UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ana Paula Ramos de Souza

MAT-Index: an index for fast multiple aspect trajectory similarity measuring

Florianópolis
2021

Ana Paula Ramos de Souza

MAT-Index: an index for fast multiple aspect trajectory similarity measuring

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do título de mestre em Ciência da Computação.
Orientadora: Prof. Vânia Bogorny, Dra.

Florianópolis
2021

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Souza, Ana Paula Ramos de
MAT-Index: an index for fast multiple aspect trajectory similarity measuring / Ana Paula Ramos de Souza ; orientadora, Vânia Bogorny, 2021.
77 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Ciência da Computação, Florianópolis, 2021.

Inclui referências.

1. Ciência da Computação. 2. Indexação. 3. Medida de Similaridade. 4. Trajetória de Múltiplos Aspectos. I. Bogorny, Vânia. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Ana Paula Ramos de Souza

MAT-Index: an index for fast multiple aspect trajectory similarity measuring

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Jugurta Lisboa Filho, Dr.
Universidade Federal de Viçosa

Prof. Carina Friedrich Dorneles, Dra.
Universidade Federal de Santa Catarina

Prof. Ronaldo dos Santos Mello, Dr.
Universidade Federal de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Ciência da Computação.

GOV.BR
Documento assinado digitalmente
PATRICIA DELLA MEA PLENTZ
Data: 23/12/2021 12:05:30-0300
CPF: 959.243.700-97
Verifique as assinaturas em https://v.ufsc.br

Prof. Patricia Della Méa Plentz, Dra.
Coordenadora do Programa

ICP-Edu
Documento assinado digitalmente
Vania Bogorny
Data: 21/12/2021 15:07:11-0300
CPF: 684.586.570-15
Verifique as assinaturas em https://v.ufsc.br

Prof. Vânia Bogorny, Dra.
Orientadora

Florianópolis, 2021.

A UM AUSENTE

Tenho razão de sentir saudade,
tenho razão de te acusar.
Houve um pacto implícito que rompeste
e sem te despedires foste embora.
Detonaste o pacto.
Detonaste a vida geral, a comum aquiescência
de viver e explorar os rumos de obscuridade
sem prazo sem consulta sem provocação
até o limite das folhas caídas na hora de cair.

Antecipaste a hora.
Teu ponteiro enlouqueceu, enlouquecendo nossas horas.
Que poderias ter feito de mais grave
do que o ato sem continuação, o ato em si,
o ato que não ousamos nem sabemos ousar
porque depois dele não há nada?

Tenho razão para sentir saudade de ti,
de nossa convivência em falas camaradas,
simples apertar de mãos, nem isso, voz
modulando sílabas conhecidas e banais
que eram sempre certeza e segurança.

Sim, tenho saudades.
Sim, acuso-te porque fizeste
o não previsto nas leis da amizade e da natureza
nem nos deixaste sequer o direito de indagar
porque o fizeste, porque te foste.

Carlos Drummond de Andrade , "Farewell".

À Thaís dos Santos Chiappetta Nogueira Salgado e Filozinha.
(in memorian)

ACKNOWLEDGEMENTS

I thank my parents for giving me their best with a huge sacrifice, allowing me to reach this moment in my life. Father, thank you for not letting me give up although I know it was very difficult and lonely for both of us. Professor Vania Bogorny, my advisor, I thank you for the opportunity and for demanding me a better work in every detail. Chiara Renso and Raffaele Perego, I have no words to describe how grateful I am for the full-time precious advice. This work would not be possible without you too.

I especially thank Ramiro Affonso de Tadeu Guerreiro, my friend and former boss at IBM, for encouraging me in the first place. I am a huge fan of you: personally, professionally, and academically. Yuri Nassar, Areli Andréia dos Santos and Vanessa Lago Machado (not necessarily in this order): my gratitude for the endless support, patience, empathy, and excellent opinions. Tarlis Portella and Osmar de Oliveira Braz Jr, thank you for the partnership. Viviane Duarte and Jéssica dos Santos Pereira, thank you for believing in me more than I do. It was a very important incentive for what seemed to be a very long and arduous journey.

Finally, I can not forget my most precious friends from CEFET/RJ that made me much better as a person and student. I dedicate this title to you, Thaís dos Santos Chiappetta Nogueira Salgado (in memorian), Carlos Augusto Domingues Zarro (Seba), and Yolanriva Marques de Cavalcanti Campos. Life separated us, but my thoughts and my heart will always be with you.

At last, this work has been partially supported by the Brazilian agencies CAPES (Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Finance Code 001), CNPQ (Conselho Nacional de Desenvolvimento Científico e Tecnológico), FAPESC (Fundação de Amparo a Pesquisa e Inovação do Estado de Santa Catarina - Project Match - Co-financing of H2020 Projects - Grant 2018TR 1266), and the European Union‘s Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie GA N. 777695 (MASTER). The views and opinions expressed in this website are the sole responsibility of the author and do not necessarily reflect the views of the European Commission.

Measuring programming progress by
lines of code is like measuring aircraft
building progress by weight.
(Bill Gates)

RESUMO

O enriquecimento semântico dos dados de mobilidade com diversas fontes de informação levou a um novo tipo de dado móvel, conhecido como *trajetória de múltiplos aspectos*. Comparar trajetórias é crucial para diversas tarefas de análise como consultas, clusterização, similaridade, classificação etc. Medir a similaridade de trajetórias de múltiplos aspectos é uma tarefa complexa e computacionalmente custosa, pois o grande número de aspectos e a heterogeneidade das dimensões espaço, tempo, e semânticas requerem diferentes tratamentos. Apenas alguns trabalhos na literatura focam na otimização de todas essas dimensões em uma solução, e, ao melhor do nosso conhecimento, nenhum propõe uma comparação ponto-a-ponto otimizada. Nesta pesquisa, é proposto o Multiple Aspect Trajectory Index (MAT-Index), uma estrutura de dados indexada para otimização de comparações ponto-a-ponto de trajetórias de múltiplos aspectos, considerando suas três dimensões básicas de espaço, tempo e semânticas em uma estrutura integrada. Avaliações quantitativas mostraram a redução do tempo de execução em até 98.1%.

Palavras-chave: Indexação. Medida de Similaridade. Trajetória de Múltiplos Aspectos.

RESUMO ESTENDIDO

Introdução

Dados de trajetórias têm sido objeto de estudo por décadas. A primeira classe de trajetória estudada foi a trajetória bruta. Coletada por GPS, esse tipo de trajetória é composto apenas por dados espaço-temporais. Os avanços tecnológicos possibilitaram integrar a Internet a vários tipos de dispositivos, permitindo assim a coleta e enriquecimento de seus dados com conteúdo semântico que deu nome a uma nova classe, a trajetória semântica. Recentemente, Mello et al. (2019) introduziu o conceito de trajetórias de múltiplos aspectos. As trajetórias de múltiplos aspectos podem ser associadas a aspectos semânticos heterogêneos, de acordo com seu significado. Resumidamente, diferentes atributos compõem cada aspecto. Uma vez que esse tipo de trajetória pode conter uma infinidade de atributos, compará-los ponto a ponto degrada a performance em problemas maiores.

Ao longo do tempo, houve diversos trabalhos de método de acesso com diferentes propósitos, incluindo trajetórias. Apesar disso, apenas poucos focavam na otimização simultânea das três dimensões (espaço, tempo, e semântica) em questão. Até onde pudemos apurar, nenhum deles considera a complexidade que uma trajetória de múltiplos aspectos tem, restringindo-se à indexação de apenas poucas palavras-chave (geralmente uma). Em muitos casos, a indexação foca em combinações exatas. Assim, mesmo pequenas diferenças entre dois pontos comparados não retornariam o conteúdo a ser processado, sendo portanto inapropriado para esse fim.

Os trabalhos pesquisados caracterizam-se por requerer uma grande quantidade de dados redundantes a fim de viabilizar uma construção mais eficiente do método. Por vezes, essas soluções requerem alocações híbridas, que acabam por necessitar de mais acessos para alcançar o conteúdo requisitado. Ainda assim, os trabalhos encontrados não focam em problemas de comparação ponto a ponto. De fato, no estado da arte, não foi possível encontrar abordagens que indexem todos os aspectos juntos com o objetivo específico de processamento da similaridade. Assim, a seguinte pergunta de pesquisa define o objetivo principal deste trabalho: *É possível construir um índice eficiente para comparação ponto a ponto de trajetórias de múltiplos aspectos que acelere o cálculo da similaridade?*

Este trabalho apresenta uma solução para esta pergunta, propondo o Multiple Aspect Trajectory Index (MAT-Index), um índice para otimização da comparação entre trajetórias de múltiplos aspectos. O índice proposto é formado da combinação de índices invertidos, onde as chaves são nomes de atributos e os valores são os atributos de seus aspectos. A principal característica do MAT-Index é agrupar pontos de trajetórias contendo atributos iguais, a fim de evitar comparações redundantes e reduzir o montante de dados armazenados. Para determinar a eficiência do índice proposto foram realizados um exemplo de cálculo de similaridade utilizando o índice proposto e avaliações quantitativas das medidas de similaridade do estado da arte *Multidimensional Similarity Measure* (MSM) – um método popular para trajetórias semânticas que se aplica a trajetórias de múltiplos aspectos –, e *MUltIple aspect Trajectory Similarity* (MUITAS) — o primeiro trabalho originalmente desenhado para esse tipo de trajetória. O exemplo de cálculo de similaridade utilizando o MAT-Index mostrou como as medidas de similaridade indexadas podem se beneficiar da aplicação do MAT-Index, reduzindo drasticamente o número de comparações necessárias. Na avaliação quantitativa foram comparados os tempos de execução e performance de escalabilidade usando os datasets públicos Foursquare e BerlinMOD previamente enriquecidos com conteúdo semântico. Os resultados quantitivos das medidas de similaridade testadas apontaram ganhos significativos em todos os cenários, com até 98.1% de redução do tempo de processamento e até 87.2% nas avaliações de escalabilidade.

Objetivos
O objetivo principal desta pesquisa é construir um índice que dê suporte a medidas de similaridade para trajetórias semânticas e de múltiplos aspectos. Para tal, foram definidos os seguintes objetivos específicos:

- Compreender o estado-da-arte de trajetórias semânticas e índices de múltiplos aspectos, atentando às suas limitações e pontos fortes;
- Criar um algoritmo extensível capaz de indexar a comparação entre pontos de trajetórias, considerando sua dimensão semântica;
- Estender o algoritmo desenvolvido para a dimensão semântica a fim de indexar as dimensões espacial e temporal juntas;
- Divulgar os resultados da pesquisa.

