Author/s:
Tedjopurnomo, DA;Li, X;Bao, Z;Cong, G;Choudhury, F;Qin, AK

Title:
Similar Trajectory Search with Spatiooral Deep Representation Learning

Date:
2021-12-01

Citation:
Tedjopurnomo, D. A., Li, X., Bao, Z., Cong, G., Choudhury, F. & Qin, A. K. (2021). Similar Trajectory Search with Spatiooral Deep Representation Learning. ACM Transactions on Intelligent Systems and Technology, 12 (6), https://doi.org/10.1145/3466687.

Persistent Link:
https://hdl.handle.net/11343/355758

Similar Trajectory Search with Spatio-Temporal Deep Representation Learning

DAVID ALEXANDER TEDJOPURNOMO, RMIT University, Australia
XIUCHENG LI, Nanyang Technological University, Singapore
ZHIFENG BAO, RMIT University, Australia
GAO CONG, Nanyang Technological University, Singapore
FARHANA CHOUDHURY, The University of Melbourne, Australia
A. K. QIN, Swinburne University of Technology, Australia

Similar trajectory search is a crucial task that facilitates many downstream spatial data analytic applications. Despite its importance, many of the current literature focus solely on the trajectory's spatial similarity while neglecting the temporal information. Additionally, the few papers that use both the spatial and temporal features based their approach on a traditional point-to-point comparison. These methods model the importance of the spatial and temporal aspect of the data with only a single, pre-defined balancing factor for all trajectories, even though the relative spatial and temporal balance can change from trajectory to trajectory. In this article, we propose the first spatio-temporal, deep-representation-learning-based approach to similar trajectory search. Experiments show that utilizing both features offers significant improvements over existing point-to-point comparison and deep-representation-learning approach. We also show that our deep neural network approach is faster and performs more consistently compared to the point-to-point comparison approaches.

CCS Concepts: • **Computing methodologies** → **Artificial intelligence**; **Knowledge representation and reasoning**; **Spatial and physical reasoning**; **Temporal reasoning;**

Additional Key Words and Phrases: Deep neural networks, spatio-temporal, trajectories, attention model

ACM Reference format:
David Alexander Tedjopurnomo, Xiucheng Li, Zhifeng Bao, Gao Cong, Farhana Choudhury, and A. K. Qin. 2021. Similar Trajectory Search with Spatio-Temporal Deep Representation Learning. *ACM Trans. Intell. Syst. Technol.* 12, 6, Article 77 (December 2021), 26 pages.
https://doi.org/10.1145/3466687

This research is supported in part by ARC DP200102611, DP180102050, and LP180100114. Gao Cong acknowledges the support by Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU), which is a collaboration between Singapore Telecommunications Limited (Singtel) and Nanyang Technological University (NTU) that is funded by the Singapore Government through the Industry Alignment Fund - Industry Collaboration Projects Grant, and a Tier-1 project RG114/19.

Authors' addresses: D. A. Tedjopurnomo and Z. Bao, RMIT University, 124 La Trobe St, Melbourne, Victoria 3000, Australia; emails: david.tedjopurnomo@student.rmit.edu.au, zhifeng.bao@rmit.edu.au; X. Li and G. Cong, Nanyang Technological University, 50 Nanyang Ave, Singapore 639798; emails: {lixiucheng, gaocong}@ntu.edu.sg; F. Choudhury, The University of Melbourne, Melbourne, Parkville, Victoria 3010, Australia; email: fchoudhury@unimelb.edu.au; A. K. Qin, Swinburne University of Technology, John Street, Hawthorn, Victoria 3122, Australia; email: kqin@swin.edu.au.

2157-6904/2021/12-ART77 $15.00
https://doi.org/10.1145/3466687

1 INTRODUCTION

With the proliferation of GPS devices installed on satnavs, trajectory data have become larger and more easily accessible. This has opened up many trajectory-driven research topics, one of which is the similar trajectory search. Given a database of trajectories and a query trajectory, similar trajectory search returns the trajectory in the database that has the highest similarity (or smallest distance) to the query. Similar trajectory search is one of the fundamental operations for spatial databases and can be applied to many downstream tasks such as co-movement identification [33], ride sharing [15], traffic analytics [30], and trajectory clustering [28].

At the heart of the most similar trajectory search is the trajectory similarity metric. Traditional metrics such as DTW [3], EDR [6], ERP [5], and LCSS [26] try to find the optimal alignment between two trajectories and calculate the similarity based on the point-to-point distances between these aligned trajectories. Many recent studies adopt a more advanced point-to-point distance approach to improve the effectiveness and efficiency of the distance computation. However, most only utilize the spatial aspect of the data while ignoring the important temporal information.

The lack of temporal information makes trajectory similarity search more rigid, affecting its downstream tasks. For instance, in traffic analytics, we may use spatial-only trajectory to find traffic hotspots. However, if we use both the spatial and temporal information, we can issue more useful queries such as finding traffic hotspots during the peak morning or evening hours. Furthermore, spatial-only trajectory search may be hindered when the query trajectory follows a popular route since too many similar results will be returned. Adding the temporal dimension allows for a more precise similar trajectory search that takes into account when the trajectory happened and its duration. Aside from the lack of temporal feature usage, point-to-point distance paradigm suffers from commonly found imperfections of the dataset, which are non-uniform sampling rate and GPS errors. We formalize these challenges below.

Challenge 1. Temporal differences between trajectories. Intuitively, the similarity between the two trajectories are based on their spatial coordinates. While generally true, this does not apply to all cases. For instance, in Figure 1, all trajectories on the left have the same timestamps. In this scenario, it makes sense to assign trajectory blue as trajectory red's most similar trajectory due to the spatial closeness. On the right, trajectory blue starts 1 hour after trajectory red and only has a 15-minute temporal span compared to red's 30 minutes even though they're spatially similar. On the other hand, trajectory black has identical timestamps but closer spatial coordinates to trajectory red compared to the left red-black pair. In this scenario, it is important to find the correct balance between the spatial and temporal aspect, which can vary from trajectory to trajectory in the same dataset and is difficult to be manually defined by users. Both the starting time and the temporal span of the trajectories are important.

Challenge 2. Non-uniform sampling rate. Due to factors such as GPS malfunctions, missing data, and different devices' specifications, different trajectories may be reported at different sampling rates, which may change the shape and context significantly. In Figure 2, there are two similar movements, denoted in red and blue+gray, reported at different sampling rates. However, due to the lower sampling rate, the blue+gray trajectory recorded the movement as a straight line, missing the detour portrayed by the grayed-out trajectory points because of the low sampling rate. Existing point-to-point-based metrics do not have any mechanism to mitigate the impact of missing trajectory points, e.g., inferring the missing points. Thus, their distance computation under the non-uniform sampling condition tends to be inaccurate.

Challenge 3. GPS errors. Due to signal interference in urban areas, GPS coordinates may contain spatial noise. In Figure 3, the red trajectory represents the actual trajectory. Due to GPS errors,

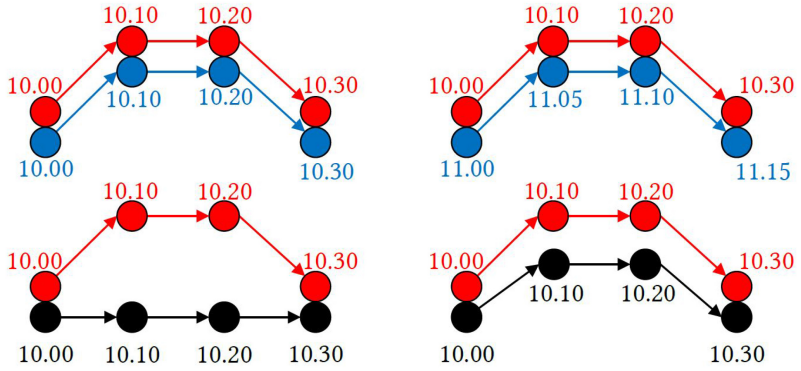

Fig. 1. Temporal differences between trajectories.

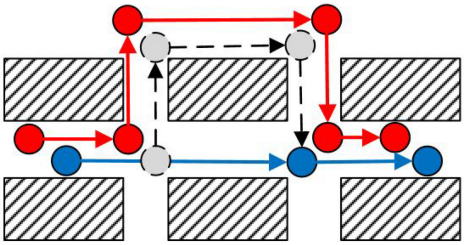

Fig. 2. Non-uniform sampling rate.

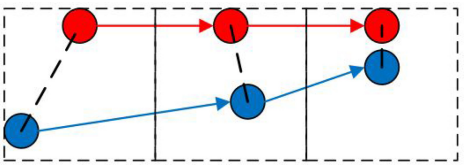

Fig. 3. GPS errors.

the trajectory may be distorted, which we visualize with the blue trajectory. While the spatial distortion may be within a certain bound (which we represent using the dashed boxes), these GPS noises between the two trajectories' points (which we represent using the dashed lines), are accumulated if we use existing point-to-point-based metrics. Thus, these metrics will assign a relatively high distance for the two trajectories even though they record the same movement.

Currently, there are only a few studies that propose a spatio-temporal similarity metric [9, 21, 37]. However, they calculate the spatial and temporal similarity separately and aggregate them using a single balancing parameter for the whole dataset, which hinders their performance because the relative importance of the spatial and temporal aspect of the data can vary from trajectory to trajectory. In addition, current literature mainly use an approach based on comparing two trajectories point-by-point, which is susceptible to non-uniform sampling rate and GPS errors. The work of Li et al. [14] addressed these two challenges by using a deep representation learning approach, but their work considers only the spatial dimension, which cannot address Challenge 1. We will describe the weaknesses of current approaches in more detail in Section 2.

In this article, we address these three challenges from a deep representation learning perspective. We utilize a deep neural network structure called GRU Sequence Autoencoder, which consists of a GRU encoder and decoder, because of its capability in processing sequence data. In tandem with this model, we utilize a spatio-temporal grid consisting of many cells to represent different areas of interest on different time periods where each combination of spatial area and time period is represented with a trainable feature vector. This enables us to learn complex spatio-temporal correlations between areas and time periods, resulting in a major improvement over using a single balancing factor. This also helps to address Challenge 1 by reducing the impact of minor temporal variations as our method assigns timestamps that are close enough into the same temporal cell.

Similarly, this also helps to address Challenge 3 since the spatio-temporal grid assigns coordinates that are sufficiently close into the same spatial cell, reducing the impacts of minor GPS errors. Furthermore, as our model transforms raw trajectory representation to a trajectory feature vector, we therefore transform the task of comparing trajectories on a point-to-point basis into the task of vector distance computation. This accelerates the trajectory similarity computation process and can be further accelerated using a KD-Tree.

