Learning to Hash for Trajectory Similarity Computation and Search

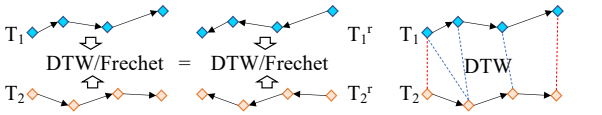
Liwei Deng[1], Yan Zhao[2,✉], Jin Chen[1], Shuncheng Liu[1], Yuyang Xia[1], Kai Zheng[1,✉]
[1]University of Electronic Science and Technology of China, China [2]Aalborg University, Denmark
{deng_liwei, chenjin, liushuncheng, xiayuyang}@std.uestc.edu.cn, yanz@cs.aau.dk, zhengkai@uestc.edu.cn

***Abstract*—Searching for similar trajectories from a database is an important way for extracting human-understandable knowledge. However, due to the huge volume of trajectories and high computation complexity of distance between trajectories, it is difficult to search for exact results, which motivates the research of approximating approaches. In this study, we propose a learning to hash method for trajectory similarity computation and search, called *Traj2Hash*, which consists of a two-channel trajectory encoder and a hash layer to encode trajectories into Euclidean and Hamming space, respectively. The embeddings of trajectories obtained from the encoder are capable of preserving the reverse symmetric property and more representative due to the reverse augmentation and the lower-bound induced read-out layer. Moreover, we design a decomposed grid representation in the encoder to make the model lighter and better. In the model training phase, we combine a weighted mean squared error loss and a ranking-based hashing loss to enable the model similarity-aware and representations self-structured, respectively, in which a fast trajectory triplet generation method is leveraged to enrich the training corpus. Extensive experiments conducted on real data offer evidence of the effectiveness and efficiency of the proposed model.**

***Index Terms*—learning to hash, top-k similar trajectory search, neural-based similarity computation**

I. Introduction

The prevalence of location-acquisition technology results in massive spatial trajectories containing a wealth of mobility information, which benefits various applications, such as trajectory clustering [1], location prediction [2]–[4], and anomaly detection [5]. Searching similar trajectories for a given query in a large trajectory database is an indispensable way to turn the blunt information into knowledge [6]–[12],. For example, Zheng et al. [13] discover gathering patterns among users via searching similar trajectories in spatial-temporal dimensions, which is helpful to reveal the users' implicit social relations. Jin et al. [14] aims to discover the identity relation via linking the same object in different datasets based on the similarity of their movement traces, which is potentially conducive to the criminal investigation. In spite of the importance of similar trajectory search, exact search on a database with massive trajectories is time-consuming due to the huge volume of trajectories and the quadratic computation complexity of distance functions, such as Dynamic Time Warping (DTW) [15], the Frechet distance [16], Edit distance with Real Penalty (ERP) [17]. To accelerate the similar trajectory search, a number of approximation approaches with tolerable search errors are developed [18]–[24].



(a) Reverse Symmetric Property (b) Lower Bound of DTW

Fig. 1. An Example

Existing approximation methods can be roughly divided into two directions. The first direction aims to accelerate the similarity computation, in which neural-based similarity computation methods are representative, such as NeuTraj [22], TrajGAT [24], TMN [25], and TrajSim2vec [23]. These methods employ deep metric learning to encode trajectories into fix-length vectors and approximate trajectory distance via Euclidean distance in latent space. With the representations of trajectories in hand, the complexity of trajectory similarity computation is reduced from O(n^2) into O(d), where n is the mean number of points in trajectories and d is the dimension of the latent space. The second direction aims to reduce the search space via pruning, in which locality sensitive hashing (LSH) is the representative method, such as Fresh [18] and tSTAT [21]. These methods encode trajectories into binary codes and search similar trajectories through table-lookup and hamming distance search. Ideally, the computation complexity of these methods is only O(1) if all the trajectories to be returned are in the same bucket.