Metodologia
A metodologia adotada consiste na realização de uma revisão da literatura com foco em publicações de alto impacto/Qualis. Utilizando mecanismos de busca como o Google Scholar, foram pesquisados tópicos relacionados a métodos de acesso a dados multidimensionais que indexassem espaço, tempo e texto, simultaneamente.
Com base nos resultados encontrados, identificou-se que poucos trabalhos focavam na otimização das três dimensões (espaço, tempo e semântica) concomitantemente. Constatou-se também que nenhum deles considera a complexidade intrínseca a uma trajetória semântica ou de múltiplos aspectos, restringindo-se apenas à consulta por poucas palavras-chave que requerem combinações exatas, inviabilizando aplicar esse tipo de solução para fins de cálculo de similaridade. Além disso, todos necessitavam manter grandes quantidades de dados redundantes para viabilizar a construção de um índice mais rápido, demandando, em alguns casos, armazenamento híbrido (usa as memórias principal e secundária), por sua vez, mais acessos para se alcançar o conteúdo desejado. Também não localizamos no estado-da-arte abordagens que indexassem todos os atributos conjuntamente com o objetivo específico de processamento da similaridade de trajetórias com vários atributos semânticos. Com base nas limitações elencadas, foi desenvolvido um índice para otimização de trajetórias contendo múltiplos atributos. A principal característica do índice proposto é agrupar pontos de trajetórias iguais, evitando comparações ponto a ponto redundantes. O índice desenhado possui uma estrutura compacta, requerendo menos espaço físico que o próprio arquivo original. A proposta foi idealizada de modo a atender às medidas de similaridade MSM e MUITAS. No entanto, o MAT-Index é potencialmente aplicável a outras medidas de similaridade do estado-da-arte que demandem comparação ponto a ponto.

Resultados e Discussão
Para determinar a eficiência da proposta, foi apresentado um exemplo de cálculo de similaridade comum às duas medidas testadas a fim de evidenciar como o processo foi simplificado; quantitativamente, foram comparados os tempos de execução do Multidimensional Similarity Measure (MSM) e do MUITAS para o processamento dos datasets Foursquare e BerlinMOD previamente enriquecidos, nas versões originais e com suportes do MAT-Index e do índice FTSM (Fast Trajectory Similarity Measure), originalmente proposto para aceleração do processamento da dimensão espacial de trajetórias semânticas.

Quantitativamente, o método proposto mostrou-se eficiente no processamento de trajetórias contendo as três dimensões, incluindo múltiplos atributos semânticos. Os resultados executados apontaram um ganho significativo no desempenho computacional em todos os cenários testados, chegando a executar até cinquenta e duas vezes mais rápido do que a mesma medida de similaridade sem o suporte do MAT-Index.

Considerações Finais

Os dados discutidos ao longo deste trabalho foram integralmente implementados e estão disponíveis em https://github.com/anapbr/MasterDegree/tree/PaperVersion. O MAT-Index provou-se um método muito eficiente no suporte ao processamento de medidas de similaridade que realizem comparação ponto a ponto a fim de obter os maiores *scores* (*best fit*) em cada caso. Contudo, o modo como o índice foi estruturado, permite que sejam obtidos os K pontos ou K trajetórias mais semelhantes a um determinado ponto consultado, um clássico problema de *K-Nearest Neighbor*. Também é possível aproveitar sua estrutura para propósitos de consultas, como em problemas de *range queries*. Assim, há uma gama de trabalhos em potencial que podem ser desenvolvidos a partir dessa pesquisa.

Palavras-chave: Indexação. Medida de Similaridade. Trajetória de Múltiplos Aspectos.

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 research, 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 in an integrated data structure. Quantitative evaluations show a processing time reduction of up to 98.1%.

Keywords: Indexing. Similarity Measure. Multiple Aspect Trajectory.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF TABLES

CONTENTS

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 (BOGORNY; BRAZ, 2012). 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* in 2016 (MELLO et al., 2019; FERRERO; ALVARES; BOGORNY, 2016).

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; suppose it is the trajectory of Peter, who wears a smartwatch and works at a smart office equipped with numerous sensors and microphones. Peter stays at *home* from *11 pm* to *8 am*. During the night, his watch collects his vital signs, including his *heart rate* and *sleeping stages*. Peter goes to *work* on *foot* when the weather is *sunny*. While he walks, he tweets a message that characterizes his *mood*. He stays at *work* from *8:30 am* to *6 pm*, where his workplace collects information about the environment like *noise*, *temperature*, *air pollution*, and *humidity*, as well as the *emotional status* of the staff. After *work*, the weather is *rainy*, so he decides to take a *taxi* to move to a *Japanese Restaurant* for dinner. The restaurant has its attributes as *open-close hours*, *price range*, spatial location, and *reviews*.

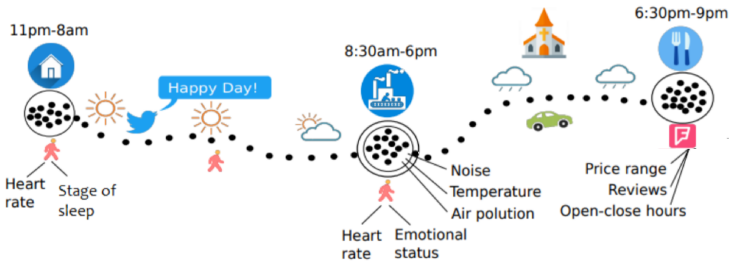

Figure 1 – Example of a Multiple Aspect Trajectory

Source: Mello et al. (2019)

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. The main challenge in trajectory data analysis is that it carries aspects that must be treated separately, as the spatial dimension. It is composed by two attributes, latitude

and longitude, and they must be considered together in order to represent a position in space. Latitude and Longitude cannot be considered as independent attributes.

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) (FURTADO et al., 2017), MSM (Multidimensional Similarity Measure) (FURTADO et al., 2016) and MUITAS (MUltIple aspect TrAjectory Similarity) (PETRY et al., 2019). 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 (BERNDT; CLIFFORD, 1994), LCSS (VLACHOS; KOLLIOS; GUNOPULOS, 2002), and EDR (Chen; Özsu; Oria, 2005). 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.

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

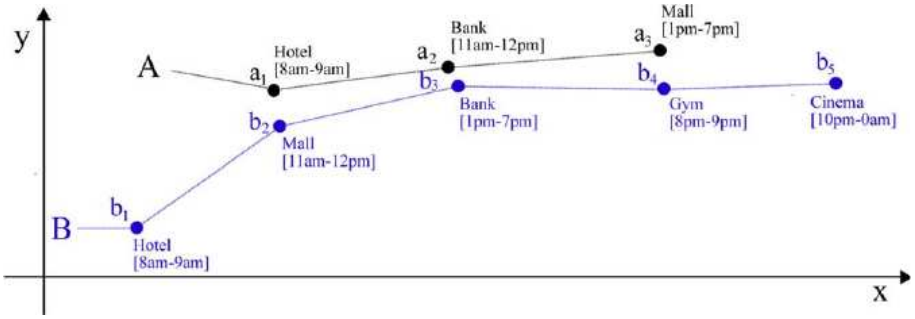

Figure 2 – Two Semantic Trajectories

Source: Furtado et al. (2016)

datasets.

In 2018, Furtado proposed FTSM (Fast Trajectory Similarity Measuring), an index for fast similarity measuring of UMS and MSM (FURTADO et al., 2018). 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 (MAHMOOD; PUNNI; 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 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 research 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 similarity running example shows how both similarity algorithms can benefit from this index and how MAT-Index can drastically reduce the number of comparisons. Indeed, a quantitative evaluation shows the running time and scalability performance over publicly available datasets enriched with semantic aspects. Comparing the performance results of the similarity algorithms with and without FTSM and MAT-Index supports, we show a gain for all tested scenarios up to 98.1% processing time reduction and up to 87.2% in scalability evaluations.

1.1 OBJECTIVES

The main objective of this research is to build an index that accelerates the similarity analysis of multiple aspect trajectories. To accomplish this purpose, the following specific objectives are defined:

- To better understand the state-of-the-art semantic trajectories and multiple aspect indexes, and well understand their limitations and strengths;
- Create an extensible index for accelerating the trajectory similarity analysis by reducing the point to point comparison;
- Propose an efficient algorithm for indexing the spatial, temporal and semantic dimensions, overcoming limitations of existing works;
- Disseminate research results by articles, presentations, reports, software, manuals, and databases for validating and publicizing solutions.

1.2 METHODOLOGY AND DISSERTATION STRUCTURE

This work applies the following methodology to accomplish the objectives proposed:

1. Perform an extensive literature review in access methods, by considering the works that index space, time, and semantic/textual data at once, and evaluate them under the following metrics: execution time, allocation method, the space required, capability to perform semantic trajectory similarity;
2. Implement the similarity measures that apply to multiple aspect trajectory: MSM, and MUITAS;
3. Define the datasets for evaluating the similarity measures implemented;
4. Design and implement an index data structure for multiple aspect trajectory similarity measuring for reducing the execution time in similarity measures, optimizing the allocation required;
5. Assess the scalability, allocation method and computational time spent of the proposed methods using real and synthetic trajectory datasets;
6. Compare the results with the original similarity measures performances, evaluating their behavior over the real trajectory datasets, under the main indexing metrics;
7. Write articles describing the method for reducing the problem;
8. Write the dissertation.

The rest of this document is structured as follows: Chapter 2 describes the concepts required to understand this work; Chapter 3 discusses related works, highlighting their limitations; Chapter 4 introduces the proposed index for multiple aspect trajectories; Chapter 5 discusses the evaluation results that assess work efficiency; finally, Chapter 6 concludes the research 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) (FURTADO et al., 2016) and the MUltIple aspect TrAjectory Similarity (PETRY et al., 2019), 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.

Definition 2.1.1 ***Multiple Aspect Trajectory*** *A Multiple Aspect Trajectory* $T = \langle p_1, p_2, ..., p_n \rangle$ *is a sequence of points, 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.*

Definition 2.1.2 ***Aspect*** *An Aspect* $a = (desc, a_{type})$ *is a relevant real-world fact for mobility data analysis. It is composed of a description (desc) and an aspect type.*

Definition 2.1.3 ***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.*

An instance of an aspect type, thus the aspect instance, is represented as a non-empty set of attribute-value pairs $a_{type_K} = \{att_1 : v_1, att_2 : v_2, ..., att_z : v_z\}$. Thus, each pair $(att_i : v_i)$ is an instantiation of a property att_i of a_{type_k} with a (atomic or multivalued) value v_i. For the sake of 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 either for sequential data as LCSS (VLACHOS; KOLLIOS; GUNOPULOS, 2002), EDR (Chen; Özsu; Oria, 2005), w-constrained Discrete Frechet distance (wDF) (Ding; Trajcevski; Scheuermann, 2008), or for trajectories as CVTI (KANG; KIM; LI, 2009), MSTP (YING et al., 2010), MTM (XIAO et al., 2012), MSM (FURTADO

et al., 2016), UMS (FURTADO et al., 2017), SMSM (LEHMANN; ALVARES; BOGORNY, 2019), and 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 focus on these measures for our indexing proposal.

