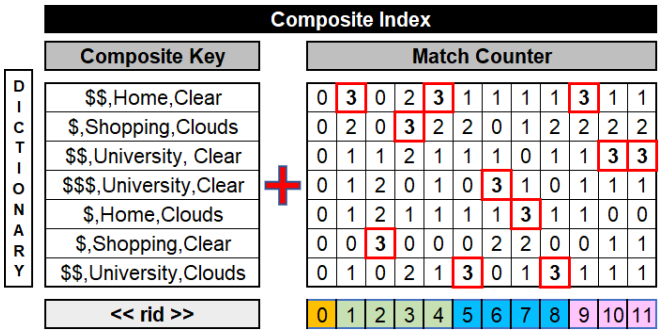

Figure 9: Semantic Index after *Match Counter* computation

4.5 Semantic Compress

The similarity algorithms for multiple aspect trajectories MSM [12] and MUITAS [26] 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 of the Compress step. For each $semCompositeKey$ in $SemCompositeIndex$ (Figure 9) (Algorithm 5, row 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                                   // Figure 10(a)
   output: CSemCompositeIndex                                   // Figure 10(b)
1  foreach < semCompositeKey, matchCounter >∈ SemCompositeIndex > do
2  |  CSemCompositeIndex.put(semCompositeKey,compressMatchCounter(MatchCounter))
3  end
4  return CSemCompositeIndex
   input  : MatchCounter
   output: CMatchCounter
5  Function compressMatchCounter(MatchCounter):
6  foreach p in Points do
7  |  CMatchCounter[p.cId] = max(CMatchCounter[p.cId], MatchCounter[p.rId])
8  end
9  return CMatchCounter
```

The Function *compressMatchCounter* in Algorithm 5 (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. Figure 10 (a) presents the top scores by trajectory point, while Figure 10 (b) the top scores by trajectory. We notice that the eleven 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 anymore. The first column of the *MatchCounter* in Figure 10 (a) corresponds to the header position in the dataset (see Figure 5 (a) at rId=0), i.e., where the name of attributes are placed. They do not contain trajectory content, being maintained at first to avoid computing the position during the load and combine process. Thus, it is discarded.

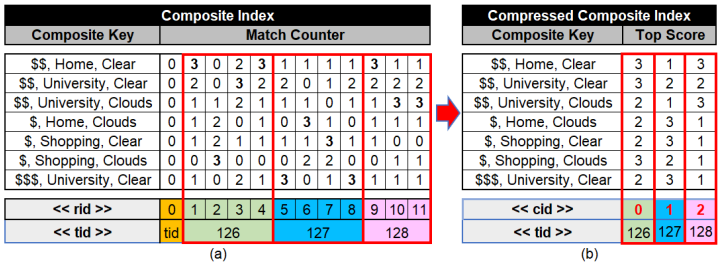

Figure 10: Composite Index Highlighted By Trajectory: (a) before and (b) after the compressing

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 *Indexes Integration* phase consolidates the matching scores into a single data structure, finalizing the MAT-index construction.

The pseudo-code for this step is depicted in Algorithm 6. For each point p (line 1), the method uses the semantic composite key of p to retrieve and store in an auxiliary variable its corresponding compressed match counter *auxCMatchCounter* (line 2). The compressed match counter holds the top scores by trajectory for a valid combination of semantic attribute values. In line 3, the method 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 method updates its $auxCMatchCounter$ 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 11 (a), and the value of the *MatchCounter* at rId position increased by 2, as shown in Figure 11 (b). By checking the maximum value, the method avoids repeating the compressing step for each tra-

jectory point. After updating the cases that match in both space and time, the method 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 method 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 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 Compresses Composite Index in Figure 11 (a) and the Composite Index in Figure 11 (b) data structures are no longer required, so they are discarded (lines 13 and 14). The Mat-Index (illustrated in Figure 11 (c)) is ready to be used.