Despite their acceptable efficiency and highly approximate searching ratio, these methods still have the following issues. First, all the neural-based similarity computation methods devote to developing a general framework, which ignores the inherent properties of each distance function. For example, as shown in Figure 1(a), the similarity between T_1 and T_2 equals the similarity between T_1^r and T_1^r under the measures DTW or Frechet, where T_1^r is the reversed version of T_1 (called as reverse symmetric property in this work). However, this property does not hold when the measure comes to the existing neural-based methods since the vectors of T_1 and T_1^r cannot be guaranteed the same through the encoding of neural networks. Moreover, the first points (or the last points) distance between T_1 and T_2 is the lower bound of the DTW distance between T_1 and T_2 as shown in Figure 1(b), which cannot be leveraged in existing studies. These discrepancies between the ground-truth distance function and the approximated algorithm hinder the neural-based method to achieve the optimal. Second, the

✉Corresponding authors: Yan Zhao and Kai Zheng.

neural-based methods will calculate all the distances between the query trajectory and the trajectories in the database, which cannot be an efficient way when the latent vectors of trajectories in the database cannot persistently in the main memory. Therefore, a data structure with compact representations to organize the latent space for pruning is needed. In addition, the LSH-based methods for different similarity measures require a specific design to achieve provable collision probability, which is difficult to generalize from one similarity measure to the others. To the best of our knowledge, the existing LSH method for trajectories only can be used when the Frechet distance or Hausdorff distance is used to measure the distance between trajectories.

To solve these problems above, we propose a learning to hash framework, namely *Traj2Hash*, which inherits the advantages of neural-based similarity computation methods and LSH to support more accurate similarity computation (in Euclidean space) or fast approximate top-k similar trajectory search (in Hamming space). Specifically, *Traj2Hash* consists of a trajectory encoder and a hash layer, in which the encoder is used to approximate the similarity between two trajectories and the hash layer is to map these trajectories into hamming space for efficiently searching. For the trajectory encoder, we devise a two-channel mechanism to process the original GPS trajectories and the spatial-aware grid trajectories, respectively. To efficiently model the grid trajectories, we develop a light-weight grid representation encoder equipped with a fast pre-training algorithm based on noise contrastive estimation. In this module, the grid space is decomposed into two single dimensions to reduce the space complexity from O(dN_g^2) to O(dN_g), where N_g indicates the number of grids along with latitude or longitude dimension. For the GPS trajectories, we employ Transformer as the our base model and incorporate the lower-bound information in the pooling operation to obtain more informative trajectory embeddings. Considering the reverse symmetric property in plenty of trajectory measurements, we concatenate the embeddings of the original trajectory and its reversed version (cf. Section III).

In the training phase, we follow the previous studies to compute the pair-wised distance for a seed set of trajectories and use the weighted mean squared error (WMSE) to match the latent Euclidean space with the distance metric space [22]. However, scalability of the capacity of the seed set is constrained due to the expensive computation complexity to obtain the ground-truth distance between trajectories. For example, measuring the distance among 10K trajectories using Frechet distance will cost more than 5 hours under the parallel run with 20 multiprocessors on our server. We argue that limited trajectories are difficult to regularize the Hamming space. To deal with that, inspired by a common rule in these trajectory distance measures that two trajectories locate in nearby region will closer than they not, we propose a ranking-based hash coding objective integrated with a fast triplets generation method to circumvent the exact distance computation between trajectories. Finally, these two objectives are linearly combined to train *Traj2Hash*.

The main contributions of this paper are summarized as follows:

- We propose a learning to hash method for trajectory data to support fast similarity computation and top-k similar search. To the best of our knowledge, *Traj2Hash* is the first learning to hash method for trajectory data. We propose a two-channel trajectory encoder, in which a light-weight grid representation coupled with a fast grids pre-training method is developed and a lower-bound induced read-out layer is specifically designed for DTW and the Frechet distance. In addition, the reverse symmetric property is also preserved through our specific model design.
- We conduct extensive experiments on two publicly datasets, i.e., Porto and ChengDu, compared with six representative baselines by searching similar trajectories in Euclidean space and Hamming space. The results demonstrate the superiority of *Traj2Hash* in terms of efficiency and effectiveness.

II. Related Works

In this section, we survey the related studies on trajectory similarity computation and learning to hash.