2.2.1 Multidimensional Similarity Measure (MSM)

The 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 $q \in Q$. The method computes the parity of two trajectories, (P,Q) and (Q,P), using the maximum matching *score* Equation 2.1.

$$parity(P,Q) = \sum_{i=1}^{|P|} max\ score(p_i, Q) | q_i \in Q\} \tag{2.1}$$

The *score* is computed according to Equation 2.2, 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 (τ_k), and scores the matches (Equation 2.3) according to their respective weights (ω).

$$score(p,q) = \sum_{k=1}^{|A|} (match_k(p,q) * \omega_k) \tag{2.2}$$

$$match_k(p,q) = \begin{cases} 1, & \text{if } \ dist_k(p,q) \leq \tau_k \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

The total similarity of two trajectories is given in Equation 2.4, by summing both parities and dividing the result by the sum of both trajectory lengths (P and Q).

$$MSM(P,Q) = \begin{cases} 0, & \text{if } |P| = 0 \text{ or } |Q| = 0 \\ \frac{parity(P,Q) + parity(Q,P)}{|P| + |Q|}, & \text{otherwise} \end{cases} \tag{2.4}$$

2.2.2 MUltIple aspect TrAjectory Similarity (MUITAS)

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 named feature. In the similarity analysis it means that the set of attributes

defined as a feature have to match to be considered in the similarity score. For example, suppose two trajectory points have the attributes (or dimensions) space, time, POI name, price, rate and weather condition. The attributes price and rate refer to the POI, so the attributes space, time and weather condition could be all independent and POI name, price and rate could be considered as a feature. In this case, the POI name, price and rate must be exactly the same of the other trajectory point to have a match. 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 2.5 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 (τ).

$$match_{f_i}(p,q) = \begin{cases} 1, & \text{if} \quad \forall a_j \in f_i, dist_j(p,q) \leq \tau_j \\ 0, & \text{otherwise} \end{cases} \tag{2.5}$$

For each feature, Equation 2.6 computes the score as the weighted sum (ω) of matching points.

$$score(p,q) = \sum_{i=1}^{|\mathscr{F}|} (match_{f_i}(p,q)) * \omega_i \tag{2.6}$$

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 2.7), and the $parity(Q,P)$.

$$parity(P,Q) = \sum_{k=1}^{|P|} max\ score(p_k, Q) | \forall q \in Q\} \tag{2.7}$$

The final similarity score between two trajectories is the sum of parities, divided by the sum of the trajectory lengths (Equation 2.8).

$$MUITAS(P,Q) = \begin{cases} 0, & \text{if } |P| = 0 \text{ or } |Q| = 0 \\ \frac{parity(P,Q) + parity(Q,P)}{|P| + |Q|}, & \text{otherwise} \end{cases} \tag{2.8}$$

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

An index is a data structure defined to significantly speed up data retrieval operations (BÜTTCHER; CLARKE; CORMACK, 2016). It may point to a physical or a logical location, where the physical points to the actual disk location while the logical holds an address that points to a physical address on the disk.

Indexes are used by all types of software, including the operating system, Database Management System (DBMS) using key data fields in a database record, resources based on file names, text within a file or unique attributes in a graphics or video file, etc. For example, the file system index in an operating system contains an entry for each file name and the starting location of the file on disk. A database index has an entry for each key field (account number, name, etc.) and the location of the record. Search engines use a very sophisticated indexing system to keep track of billions of pages on the Web.

There are several indexes developed specifically for spatial data. Examples are the Quad-tree (FINKEL; BENTLEY, 1974), Z-Ordering Tree (MORTON, 1966), and (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 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*) (BÜTTCHER; CLARKE; CORMACK, 2016). 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	
REF	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	Value
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 various 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

An extensive list of access methods for trajectories is organized in (MAHMOOD; PUNNI; AREF, 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 text with space and/or time data.

However, no works index data focusing on similarity measuring purposes. Indexing only part of the data like in both top-k and range queries solutions is unfeasible. In this case, or it would lead to approximate solutions then possibly propagating errors, or it would require getting original data when the content is not indexed. So, the multiple access and the naive comparisons comprise the performance. The related state-of-the-art works found along with the research are following briefly described:

The **Grid index for Activity Trajectories** (GAT) (Zheng et al., 2013) addresses the Activity Trajectory Similarity Query (ATSQ). An activity trajectory is composed of semantic keywords, analogous to a semantic trajectory. GAT hierarchically divides the entire spatial region into quad cells, using a space-filling curve to assign the multidimensional cells into a 1-dimensional domain. It implements four data structures: *(i)* The *Hierarchical Inverted Cell List (HICL)* is a multi-level grid data structure that, for each activity, builds a tree using the spatial references as nodes and leaves. Because of the high allocation demand of this structure, only the upper levels of the tree are in the main memory, and lower levels are stored in the secondary memory; *(ii)* The *Inverted Trajectory List (ITL)* stores each cell of the HICL in an inverted index that stores the lists of activities belonging to them and the trajectories where they happen. This structure does not keep detailed information about each trajectory individual point. Hence, ITL can be stored within the main memory in most cases; *(iii)* The *Trajectory Activity Sketch (TAS)* stores in the main memory, for each trajectory, the trajectory activities that are assigned to continuous numerical IDs. Then, the activities are summarized, sorted by the frequency and partitioned into intervals. The idea of this data structure is to quickly filter out trajectories that do not match the query requirements, preventing retrieval from the disk unnecessary detailed information for the query goal; *(iv) Activity Posting List (APL)* allocates, for each trajectory, the activities belonging to them and the trajectory points that has the activity. This data structure is stored on disk due to its high space requirement and will be retrieved only when the distance with the query needs to be evaluated.

The **Adaptive Frequent Item Aggregator** (AFIA) (Skovsgaard; Sidlauskas; Jensen, 2014) identifies the top-k frequent terms (keywords), in a specific spatio-temporal range. This index creates multiple spatial grids with different granularities to partite the space. For each grid granularity and each cell into it, the method stores a summary of the most frequent terms in that cell. For temporal support, it creates new instances of grid cells periodically. It also creates temporal cells at multiple time-granularities. Each grid cell maintains frequencies of terms for all supported temporal granularities, e.g., hour, day, week, and month. To process a query with a specific Spatio-temporal range, it partitions the query range into several coarser regions. The

aggregates from these regions are combined to get the final top-k result. Indeed, this index changes the size of the summaries dynamically to adapt to changes in the number of frequent terms within grid cells. The AFIA approach seeks a solution capable of supporting streams with higher rates than Twitter currently sees (5,000 tweets/sec). It focuses on semantics, besides is not ready to similarity measures as it is not the focus of this work.

Mercury (Magdy et al., 2014) uses a partial in-memory pyramid approach to support top-k spatio-temporal and textual queries over microblogs. The pyramid structure is a multi-level partitioning of the indexed space. In the spatial pyramid, each cell level is partitioned into four equal cells in the subsequent level. Each pyramid cell maintains a list of the microblogs that have arrived in the spatial range of the corresponding cell during the past time units. The index orders the microblogs within a cell according to their arrival timestamps. Microblogs are periodically bulk-inserted into Mercury using a main-memory buffer to reduce the insertion overhead. It periodically checks the content spans of the pyramid cells to split or delete levels according to the number of quadrants, avoiding an extremely deep data structure. The microblogs score according to a ranking function of the spatial microblog spatial proximity of the query and the time recency of the microblog. The index also uses a pyramid cell priority queue to visit them according to a ranking function. It depends on the spatial proximity between the cell and the query location, and the most recent microblog timestamp in every pyramid cell. Also, during query processing, a list of the top-k microblogs is maintained. This list is sorted in the order of the scores of the microblogs. It gets updated as the pyramid cells are being visited.

The **Grid index Keyword index** (GiKi) (Zheng et al., 2015) answers the Approximate Keyword Query of Semantic Trajectories (AKQST). An AKQST is a set of keywords required to retrieve the k most relevant semantic trajectories or sub-trajectories required to cover all the query keywords (on approximate keyword-matching, e.g., to tolerate any misspelled keywords) and have the shortest travel distance. GiKi consists of a Semantic Quad-tree (SQ-tree) and a Keyword-Reference Index (K-Ref). It implements a two-step function to define the relevance of the trajectories: in the first step, it aggregates the trajectory travel distances; And, in the second step, it calculates the similarity between the trajectory keywords and the query keywords. The SQ-Tree construction basis on a multi-level grid-partitioning of the indexed activity-trajectories. To build the spatial quad-tree within an SQ-tree, it uses Grid cells that overlap the trajectories. A non-leaf node in the SQ-tree that contains an identifier of the corresponding grid cell in the multi-level grid, the pointers to children nodes of the quad-tree, and a signature of all the keywords covered within the corresponding grid cell in the multi-level grid. The signature of keywords is a MinHash of all the keywords covered by the Quad-Tree node. It is possible to calculate MinHash keywords signatures by generating the multiple hash functions over all the n-grams keywords covered by a Quad-Tree node. A leaf node in the SQ-Tree contains the keyword signature and pointers to the indexed trajectories. K-Ref is a textual index maintained per trajectory to speed up the string edit-distance computation. The K-Ref uses the K-Means Clustering to identify the keyword clusters per trajectory. It chooses as keywords clusters, for every cluster, a representative reference keyword. Then, keywords of a trajectory are indexed

based on their distance to the reference keywords using a B+-Tree. AKQST uses the SQ-tree and K-Ref to identify candidate trajectories that have keywords similar to the query keywords. Then, to identify the k most-relevant trajectories, the relevance of the candidate trajectories is calculated.