Algorithm 6: Dimensions Integration

```
   input  : SemCompositeIndex, CSemCompositeIndex                    // Fig 10
   output: MATIndex                                                 // Figure 11(c)
 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]=
   |   |    max(auxCMatchCounter[Points.get(rId).cId],
   |   |    SemCompositeIndex.get(p.semCompositeKey)[rId]+2)
 6 |   end
 7 |   oneMatch = p.spatialMatches XOR p.temporalMatches
 8 |   foreach rId in oneMatch do
 9 |   |   auxCMatchCounter[Points.get(rId).cId]=
   |   |    max(auxCMatchCounter[Points.get(rId).cId],
   |   |    SemCompositeIndex.get(p.semCompositeKey)[rId]+1)
10 |   end
11 |   MATIndex.put(p.rId, auxCMatchCounter)
12 end
13 CSemCompositeIndex.clear();
14 SemCompositeIndex.clear();
15 return MATIndex
```

Going back again to our example in Figure 5, each trajectory point has now 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 Figure 11 (c) the value 5. By updating the spatial and temporal matches, the pairs of trajectory points {1,2}, {8,11}, and {4,10} do spatially match. The pairs of trajectory points {9,10}, {3,11}, and {3,6} do temporally match. All matches are updated in the final MAT-Index in Figure 11 (c).

Note that the final score in Figure 11 (c) is the maximum between the result incremented

in Figure 11 (b) and the auxiliary top score preserved in Figure 11 (a), as explained in Algorithm 6. For instance, updating the match between rIds 3 and 11, starting by rId=11 (¡*$$, University, Clouds*¿). The trajectory point rId=3 corresponds to the trajectory cId=0. In Figure 11 (b), the composite key ¡*$$, University, Clouds*¿ at position rId=3 has score of 3. It is higher than the old top score 2 in Figure 11 (a). So, the method updates the score of trajectory point rId=11 at cId=0 to 3 in the MAT-Index Figure 11 (c). The method repeats this process for all matches. The updates are highlighted in Figure 11 (c).

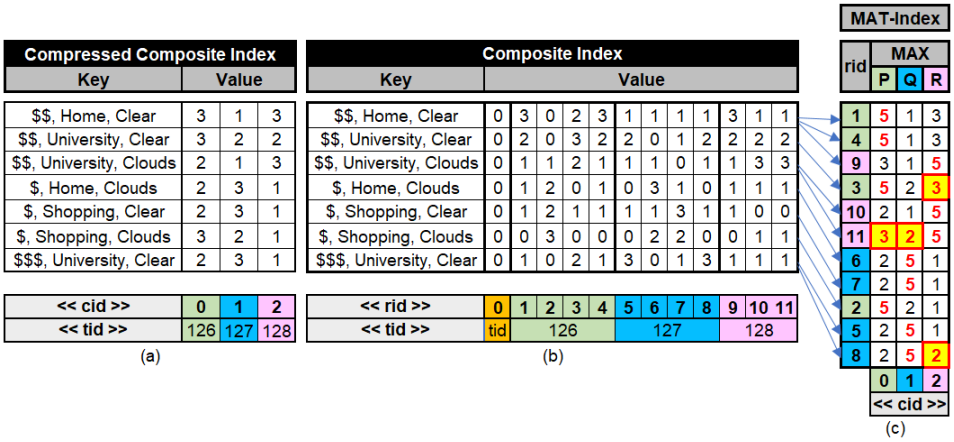

Figure 11: MAT-Index (c) after Integration

The final MAT-Index data structure, depicted in Figure 11 (c), 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 is 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 a complexity $O(N)$, with N 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. Indeed, we have here again an $O(N)$ 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 to 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 have thus an $O(2\tau N)$ 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 precisely N times each attribute in A, i.e., $A \times N$ times. Considering that N tends to be much bigger than A in order of magnitude, the method has a linear complexity of $O(AN)$.