A typical sequence autoencoder employs only one loss function—the reconstruction loss. Using this default configuration, it is not able to accurately capture the spatio-temporal context of trajectories and handle the three challenges. Therefore, we design a composite loss that incorporates three loss functions and use the reconstruction loss as one of its three components. Each loss function addresses the challenges in different ways, with each operating on different representations of the data. The first loss function, the *representation loss*, operates on the feature vector representation of trajectories output by the encoder. This helps to address all three challenges. The second loss function is the reconstruction loss, but in order to reduce ambiguity, we use the term *point-to-point loss*. It operates on the predicted trajectory, which is the output from the decoder. This helps to address all three challenges. Finally, for the third loss function, the *pattern loss*, we introduce a novel idea of dividing trajectories to segments based on the timestamp. We then learn the spatial and temporal pattern for each segment and use this pattern representation in the loss function. This helps address Challenges 1 and 2. We will describe in detail how these loss functions address different challenges in Section 4. For the learning process, we inject synthetic noise into our dataset to simulate data imperfections expressed in the aforementioned three challenges. Our primary goal is to address Challenge 1 by showing that our method outperforms the traditional spatio-temporal similarity metric that uses only a single balancing factor, as well as the state-of-the-art spatial-only deep neural network approach. Our secondary goals are to address Challenges 2 and 3 by showing that our model is robust to non-uniform sampling rates and GPS errors. To the best of our knowledge, this is the first work that utilizes a deep representation learning approach for spatio-temporal trajectories. To summarize, here are the contributions of our paper:

- We use a sequence autoencoder model to transform the trajectory distance computation task into a vector distance computation task, which is much more efficient.
- We utilize deep representation learning and propose three novel loss functions to address the three challenges. They are the representation loss, point-to-point loss, and pattern loss. The pattern loss utilizes a novel method to capture trajectory movement patterns.
- We conduct extensive experiments to compare our method with several traditional similarity metrics which we extend to the spatio-temporal case and a state-of-the-art spatial-only deep representation learning approach. The results consistently show the superiority of our approach over both the traditional metrics and the deep learning–based metrics.
- In our experiments, we also perform exploratory analyses on four spatio-temporal point-to-point trajectory similarity metrics and show how they may fail in handling one or more of the aforementioned challenges faced in similar trajectory search scenarios.

2 RELATED WORK

In this section, we explore the related work from three perspectives: an overview of traditional similarity metrics, more modern variations of the traditional point-to-point comparison, and deep representation learning approach. It is worth highlighting that the vast majority of studies in each of the three perspectives have a focus on spatial-only trajectory. Note that, although we refer to them as similarity metrics, most calculate the distance between trajectories; a measure of dissimilarity. In such cases, the smaller the distance, the more similar the trajectories.

2.1 Traditional Similarity Metrics

Traditional trajectory similarity metrics rely on the pairwise matching of the points between trajectories to compute the distance. These methods are traditionally applied to one-dimensional time series data rather than the 2D (3D) spatial (spatio-temporal) trajectories. The earliest and simplest method is the Euclidean distance where every pair of points is sequentially checked, one from each trajectory. The summation of the Euclidean distances between each such pair is the final distance between the two trajectories. This simple method suffers from major drawbacks: it can only be applied to equal-length trajectories, and it is very sensitive to differing sampling rates and alignments. To address these drawbacks, the **Dynamic Time Warping** (**DTW**) method is proposed in [3]. Unlike Euclidean distance, DTW allows each point of a trajectory to be checked against multiple points in another trajectory to find the optimal pairwise matching, where a point is matched to the point with the shortest distance. Another trajectory similarity metric, **Edit Distance with real Penalty** (**ERP**), is proposed by Chen and Ng [5]. Unlike DTW, this distance function does not perform a one-to-many point matching. If a point needs to be re-matched to find the optimal alignment, ERP instead uses a *gap* element, defined as a stationary point in the metric space, to be used as a reference point. Then, it calculates the distance of the current point to the gap element, a process that substitutes the re-matching process in DTW. This makes ERP a *metric* as it obeys the triangle inequality, enabling the use of many indexing structures.

All of the aforementioned distance functions rely on real distances. That is, the trajectory distance calculation is based on the summation of the actual distance between points. Consequently, these methods are not robust to outliers. The following two methods, **Longest Common Subsequence** (**LCSS**) [26] and **Edit Distance on Real sequence** (**EDR**) [6] perform an alternative computation that sums up the number of matching points between two trajectories, where a match occurs when the distance between two points is smaller than a pre-defined parameter ϵ. This approach is robust to outliers as a non-matching ordinary point and non-matching outlier point contribute equally to the calculation, no matter how far apart the distances are. The difference between LCSS and EDR is that LCSS measures the similarity between the two trajectories while EDR is a measure of dissimilarity. Specifically, given a pair of trajectories X and Y, LCSS measures the number of matching pairs of trajectory points appearing in the same relative order, but not necessarily contiguous, while EDR performed a series of insertion, deletion, or replacement operations to transform X into Y. For further reading regarding these metrics, we refer viewers to the survey of Wang et al. [27].

Utilizing both the spatial and temporal dimensions of the data is crucial. So far, these methods perform spatial-only similar trajectory search, but extension to spatio-temporal is possible. To do this, one can treat the temporal feature as another dimension of the data and transform the problem from 2D trajectory similarity computation to 3D. One can employ the approach used by Shokoohi-Yekta et al. [23] for this task, in which they performed a generalization of the traditional one-dimensional DTW to the multidimensional case and defined two possible approaches: (1) Independent, in which DTW is calculated on each dimension separately and the distances are summed and (2) Dependent, where multidimensional DTW is calculated on all dimensions at once.

While these extensions are possible, a major weakness of this approach is that these methods cannot dynamically adjust the balance between the spatial and temporal aspect of the data. As we mentioned in Section 1, the relative importance between these two aspects can change from trajectory to trajectory. In addition, these traditional trajectory similarity metrics are slow to compute; since these methods rely on finding the optimal alignment between trajectories, they use a dynamic programming approach that has an $O(n^2)$ complexity. Rakthanmanon et al. [19] proposed the UCR suite to accelerate the computation for DTW only. However, the UCR suite is only applicable for time-series data, which is one-dimensional.

In this work, we use DTW, EDR, ERP, and LCSS as the baseline models and assess their performance, sensitivity to parameters, and efficiency. In order to perform a fair comparison with our spatio-temporal approach, we extend DTW, EDR, ERP, and LCSS to the spatio-temporal case. We will describe such extensions in Section 5.

2.2 Variants of Traditional Approach

Many of the more recent works approached the trajectory similarity computation from a non-deep-learning perspective, relying on a more optimized point-to-point comparison paradigm akin to the aforementioned traditional methods. Most of these studies only calculate the spatial distance between the trajectories. **Edit Distance with Projections** (**EDwP**) [20] uses the temporal feature of trajectory data, but only to facilitate the calculation of the spatial distance. Specifically, the temporal information is used to calculate the speed of an object moving from one location to another. When performing a point-to-point match, EDwP may perform an insertion operation that projects the point of one trajectory to another based on the speed. Su et al. [25] proposed a method to address the challenge of non-uniform sampling rate. They used an anchor-based calibration system that aligns trajectory points to a set of pre-defined, dataset-independent anchor points. Their work utilized the temporal information to perform a more accurate trajectory calibration. Thus, while these works use temporal information, it is only to facilitate the spatial distance computation and we consider them to be spatial-only. Chen et al. [7] defined a novel query called **k Best-Connected Trajectories** (**k-BCT**). A k-BCT query allows a spatial-only querying of trajectories by specifying several locations that are not necessarily ordered. The trajectories returned by k-BCT are trajectories that best connect the provided locations spatially. Wang et al. [29] defined a novel trajectory similarity function called LORS that operates on map-matched trajectories, unlike conventional similar trajectory search metrics that are based on trajectory points. Finally, several other researches have also focused on distributed similar trajectory search [22, 32, 35].

Spatio-temporal, point-to-point similarity metrics are relatively rare. Dan et al. [9] used a two-level grid index for the temporal and spatial dimension of the data. They also utilized a time-first searching framework and a triangle inequality approach to prune trajectories, improving the efficiency. Zhao et al. [37] performed a database grid indexing and a query partitioning approach to accelerate the computation. These two methods compute spatial and temporal distance separately and then aggregates them using single balancing factor. As we have mentioned in Section 1, the importance of the spatial and temporal aspect of a trajectory varies greatly amongst datasets and even amongst data points within the datasets. Thus, applying the same balancing factor for the whole dataset is not the right approach.

2.3 Deep Representation Learning for Trajectory Similarity Computation

Representation learning is a task of representing raw data in a way that makes it easier to extract useful information for classifiers or other predictors. The performance of machine learning methods is heavily reliant on the data representation—the choice of features that represent the data [2]. As the moniker suggests, deep representation learning concerns the usage of deep neural network models to perform representation learning. This technique has been applied to research fields such as **Natural Language Processing** (**NLP**) [16, 18] and image recognition [12, 24].

Despite the popularity of deep representation learning, its application in trajectory analysis is rare. Yuan et al. [36] uses a neural network for **origin-destination** (**OD**) travel time estimation. Yao et al. [34] used a **Long Short-Term Memory** (**LSTM**) network imbued with a novel spatial attention memory to approximate existing traditional trajectory similarity metric while reducing their computational complexity. The work of Li et al. [14] is the most similar to ours as it performs a similar trajectory search based on deep representation learning. They used an

Table 1. Important Notations and Definitions

Notation	Definition
X	Input query trajectory
$\mathcal{X}$	Set of query trajectories
Y	Ground truth trajectory
Y^*	Result of a most similar trajectory query
$\mathcal{Y}$	A database of ground truth trajectories
x_t	Query trajectory point at time t
y_t	Ground truth trajectory point at time t
d	An arbitrary trajectory similarity function

NLP-inspired approach to transform raw latitude and longitude information in trajectory points to feature vectors and utilized a deep neural network to learn the overall trajectory's vector representation. This representation is then used in place of the raw trajectory points. Furthermore, they used noise contrastive estimation in their loss function for better efficiency. Finally, Wang et al. [31] addresses the similar subtrajectory search problem using deep representation learning. These works only consider the spatial information. Extending these models to the spatio-temporal case is not straightforward because of the complex interactions between different locations at different times. To the best of our knowledge, ours is the first work that combines spatio-temporal data with a deep representation learning approach for the similar trajectory search problem.