The **Grid and KR*-Tree** (GKR) has a hybrid structure that combines the SETI access method to index the moving object trajectories and a KR*-Tree to index the spatial-textual objects. It uses a grid to partition the space into disjointed cells of the same size. The partitions store the trajectory segments contained into the corresponding space. One or more disk pages store the resultant segments. Each page contains the sets of segment keywords of a trajectory. Finally, the pages are indexed through a KR*-Tree, whose structure associates index nodes to keywords and organizes the pages of the disk accordingly to their temporal proprieties. To proceed to the query, suppose a spatio-temporal, and textual trajectory with a set of keywords. First, it finds the candidate grid-cells that overlap the queried spatial interval. Then, for the corresponding KR*-Trees of the candidate grid-cells traversed, it finds the nodes with timestamp overlaps that contain a keyword from the queried set. The corresponding pages disk of these nodes is further filtered in two steps to discard false-positive trajectory segments in the spatio-temporal dimensions, and to remove trajectory segments that do not fully cover the set of query keywords.

The **Inverted OC-tree** (IOC-tree) Han et al. (2015) answers Spatio-temporal and textual filter queries on trajectories. An inverted index basis the IOC-tree, where query processing performs by filtering the indexed data using a keyword-first strategy. In the IOC-Tree, each keyword has an Octree built by recursively dividing the Spatio-temporal space into eight nodes. Leaf nodes are encoded using the 3D Morton code, where disk stores the non-empty leaf nodes in a one-dimensional structure ordered by Morton codes. An octree node also maintains a signature that contains a summary of the trajectory information within that node. This signature filters out the non-qualifying nodes and the signature gets updated during the insertion/deletion of trajectories. The disk stores exact trajectory information per non-leaf nodes. Query processing is performed by dividing the nodes into three types of Spatio-temporal constraint return: Nodes that do not satisfy the constraint; Nodes that partially covered range constraint; and Nodes that fully cover the query range. After the signature, a test is performed to filter out the non-qualifying nodes. Candidate trajectories are loaded from disk and validated to get the final result.

The **IR-With-Identifiers** (IRWI-Tree) (Issa; Damiani, 2016) is a hybrid index framework for spatial-textual trajectories defined over continuous temporal domains. Similarly to an IR-tree, the IRWI data structure consists of an R-tree augmented with inverted lists that associate each label with a compressed representation of a corresponding trajectory identifier. The IRWI-tree internal nodes contain summaries of the trajectory units in the leaf level, which entries are represented as trajectory units u = [I, L, Seg], for I the interval, L the textual content, and Seg the segmented spatial location. The IRWI-Tree efficiently answers the sequenced queries over trajectories by splitting the original ones into multiple simple queries processed in

parallel. The result set contains only the trajectories that satisfy all simple queries in the proper order. The indexing techniques were implemented in Secondo's development environment.

The **GeoTrend** index (Magdy et al., 2016) applies for real-time microblogs to find out the trends in arbitrary spatial regions with the support of trending measures. GeoTrend is an access method for identifying the trending keywords within recent microblogs in a specific spatial region. GeoTrend adopts a hybrid Spatio-temporal and textual data structure that builds on the incomplete pyramid structure for spatial indexing. Every cell in the pyramid maintains a textual index. The textual index is a hash table that stores keywords aggregate statistics in the microblogs over the past time-period. The length of the time duration T depends on the availability of the main memory. The keyword aggregate statistics k is a set of N counters. Each counter stores the number of microblogs containing Keyword k for a partial time-interval of length NT. GeoTrend uses an expiration technique to evict the obsolete aggregates. When the index is under high workload, GeoTrend adopts a load-shedding technique that evicts the less-important aggregates that are less likely to contribute to any query answer. The main difference between GeoTrend and Mercury is that Mercury searches for individual microblogs. At the same time, GeoTrend uses aggregates over microblogs to identify the trending keywords.

The **Fast Trajectory Similarity Measuring** (FTSM) (FURTADO et al., 2018) is a branch and bound strategy that focus on indexing the spatial dimension for fast computing the MSM. It consists of pruning out the trajectory points that exceed the similarity threshold plus the length of the trajectory segment analyzed, preventing a point-to-point comparison of trajectory segments most distant. FTSM starts considering the entire trajectory as a segment and then the trajectory is continually divided by the middle, repeating the punning process until the segment corresponds to a trajectory point. This method can reach a quadratic cost in the worst cases where there are no points eliminated during the process. This method was used in Chapter 5 in comparison to the MAT-Index. Although FTSM performed efficiently in (FURTADO et al., 2018) for spatial indexing, the method requires a threshold to calculate and prevent comparisons. Since semantics in general does not admit a threshold as a match condition, the method can not be adapted. Thus, we decided to execute the original FTSM proposal where the index provided a list of matches to the similarity measures that only computed the semantic and temporal dimensions to calculate the final score. However, MSM and MUITAS performances do not improve when using FTSM for computing all trajectory dimensions. The reason is that first the FTSM is executed, computing and storing the pairs of trajectory points that match. Then, MSM computes semantics and time dimensions, besides always retrieve the list of matches for all trajectory points, no matter it matches or not. In other words, FTSM computes the entire dataset in until quadratic cost, storing the results in an array of matches, then MSM equally computes the similarity in quadratic cost, only preventing the spatial comparison itself, but demanding to retrieve the FTSM list of matches. This overhead explains the results presented in Subsection 5.2.1.

R-trees with STLs (AHMED et al., 2017) is a disk-based index that provides exact answers to the top-k Frequent Spatio-Temporal query (the kFST query, for short). This query

identifies the most frequent terms in a specific Spatio-temporal range. This index extends the nodes of a multi-dimensional R-tree with sorted terms lists (STL, for short). An STL of an R-tree node, say N, is a list that contains the frequencies of terms of the objects covered by Node N. This list is sorted based on the frequencies of terms. The STLs are added to both leaf and internal nodes of the underlying R-tree to improve the query performance. To reduce the memory overhead of the index, STLs store the frequencies of the most frequent terms estimated analytically.

Liu et al. (2017) proposed the **ST-Tree** answers semantic-aware similarity queries on activity trajectories. Instead of adopting exact or approximate keyword matching, the semantic-aware similarity query considers the semantic similarity between the keywords representing the activities of the trajectories. This query attempts to identify the k most-relevant activity-trajectories to a specific set of keywords and spatial locations. A function defines relevance as the spatial proximity between the locations of the trajectories and the locations of the query; and the semantic similarity between the keywords representing the trajectories' activities and the keywords of the query. To measure the semantic similarity of the keywords, Latent Dirichlet Allocation (LDA) is used to map the keywords of activities into a high-dimensional vector that represents the semantics of the keywords. The ST-tree integrates the quad-tree with Locality Sensitive Hashing (LSH). LSH is used to reduce the dimensions of the LDA representation and to ensure that relevant activity trajectories are assigned to the same bucket with high probability. In the ST-tree, activity trajectories are first indexed using a quad-tree. Every leaf node in the quad-tree points to an LSH structure for each trajectory point indexed by this leaf node. Query processing in the ST-tree uses the quad-tree to find the candidate trajectories that are close to the location of the query. Then, LSH is probed to identify the semantically similar activity-trajectories.

Bubble Bucket Tree (BB-Tree) (Sprenger; Schäfer; Leser, 2019) is a spatial index structure for processing multidimensional workloads in main memory. A BB-Tree consists of two components: A k-ary search tree for pruning and searching and a set of Bubble Buckets. Inner nodes of the IST recursively split the data space into k disjoint subsets according to a delimiter dimension and k-1 values. Data are kept in regular BB, which hold up to a predefined limit of m-dimensional data objects, and can dynamically expand to cope with a larger number of objects. The idea is to linearize the inner search tree and manages it in a cache-optimized array, with optimized re-organizations when data changes. When queried, inner nodes are used to reduce the data space. Once, all irrelevant subtrees have been pruned, the remaining BB are scanned to determine the results.

Chen et al. (2020) proposed the hybrid structure **S^2R-Tree**, which integrates not only spatial and semantic information in a transparently, but also represents semantic information by coordinates based on low dimensions pivot. The existing indexing and search methods have a limited removal effect due to the high dimensionality in the semantic space, making query efficiency a severe problem. The spatial mechanism transforms high-dimensional semantic vectors into a small-sized space. So, instead of indexing objects in the original semantic space, the di-

mension reduction allowed a more effective pruning effect. The method uses pivot-based principles for space transformation and partitioning. Hence, the high-dimensional semantic vectors can be mapped to low-dimensional coordinates with high data variation. The S^2R-Tree hybrid indexing structure seamlessly integrates spatial and semantic information, but also represents semantic information by pivot-based coordinates in low dimensions. That way, the pruning effect can be significantly improved. In addition, Chen et al. (2020) developed an efficient and accurate SKQ (Spatial Keyword Queries) processing algorithm at the top of the S^2R-tree. This algorithm can significantly reduce the space for large-scale research. To calculate the distance between an object and a query, euclidean distance, and normalization using a sigmoid function.

Time-weighted Term List (TwTL) (Dam et al., 2021) employs a spatial index using a quad-tree data structure to index posts augmented by top-k time-weighted term lists (TwTL) and a bulk updating technique to support the fast digestion of social post streams. The method adopts a location-based time-decaying query technique to retrieve recently frequent terms within a user-specified region of interest. The decaying time-weighted frequency term is based on an exponentially higher score to terms posted most recently. These top-k term lists are employed in the aggregation step to produce the final results so the incoming queries can be efficiently processed.

3.1 TYPES OF INDEX APPROACHES

Regarding the works that process space, time, and semantic data, we briefly discuss their limitations, explaining why each one does not fit our goal to make point-to-point trajectory similarity faster.

3.1.1 Retrieving Approaches

We can retrieve information by using exact and partial querying approaches according to what we want to accomplish. The exact approaches (Skovsgaard; Sidlauskas; Jensen, 2014; Han et al., 2015; Magdy et al., 2016) retrieve only information containing all queried data. This condition allows us to apply pruning strategies that are very useful for speeding up the search. However, pruning the content does not let us acquire cases that do not entirely coincide. In other words, we can only determine the aspects that fully match, not being possible to weigh how similar two elements not retrieved are. Instead, the partial match approaches (Zheng et al., 2013; Magdy et al., 2014; Zheng et al., 2015; Mehta; Skoutas; Voisard, 2015; Issa; Damiani, 2016; LIU et al., 2017; Wang et al., 2017; AHMED et al., 2017; Chen et al., 2020) allow us to distinguish two compared characteristics independently of having something in common, not restricting access to the whole dataset, which can work for similarity purposes.