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 $O(N)$. In the worst case, all the composite keys are distinct, resulting in a complexity of $O(N^2)$. Both extremes are unlikely. For instance, considering the assessed datasets, the *Foursquare* has 7107 keys. The *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 MAT-Index performance tended to a linear behavior, as discussed along with Section 5.

Finally, the *Dimensions Integration* method depicted in Algorithm 6 updates only the cases that match. Since it maintains *BitSet*s for spatial and temporal matches, the logical operations prevent double retrieving the same content, which is done in linear time.

After analyzing all MAT-Index methods, we can conclude that the MAT-Index bottleneck lies in the compressing step complexity that can vary from linear to quadratic depending on the number of semantic composite keys. Again, two unlikely situations, 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 represents a minimal time compared to the original similarity measures performance.

5 Evaluating MAT-index for Trajectory Similarity Measuring

In this section we evaluate the performance of MSM and MUITAS using MAT-index. We also compare MAT-index with FTSM [11], a spatial index proposed to accelerate the comparison of the spatial dimension of MSM. MAT-Index is general and can be potentially 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 of 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*, that 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 a dataset composed of three attributes att_1, att_2, and att_3. MSM pairwise compare all three attributes separated, while MUITAS can process different feature configurations like $\mathcal{F} = \{\{att_1\}, \{att_2\}, \{att_3\}\}$ (where each feature has one attribute, leading to MSM definition), $\mathcal{F} = \{\{att_1\}, \{att_2, att_3\}\}$, 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 throughout 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 the quantitative evaluation of MAT-Index, focusing on time and scalability metrics.

5.1 Qualitative Evaluation

We start this subsection presenting a scenario where 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 to compare 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 with the parity (MSM Equation 1, MUITAS Equation 7) computation, thus skipping for MSM Equations 3 and 2, and for MUITAS Equations 5 and 6. Accessing MAT-Index data structure to get the top scores consolidated – Figure 11 (c) –, we compute the final score in linear time of $|P| + |Q|$ accesses whose contents are added, while the original problem would require quadratic cost by comparing all points of P with all points of Q, for each dimension, i.e., $|P| \times |Q| \times |A|$ comparisons, being A the number of attributes processed (space, time, and semantic attributes).

Going back to the running example in Figure 5 and considering the features $\mathcal{F} = \{\{price\}, \{poi\}, \{weather\}\}$, suppose we want to measure the similarity between trajectory ids 126 and 128. For reference, the first column (tid) of the dataset in Figure 5 (a) identifies the trajectory ids and its points. The first step to find the similarity score $Sim(126, 128)$ is to compute the parities by querying the top scores of the points of both trajectories in MAT-Index (Figure 11 (c)). Starting from the computation of $parity(126, 128)$, we recall that the trajectory 126 ($cId = 0$) is composed of four points with $rIds = \{1, 2, 3, 4\}$. It is necessary to retrieve the value in $key = rId$ at position $cId = 2$ (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,

$$parity(126, 128) = \frac{\sum_{rId=1}^{4} max(rId, cId)}{|\mathcal{F}|} = \frac{3 + 1 + 3 + 3}{5} = 2$$

The process must be repeated by inverting the references for all points of 128 to get the $parity(128, 126)$, thus summing the retrieved scores at position $cId = 0$.

$$parity(128, 126) = \frac{\sum_{rId=9}^{11} max(rId, cId)}{|\mathcal{F}|} = \frac{3 + 2 + 3}{5} = 1.6$$

After calculating the parities, the final similarity score $Sim(126, 128)$, i.e., MSM(126,128) and MUITAS(126,128) are the sum of parities divided by the sum of each trajectory length (number of points), such that:

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

The presented process can be repeated for any pair of trajectories, using their rIds 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: first, 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 the 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 in the processing times.