Trajectory Similarity Computation. Trajectory similarity computation aims to determine how similar two trajectories are. Traditional trajectory measures, such as DTW and the Frechet distance, usually follow the matching-and-aggregation paradigm based on dynamic programming. Thus, they suffer from quadratic computation complexity, which is intolerable on huge volumes of trajectories to be compared. To accelerate the computation, various approximation algorithms are proposed, which can be roughly divided into two directions, i.e., traditional methods and neural-based methods, according to the learnability. As a type of fast version of DTW, cDTW [26], [27] is a representative method in the first direction, which constrains the window width in the matching process. Despite its effectiveness in some situations, it is short of generalization [28] and accuracy [22] in practice due to the hand-crafted heuristics. To alleviate these problems, the methods in the other direction adopt neural networks to approximate the ground-truth distance, in which the trajectories are encoded into Euclidean space through the metric learning technique. Then, the computation of the original distance such as DTW can be approximated through the Euclidean distance between the encoded vectors. NeuTraj [22] demonstrates around 60$\times$ faster than the original DTW while achieving an acceptable approximation ratio on the Porto dataset. TrajSim2Vec [23] designs an auxiliary loss to simulate the matching process of the dynamic programming-based trajectory measures, such as DTW and Frechet. Recently, TrajGAT [24] is proposed to deal with long trajectories, in which the transformer with PR-quadtree enhanced attention mechanism is used to extract the long-term information. Although these methods perform well in general, they suffer from two shortcomings. First, the properties (e.g., the reverse symmetric property and the lower bound of two trajectories) of each distance function are not

taken into account. In our experiments, we observe that with the guidance of these properties, the model accuracy can be improved. Second, the goal of accelerating the top-k similar trajectory search is to speed up the computation of similarity between trajectories while the searching space is not pruned, which may be impractical in large databases.

Learning to Hash. The hash technique aims to encode objects as integers or into Hamming space, where the objects can be fastly searched through table-lookup. Different from the hash for fast point search, locality sensitive hashing is to support similar search [29]–[34], which designs a hash function with provable collision probability to map the objects within distance r into the same bucket. For instance, Fresh [18] encodes trajectories into integers through random grids shift and multiply-shift hashing, and PM-LSH combines E2LSH [31] and PM-Tree [35] to achieve query-aware hashing for high-dimensional vectors. These methods are data-independent when constructing the hash tables and thus may be poor in efficiency and effectiveness for some datasets, which inspires researchers to develop hash methods based on the data distribution [36]–[41]. For example, HashNet [37] encodes images with similar labels into nearby buckets in Hamming space for fast image retrieval. HashGNN [40] learns the hash coding of items for fast recommendation through retrieval. Despite the numerous learning to hash methods that have been proposed for many data types, such as images and audio, learning to hash for trajectories is still unexplored.

III. Preliminaries

We proceed to present the necessary preliminaries and then define the problem addressed.

A. Basic Concepts

Definition 1 (GPS Trajectory): A GPS trajectory, denoted as T, is a sequence of timestamped locations with the form of $((l_1, t_1), (l_2, t_2), ..., (l_n, t_n))$, where l_i $(1 \leq i \leq n)$ stands for a GPS point coupled with the longitude, e.g., lon_i, and latitude, e.g., lat_i, at timestamp t_i.

In this paper, we only consider the spatial trajectory. Thus, the temporal information, e.g., t_i, is ignored in the following.

Definition 2 (Grid Trajectory): A grid trajectory, denoted as T_g, is a sequence of grids with the form of $(g_1, g_2, ..., g_n)$, where g_i $(1 \leq i \leq n)$ presents the grid number. It is obtained by partitioning the studied space into equal-size grids and mapping the GPS points of T into the located grids.

Figure 2 shows an example of a GPS trajectory T (e.g., the sequence of blue rhombuses) and its corresponding grid trajectory T_g (e.g., the sequence of red rectangles). It should be noted that the grid trajectory T_g can be seen as a coarse version of T, which is helpful in improving the modeling of spatiality [42], [43] and quality of hashing.

Definition 3 (DTW and the Frechet distance): DTW and the Frechet distance are functions based on dynamic programming to measure how the closeness of two trajectories. Their state transition equations can be described as follows:

$$\begin{aligned} D_{i,j} &= min(D_{i-1,j}, D_{i,j-1}, D_{i-1,j-1}) + d(T_1^i, T_2^j) \\ F_{i,j} &= max(min(F_{i-1,j}, F_{i,j-1}, F_{i-1,j-1}), d(T_1^i, T_2^j)) \end{aligned} \tag{1}$$

where p_i and q_j present the i-th and j-th point for the two input trajectories T_1 and T_2, $d(,)$ is the Euclidean distance between GPS points.

Lemma 1 (Lower-bound of DTW): The Euclidean distance between the first points or the last points, e.g., $d(T_1^0, T_2^0)$ or $d(T_1^n, T_2^n)$, of the input two trajectories T_1 and T_2, is the lower-bound of DTW, i.e., $d(T_1^0, T_2^0) \leq DTW(T_1, T_2)$, which always holds for any two trajectories.