3.1.2 Allocation Approaches

Indexes can keep different and redundant data structures connected in a try to optimize the access method. However, depending on the amount of memory allocated, this approach can force the operating system to excessively transfer data between memory levels, delaying the access in a known problem named *Thrashing*. In the state-of-the-art, we observed three allocation ways of how related works process the data: disk, memory or hybrid allocations. Works that adopt disk solutions (Issa; Damiani, 2016; AHMED et al., 2017) avoid the system collapse since the disk space is less limited than the memory. However, the transference between primary and secondary memory levels demands a slower device to access and transfer data. This bottleneck can significantly impact the performance in cases that require constant access. Point-to-point comparison is a quadratic problem with high access demand, thus highly affected by disk solutions. Due to the mentioned issues related to the disk latency, some works like (Zheng et al., 2013; Zheng et al., 2015; Han et al., 2015; LIU et al., 2017) adopts hybrid strategies. Although this approach is less slower than the pure disk allocation methods, it still requires multiple accesses to retrieve data from disk and thus locate it in the RAM memory before to start the processing. Thus, disk and hybrid allocation are inefficient for large and complex data, including trajectories datasets. Another approach to minimize the processing and allocation workload is to develop approximate solutions like in Skovsgaard, Sidlauskas e Jensen (2014), Zheng et al. (2015). For similarity search purposes, the approximation would propagate a possible error to all other related comparisons, affecting the reliability of the score. Finally, (Skovsgaard; Sidlauskas; Jensen, 2014; Magdy et al., 2014; Mehta; Skoutas; Voisard, 2015; Magdy et al., 2016; Wang et al., 2017; Chen et al., 2020; Dam et al., 2021) proposes indexes using memory allocation solutions not comprising the performance.

3.1.3 Pruning Strategies

A common strategy to accelerate the processing is to reduce the set of elements to compare. Problems that demand the k most similar elements (TOP-K) allow us to design access methods (Zheng et al., 2013; Skovsgaard; Sidlauskas; Jensen, 2014; Magdy et al., 2014; Zheng et al., 2015; Magdy et al., 2016; AHMED et al., 2017; Wang et al., 2017; LIU et al., 2017; Dam et al., 2021) that adopt pruning strategies avoiding comparisons that never lead to a potential response. Another pruning strategy is to design the index building focusing on spatial (Magdy et al., 2016; Dam et al., 2021), temporal (Issa; Damiani, 2016; Sprenger; Schäfer; Leser, 2019; Chen et al., 2020), or spatio-temporal (Skovsgaard; Sidlauskas; Jensen, 2014; Magdy et al., 2014; Mehta; Skoutas; Voisard, 2015; Han et al., 2015; Dam et al., 2021) ranges, equally avoiding comparisons in cases that never lead to a match. However, both pruning techniques intend to accelerate only the targeted cases. Comparing elements that are not in the same range equally demands brute force processing. The multiple aspect trajectory comprises three dimensions, including heterogeneous data. Access methods that apply to space, time, and textual data

are indexed using different linked data structures, never consolidating them as one. MSM and MUITAS similarity measures pairwise compare each attribute counting the number of matches. If the access method implements separated structures for each data type, retrieving each attribute match does not prevent the quadratic point-to-point operation. Thus, it is necessary to consolidate each point-to-point number of matches to effectively improve the performance.

3.2 COMPARISON AMONG STATE OF THE ART METHODS

The indexes discussed in previous sections can handle applications involving trajectories that contain semantic data. However, they presented limitations that impacted their use for the aims of this work. Table 1 summarizes their main characteristics as follows: *Work* contains the index name and its authors; *Application/Data* summarizes the purpose that initially motivated the work and the data types as processed, highlighting the core in **bold**; *Allocation* presents the storage method adopted along with the index building; *Retrieving* concerns the search condition implemented by the technique being *Partial* or *Exact*; *Pruning* presents if the work applies any criteria to exclude a set of elements from comparison.

Queries, top-k, and range problems have a parameter to search for, while similarity requires fully comparing the data (no matter how different they are) to return a score expressing how similar the compared elements are. 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.

Table 1 – Comparative State-of-the-Art for Spatio-Temporal and Textual Access Methods

Work	Application/Data	Approach		
		Allocation	Retrieving	Pruning
GAT (Zheng et al., 2013)	Activity (Semantic) Trajectories **Search**	Hybrid	Partial	Yes
AFIA (Skovsgaard; Sidlauskas; Jensen, 2014)	**Real-time** Keyword **Search** for Microblogs Trending Topics	Memory	Exact	Yes
Mercury (Magdy et al., 2014)	**Real-time** Keyword **Search** for Microblogs Trending Topics	Memory	Partial	Yes
GiKi (Zheng et al., 2015)	Approximate Spatio-Temporal Keyword **Search**	Hybrid	Partial	Yes
GKR*-Tree (Mehta; Skoutas; Voisard, 2015)	Spatio-temporal Keyword **Search**	Memory	Partial	Yes
IOC-Tree (Han et al., 2015)	Spatial Keyword **Queries**	Memory	Exact	Yes
IRWI-Tree (Issa; Damiani, 2016)	Keyword **Queries on Aligned Spatio-Textual** Trajectories	Disk	Partial	Yes
GeoTrend (Magdy et al., 2016)	Spatial Trending **Queries on Real-time** Microblogs	Memory	Exact	Yes
R-trees with STLs (AHMED et al., 2017)	**TOP-k** Keywords on Spatio-Temporal **Range Queries**	Disk	Partial	Yes
ST-Tree (LIU et al., 2017)	**Approximate** Activity (Semantic) Trajectory **Search**	Hybrid	Partial	Yes
FTSM (FURTADO et al., 2018)	Spatial Support for **Similarity Measuring**	Memory	Partial	Yes
BB-Tree (Sprenger; Schäfer; Leser, 2019)	Multidimensional **Range Queries**	Memory	Partial	Yes
S^2R-Tree (Chen et al., 2020)	Spatial Keyword **Queries**	Memory	Partial	Yes
TwTL (Dam et al., 2021)	**Top-k** Trending Topics on Spatial **Range Queries**	Memory	Exact	Yes
MAT-Index (Souza et al., 2021)	Spatial, Temporal and Semantics Support for **Similarity Measuring**	Memory	Partial	No

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.

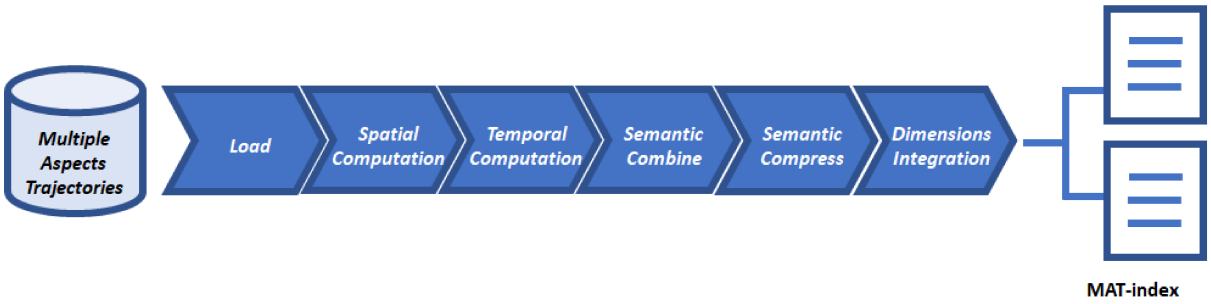

Figure 4 – MAT-Index Flow

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 Subsection 2.2.2) contains only one attribute, thus $\mathscr{F} = \{\{price\}, \{poi\}, \{weather\}\}$. 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 header of the dataset, is processed by the method *addAttributeToTwoLevelIndex* (Line 1). It saves the

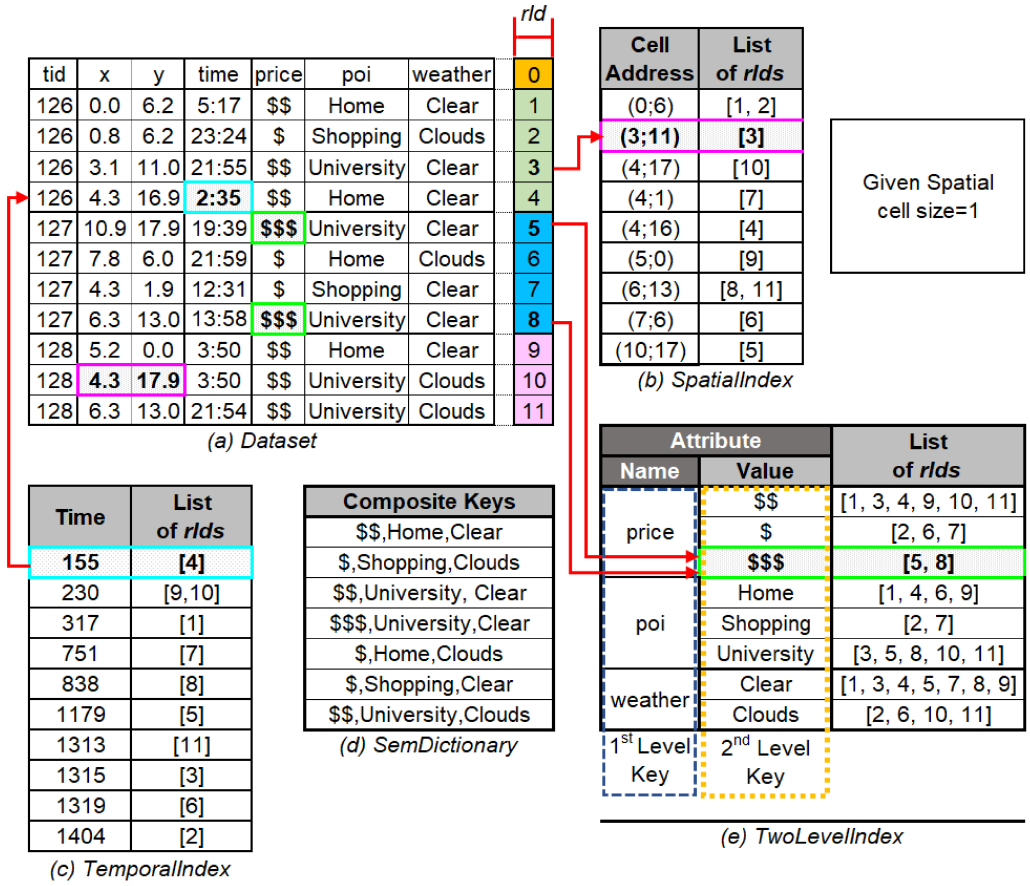

Figure 5 – Running example with the spatial (b), temporal (c), and semantic (d and e) data as preliminarily indexed

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* shown in Figure 6 (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

Algorithm 1: Load

input : Dataset // Figure 5(a)
output: SpatialIndex, TemporalIndex, SemDictionary, TwoLevelIndex
addAttributeToTwoLevelIndex(header)// Saves header (*rId*=0) - Fig 5(e) 1^{st} 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) 2^{nd} level
end
return SpatialIndex, TemporalIndex, SemDictionary, TwoLevelIndex

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 6 left) do automatically match, since they are below the threshold, thus not requiring the 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.