Summary. Current similar trajectory search literatures suffer from two main deficiencies: they do not combine the spatial and temporal features, and for the ones that do, they utilize a rigid approach based on a pre-defined balancing factor for the whole dataset. We address the first problem by modeling our data in a way that spatial and temporal correlation can be learned together and the second problem by using a deep neural network that can learn to assign the correct spatial and temporal balance based on the needs of individual trajectories.

3 PROBLEM DEFINITION AND PRELIMINARIES

In this section, we will first formalize the spatio-temporal similar trajectory search problem. Afterward, we will provide some background for the deep neural network model we use for the task of similar trajectory search.

3.1 Problem Definition

We will first introduce several important concepts used in this section. Frequently used notations are presented in Table 1. Then, we will formally define the problem statement.

Definition 3.1 (Spatio-temporal Trajectory). A spatio-temporal trajectory is defined as a sequence of trajectory points. Each trajectory point is a triplet of latitude, longitude, and timestamp information.

Definition 3.2 (Spatio-temporal most Similar Trajectory Search Problem). Given a query trajectory X, a trajectory database $\mathcal{Y}$, and an arbitrary trajectory similarity function d, this search problem aims to find a trajectory Y^* that has the smallest distance (or highest similarity) to X. Formally,

$$\{Y^* \in \mathcal{Y} \mid d(X, Y^*) < d(X, Y), \forall Y \in \mathcal{Y} \setminus \{Y^*\}\}. \tag{1}$$

Problem Statement. Given a collection of query trajectories and a collection of database trajectories $\mathcal{Y}$, our aim is to learn a similarity function d that can accurately answer the spatio-temporal

most similar trajectory search queries. We use the mean rank to measure the accuracy, which will be elaborated in Section 5.

3.2 Preliminaries

The similar trajectory search problem relies on the similarity function d. This function calculates the distance between the trajectories, where smaller distance means greater similarity. In this work, we explore the usage of deep representation learning combined with spatio-temporal features. There is a wide array of deep neural network models that can be employed for deep representation learning of trajectory data. In our work, we use a sequence autoencoder. Sequence autoencoder is a deep neural network structure that combines two **Recurrent Neural Networks** (**RNNs**); one acts as an encoder and the other acts as the decoder. The encoder takes an input sequence and learns a vector representation of that input, while the decoder takes this vector representation and produces a prediction in the form of an output sequence. This sequence autoencoder is trained using the backpropagation algorithm, which is computed using the gradient of the loss. The loss function is a measure of the model's performance, where a smaller loss equals better performance. A standard sequence autoencoder is trained to reconstruct a ground truth sequence. Thus, a typical loss function for sequence autoencoder is the reconstruction loss, which captures how well the decoder part of the model reconstructs the ground truth sequence.

Sequence autoencoders are commonly applied to NLP tasks such as neural machine translation [1] because it is a sequence-to-sequence problem. Similar trajectory search can also be modeled as a sequence-to-sequence problem. In the context of similar trajectory search, each GPS recording is treated as one part of the input sequence. Thus, using sequence autoencoders, we can train our model to reconstruct the target trajectory. We will describe this process in more detail in Section 4.4.

While the vanilla RNN structure can be used, it can suffer from the vanishing and exploding gradient problem. These two problems severely diminish the model's learning capability for very long sequences. In order to address these problems, two approaches, the LSTM [10, 11] and **Gated Recurrent Uni**t (**GRU**) [8] were proposed. We use the latter for our work as it contains fewer parameters to train. GRU has two gates: a reset gate and an update gate. These gates control the balance between keeping the information from the previous part of a sequence and incorporating new information. Essentially, these gates imbue the model with the ability to not only learn and retain new information, but also to forget old information that is no longer relevant and can be harmful for the learning process. Using these information gates, GRU can avoid the vanishing and exploding gradient problem, allowing it to be effective even for very long sequences.

4 OUR METHODOLOGY

We use the GRU Sequence Autoencoder model with Attention in our work, which can be seen in Figure 4. This model takes two inputs, the input trajectory X and target trajectory Y, which in this example contain t triplets of latitude, longitude, and timestamp values; i.e., $\{xlat, xlon, xtime\}$ and $\{ylat, ylon, ytime\}$, respectively. The model will then output a reconstruction Y' of the target trajectory Y. The more similar Y' is to Y, the better our model performs.

Each input trajectory point $x \in X$ is first transformed into a vector embedding. This process is described in Section 4.1. Afterward, they are fed to the encoder, which consists of the encoder GRU cells represented as h, to produce a feature vector representation VX of the input. This process transforms the trajectory similarity computation task into the simpler vector distance calculation. Given two trajectories, the distance between them is simply the Euclidean distance between their vector representations. While a point-to-point trajectory similarity computation using raw

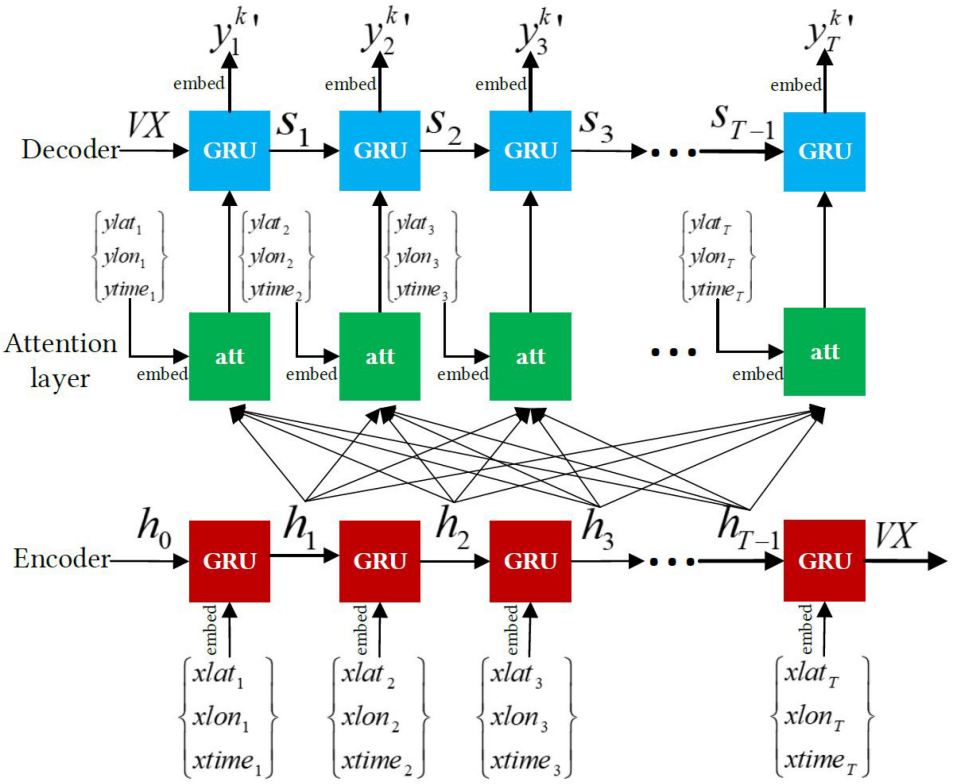

Fig. 4. The full architecture of our model.

trajectory data has a computational complexity of $O(n^2)$, Euclidean distance computation is only $O(n)$. In addition to VX, the state of each encoder GRU cell h is fed to the attention layer.

For the model training, we utilize an attention layer in our work. This layer takes two inputs, the encoder states h and the target trajectory Y, and incorporates them together. At every timestep, the attention layer determines which parts of the input sequence, as processed by the encoder, are the most salient for the reconstruction process in the decoder. The decoder part of the network has two inputs and one output. The first is the vector representation VX of the input trajectory from the encoder. From this vector, we seek to output a reconstruction Y' of the target trajectory Y. The second input to the decoder is the output of the attention layer. This output helps the decoder to improve the reconstruction accuracy by focusing on the important parts of the input. The output of the decoder undergoes a final embedding process to produce $Y^{k\prime}$ with the points $[y_1^{k\prime}, y_2^{k\prime}, y_3^{k\prime}, \ldots, y_T^{k\prime}]$. This reconstruction and embedding process will be presented in detail in Section 4.4.

To train our model, we use the backpropagation algorithm that will adjust the parameters of the model. A standard sequence autoencoder uses only one loss function—the reconstruction loss. However, a standard sequence autoencoder loss cannot address the three challenges we mentioned in Section 1. We explain our reasoning in Section 4.4. Thus, we propose three loss functions that each update different parts of the model. Before we describe the loss functions, We will first describe how we perform data modeling; from the raw latitude, longitude, and timestamp triplet, to a vector representation. We will then describe our three loss functions. These loss functions contribute to the backpropagation separately in this ordering: representation loss, point-to-point loss, and pattern loss. They are described in detail in Sections 4.3, 4.4, and 4.5, respectively.

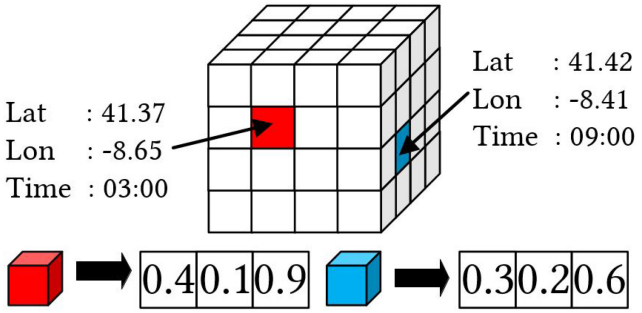

Fig. 5. Performing a lookup using the 3D spatio-temporal grid.

4.1 From Raw Data to Vector Representation

Raw latitude, longitude, and timestamp information can be used for the similarity search task. However, these raw data do not contain important contextual information such as the definition of spatial regions, time periods, and the interaction between the spatial and temporal aspect of the data.

To address this problem, we use the idea of spatio-temporal grids. We first divide the spatial area (e.g., a city) into a 2D grid of square-shaped cells. Then, we divide the 24-hour time range into equal-length time periods. We then combine the two to create a 3D spatio-temporal grid. Given a raw trajectory point, we perform a lookup on this grid to find which cell it belongs to. Each cell is then represented with a feature vector that contains its spatio-temporal context, which captures the proximity of the spatial areas and time periods, and the relationship between them. For instance, neighboring cells may be represented with similar feature vectors (e.g., adjacent city area, both during morning rush hours). For clarification, a grid is the collection of all spatio-temporal cells. When we refer to a grid, we refer to the entire collection of cells and when we refer to a cell, we refer to a single cell within the grid.