Proof. According to the definition of DTW, we can rewrite it as $DTW(T_1, T_2) = \sum_{i=0}^{\max(n_1,n_2)} d(T_1^{i_1}, T_2^{i_2})$, where i_1 and i_2 are the indices to indicate the matching point pair in T_1 and T_2, respectively, and i_1 and i_2 equal to 0 (n_1 and n_2) if $i = 0$ ($i = \max(n_1, n_2)$). After expanding the rewriten equation, we have $DTW(T_1, T_2) \geq d(T_1^0, T_2^0)$ or $DTW(T_1, T_2) \geq d(T_1^{n_1}, T_2^{n_2})$. We refer to [44], [45] for more details.

Note that Lemma 1 also remains valid for the Frechet distance since the first points always match the input two trajectories in the Frechet distance. Although this lower-bound seems loose for pruning, it provides a convenient way to be a guide to summary more representative embedding of trajectories (cf. Section V-D).

Definition 4 (Reverse Symmetric Property): A trajectory distance function D satisfies reverse symmetric property if $D(T_1, T_2)$=$D(T_1^r, T_2^r)$ where T_1^r and T_2^r are the reversed version of T_1 and T_2, e.g., if T=$((l_1, t_1), (l_2, t_2), ..., (l_n, t_n))$, we can get T_r=$((l_n, t_n), ..., (l_2, t_2), (l_1, t_1))$.

Lemma 2: DTW, the Frechet distance, and the Hausdorff distance satisfy the reverse symmetric property.

It should be noted that the encoded vectors from existing neural-based similarity computation methods do not in compliance with this property. For example, assume f_n is a neural network that encode trajectories into vectors. T^* and T_r^* present a trajectory and its corresponding reversed version, where h^{T^*} and $h^{T_r^*}$ are their encoded vectors through f_n. The Euclidean distance (i.e., $E(.,.)$) is used to measure the distance between vectors to approximate the trajectory measures (e.g., DTW and Frechet). For existing methods, $E(h^{T^1}, h^{T^2})$ is not constrained to equal $E(h^{T_r^1}, h^{T_r^2})$, where h^{T^*} and $h^{T_r^*}$ are usually different due to the different input of f_n. Thus, a discrepancy between the ground-truth and the approximated one may hinder these approaches to be optimal. To tackle this problem, we use Lemma 3 to demonstrate our solution.

Lemma 3: Using the concatenation of h^{T^*} and $h^{T_r^*}$ as the final representation, i.e., $h_f^{T^*}$=$[h^{T^*}, h^{T_r^*}]$, of trajectory T^* can satisfy the reverse symmetric property (i.e., $E(h_f^{T^1}, h_f^{T^2}) = E(h_f^{T_r^1}, h_f^{T_r^2})$).

Proof.

$$E(h_f^{T^1}, h_f^{T^2}) = E([h^{T^1}, h^{T_r^1}], [h^{T^2}, h^{T_r^2}]) \tag{2}$$

$$= E([h^{T_r^1}, h^{T^1}], [h^{T_r^2}, h^{T^2}]) \tag{3}$$

$$= E(h_f^{T_r^1}, h_f^{T_r^2}) \quad (4)$$

Equation 3 is based on that simultaneously exchange the dimension of vectors will not change their Euclidean distance. Equation 4 holds since the Since the reverse version of T_r^1 is T^1, and $h_f^{T_r^1} = [h^{T_r^1}, h^{T^1}]$ according to the definition of $h_f^{T^*}$.

Lemma 3 shows that if we augment the original trajectory with its reversed version and concatenate the output of the neural network, we can preserve the reverse symmetric property existed in the to-be-approximated measures. It should be noted that there are other operations, such as element-wise sum [1], that can also help the neural-based approximation approaches to meet the reverse symmetric property. We only consider the concatenation operation in this work.

B. Problem Statement

Given a trajectory database containing N trajectories and a distance function $f(.,.)$ (e.g., DTW, the Frechet distance, or the Hausdorff distance), our problem is to learn a data-dependent and distance-aware hash function to support approximate top-k similar trajectory search, which can be divided into two goals: (1) learning an approximate similarity function $g(.,.)$ to convert trajectories into Euclidean space such that computing $g(.,.)$ is efficient while the difference $|f(.,.) - g(.,.)|$ is minimized; and (2) learning a hash function to structure the latent vectors in Hamming space for efficient search space reduction.