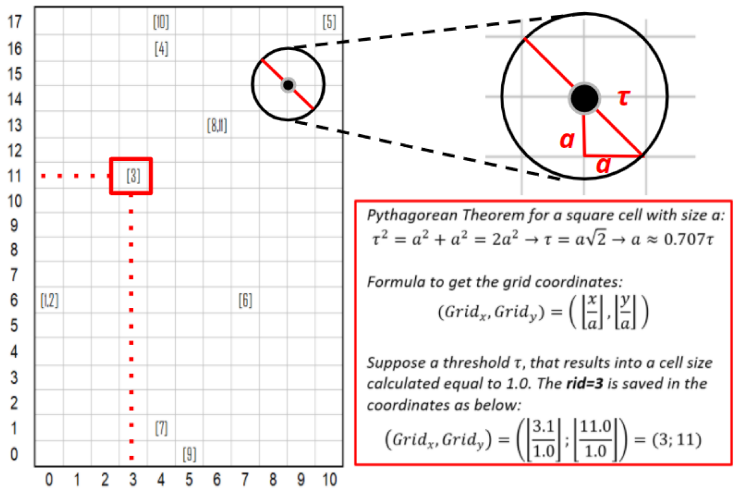

Figure 6 – Spatial Allocation Method

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 *getCandidate-Cells* 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.

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

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
```

{8,11} automatically match as well as {1,2} because they are in the same cell. The pairs of *rId*s {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 *rId*s, 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 a list of temporal matches 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).

Figure 7 (a) shows the temporal data as stored in the Load phase, and (b) the corresponding list of matches. All *rId*s in Figure 7 (a) shared the same list of matches. In the

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 = ⋃_{i=time−τ}^{time+τ} TemporalIndex.get(i).rIds   // τ: temporal threshold
3   foreach rId ∈ 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
```

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 *rId*s where, for instance, the cells 1313 and 1315 are within the threshold, so the list of *rId*s of both cells, i.e., *rId* [3] and *rId* [11] are added to both corresponding Matching *rId*s entries in Figure 7 (b). Additionally, 1315 and 1319 are within the threshold of 5 minutes, thus the process is repeated and the *rId*s [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 (*rId*s 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.

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]

TemporalIndex
(a)

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

List of Matches
(b)

Figure 7 – Result after the Temporal Computation

The temporal approach brings two benefits: *(i)* the allocation by unit allows us to get the matches by aggregating the *rId*s 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 *rId*s 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*.

4.4 SEMANTIC COMBINE

The *SemanticCombine* 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*, which 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.

Algorithm 4: Semantic Combine

```
   input : SemDictionary                                          // Figure 5 (d)
   output: SemCompositeIndex                                      // Figure 9
1  foreach semCompositeKey ∈ 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 ∈ F do          // Being each feature f a set of attributes
   |  |    // rIds where all feature attributes values in f match (Eq.2.5)
7  |  |  foreach rId ∈ ⋂_{i=1}^{|f|} TwoLevelIndex(attributeValue)_i.rIds do
8  |  |  |  ++MatchCounter[rId]
9  |  |  end
10 |  end
11 |  return MatchCounter
```

Function *getMatchCounter* in 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 $\mathscr{F}$ 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 *rId*s where all attributes match for f (row 7). Then, only the *rId*s 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*>, considering the features $\mathscr{F} = \{\{price\}, \{poi\}, \{weather\}\}$. 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.

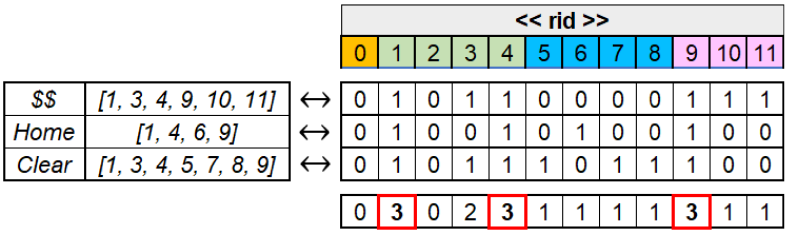

Figure 8 – Example of Match Counter computation

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.

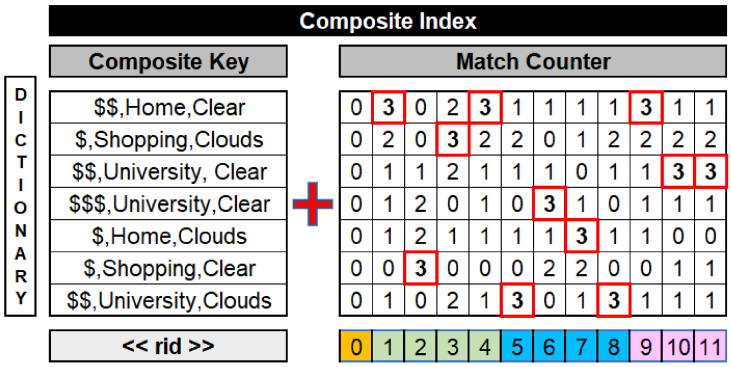

Figure 9 – Semantic Index after *Match Counter* computation

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 *rId*s {1,4,9}. It also matches with the *rId*=3 in two of the three semantic attribute values, only once with the *rId*s={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.

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 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.

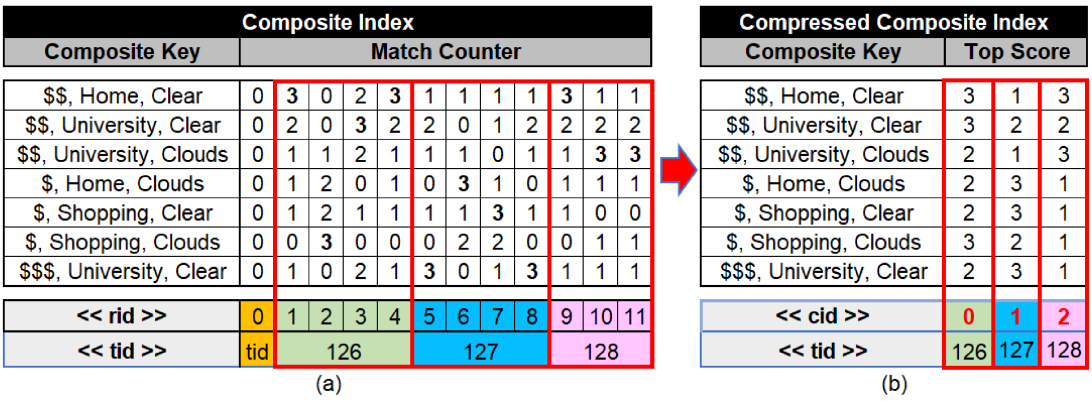

Figure 10 – Composite Index Highlighted By Trajectory: (a) before and (b) after the compressing

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 *rId*s. 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 trajectory 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 si-

multaneously. 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 *rId*s 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).

The final MAT-Index data structure, depicted in Figure 11 (c), considers the spatial,

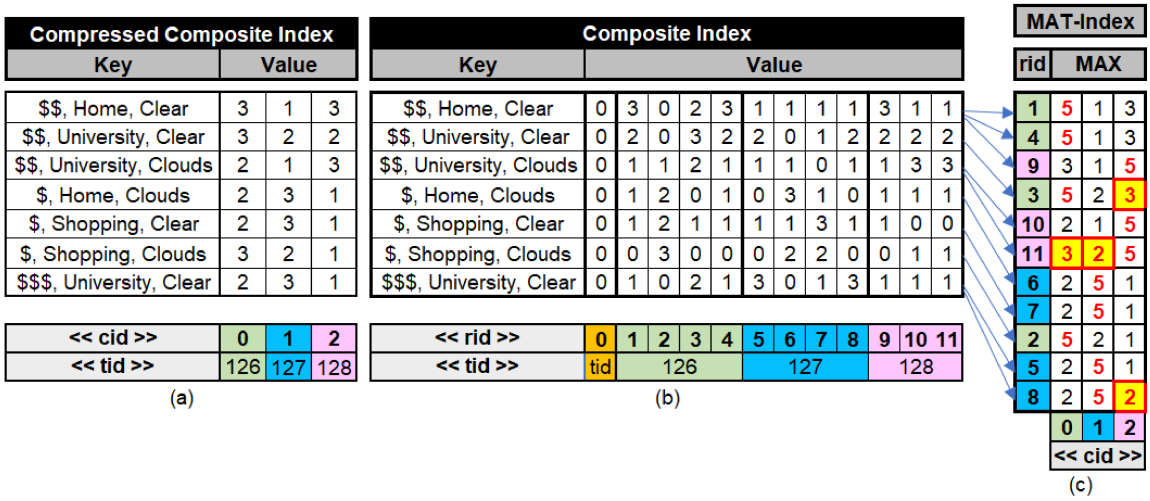

Figure 11 – MAT-Index (c) after Integration

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 tra-

jectory 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.

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 (FURTADO et al., 2018), 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 the public GitHub repository in https://github.com/anapbr/MasterDegree/tree/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 $\mathscr{F}$ 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 $\mathscr{F} = \{\{att_1\},\{att_2\},\{att_3\}\}$ (where each feature has one attribute, leading to MSM definition), $\mathscr{F} = \{\{att_1\},\{att_2,att_3\}\}$, or even all attributes together in the same feature $\mathscr{F} = \{\{att_1,att_2,att_3\}\}$.

We evaluate MAT-index considering a similarity running example and the quantitative aspects: Section 5.1 shows 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 SIMILARITY RUNNING EXAMPLE

We start this section presenting a scenario where both MSM and MUITAS 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 methods without index support do. It means that both similarity measures can start with the parity (MSM Equation 2.1, MUITAS Equation 2.7) computation, thus skipping for MSM Equations 2.2 and 2.3, and for MUITAS Equations 2.5 and 2.6. 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 summed, while the method without support 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 with $\mathscr{F} = \{\{price\},\{poi\},\{weather\}\}$, suppose we want to measure the 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 is to find the similarity score $Similarity(126,128)$ 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,

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

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 $cId = 0$.