All quantitative experiments were performed in an Intel® Core™ i7-9750H Coffee Lake CPU @ 2.60GHz (12MB cache), 32GB Crucial Dual-Channel @ 1330MHz (19-19-19-43), 500GB Samsung SSD 970 EVO Plus, and 4GB NVIDIA GeForce GTX 1650 in a Windows 10 Education 64-bit, using 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 in two parts: *(i)* how faster do both similarity measures process the entire dataset when employing MAT-index and FTSM access methods and; *(ii)* how do 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. The Foursquare NYC [32] dataset contains real data, while the other is a benchmarck dataset generated by BerlinMOD [6]. These datasets have different characteristics, including the number of points and trajectory average sizes that make them suitable for evaluating MAT-Index.

The Foursquare NYC contains semantically enriched check-ins of users collected from April 2008 to October 2010. The Foursquare API[1] provided the semantic information related to the POI: category (*root-type*), subcategory (*type*), and *price* – this last one is a numeric classification; the Weather Wunderground API[2] provided the *weather* conditions. The Berlin Moving Object Dataset (*BerlinMOD*) is a public spatio-temporal dataset containing moving point data, simulating workers commuting between their homes and workplaces on the real street network of the German capital Berlin during two days. It holds two semantic attributes re-

[1]https://developer.foursquare.com/
[2]https://www.wunderground.com/weather/api/

garding the *type* of the user and the transportation *mode*. In both datasets, the timestamp provided the weekday (*day*).

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

Dataset	Description	
Foursquare NYC [32]	Traj Size:	$209,97 \pm 188,36$
	# of Traj.:	3,079
	# of Points:	227,403
	# of Users:	1,083
	Attributes:	Latitude, longitude, time, weekday, price, weather, POI type and root-type.
SECONDO BerlinMOD [6]	Traj Size:	$417,24 \pm 279,68$
	# of Traj.:	1,797
	# of Points:	346,657
	# of Users:	141
	Attributes:	Latitude, longitude, start time, weekday, type of user, and transportation mode.

Table 1: Summary of the trajectory datasets used in the experiments.

Figure 12 shows the running times comparison for each dataset considering MSM and MUITAS original implementations, using FTSM that indexes only the spatial dimension and MAT-Index, developed to consider all trajectory 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:

- $\underline{NYC^1}$ This scenario considers each semantic attribute as individual features, thus, it can be applied to both measures. In other words, we have 5 features f composing the set $\mathcal{F} = \{f_1, ..., f_5\}$, such that $f_1 = \{day\}$, $f_2 = \{price\}$, $f_3 = \{weather\}$, $f_4 = \{root\text{-}type\}$, and $f_5 = \{type\}$;
- $\underline{NYC^2}$ This scenario considers the *root-type* and the *type* attributes as semantically related, i.e., they are processed together in the same feature. In other words, we have 4 features f for 5 semantic attributes, which cannot be processed by MSM. Consequently, only the performances of MUITAS are plotted for this case. The set of features $\mathcal{F} = \{f_1, ..., f_4\}$ is such that $f_1 = \{day\}$, $f_2 = \{price\}$, $f_3 = \{weather\}$, $f_4 = \{\underline{root\text{-}type,\ type}\}$.