We display two examples in Figure 5. The first trajectory point has a latitude, longitude, and timestamp of {41.37, −8.65, 03:00}. After performing a lookup on the 3D grid, we find the cell colored in red which contains the point. For instance, the red cell may cover the latitude range from 41.30 to 41.40, the longitude range from −8.70 to −8.60, and the time range from 00:00 to 06:00. We do the same for the second point to find the blue cell, which covers the latitude range from 41.40 to 41.50, the longitude range from −8.50 to −8.40, and the time range from 06:00 to 12:00.

The lookup process matches the raw trajectory point to a spatio-temporal cell. Each cell is represented with a feature vector that contains the spatio-temporal context. For instance, the red and blue cells are represented with {0.4, 0.1, 0.9} and {0.3, 0.2, 0.6}, respectively. We initially randomize the vector of the cells. In the training process, the values are adjusted so that they can more accurately reflect the context of the cells. Cells that are closer to each other spatially and temporally tend to have similar context and thus their vector should to be similar. This is similar to the training process of Word2Vec [16, 17], which is a technique used in NLP applications. It transforms words in a corpora to a feature vector representation, which can be trained to capture the semantic context of the words. The closer two words are in the vector space, the more similar their context. When training a Word2Vec corpus from scratch, each word is assigned a feature vector of random values. As the corpus is trained, these vectors are adjusted considering the co-occurrence of neighboring words in a sentence such that semantically similar words are placed closer in the vector space. Comparatively, our context relates to the spatio-temporal rather than vocabulary. With these steps, we transform the raw trajectory coordinates into a vector representation that can be fed to the model. This process is called “embedding.”

A drawback of this approach is that the number of cells can grow very large. In order to maintain satisfactory model efficiency while losing as little information as possible, we only keep cells that are hit by a certain number of trajectory points. These cells are referred to as hot cells and the minimum number of hits is referred to as the hot-cell threshold.

This new data representation addresses the "temporal differences between trajectories" challenge and "GPS errors" challenge (Challenges 1 and 3 in Section 1) in the following way:

— Challenge 1. Temporal differences between trajectories. Each spatio-temporal cell covers a pre-defined temporal span (i.e., a 15-minute time span). Thus, two trajectory points that are only a few minutes apart are still treated as the same trajectory point because they are still in the same time span and thus will be represented with the same vector embedding.
— Challenge 3. GPS errors. Each spatio-temporal cell covers a pre-defined spatial area (i.e., a 500 meters × 500 meters area). When a trajectory recording is subject to spatial noise, which is minor relative to the spatial area, it is unlikely that the distorted point will move to another cell. Thus, even with the noise, the distorted trajectory points will still be represented with the same vector embedding as the original.

4.2 Training Data Modeling

One of the main challenges of the similar trajectory search problem is the lack of dataset with well-defined query and ground truth trajectory pairs. As a workaround, we adapt the approach used by Li et al. [14] to produce synthetic query and ground truth pairs. However, we also add one more type of distortion, the temporal distortion, to this approach. For a "ground truth" trajectory Y, we create a copy of it called the "query" trajectory X and apply several distortions to that copy. This process contributes to the solving of the three challenges by introducing synthetic distortions and downsampling that reflect the real-life data imperfections. These distortions are as follows:

— **Trajectory downsampling.** We randomly remove some points from the copy. The number of points removed is controlled by the parameter downsampling rate d_r. The purpose of this distortion is to simulate the non-uniform sampling rate in real data, relevant to Challenge 2.
— **Spatial distortion.** We randomly select some points from the copy and shift the spatial coordinates to a random location within 30 meters from the original. The number of points to be selected is controlled by the parameter spatial distortion rate d_s. The purpose of this distortion is to simulate real GPS errors, relevant to Challenge 3.
— **Temporal distortion.** For all points in a trajectory, we add or subtract a random number of minutes. All points will receive the same temporal distortion to ensure that the travel speed and point ordering do not change. The maximum number of minutes is controlled by the parameter temporal distortion d_e. The purpose of this distortion is to simulate temporal variation between trajectories, relevant to Challenge 1.

Finally, to improve the training process, for each trajectory, we sample a different, random trajectory from the dataset and apply the same distortions as Y. We call this trajectory as the "negative" trajectory N, which will be used in the representation loss. Next, we will introduce how to train our model, where three loss functions are proposed to make our model robust to the three challenges mentioned in Section 1.

4.3 Representation Loss

This loss function operates directly on the vector representation. We perform the token embedding lookup shown in Figure 5 for the three trajectories X, Y, and N. Then, we feed the trajectories to the encoder, shown in the bottom part of Figure 4, to produce their vector representations VX,

VY, and VN. Training the model to produce VX and VY, which are similar to each other, makes it more robust to the three distortions mentioned before. This is because the query trajectory X is derived from the ground truth trajectory Y. Intuitively, they should have similar representations. By introducing some distortions to X and downsampling it, we are training the model to recognize that trajectory X and Y are similar despite the distortions and non-uniform sampling rate.

While we can use only X and Y for the training, it is not an optimal approach because simply training the model to learn similar VX and VY allows the model to understand the concept of spatio-temporal trajectory similarity, but not spatio-temporal trajectory dissimilarity. That is, it can learn which spatio-temporal cells are close to which, but cannot learn to assign a large distance (or small similarity value) to cells that are far apart. To address this problem, we implement the triplet loss function [4] with negative sampling. The triplet loss function takes three inputs: the anchor sample, the positive sample, and a negative sample. Here, we use the query trajectory X as the anchor sample, ground truth trajectory Y as the positive sample, and N as the negative sample. We use the three samples' vector representation for the equation below.

$$\mathcal{L}_1(VX, VY, VN) = max(\|VX - VY\|_2 - \|VX - VN\|_2 + \gamma, 0), \tag{2}$$

where VX, VY, and VN are the vector representation of the query, ground truth, and negative trajectory, respectively, and γ is a hyperparameter that defines the margin between the anchor-positive pair and anchor-negative pair. This loss function helps our model to identify that the anchor-positive pairing is more similar compared to anchor-negative, which enables our model to learn both the concept of spatio-temporal similarity and dissimilarity for both the trajectories and the spatio-temporal cells. This loss function addresses the challenges mentioned in Section 1 in the following way:

- Challenge 1. Temporal differences between trajectories. We apply the temporal distortion to the query trajectory and train our model to identify the query and the ground truth as similar trajectories despite the distortion, making our model more robust to these temporal differences. In addition, the negative sampling approach improves the training by introducing the concept of dissimilarity for the different spatio-temporal cells.
- Challenge 2. Non-uniform sampling rate. We train our model to produce similar representations for the query and ground truth trajectories. Since the query trajectory is a copy of the ground truth trajectory after the trajectory downsampling, we make our model more robust to this downsampling and thus, robust to differing sampling rate between trajectories.
- Challenge 3. GPS errors. Since we apply the spatial distortion to the trajectories, training our model to produce similar representations for the query and ground truth will increase its robustness to GPS errors. Also, using the triplet function and negative sampling, we train our model to identify dissimilar trajectories, consequently training our model to assign different vector representations for different spatio-temporal cells.

4.4 Point-to-Point Loss

Defining the similarity of two vector representations solely by the Euclidean distance between them is not enough. In order to improve the training performance, we also test the vector representation's performance in reconstructing the ground truth by using the decoder. As seen in Figure 4, we feed the vector representation VX to the decoder, producing an output prediction sequence. In this subsection, we refer to a "point" as the contents of the ground truth trajectory Y (e.g., y_1, y_2) and predicted trajectory Y' (e.g., y'_1, y'_2). These points are called the ground truth point and predicted point, respectively. We will then compare Y' to Y to see how well our model recreates the input.

The typical way to measure the reconstruction accuracy is to calculate the similarity between the original and reconstructed sequence. A common, NLP-inspired approach is to model the decoding accuracy by how well the model predicts the correct cell ID (or in the context of NLP, predicting the correct word). This step requires a softmax layer on top of the decoder which outputs one probability for each cell. For instance, if there are five spatio-temporal cells and the predicted point at time t has a cell ID of 3, then the ideal softmax output probability will be [0, 0, 1.0, 0, 0]. Unfortunately, this approach has two major flaws:

—Since the number of cells can be very large, a softmax output needs to be generated for every cell, making the model unwieldy and slow to operate. For instance, if there are 100,000 cell IDs, which is a reasonable number for this task, for a ground truth trajectory of length 30, we need to output 3 million softmax probabilities.

—The training does not take into account the spatio-temporal proximity of each cell. This means that two wrong predictions may be penalized equally even if one of them should be penalized heavier than the other. For instance, if a target cell ID has a time range of 00:00 to 00:15 and the two wrong predictions have a time range of 00:15 to 00:30 and 11:15 to 11:30, both wrong predictions may be penalized equally even though the second prediction is much further from the target. This is because softmax does not take into account the spatial and temporal properties of the data; it only identifies when a prediction is right or wrong.

In order to address this problem, we adapt a sampling-based approach. For every spatio-temporal cell, we find its k-nearest cells and calculate a "weight" value for each. The closer the cells are, the greater the weight. Then, we replace the task of predicting the correct cell ID to the task of predicting these k weights. For instance, if the ground truth point has a cell ID of 7 and we set k to be 5, we may find that the nearest cells of cell ID 7 have the IDs [11,13,2,6,10]. We then calculate the weight of cell 7 and these five cells and find the values [0.45, 0.33, 0.2, 0.15, 0.09]. Thus, instead of using the softmax prediction to output probabilities for all cell IDs, we only predict these five values. In our experiments, we set k to be 10. Using this setting, if we have 100,000 spatio-temporal cells and an output trajectory of length 30, instead of predicting 3 million softmax outputs, we only predict 300 cell weights. The spatio-temporal weight generation process is detailed in Algorithm 1.