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

After calculating the parities, the final similarity score $Similarity(126,128)$, 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

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

The presented process can be repeated for any pair of trajectories, using their *rId*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: 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 and FTSM supports. 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 (YANG et al., 2015) dataset contains real data, while the other is a benchmarck dataset generated by BerlinMOD (DüNTGEN; BEHR; GüTING, 2009). 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 regarding the *type* of the user and the transportation *mode*. In both datasets, the timestamp provided the weekday (*day*).

Table 2 summarizes the main characteristics of the datasets. *Traj Size* refers to the trajectory average length followed by the $\pm$ signal with its standard deviation; the *# of Users* attribute indicates the number of distinct anonymous users tracked; 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 (YANG et al., 2015)	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 (DüNTGEN; BEHR; GüTING, 2009)	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 2 – Summary of the trajectory datasets used in the experiments.

Figure 12 shows the running times comparison for each dataset considering MSM and MUITAS implementations without any support, 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:

[1] https://developer.foursquare.com/

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

- $\underline{NYC^1}$ 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 $\mathscr{F} = \{f_1, ..., f_5\}$, 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 $\mathscr{F} = \{f_1, ..., f_4\}$ is such that $f_1 = \{day\}$, $f_2 = \{price\}$, $f_3 = \{weather\}$, $f_4 = \{\underline{root\text{-}type, type}\}$.

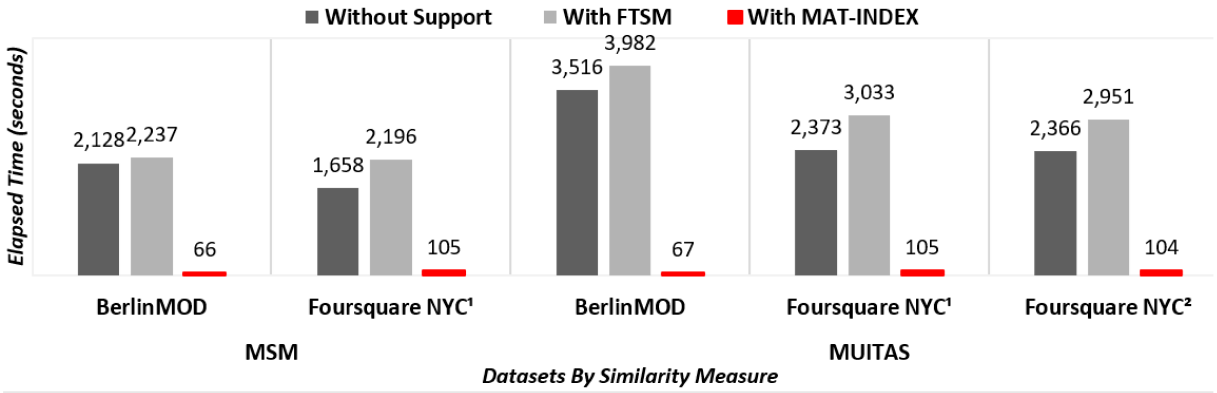

Figure 12 – Similarity computation elapsed time by dataset

By observing the results in Figure 12, we can notice that, in the implementations *Without Support* , MUITAS performed worse than MSM in all datasets due to the computation overhead for checking the semantic related attributes. However, using MAT-Index support results up to 52 times faster, reducing between 93.6% and 98.1% the execution times. We 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 in all datasets. The results with MAT-Index that is able to treat all dimensions in an integrated data structure are much better. 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 2) 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 $\mathscr{F} = \{\{price\}\}$, only the weather $\mathscr{F} = \{\{weather\}\}$, etc.; with 2 attributes, we can process the price and weather $\mathscr{F} = \{\{price\}, \{weather\}\}$ 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 implementation without index support 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.

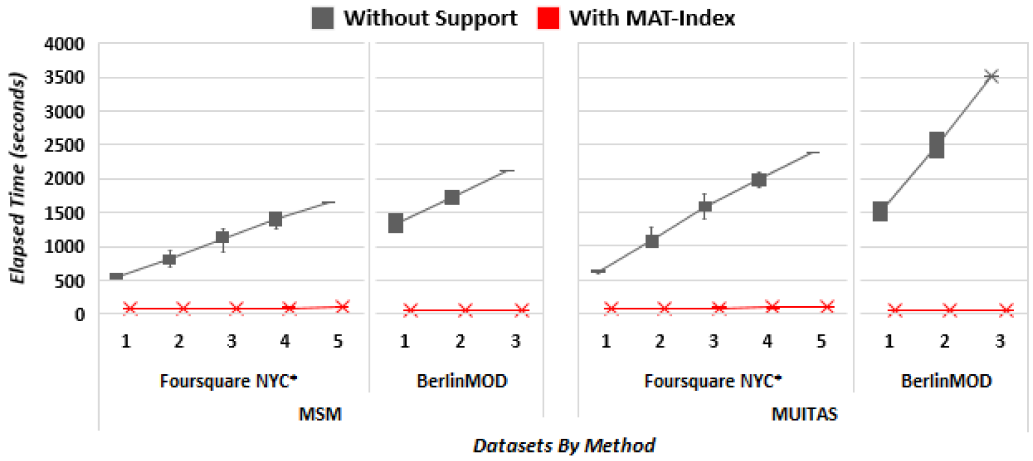

Figure 13 – Elapsed Times By the Number of Semantic Attributes

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 $\mathscr{F} = \{\{price\}\}$ produces only five distinct price possibilities (-1, 1, 2, 3, and 4); the weather $\mathscr{F} = \{\{weather\}\}$ has 6 distinct possibilities (Clear, Clouds, Fog, Unknown, Rain, and Snow). However, $\mathscr{F} = \{\{price\}, \{weather\}\}$ has 29 distinct possibilities in the dataset (one less than if we individually combine price and weather values). The chart of 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.

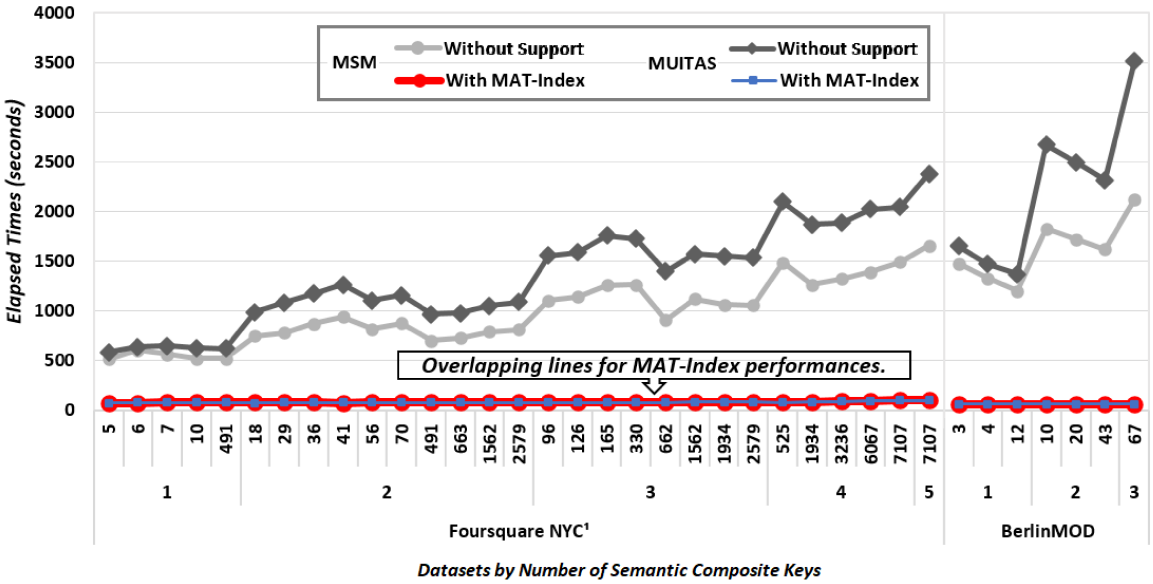

Figure 14 – Elapsed Times By the Number of Composite Keys

We can notice how the dataset size and the number of attributes generate an explosion in the elapsed times for the implementations without any index support. MAT-Index performance is very stable for both MSM and MUITAS similarity measures, represented by the overlapping lines in the chart since the execution times are very similar. The 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 (FERRERO et al., 2020) 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.

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 2) reported in Subsection 5.2.1 contain trajectories longer than all the datasets used in this scalability experiment. Therefore,

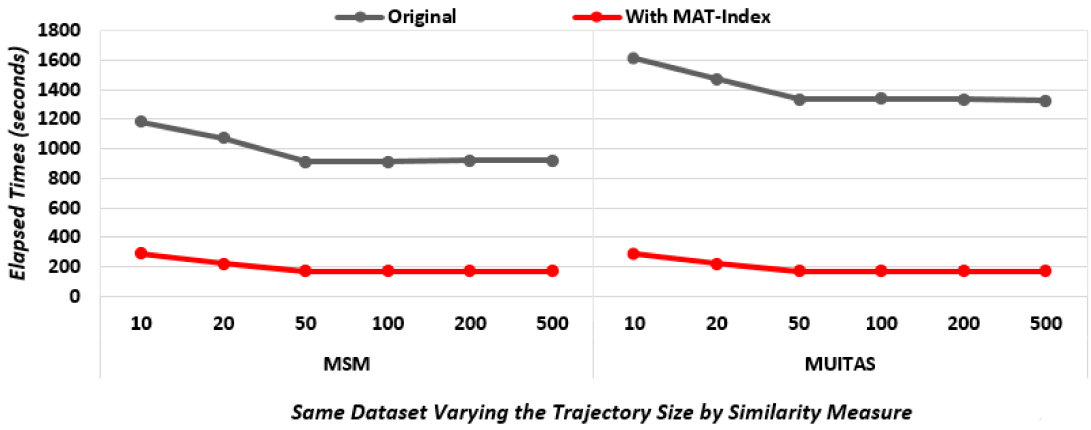

Figure 15 – Elapsed Times Varying the Trajectory Size for the Same Content

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%.

Similarity measures provide thresholds and matching criteria. In the case of changing, MAT-Index can compute the dimension affected independently and then reintegrate the index, saving the intermediate computation of the other unchanged dimensions. All MAT-Index elapsed times presented in this chapter include the index construction and the similarity computation itself, being the index building a very small part of the total. Indeed, considering the building costs discussed in Section 4.7 of each dimension, the update represents a minimal time compared to the similarity measures performance without index support, even when the entire rebuild is required.

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 (PARETO, 1906), 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

Measuring the similarity of multidimensional data as trajectories is a very costly task used on clustering, nearest neighbor queries, and other crucial problems in the literature. Several similarity measures can not deal with the high volume of real datasets because of point to point comparisons, which is very time consuming and leads to the need of index data structures capable of speeding up the computation.