By observing the results in Figure 12, we can notice 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 can notice in the Figure 12 that, although FTSM efficiently indexes the spatial dimension in average cases, its integration to both MSM and MUITAS similarity measures for space, time and semantics overloaded the processing time

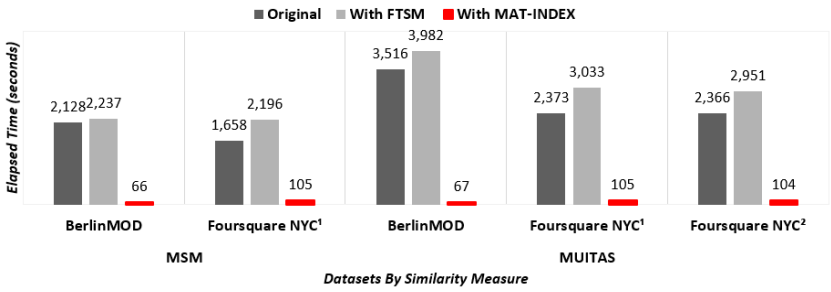

Figure 12: Similarity computation elapsed time by dataset

in all datasets. However, MAT-Index treats all dimensions in an integrated data structure; thus, the results are much better, reducing between 93.6% and 98.1% of the execution times, i.e., up to 52 times faster. 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 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 resulted in being better also 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 amount 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 1 to 5 attributes, resulting in 31 distinct possibilities. The three semantic attributes from the BerlinMOD dataset were analogously distributed, resulting in 7 other processing possibilities, totaling 38 scenarios. For instance, a possible scenario with 1 attribute for the Foursquare dataset can be only the price $\mathcal{F} = \{\{price\}\}$, only the weather $\mathcal{F} = \{\{weather\}\}$, etc.; with 2 attributes, we can process the price and weather $\mathcal{F} = \{\{price\}, \{weather\}\}$ or other pairs of semantic attributes, and so on.

Figure 13 shows, for each dataset, the elapsed time variability considering the number of processed attributes. We can notice that MAT-Index is stable and faster in all tested scenarios. The BerlinMOD dataset elapsed time varies between 63 and 67 seconds, and the Foursquare NYC between 76 and 105 seconds. The original implementation stays between 1,200 and 3,516 seconds for the BerlinMOD, and between 519 and 2,380 seconds for the 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 increase. This characteristic is

particularly important for multiple aspect trajectories since they tend to have a large number of attributes.

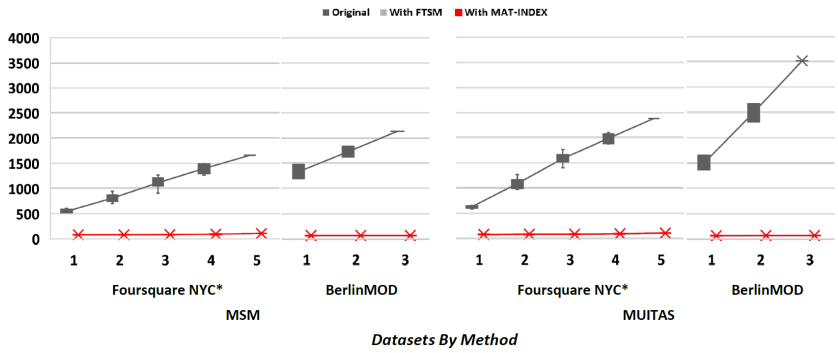

Figure 13: Elapsed Times By the Number of Semantic Attributes

In order to better understand how the variability of values affects the 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 $\mathcal{F} = \{\{price\}\}$ produces only five distinct price possibilities (-1, 1, 2, 3, and 4); the weather $\mathcal{F} = \{\{weather\}\}$ has 6 distinct possibilities (Clear, Clouds, Fog, Unknown, Rain, and Snow). However, $\mathcal{F} = \{\{price\}, \{weather\}\}$ has 29 distinct possibilities in the dataset (one less than if we individually combine price and weather values). The chart Figure 14 shows the number of composite keys sorted in ascending order, grouped by the number of processed attributes. We segregate both results by dataset – indicated at the top – to compare how the length influences the performance.