ALGORITHM 1: k-nearest spatio-temporal cell lookup

Input
  C List of all spatio-temporal cells
  θ Spatial scaling parameter
Output
  C^k A dictionary of all spatio-temporal k-nearest cells
$C^k \leftarrow \{\}$
for $c \in C$ **do**
  $W \leftarrow []$
  $G \leftarrow find\_k\_nearest\_cells(c)$
  for $g \in G$ **do**
    $d \leftarrow \|c.centroid - g.centroid\|_2$
    $w \leftarrow exp(-d \times \theta)$
    $W.append(w)$
  for $w \in W$ **do**
    $w \leftarrow w/sum(W)$
  $C^k[c.id] \leftarrow W$
return C^k

For every cell in the dictionary, we first find its k-nearest cells and then we calculate the weight of all k cells in lines 10–15. The closer the centroids of two cells are, the higher the weight. The exponential function and the θ parameter are adapted from [14]. The exponential function encourages the model to assign a greater weight to closer cells and θ further controls the distance-to-weight conversion. A small value penalizes far away cells and vice versa. We use a θ value of 0.8. Using this new representation, we calculate the loss function between the target trajectory Y and the predicted trajectory Y'. With the dictionary C^k defined, we can use it to transform our target trajectory Y. Initially, it is a sequence Y of spatio-temporal cell IDs. Now, we can use these IDs to look up C^k and find the k weights of each ID, resulting in the sequence Y^k, consisting of points y_t^k. The output from the model $Y^{k'}$ will be compared with Y^k using the loss function defined below. To get $Y^{k'}$, consisting of points $y_t^{k\prime}$, we utilize the fully connected layer in our GRU Sequence Autoencoder that is connected to the preceding hidden layer. The loss is then calculated as below:

$$\mathcal{L}_2(Y^k, Y^{k\prime}) = \sum_{t=1}^{T} \left\| y_t^k - y_t^{k\prime} \right\|_2, \tag{3}$$

where Y^k is the transformed ground truth trajectory, $Y^{k\prime}$ is the predicted trajectory, and T is the length of Y^k and $Y^{k\prime}$. Using this new representation addresses both of the aforementioned problems with using the softmax output. Unlike the softmax function that requires calculating the probability of every cell, we are only concerned with each cell's k-nearest neighbors and we also penalize further cells greater than closer ones. Additionally, this loss function addresses the three challenges mentioned in Section 1 in the following way:

- Challenge 1. Temporal differences between trajectories. Using this representation, we avoid calculating the raw timestamp differences. Also, if two trajectory points are close enough in both the space and time dimension, they will be represented with the same k-cell weights.
- Challenge 2. Non-uniform sampling rate. This loss function trains the sequence autoencoder to reproduce the original trajectory input given the downsampled query trajectories. This process can be seen as a trajectory imputation, which addresses this challenge by implicitly upsampling the trajectories and then performing the point-to-point comparison.
- Challenge 3. GPS errors. We predict a cell's k-nearest cells weight for every output as opposed to comparing the raw geographical distances directly. This means that if a trajectory point of the query is close enough to the ground truth, they will be classified as the same cell. Thus, we do not allow the spatial noise to factor into the loss computation.

4.5 Pattern Loss

To the best of our knowledge, there is no prior work that has explored the concept of trajectory movement patterns; they rely solely on point-to-point comparisons. Here, we define a trajectory pattern as a sequence of trajectory points that together represent a short-term movement pattern. To assign the sequence of trajectory points to sequences, we use their timestamps. We first divide the full 24-hour range into equal-length and equal-spaced temporal segments. The length of the temporal segments is referred to as the "span" and is controlled by the parameter δ_p. The space between the temporal segments is referred to as the "stride" and is controlled by the parameter δ_t. As an example, setting δ_p to be 30 minutes and δ_t to be 15 minutes, the temporal segments are [00:00-00:30, 00:15-00:45, 00:30-01:00, ...].

We illustrate the process of generating temporal segments using an example trajectory in Figure 6 and setting δ_p to be 30 minutes and δ_t to be 15 minutes. With this setting, there are four temporal segments ts displayed in Table 2. We then generate a trajectory pattern for each segment by calculating the total traveled distance and elapsed time within the segment. The spatial pattern

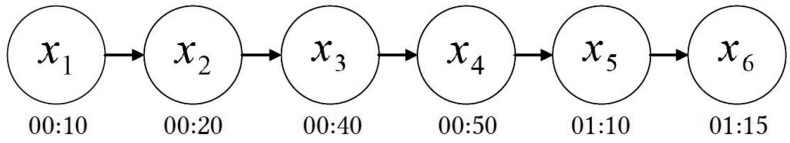

Fig. 6. Example trajectory for the time span division.

Table 2. Assignment of Points to Time Spans

Time span		Points	Timestamp
ts_1	00:00–00:30	x_1	00:10
		x_2	00:20
ts_2	00:15–00:45	x_2	00:20
		x_3	00:40
ts_3	00:30–01:00	x_3	00:40
		x_4	00:50
ts_4	00:45–01:15	x_4	00:50
		x_5	01:10
		x_6	01:15

is defined in Equation (4).

$$ps = 0 + \sum_{t=2}^{T} \left\| x_t^s - x_{t-1}^s \right\|_2, \tag{4}$$

where ps is the spatial pattern of a temporal span ts, T is the number of points in the temporal span ts, and x_t^s is the spatial coordinates of the t-th point of ts. To produce the temporal pattern, we transform the raw timestamp (minutes in a day) to cyclical features. If we do not use the cyclical features and model the timestamps linearly, timestamps going past midnight (e.g., 23.58 and 00.01) will be given a very large distance even though they are only minutes apart. This process requires transforming the raw timestamp x^e into sin and cos features, which are calculated using Equation (5).

$$\begin{aligned} x^{e.sin} &= \left(sin\left(\frac{x^e \times 2\pi}{1{,}440} \right) + 1 \right) \times \frac{1}{2}, \\ x^{e.cos} &= \left(cos\left(\frac{x^e \times 2\pi}{1{,}440} \right) + 1 \right) \times \frac{1}{2}. \end{aligned} \tag{5}$$

The value 1,440 is used as it is the number of minutes in a day. Using this transformation, all sin and cos values will be between 0 and 1. We use these new features as the cyclical timestamp features $x^{e\prime}$ in place of the raw timestamp x^e, and define the temporal pattern below.

$$pe = 0 + \sum_{t=2}^{T} \left\| x_t^{e\prime} - x_{t-1}^{e\prime} \right\|, \tag{6}$$

where pe is the temporal pattern of a temporal span ts, T is the number of points in the temporal span ts, and $x_t^{e\prime}$ is the temporal features of the t-th point of ts. We combine the spatial pattern ps and temporal pattern pe into a combined pattern feature p. We apply this pattern transformation for the ground truth trajectory Y to produce a sequence of patterns PY. To output the predicted trajectory pattern PY', we feed the feature vector representation VX into a small fully connected

layer. The pattern loss compares PY and PY' as below.

$$\mathcal{L}_3(PY, PY') = \sum_{t=1}^{T} \left\| py_t - py'_t \right\|_1, \tag{7}$$

where PY is the ground truth trajectory's pattern features, PY' is the predicted trajectory pattern features, T is the number of trajectory patterns in PY and PY', py_t is the t-th pattern of PY, and py'_t is the t-th pattern of PY'. This loss function addresses the challenges of temporal differences between trajectories and non-uniform sampling rate in the following way:

- Challenge 1. Temporal differences between trajectories. The division of trajectory points to patterns rely on the timestamp. Intuitively, trajectories that are similar to each other also have a similar temporal span. Thus, even if two trajectories start at a slightly different time, as long as they span the similar duration, they will be treated as being similar.
- Challenge 2. Non-uniform sampling rate. Since we use the timestamp to divide trajectories to patterns, regardless of the difference in the number of points, similar trajectories will be represented with the same number of patterns. Additionally, if we consider trajectories as patterns instead of points, no matter how different the number of points are, the spatial and temporal movement pattern are still preserved.

These three losses are used to update the network. We use the backpropagation algorithm, which utilizes the gradient of the loss functions. The representation loss and the pattern loss updates the encoder while the point-to-point loss updates both the encoder and the decoder. By using a vector representation, we transformed the task of similar trajectory search into a task of computing Euclidean distance between vectors, reducing the computational complexity from $O(n^2)$ to $O(n)$.

Summary. In this section, we first described our data modeling. We then show, step-by-step, how our model performs a similar trajectory search. Finally, we showed the the representation loss, point-to-point loss, and the pattern loss that each contribute differently for the training.

5 EXPERIMENTS

In this section, we will first describe our experimental setup, including the dataset used, the baseline models used for comparison, and our model's hyperparameters. Afterward, we will show and analyze the results of three sets of experiments, which answer the following research questions:

RQ1. How much of an improvement does our model achieve over existing models? To answer this question, we compare our method with two classes of models: spatial-only deep neural network model, and traditional spatio-temporal similarity metric. More details and results are available in Section 5.2.

RQ2. How efficient is our model? To answer this question, we analyze the time taken for our model to complete a similar trajectory search query and compare it with the baseline models. More details and results are available in Section 5.3.

RQ3. What are the impacts of different hyperparameters, the three loss functions we developed, and the usage of both spatial and temporal aspects of the data? To answer this question, we perform a comprehensive ablation test on our model and assess the performance of each permutation. More details and results are available in Section 5.4.

Table 3. Porto and Chengdu Dataset Details

	Porto	Chengdu
Time range	July 2013–June 2014	1 Nov 2016–7 Nov 2016
Sampling rate	15 seconds	2–4 seconds
# of trajectory	1.2 million	720,000
Avg. # of points in trajectory	57 points	273 points

5.1 Experimental Setup

Dataset. We use two datasets: the Porto taxi dataset[1] same as [14], and the Didi Chengdu dataset.[2] For both datasets, we prune trajectories that are overly short or long because they are likely to be erroneous recordings. For the Porto data, we prune trajectories that are shorter than 30 and then remove the 1% longest trajectory, resulting in the removal of trajectories longer than 206. Since the Chengdu dataset is sampled at about five times the rate of Porto, we multiply the minimum and maximum trajectory length by 5, thus removing trajectories shorter than 150 and longer than 1,030. We use a span δ_p of 30 minutes and stride δ_t of 15 minutes for both. More details are in Table 3. For the Chengdu data, we only report the efficiency experiments and some effectiveness experiments due to space limits and due to how all models perform similarly. The similar performance is due to the Chengdu data being more concentrated, with many trajectories covering the same roads compared to Porto's more balanced distribution. This can be seen in the two datasets' heat maps in Figures 7 and 8.

Preparation of test data. As there is no ground truth data for similar trajectory search, we have to synthetically generate it. For this task, we adopt the setup used in the state-of-the-art [14]. We select a number of trajectories as the set of query trajectories Q. For the Porto data, this value is 10,000. For the Chengdu data, we reduce this value to 1,000 due to its more granular sampling rate and longer trajectories. We then select a set of m (parameter to be evaluated) trajectories denoted by P. For each trajectory $T_b \in Q$, we create two subtrajectories T_a and T'_a by alternatively taking points from it. To simulate noisy trajectory data, we apply the downsampling, spatial distortion, or temporal distortion, described in Section 4.2, to T_a. We collect both sets of subtrajectories into two datasets $D_Q = \{T_a\}$ and $D'_Q = \{T'_a\}$. We use the same process on P to produce two datasets, D_P and D'_P. Finally, we use D_Q as the final set of **query trajectories** and $D'_Q \cup D'_P$ as the final set of **database trajectories** $\mathcal{Y}$. We assess the model's performance on different downsampling, spatial distortion, and temporal distortion rate in Section 5.2. The intuition is to simulate performing a similar trajectory search over a trajectory database. For each query trajectory $T_a \in D_Q$, its other half T'_a resides in the database $\mathcal{Y}$ and is a synthetic ground truth that should be ranked highly when we perform a similar trajectory search. The three distortions introduced to T_a simulate the noisiness of real trajectory data. The greater the three distortions and the larger the database size (controlled by m), the more difficult this task becomes.