Current indexes were not developed for efficiently manage all trajectory dimensions for similarity analysis. They also process a limited number of keywords (in general, only one). The limitations include intrinsically pruning techniques that are applied to speed up the processing. However, these pruning techniques do not perform a point-to-point analysis. The pruning techniques commonly involve top-k, range queries, and exact querying approaches. The top-k and range queries problems allow solutions removing candidates (so data) that never will reach a set at a certain point, or even discard information outside a spatial and/or temporal limit.

In this work we proposed MAT-Index 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 (MSM) and MUltIple aspect TrAjectory Similarity (MUITAS).

Regarding the index performance analysis, we show throughout a similarity running example how MAT-Index supports both MSM and MUITAS and drastically reduces the number of 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 in all scenarios of up to 98.1% in running time and 87.2% in scalability.

It is worth recalling that the execution times on experiments include the indexing building and the similarity computation. The similarity is still a quadratic problem that, with MAT-Index, depends on the number of trajectories (the method without index support depends on the number of trajectory points). Still, most of the execution relates to the comparison itself. Rebuilding the index does not significantly impact the performance, although threshold changes do not require full index rebuilding since it is possible to process again only the affected dimension and the final dimensions integration step. Datasets changes involving trajectory point inserting or updating would demand a full index reconstruction. Therefore, it is still is a significant gain compared to the original implementation, despite the re-execution..

Concerning future works, we would like to evaluate MAT-Index using more semantically enriched trajectory datasets. Currently, MAT-Index treats the three trajectory dimensions individually before integrating their computation results. When we have spatial or temporal threshold changes, we can compute only the dimension affected once more and then reintegrate the result. However, another opportunity is to develop an efficient update when new trajectory points are added to the dataset, although index building represents a tiny part of the elapsed

time presented along with the experiments for the similarity processing. Finally, problems involving range and top-k queries can benefit from MAT-Index intermediate data structures. For instance, the match counter structure favors creating a ranking of trajectory points by the number of matching attributes, yet, simple queries can be resolved varying from a single access to linear time.

BIBLIOGRAPHY

AHMED, P. et al. Efficient computation of top-k frequent terms over spatio-temporal ranges. In: **Proceedings of the 2017 ACM International Conference on Management of Data**. New York, NY, USA: Association for Computing Machinery, 2017. (SIGMOD '17), p. 1227–1241. ISBN 9781450341974. Disponível em: https://doi.org/10.1145/3035918.3064032.

BERNDT, D. J.; CLIFFORD, J. Using dynamic time warping to find patterns in time series. In: SEATTLE, WA. **KDD workshop**. [S.l.], 1994. v. 10, p. 359–370.

BOGORNY, V.; BRAZ, F. J. Introdução a trajetórias de objetos móveis: Conceitos, armazenamento e análise de dados. Univille, v. 1, 2012.

BÜTTCHER, S.; CLARKE, C. L.; CORMACK, G. V. **Information retrieval: Implementing and evaluating search engines**. [S.l.]: Mit Press, 2016.

Chen, L.; Özsu, M. T.; Oria, V. Robust and fast similarity search for moving object trajectories. In: ACM. **Proceedings of the 2005 ACM SIGMOD international conference on Management of data**. [S.l.], 2005. p. 491–502.

Chen, X. et al. S2r-tree: a pivot-based indexing structure for semantic-aware spatial keyword search. **GeoInformatica**, 07 2020.

Dam, T.-L. et al. Efficient top-k recently-frequent term querying over spatio-temporal textual streams. **Information Systems**, v. 97, p. 101687, 2021. ISSN 0306-4379. Disponível em: https://www.sciencedirect.com/science/article/pii/S0306437920301368.

Ding, H.; Trajcevski, G.; Scheuermann, P. Efficient similarity join of large sets of moving object trajectories. In: **2008 15th International Symposium on Temporal Representation and Reasoning**. [S.l.: s.n.], 2008. p. 79–87.

DüNTGEN, C.; BEHR, T.; GüTING, R. Berlinmod: A benchmark for moving object databases. **VLDB J.**, v. 18, p. 1335–1368, 12 2009.

FERRERO, C. et al. Mastermovelets: discovering heterogeneous movelets for multiple aspect trajectory classification. **Data Mining and Knowledge Discovery**, v. 34, p. 652 – 680, 2020.

FERRERO, C. A.; ALVARES, L. O.; BOGORNY, V. Multiple aspect trajectory data analysis: research challenges and opportunities. In: CAMPELO, C. E. C.; NAMIKAWA, L. M. (Ed.). **XVII Brazilian Symposium on Geoinformatics - GeoInfo 2016, Campos do Jordão, SP, Brazil, November 27-30, 2016**. MCTIC/INPE, 2016. p. 56–67. Disponível em: http://urlib.net/8JMKD3MGPDW34P/3NDC42E.

FINKEL, R.; BENTLEY, J. Quad trees: A data structure for retrieval on composite keys. **Acta Inf.**, v. 4, p. 1–9, 03 1974.

FURTADO, A. et al. Unveiling movement uncertainty for robust trajectory similarity analysis. **International Journal of Geographical Information Science**, v. 32, p. 1–29, 09 2017.

FURTADO, A. S. et al. Unveiling movement uncertainty for robust trajectory similarity analysis. **Int. J. Geogr. Inf. Sci.**, Taylor & Francis, Inc., USA, v. 32, n. 1, p. 140–168, jan. 2018. ISSN 1365-8816. Disponível em: https://doi.org/10.1080/13658816.2017.1372763.

FURTADO, A. S. et al. Multidimensional similarity measuring for semantic trajectories. **Transactions in GIS**, Wiley Online Library, v. 20, n. 2, p. 280–298, 2016.

Han, Y. et al. Spatial keyword range search on trajectories. In: RENZ, M. et al. (Ed.). **Database Systems for Advanced Applications**. Cham: Springer International Publishing, 2015. p. 223–240. ISBN 978-3-319-18123-3.

Issa, H.; Damiani, M. L. Efficient access to temporally overlaying spatial and textual trajectories. In: **2016 17th IEEE International Conference on Mobile Data Management (MDM)**. [S.l.: s.n.], 2016. v. 1, p. 262–271.

KANG, H.-Y.; KIM, J.-S.; LI, K.-J. Similarity measures for trajectory of moving objects in cellular space. In: ACM. **Proceedings of the 2009 ACM symposium on Applied Computing**. [S.l.], 2009. p. 1325–1330.

LEHMANN, A. L.; ALVARES, L. O.; BOGORNY, V. Smsm: A similarity measure for trajectory stops and moves. **International Journal of Geographical Information Science**, 2019.

LIU, H. et al. Semantic-aware query processing for activity trajectories. In: **Proceedings of the Tenth ACM International Conference on Web Search and Data Mining**. New York, NY, USA: Association for Computing Machinery, 2017. (WSDM '17), p. 283–292. ISBN 9781450346757. Disponível em: https://doi.org/10.1145/3018661.3018678.

Magdy, A. et al. Geotrend: spatial trending queries on real-time microblogs. In: . [S.l.: s.n.], 2016. p. 1–10.

Magdy, A. et al. Mercury: A memory-constrained spatio-temporal real-time search on microblogs. In: **2014 IEEE 30th International Conference on Data Engineering**. [S.l.: s.n.], 2014. p. 172–183.

MAHMOOD, A.; PUNNI, S.; AREF, W. Spatio-temporal access methods: a survey (2010 - 2017). **GeoInformatica**, 10 2018.

MEAGHER, D. **Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer**. [S.l.: s.n.], 1980.

Mehta, P.; Skoutas, D.; Voisard, A. Spatio-temporal keyword queries for moving objects. In: **Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems**. New York, NY, USA: Association for Computing Machinery, 2015. (SIGSPATIAL '15). ISBN 9781450339674. Disponível em: https://doi.org/10.1145/2820783.2820845.

MELLO, R. d. S. et al. Master: A multiple aspect view on trajectories. **Transactions in GIS**, Wiley Online Library, 2019.

MORTON, G. M. A computer oriented geodetic data base and a new technique in file sequencing. International Business Machines Company New York, 1966.

PARETO, V. Manuale di economica politica, societa editrice libraria. **Manual of political economy**, v. 1971, 1906.

PETRY, L. M. et al. Towards semantic-aware multiple-aspect trajectory similarity measuring. **Transactions in GIS**, v. 23, p. 960–975, 2019.

Skovsgaard, A.; Sidlauskas, D.; Jensen, C. S. Scalable top-k spatio-temporal term querying. In: **2014 IEEE 30th International Conference on Data Engineering**. [S.l.: s.n.], 2014. p. 148–159.

Souza, A. P. R. de et al. Mat-index: An index for fast multiple aspect trajectory similarity measuring. **Transactions in GIS**, 2021.

SPACCAPIETRA, S. et al. A conceptual view on trajectories. **Data & knowledge engineering**, Elsevier, 2008.

Sprenger, S.; Schäfer, P.; Leser, U. Bb-tree: A practical and efficient main-memory index structure for multidimensional workloads. In: **EDBT**. [S.l.: s.n.], 2019.

VLACHOS, M.; KOLLIOS, G.; GUNOPULOS, D. Discovering similar multidimensional trajectories. In: IEEE. **Proceedings 18th international conference on data engineering**. [S.l.], 2002. p. 673–684.

Wang, S. et al. Answering top-k exemplar trajectory queries. In: **2017 IEEE 33rd International Conference on Data Engineering (ICDE)**. [S.l.: s.n.], 2017. p. 597–608.

XIAO, X. et al. Inferring social ties between users with human location history. **Journal of Ambient Intelligence and Humanized Computing**, Springer, v. 5, n. 1, p. 3–19, December 2012.

YANG, D. et al. Modeling user activity preference by leveraging user spatial temporal characteristics in lbsns. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, IEEE, v. 45, n. 1, p. 129–142, 2015. ISSN 2168-2216.

YING, J. J.-C. et al. Mining user similarity from semantic trajectories. In: ACM. **Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks**. [S.l.], 2010. p. 19–26.

Zheng, B. et al. Approximate keyword search in semantic trajectory database. In: **2015 IEEE 31st International Conference on Data Engineering**. [S.l.: s.n.], 2015. p. 975–986.

Zheng, K. et al. Towards efficient search for activity trajectories. In: **2013 IEEE 29th International Conference on Data Engineering (ICDE)**. [S.l.: s.n.], 2013. p. 230–241.