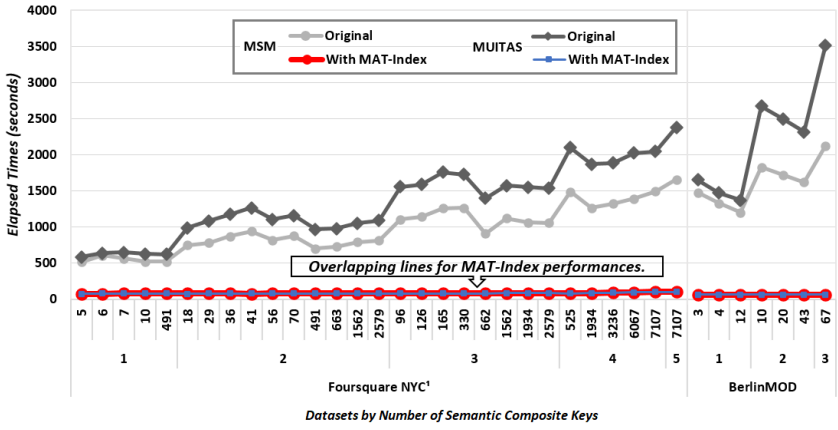

Figure 14: Elapsed Times By the Number of Composite Keys

We can notice how the dataset size and the number of attributes generate an explosion in the elapsed times for the original implementations. However, we also notice 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 fastly processed than the Foursquare dataset, even if they have the same number of attributes. In conclusion, MAT-Index successfully reduces the execution time in a percentage ranging from 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 in [7] with 200,000 points, having as attributes *latitude*, *longitude*, *time*, week*day*, *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 computational time spent by each similarity measure with and without MAT-Index support is reported in Figure 15.

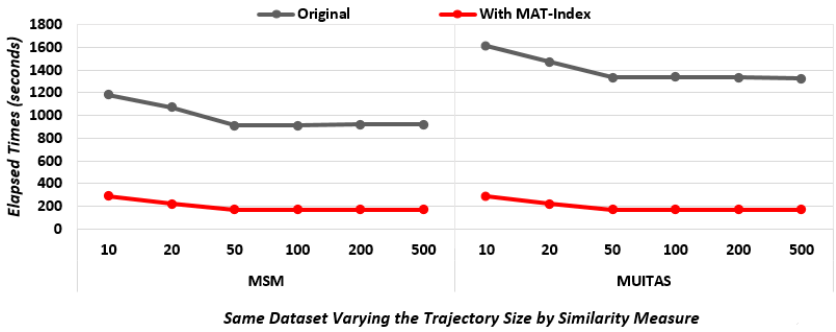

Figure 15: Elapsed Times Varying the Trajectory Size for the Same Content

It is worth noticing 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 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, the larger BerlinMOD dataset performed even faster than the Foursquare dataset. The larger a dataset is, the more it tends to present spatial, temporal and semantic data repetition. For instance, most attributes like weekdays, price categories, ratings, and even POIs are finite sets. Besides, according to the Principle of Pareto [25], 20% of the causes compass 80% of the problems, reinforcing the repetition tendency. Thus, MAT-Index perfectly fits the aims of this work that is to make faster the processing of similarity for larger datasets.

6 Conclusions and Future Work

Similarity measuring multidimensional data 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 intends to fill this gap by indexing the semantic content with multiple attributes, having spatial and temporal dimensions into a compact data structure, assuring 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 Multidimensional Similarity Measure and MUltIple aspect TrAjectory Similarity (MUITAS). On the one hand, we qualitatively show how the MAT-Index support for MSM and MUITAS drastically reduces the needed comparisons. On the other hand, we compute the index performance in running time exploiting two public datasets and scalability using one synthetic dataset. Experiments show an improvement of 98.1% in running time and 87.2% in scalability.

Future works include: (1) experiment MAT-Index with additional semantically enriched trajectories datasets; and (2) develop an efficient way to update the index without the need of fully 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 88887.310782/2018-00). 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 programme under Grant Agreement 777695 (MASTER[3]). The views and opinions expressed in this paper are the sole responsibility of the author and do not necessarily reflect the views of the European Commission.

References

[1] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[2] Stefan Büttcher, Charles LA Clarke, and Gordon V Cormack. *Information retrieval: Implementing and evaluating search engines*. Mit Press, 2016.