Preparation of training data. For the training of our model, we select 500,000 trajectories as the set of query trajectories Q. For every trajectory $T_b \in Q$, we apply the downsampling with rates [0, 0.2, 0.4, 0.6], spatial distortion with rates [0, 0.2, 0.4, 0.6], and a temporal distortion rate of 15 minutes. Thus, for every T_b, we generate 16 matching distorted trajectories T'_b.

Evaluation metric. To compare our model with all baselines, we use the mean rank as the evaluation metric. Specifically, we use every trajectory $T_a \in D_Q$ to query the database and find the rank

[1] http://www.geolink.pt/ecmlpkdd2015-challenge/dataset.html.
[2] https://outreach.didichuxing.com/appEn-vue/personal?id=2.

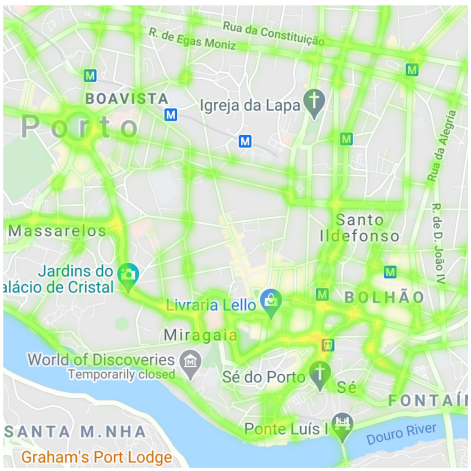

Fig. 7. Heat map of the Porto data.

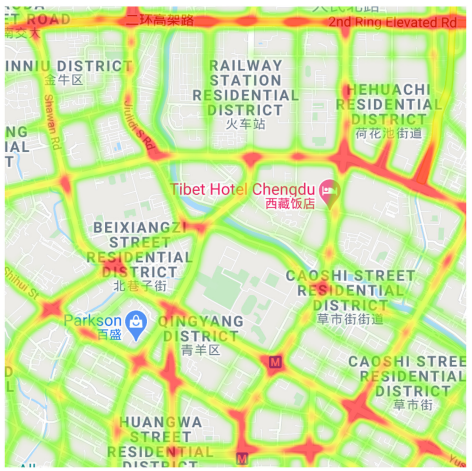

Fig. 8. Heat map of the Chengdu data.

of every matching trajectory $T'_a \in \mathcal{Y}$. We then calculate the mean of these ranks. This metric is also used in [14] and is useful as it directly reports the model performance in real applications.

Baselines. We categorize the baseline models into two categories: spatial-only deep neural network and spatio-temporal traditional.

(1) **Spatial-only deep neural network.** We use the state-of-the-art t2vec model [14] as a baseline. Similar to our work, t2vec first transforms input trajectories to a vector representation and performs similarity search using this new representation. t2vec uses a Sequence Autoencoder with three GRU layers for the encoder and three GRU layers for the decoder. These GRU layers have a hidden unit size of 256 and are trained using the Adam optimizer with an initial learning rate of 0.001.

(2) **Spatio-temporal traditional.** We use several traditional trajectory similarity metrics for comparison, which are DTW [3], EDR [6], ERP [5], and LCSS [26]. Since they were designed for a 1D time series, we extend them for spatio-temporal data by performing two separate distance calculations on the spatial and temporal features, and then combine the score by using a balancing factor α. More specifically, we use the following formula: $d(t_a, t_b) = \alpha \times d_s(t_a, t_b) + (1 - \alpha) \times d_e(t_a, t_b)$. Here, d is the combined distance function, (t_a, t_b) is a pair of input trajectories, α is the balancing factor, d_s is the spatial trajectory similarity function, and d_e is the temporal trajectory similarity function. The spatial and temporal trajectory similarity function can be any one of the four traditional similarity metrics we mentioned. For instance, if we are using DTW, the formula is $DTW(t_a, t_b) = \alpha \times DTW_s(t_a, t_b) + (1-\alpha) \times DTW_e(t_a, t_b)$, where DTW_s (DTW_e) is the DTW applied to the spatial (temporal) features of the input trajectory pairs. We refer to the spatio-temporal extensions of these four methods as ST-DTW, ST-EDR, ST-ERP, and ST-LCSS. We also Z-normalize all three dimensions of the data. Finally, we set the hyperparameter ϵ for the matching threshold in ST-EDR and ST-LCSS to a quarter of the maximum standard deviation, as recommended by Chen et al. [6].

Hyperparameters. We categorize our model's hyperparameters and describe them each below:

(1) **Sequence autoencoder parameters.** Our GRU Sequence Autoencoder model with attention uses a stack of three GRU layers for the encoder and another stack of three for the decoder.

Table 4. Mean Rank on Different Database Sizes

Database size	20k	40k	60k	80k	100k
t2vec	2.1294	3.1695	4.2832	5.5080	6.5470
ST-DTW	1.3122	1.4988	1.5824	1.5918	1.6854
ST-EDR	1.0281	1.0478	1.0528	1.0518	1.0611
ST-ERP	1.0050	1.0104	1.0124	1.0187	1.0245
ST-LCSS	1.0318	1.0488	1.0637	1.0751	1.0812
Our model	1.0132	1.0193	1.0230	1.0262	1.0278

All GRUs have a cell size of 256. The output vector representation for the trajectories have a size of 512.

(2) **Spatio-temporal cells.** Every spatio-temporal cell has a spatial coverage of 500×500 meters and a temporal coverage of 30 minutes. We use a hot-cell threshold value of 10, i.e., cells that are hit by fewer than 10 trajectory points are removed. This leaves 73,310 remaining hot cells for the Porto data and 13,872 for Chengdu. The Chengdu dataset's smaller number of cells is due to the much smaller spatial region compared to Porto.

(3) **Model training.** We train our model for one epoch through the dataset and use the Adam optimizer [13] with the recommended setting: initial learning rate = 0.001, $\beta_1 = 0.9$, $\beta_2 = 0.999$. We also employ a learning rate decay that linearly shrinks the learning rate after every training sample to a minimum of 0.0001. With these parameter settings, it takes our model about 12.5 hours to train for the Porto data and 51 hours for the Chengdu data.

5.2 Evaluation of Model Effectiveness

In this section, we perform four sets of experiments with two main purposes. Firstly, we would like to assess the performance improvements when using both the spatial and temporal feature of trajectory data. To answer this question, we compare our model with a state-of-the-art model t2vec [14], which also uses a deep representation learning approach, but only the spatial feature of the data. Secondly, we would like to assess the performance improvements of using a deep representation learning approach over the traditional point-to-point approach. To answer this question, we compare our model with our four spatio-temporal baselines: ST-DTW, ST-EDR, ST-ERP, and ST-LCSS. For all ST models, we iterate through an α value of {0.5, 0.6, 0.7, 0.8, 0.9, 1}. We present an analysis on the impact of α in Section 5.5.

The four experiments evaluate the effectiveness of our model w.r.t. the varying database size, downsampling rates, spatial distortion rates, and temporal distortion rates, respectively. For the Chengdu data, we only report the downsampling experiment results due to space constraints. For the database size experiment, we vary the database size m with the values {20k, 40k, 60k, 80k, 100k} while for the remaining experiments, we keep m as 100k. For the downsampling experiment, we vary the rate d_r to {20%, 30%, 40%, 50%}. For the spatial distortion experiment, we vary the rate d_s to {20%, 30%, 40%, 50%}. Finally, for the temporal distortion experiment, we vary the rate d_e to {5 minutes, 10 minutes}. We only use small d_e because the further apart in time two trajectories are, the less likely they are to be similar. A small value of 5 and 10 minutes are still close enough to treat two trajectories as being similar.

5.2.1 *Experiment Results.*

Database size experiment. From the results shown in Table 4, we find our model scales much better compared to t2vec. It confirms that including the temporal information adds another

Table 5. Mean Rank on Different Downsampling Rates (Porto)

Downsampling rate	20%	30%	40%	50%
t2vec	6.8838	7.3831	8.0531	10.0292
ST-DTW	1.4487	1.3956	1.4122	1.5696
ST-EDR	2.6815	4.6804	12.1010	23.2293
ST-ERP	2.2195	3.3803	6.3838	14.1944
ST-LCSS	2.2051	2.7106	3.1864	4.0918
Our model	1.1140	1.2984	1.5510	1.9802

Table 6. Mean Rank on Different Downsampling Rates (Chengdu)

Downsampling rate	20%	30%	40%	50%
t2vec	10.0820	10.1650	10.3720	10.6300
ST-DTW	1.0000	1.0000	1.0010	1.0000
ST-EDR	1.1820	1.4620	1.6570	2.1430
ST-ERP	1.0350	1.1190	1.3570	1.7160
ST-LCSS	1.4310	1.6190	1.9340	2.0560
Our model	1.0020	1.0030	1.0060	1.0270

Table 7. Mean Rank on Different Spatial Distortions

Spatial distortion rate	20%	30%	40%	50%
t2vec	6.9866	7.0486	6.6526	6.3305
ST-DTW	1.8391	1.8375	1.8353	1.8350
ST-EDR	1.0683	1.0697	1.0681	1.0680
ST-ERP	1.0146	1.0143	1.0136	1.0150
ST-LCSS	1.0940	1.0944	1.0952	1.0955
Our model	1.0359	1.0312	1.0298	1.0324

Table 8. Mean Rank on Different Temporal Distortions

Temporal distortion rate	5 minutes	10 minutes
t2vec	-	-
ST-DTW	3.1974	12.3873
ST-EDR	1.5072	5.2892
ST-ERP	3.4112	14.8540
ST-LCSS	1.6871	3.6839
Our model	1.6746	3.8275

dimension to the data, which helps it distinguish between trajectories that are only spatially similar, but have significantly different timestamp. Compared to the four spatio-temporal baselines, we managed to outperform ST-DTW, ST-EDR, and ST-LCSS while losing slightly to ST-ERP.