[3] http://www.master-project-h2020.eu/

[3] Lei Chen, M Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502. ACM, 2005.

[4] Xinyu Chen, Jiajie Xu, Rui Zhou, Pengpeng Zhao, Chengfei Liu, Junhua Fang, and Lei Zhao. S2r-tree: a pivot-based indexing structure for semantic-aware spatial keyword search. *GeoInformatica*, 07 2019.

[5] H. Ding, G. Trajcevski, and P. Scheuermann. Efficient similarity join of large sets of moving object trajectories. In *2008 15th International Symposium on Temporal Representation and Reasoning*, pages 79–87, 2008.

[6] Christian Düntgen, Thomas Behr, and Ralf Güting. Berlinmod: A benchmark for moving object databases. *VLDB J.*, 18:1335–1368, 12 2009.

[7] C. Ferrero, Lucas May Petry, L. O. Alvares, Camila Leite da Silva, W. Zalewski, and Vania Bogorny. Mastermovelets: discovering heterogeneous movelets for multiple aspect trajectory classification. *Data Mining and Knowledge Discovery*, 34:652 – 680, 2020.

[8] Carlos Andres Ferrero, Luis Otávio Alvares, and Vania Bogorny. Multiple aspect trajectory data analysis: research challenges and opportunities. In Cláudio E. C. Campelo and Laércio Massaru Namikawa, editors, *XVII Brazilian Symposium on Geoinformatics - GeoInfo 2016, Campos do Jordão, SP, Brazil, November 27-30, 2016*, pages 56–67. MCTIC/INPE, 2016.

[9] Raphael Finkel and Jon Bentley. Quad trees: A data structure for retrieval on composite keys. *Acta Inf.*, 4:1–9, 03 1974.

[10] Andre Furtado, Luis Alvares, Nikos Pelekis, Yannis Theodoridis, and Vania Bogorny. Unveiling movement uncertainty for robust trajectory similarity analysis. *International Journal of Geographical Information Science*, 32:1–29, 09 2017.

[11] Andre Salvaro Furtado, Luis Otavio Campos Alvares, Nikos Pelekis, Yannis Theodoridis, and Vania Bogorny. Unveiling movement uncertainty for robust trajectory similarity analysis. *Int. J. Geogr. Inf. Sci.*, 32(1):140–168, January 2018.

[12] Andre Salvaro Furtado, Despina Kopanaki, Luis Otavio Alvares, and Vania Bogorny. Multidimensional similarity measuring for semantic trajectories. *Transactions in GIS*, 20(2):280–298, 2016.

[13] Yuxing Han, Liping Wang, Ying Zhang, Wenjie Zhang, and Xuemin Lin. Spatial keyword range search on trajectories. In Matthias Renz, Cyrus Shahabi, Xiaofang Zhou, and Muhammad Aamir Cheema, editors, *Database Systems for Advanced Applications*, pages 223–240, Cham, 2015. Springer International Publishing.

[14] H. Issa and M. L. Damiani. Efficient access to temporally overlaying spatial and textual trajectories. In *2016 17th IEEE International Conference on Mobile Data Management (MDM)*, volume 1, pages 262–271, 2016.

[15] Hye-Young Kang, Joon-Seok Kim, and Ki-Joune Li. Similarity measures for trajectory of moving objects in cellular space. In *Proceedings of the 2009 ACM symposium on Applied Computing*, pages 1325–1330. ACM, 2009.

[16] Andre Luis Lehmann, Luis Otavio Alvares, and Vania Bogorny. Smsm: A similarity measure for trajectory stops and moves. *International Journal of Geographical Information Science*, 2019.

[17] Huiwen Liu, Jiajie Xu, Kai Zheng, Chengfei Liu, Lan Du, and Xian Wu. Semantic-aware query processing for activity trajectories. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, WSDM '17, page 283–292, New York, NY, USA, 2017. Association for Computing Machinery.

[18] A. Magdy, M. F. Mokbel, S. Elnikety, S. Nath, and Y. He. Mercury: A memory-constrained spatio-temporal real-time search on microblogs. In *2014 IEEE 30th International Conference on Data Engineering*, pages 172–183, 2014.

[19] Amr Magdy, Ahmed M. Aly, Mohamed F. Mokbel, Sameh Elnikety, Yuxiong He, Suman Nath, and Walid G. Aref. Geotrend: Spatial trending queries on real-time microblogs. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPACIAL '16, pages 1–10, New York, NY, USA, 10 2016. Association for Computing Machinery.

[20] Ahmed Mahmood, Sri Punni, and Walid Aref. Spatio-temporal access methods: a survey (2010 - 2017). *GeoInformatica*, 10 2018.

[21] Donald Meagher. Octree encoding: a new technique for the representation, manipulation and display of arbitrary 3-d objects by computer. rensselaer polytechnic institute. *Image Processing Laboratory*, 1980.

[22] Paras Mehta, Dimitrios Skoutas, and Agnès Voisard. Spatio-temporal keyword queries for moving objects. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '15, New York, NY, USA, 2015. Association for Computing Machinery.

[23] Ronaldo dos Santos Mello, Vania Bogorny, Luis Otavio Alvares, Luiz Henrique Zambom Santana, Carlos Andres Ferrero, Angelo Augusto Frozza, Geomar Andre Schreiner, and Chiara Renso. Master: A multiple aspect view on trajectories. *Transactions in GIS*, 2019.

[24] Guy M Morton. *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. International Business Machines Company, 1966.

[25] Vilfredo Pareto. Manuale di economica politica, societa editrice libraria. *Manual of political economy*, 1971, 1906.

[26] Lucas May Petry, Carlos Andres Ferrero, Luis Otavio Alvares, Chiara Renso, and Vania Bogorny. Towards semantic-aware multiple-aspect trajectory similarity measuring. *Transactions in GIS*, 23:960–975, 2019.

[27] A. Skovsgaard, D. Sidlauskas, and C. S. Jensen. Scalable top-k spatio-temporal term querying. In *2014 IEEE 30th International Conference on Data Engineering*, pages 148–159, 2014.

[28] Stefano Spaccapietra, Christine Parent, Maria Luisa Damiani, Jose Antonio de Macedo, Fabio Porto, and Christelle Vangenot. A conceptual view on trajectories. *Data & knowledge engineering*, 2008.

[29] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *Proceedings 18th international conference on data engineering*, pages 673–684. IEEE, 2002.

[30] S. Wang, Z. Bao, J. S. Culpepper, T. Sellis, M. Sanderson, and X. Qin. Answering top-k exemplar trajectory queries. In *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, pages 597–608, 2017.

[31] Xiangye Xiao, Yu Zheng, Qiong Luo, and Xing Xie. Inferring social ties between users with human location history. *Journal of Ambient Intelligence and Humanized Computing*, 5(1):3–19, December 2012.

[32] Dingqi Yang, Daqing Zhang, Vincent. W. Zheng, and Zhiyong Yu. Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 45(1):129–142, 2015.

[33] Josh Jia-Ching Ying, Eric Hsueh-Chan Lu, Wang-Chien Lee, Tz-Chiao Weng, and Vincent S Tseng. Mining user similarity from semantic trajectories. In *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks*, pages 19–26. ACM, 2010.

[34] B. Zheng, N. J. Yuan, K. Zheng, X. Xie, S. Sadiq, and X. Zhou. Approximate keyword search in semantic trajectory database. In *2015 IEEE 31st International Conference on Data Engineering*, pages 975–986, 2015.

[35] K. Zheng, S. Shang, N. J. Yuan, and Y. Yang. Towards efficient search for activity trajectories. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 230–241, 2013.