Downsampling rate experiment. This experiment assesses each model's performance in addressing Challenge 2 outlined in Section 1, *non-uniform sampling rate*. The results are displayed in Tables 5 and 6. Our model managed to outperform t2vec on all experiments. Our model also managed to greatly outperform ST-EDR and ST-ERP on the Porto data. In the 50% drop rate experiment, the most difficult task of the experiments, they achieved a mean rank about 12 times and 7 times worse than our model, respectively. On the Chengdu dataset, all spatio-temporal models perform relatively well.

Spatial distortion experiment. This experiment assesses each model's performance in addressing Challenge 3 outlined in Section 1, *GPS errors*. The results are displayed in Table 7. Our model manages to outperform t2vec, showing the importance of using temporal features of the data. Amongst the other baselines, ST-DTW performed the worst.

Temporal distortion experiment. This experiment assesses each model's performance in addressing Challenge 1 outlined in Section 1, *temporal differences*. We exclude t2vec as it is spatial only. Table 8 shows the results. Our model slightly loses to ST-EDR in the 5-minute experiment and ST-LCSS in the 10-minute experiment, but performed better than the other models in the remaining experiments, with ST-DTW and ST-ERP lagging behind in performance.

5.2.2 Experimental Analysis on Effectiveness. From the results of these four sets of experiments, we can see that even though our model did not outperform the baselines on all experiments, it offers by far the most consistent performance. The state-of-the-art model t2vec uses only the spatial

feature of the data and thus its performance lags behind the other spatio-temporal baselines. Next, we conduct a further analysis on the pros and cons of each spatio-temporal baseline.

ST-DTW Evaluation. ST-DTW is robust to non-uniform sampling rates, but amongst the spatio-temporal methods, it performed relatively poorly on the remaining experiments. ST-DTW's relatively poor performance on the spatial and temporal distortion experiments is caused by its usage of the actual distance between the trajectory points. Thus, the distance computation takes into account the spatial noise and the temporal differences between the trajectories, meaning it is not robust to these noises. It is also not very scalable to large datasets relative to the other models.

ST-EDR Evaluation. ST-EDR is robust to spatial distortions, scales well, and has decent robustness to temporal distortions, but it performs poorly w.r.t. non-uniform sampling rate because of the edit distance calculation. Recall that given a pair of query and ground truth trajectory X and Y, EDR performs a set of insertion, deletion, and replacement operations to transform X to Y. As the number of points removed from both trajectories increases, the number of operations to do this transformation from X to Y dramatically increases. Even if Y is the ground truth trajectory for X, EDR can mistakenly assign a very high edit distance for this pair of trajectories. As can be seen from the result, the performance rapidly degrades as the downsampling rate increases.

ST-ERP Evaluation. ST-ERP has good scalability and handles spatial distortion well, but it does not handle non-uniform sampling rate and temporal distortion well. At its core, ERP is a metric that is still based on edit distance, similar to EDR. The difference is that instead of counting the number of operations to match the two trajectories, ERP compared the distance of a point to a gap element if the matching point is not close enough. In the case of the downsampling experiment, when the number of non-matching points increases due to a large number of points being removed from a matching X and Y, the performance rapidly degrades as the distances between the gap element to both trajectories are accumulated, even though this Y might be the ground truth for X. Additionally, ST-ERP performed the best in the spatial distortion experiment, but the worst in the temporal distortion experiment, showing that the method of using gap element does not perform well on the 1D temporal feature relative to the 2D spatial feature.

ST-LCSS Evaluation. ST-LCSS has decent scalability and robustness to spatial and temporal noises, but it is not particularly robust non-uniform sampling rates. The performance improvements of ST-LCSS relative to the edit-distance-based functions on the downsampling experiment can be attributed to the fact that LCSS counts the number of matching points in X and Y in the same relative order, but not necessarily contiguous. Edit-distance-based functions incur a large penalty if there is a significant gap between two matching points' position in the trajectory, e.g., if x and y is a match where $x \in X$ is in the 5-th position while $y \in Y$ is in the 11-th position. LCSS ignores this gap, which means that the downsampling rate has a much reduced impact to its performance. However, LCSS can still falter when a non-matching trajectory has more matching points compared to the ground truth due to the trajectory length. For instance, consider a pair of trajectories X and Y both with a length of 5 and LCSS assigns a value (recall that LCSS measures trajectory similarity, so the larger the better) of 4, which is large relative to the trajectory length. An arbitrary trajectory Y' with a length of 25 can still be counted as X's most similar trajectory if the LCSS value between them is 5 even though only 20% of the points in Y' are matched, as opposed to Y where there is an 80% match. To summarize, the performance of edit-distance-based methods degrade because it assigns a disproportionately large distance/dissimilarity value for matching trajectories while LCSS degrades because it assigns a disproportionately large similarity value for non-matching trajectories.

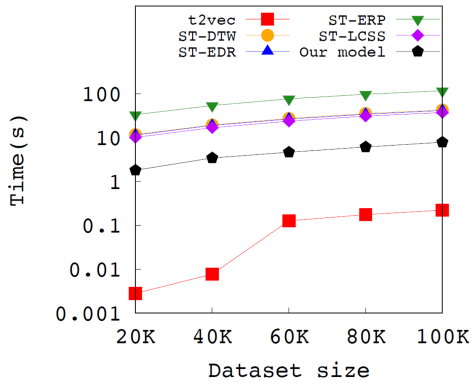

Fig. 9. Efficiency evaluation (Porto).

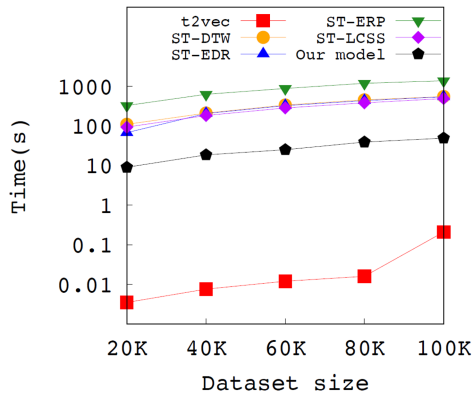

Fig. 10. Efficiency evaluation (Chengdu).

Chengdu result. For the Chengdu downsampling experiment, the performance of all models does not degrade as much as Porto when we increase the downsampling rate. The reason is that Chengdu does not have as many spatio-temporal cells as Porto due to the relatively small area. In addition, the Chengdu data has a more granular sampling rate, making it more robust to downsampling. t2vec performs worse compared to the other models because of the concentrated trajectories, as can be seen in Figure 8. Consequently, there are many trajectories traveling the same route. Without the timestamp information to distinguish the different trajectories, this spatial-only method falters.

Summary. As can be seen from the discussion of these four baselines, each of the four point-to-point based comparisons suffers from unique flaws on different experiments. Thus, in the presence of noisy real data, they do not adapt well. In contrast, our deep-neural-network-based, spatio-temporal model adapts very well to all circumstances and thus is more useful in real application scenarios.

5.3 Evaluation of Model Efficiency

In this section, we evaluate all models' evaluation efficiency. To do this, we report the average time taken to query a database whose size m we vary from 20k to 100k The results are displayed in Figures 9 and 10. The most efficient model is t2vec, followed by our model. Both models use a deep neural network model to transform trajectories from a raw representation to a vector representation and then use a KD-Tree to facilitate faster querying. t2vec is more efficient compared to our model because it uses only the 2D spatial factors while ours uses the 3D spatio-temporal factors.

The traditional point-to-point-based models are significantly slower since they have a computational complexity of $O(n^2)$. Additionally, we need to perform one computation each for the spatial and temporal aspect. To the best of our knowledge, there is no efficiency optimization for multidimensional trajectory distance computations. In the case of DTW, Rakthanmanon et al. [19] proposed the UCR suite to improve its efficiency, but this is only applicable for 1D time series data. Both the Porto and Chengdu data exhibit similar trends.

5.4 Ablation Test

Here, we will explore the impact of each loss function and also test our model's performance given different hyperparameter values. We use the Porto data and perform the downsampling experiment with the rate d_r set to {40%,50%}. We use these two experiments because amongst all experiments, it is the most difficult to improve performance for them. For all the tables in this

Table 9. Our Model's Performance with Different Loss Functions

Losses	Training time	Drop rate	
		40%	50%
$\mathcal{L}_1$	44,820	1.49	2.17
$\mathcal{L}_2$	43,196	1,118.33	1,929.04
$\mathcal{L}_3$	44,320	3,456.34	5,777.53
$\mathcal{L}_{All}$	**43,492**	**1.55**	**1.98**

Table 10. Our Model's Performance with Different Spatial Cell Sizes

Spatial cell size	# of cells	Training time	Drop rate	
			40%	50%
100	521,954	69,140	41.40	87.25
500	**73,310**	**43,492**	**1.55**	**1.98**
1,000	27,735	40,165	1.67	2.19

Table 11. Our Model's Performance with Different Temporal Cell Sizes

Temporal cell size	# of cells	Training time	Drop rate	
			40%	50%
5 min	294,388	48,640	2.48	4.35
10 min	175,790	47,795	2.15	4.38
15 min	127,828	46,315	1.53	2.74
30 min	**73,310**	**43,492**	**1.55**	**1.98**

Table 12. Our Model's Performance with Different Hot-Cell Thresholds

Hot-cells threshold	# of cells	Training time	Drop rate	
			40%	50%
10	**73,310**	**43,492**	**1.55**	**1.98**
50	51,924	43,011	1.73	2.77
100	41,970	41,652	1.93	3.41

Table 13. Comparison of Our Spatio-Temporal and Spatial-Only Models

Model	# of cells	Training time	Drop rate	
			40%	50%
Spatio-temporal	**73,310**	**43,492**	**1.55**	**1.98**
Spatial only	3,727	39,377	13.89	23.92

section, the rows that are bolded signify the default setting we used for all prior experiments. The training time is measured in seconds.

Effectiveness of different loss functions. Table 9 shows our model's performance when using different loss functions. The losses in the table refer to the following: $\mathcal{L}_1$ that uses only the representation loss; $\mathcal{L}_2$ that uses only the point-to-point loss; $\mathcal{L}_3$ that uses only the pattern loss; $\mathcal{L}_{All}$ that uses all losses. The representation loss operates directly on the feature vector representation, directly impacting the evaluation performance and reaching a performance close to our default setting. The point-to-point loss and pattern loss are intended to be supplementary losses and thus, performed very poorly when used on their own. The default setting uses all loss functions, which showed that these supplementary losses indeed help the main representation loss.

Effectiveness of different spatial cell sizes. Table 10 shows the performance of our model on different spatial cell sizes. We can see that using a small cell size results in longer training time yet with worse performance. This is because a large number of cells require a lot more training to properly adjust the vector values of all spatio-temporal cells. This problem is compounded by the fact that for each cell, there are significantly fewer number trajectory points that are hit by it. This means that there is insufficient data to train each cell properly. On the contrary, if the size of the cell becomes too large, the performance may degrade because points that are further away are now grouped together in the same cell and are represented with the same feature vector, creating

a lot of false-positive cases where dissimilar cells are treated the same. Thus, the task of finding a proper spatial cell size depends greatly on the dataset and must be done with care.

Effectiveness of different temporal cell sizes. Table 11 shows our model's performance using different temporal cell sizes. We observe the same trend as the spatial cell experiment, in that using a cell size that is too small results in a larger number of cells, longer training time, and worse performance. This value also greatly depends on the dataset. Thus, this feature must also be explored carefully.

Effectiveness of different hot-cell thresholds. Here, we evaluate the impact of different hot-cell thresholds δ. Recall that δ defines how many times a cell must be hit in order for it to be treated as a hot cell. Non-hot cells are then removed from the list of all spatio-temporal cells. The results are displayed in Table 12. The larger the threshold, the fewer the remaining cells, resulting in a shorter training time. The performance worsens as we increase the threshold because when we remove too many points, the trajectories become lossier. As a result, relatively uncommon trajectory points that represent the uniqueness of a particular trajectory may be removed, making it less distinguishable from the other similar trajectories.

Comparison between spatio-temporal and spatial-only model. Finally, we evaluate the effectiveness of using both the spatial and temporal aspects of the data. For this experiment, we modified our data so that it only contains the spatial cells. We do this by increasing the temporal cell size to 1,440, which ensures that there is only one temporal cell for all trajectory points. The result is displayed in Table 13. As can be seen, when we only use the spatial aspect of the data, the performance degrades significantly, showing that we need both aspects.

5.5 Importance of Spatial and Temporal Factors

In this section, we perform a set of exploratory analyses on the four spatio-temporal, point-to-point baseline models ST-DTW, ST-EDR, ST-ERP, and ST-LCSS. The purpose is to demonstrate the importance of utilizing both spatial and temporal factors, as well as to show the sensitivity of these models to the balancing factor α. We use an α value from 0.5 to 1, where a larger α value puts more weight on the spatial aspect of the data. We conduct this experiment on all database size, downsampling rate, spatial distortion, and temporal distortion experiments, but due to space limitations, we only display the results from the 50% drop rate experiments in Table 14. From these experiments, we observe several trends.

Firstly, we find that different models respond differently to increasing amounts of α. The performance of ST-DTW, ST-EDR, and ST-LCSS tend to worsen as we increase α, while ST-ERP's performance tends to improve. In addition, for some models, even a 0.1 increase can result in significant changes; for ST-ERP, the performance improves greatly when α goes from 0.5 to 0.6. We can see that these four baselines are quite sensitive to α. Thus, when using methods that use a balancing factor, we need to iterate over a range of α values to find the best one, adding more overhead to the already time-consuming point-to-point comparisons.

A constant observation from these results is that for all methods, increasing α from 0.9 to 1 dramatically worsens performance. Additionally, with the exception of ERP, this value achieved by far the worst performance out of the remaining α values. When α is 1, the temporal aspect of the data is ignored. Thus, from this observation, we can see that for all models, both the spatial and temporal information is important. For this particular experiment, the best performance tends to be when we use an equal balance between the spatial and temporal aspects of the data, except for ST-ERP which favors a relatively more spatial-oriented α.

Table 14. Experiment Results of All Baselines on Different Choices of α

α	ST-DTW	ST-EDR	ST-ERP	ST-LCSS
0.5	1.5696	23.2293	29.7937	4.0918
0.6	1.6254	22.9506	20.6814	4.1011
0.7	1.6830	23.8497	16.1187	4.1194
0.8	1.7300	26.4362	14.1944	4.1997
0.9	1.7947	32.2287	14.7551	4.6906
1	2.6350	41.5255	19.1225	6.4034
Mean	1.8396	28.3700	19.1110	4.6010

6 CONCLUSION

In this article, we propose a novel spatio-temporal deep representation learning model for similar trajectory search. Our model manages to address the weaknesses of two types of trajectory similarity search methods. Compared to the state-of-the-art deep neural network solution, we utilize both spatio-temporal aspects of the data rather than spatial-only and managed to show significant improvements. Compared to the traditional methods that we extend to handle the spatio-temporal case, we show that our model offers significantly better consistency for all experiments. We also show that these traditional methods are slow and require tuning to find the best spatio-temporal balancing factor α which is unrealistic for end users. On the contrary, our deep representation learning approach can dynamically find the balance between the spatial and temporal aspects.

REFERENCES

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *CoRR* abs/1409.0473. https://arxiv.org/abs/1409.0473.

[2] Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. 2012. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR* abs/1206.5538. https://arxiv.org/abs/1206.5538.

[3] Donald J. Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series. In *KDD*. 359–370.

[4] Gal Chechik, Varun Sharma, Uri Shalit, and Samy Bengio. 2010. Large scale online learning of image similarity through ranking. *Journal of Machine Learning Research* 11 (2010), 1109–1135.

[5] Lei Chen and Raymond Ng. 2004. On the marriage of Lp-norms and edit distance. In *VLDB*, Vol. 30. 792–803.

[6] Lei Chen, M. Tamer Özsu, and Vincent Oria. 2005. Robust and fast similarity search for moving object trajectories. In *SIGMOD*. 491-502.

[7] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, Yu Zheng, and Xing Xie. 2010. Searching trajectories by locations: An efficiency study. In *SIGMOD*. 255–266.

[8] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. CoRR abs/1406.1078. https://arxiv.org/abs/1406.1078.

[9] Tangpeng Dan, Changyin Luo, Yanhong Li, Bolong Zheng, and Guohui Li. 2019. Spatial temporal trajectory similarity join. In *APWeb-WAIM*, Vol. 11642. 251–259.

[10] Felix A. Gers, Fred Cummins, and Jürgen Schmidhuber. 2000. Learning to forget: Continual prediction with LSTM. *Neural Computation* 12 (2000), 2451–2471.

[11] Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[12] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. 2015. Spatial transformer networks. In *NIPS 28*. 2017–2025.

[13] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980. https://arxiv.org/abs/1412.6980.

[14] Xiucheng Li, Kaiqi Zhao, Gao Cong, Christian S. Jensen, and Wei Wei. 2018. Deep representation learning for trajectory similarity computation. In *ICDE 34*. 617–628.
[15] Hui Luo, Zhifeng Bao, Farhana Choudhury, and J. Shane Culpepper. 2019. Dynamic ridesharing in peak travel periods. *IEEE Transactions on Knowledge and Data Engineering* 33, 7 (2021), 2888–2902. https://doi.org/10.1109/TKDE.2019.2961341
[16] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. arXiv:1301.3781. https://arxiv.org/abs/1301.3781.
[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed representations of words and phrases and their Compositionality. arXiv:1310.4546. https://arxiv.org/abs/1310.4546.
[18] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. arXiv:1802.05365. https://arxiv.org/abs/1802.05365.
[19] Thanawin Rakthanmanon, Bilson Campana, Abdullah Mueen, Gustavo Batista, Brandon Westover, Qiang Zhu, Jesin Zakaria, and Eamonn Keogh. 2012. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*. 262–270.
[20] Sayan Ranu, Deepak Padmanabhan, Aditya D. Telang, Prasad Deshpande, and Sriram Raghavan. 2015. Indexing and matching trajectories under inconsistent sampling rates. In *ICDE*. 999–1010.
[21] Shuo Shang, Ruogu Ding, Kai Zheng, Christian Jensen, Panos Kalnis, and Xiaofang Zhou. 2014. Personalized trajectory matching in spatial networks. *The VLDB Journal* 23 (2014), 449–468.
[22] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. 2018. DITA: Distributed in-memory trajectory analytics. In *SIGMOD*. 725–740.
[23] Mohammad Shokoohi-Yekta, Bing Hu, Hongxia Jin, Jun Wang, and Eamonn Keogh. 2016. Generalizing DTW to the multi-dimensional case requires an adaptive approach. *Data Mining and Knowledge Discovery* 31 (2016), 1–31.
[24] Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556. https://arxiv.org/abs/1409.1556.
[25] Han Su, Kai Zheng, Haozhou Wang, Jiamin Huang, and Xiaofang Zhou. 2013. Calibrating trajectory data for similarity-based analysis. In *SIGMOD*. 833–844.
[26] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. 2002. Discovering similar multidimensional trajectories. In *ICDE*. 673–684.
[27] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, and Gao Cong. 2020. A survey on trajectory data management, analytics, and learning. arXiv:2003.11547. https://arxiv.org/abs/2003.11547.
[28] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Timos Sellis, and Xiaolin Qin. 2019. Fast large-scale trajectory clustering. *Proceedings of the VLDB Endowment* 13, 1 (2019), 29–42.
[29] Sheng Wang, Zhifeng Bao, J. Shane Culpepper, Zizhe Xie, Qizhi Liu, and Xiaolin Qin. 2018. Torch: A search engine for trajectory data. In *SIGIR*. 535–544.
[30] Sheng Wang, Yunzhuang Shen, Zhifeng Bao, and Xiaolin Qin. 2019. Intelligent traffic analytics: From monitoring to controlling. In *WSDM*. 778–781.
[31] Zheng Wang, Cheng Long, Gao Cong, and Yiding Liu. 2020. Efficient and effective similar subtrajectory search with deep reinforcement learning. arXiv:2003.02542. https://arxiv.org/abs/2003.02542.
[32] Dong Xie, Feifei Li, and Jeff M. Phillips. 2017. Distributed trajectory similarity search. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1478–1489.
[33] Munkh-Erdene Yadamjav, Zhifeng Bao, Baihua Zheng, Farhana M. Choudhury, and Hanan Samet. 2020. Querying recurrent convoys over trajectory data. *ACM TIST* 11, 5 (2020), 1–24.
[34] D. Yao, G. Cong, C. Zhang, and J. Bi. 2019. Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach. In *ICDE*. 1358–1369.
[35] Haitao Yuan and Guoliang Li. 2019. Distributed in-memory trajectory similarity search and join on road network. In *ICDE*. 1262–1273.
[36] Haitao Yuan, Guoliang Li, Zhifeng Bao, and Ling Feng. 2020. Effective travel time estimation: When historical trajectories over road networks matter. In *SIGMOD*. 2135–2149.
[37] Peng Zhao, Weixiong Rao, Chengxi Zhang, Gong Su, and Qi Zhang. 2020. SST: Synchronized spatial-temporal trajectory similarity search. *GeoInformatica* (2020), 1–24.

Received November 2020; revised March 2021; accepted May 2021