IV. Methodology

In this section, we first give an overview of the framework and then provide specifics on each component in the framework.

A. Overview of Traj2Hash

The framework of *Traj2Hash* is shown in Figure 2. It takes the trajectory triplets generated by the fast triplet generation technique (see Section IV-F) and a set seeds of trajectories as input and consists of five key components, trajectory augmentation, light-weight grid representation encoder, attention-based trajectory encoder, hash layer, and optimization. In the trajectory augmentation, the input GPS trajectory is converted into a grid trajectory, and then the reversed versions of them are both generated. Next, the grid trajectory and the GPS trajectory are put into the light-weight grid representation component and the attention-based trajectory encoder, respectively, to summarize trajectory information. After that, the latent vectors of the grid and GPS trajectories in Euclidean space are delivered into the hash layer to generate the representation of the trajectory. The hash layer combines the representation of the trajectory and its reversed version and then maps them into Hamming space. In the model optimization, *Traj2Hash* combines the metric learning technique and the hash ranking objective. For the metric learning part, we follow NeuTraj [22] to randomly sample a set of trajectories τ as seeds and computes a symmetric $N \times N$ distance matrix D as supervision to enforce similarity of the vectors in Euclidean space to approximate the similarity function. The hash ranking objective equipped with fast triplets generation method is leveraged to structure the vectors in Hamming space, in which plenty of trajectory triplets (e.g., (T, T^p, T^n)) are generated and the objective function is to force the positive trajectory (e.g., T^p) in Hamming space more similar to the anchor trajectory, e.g., T, than the negative one (e.g., T^n).

[1]Actually, the element-wise sum should not be considered when approximating DTW, Frechet, and Hausdorff since it may occur a unexpected property, i.e., $E(h_f^{T^1}, h_f^{T^2}) = E(h_f^{T^1}, h_f^{T_r^2}) = E(h_f^{T_r^1}, h_f^{T^2}) = E(h_f^{T_r^1}, h_f^{T_r^2})$.

B. Trajectory Augmentation

As shown in Figure 2, we firstly augment the GPS trajectory T with grid trajectory T_g, in which the studied space is divided into equal-size grids, and the GPS points in the trajectory are converted into grid identities. The grid trajectory can be regarded as the first hashing for a GPS trajectory, i.e., two trajectories with the same grid sequence indicate the strong similarity between them, which also provides complementary information for the spatiality of the GPS trajectory. From Lemma 2, we can see that the distance function to be approximated satisfy the reverse symmetric property. However, existing neural-based methods for similarity computation do not hold. The conflict between them will degenerate the effectiveness of the approximation. Inspired by Lemma 3, we augment T and T_g with their reversed version T_r and T_{g_r} and parallelly encode these four augmentations into the following components.

C. Light-Weight Grid Representation Encoder

It is obvious that the larger the grid size, the coarse extent the grid trajectory has. The extremely coarse grids will be useless and even harm the performance of modeling spatiality. Thus, the grid size is usually set to relatively small, e.g., 50m×50m [22]. However, such small grid size will result in a huge number of grids, e.g., the Porto dataset is split into around 1.2M (i.e., 1100×1100) grids in NeuTraj [22]. Allocating each grid an independent embedding consume lots of memory and is difficult to be trained with limited training samples. To solve this problem, we propose a decomposed grid representation method. Specifically, instead of allocating each grid an independent embedding, we present each grid with its coordinate and induce the embedding of the grid from the combination of the embeddings of the coordinate. Suppose the coordinate of g_i is (x_i, y_i). The embedding of g_i can be obtained as follows:

$$\boldsymbol{e}_{g_i} = com(\boldsymbol{e}_{x_i}, \boldsymbol{e}_{y_i}) \quad (5)$$

where $\boldsymbol{e}_{x_i}$, $\boldsymbol{e}_{y_i}$, and $\boldsymbol{e}_{g_i}$ are the embeddings of x_i, y_i and g_i, respectively. The com is a combination function. In this paper, we employ the sum operation to aggregate the embeddings of x_i and y_i. With this decomposition representation, the parameters to be learned are the embeddings of the coordinates, i.e., embeddings of x and y rather than g. For example, suppose the studied space is split into 1100×1100 grids. We only need to learn 2×1100 coordinate embeddings rather than 1.2M

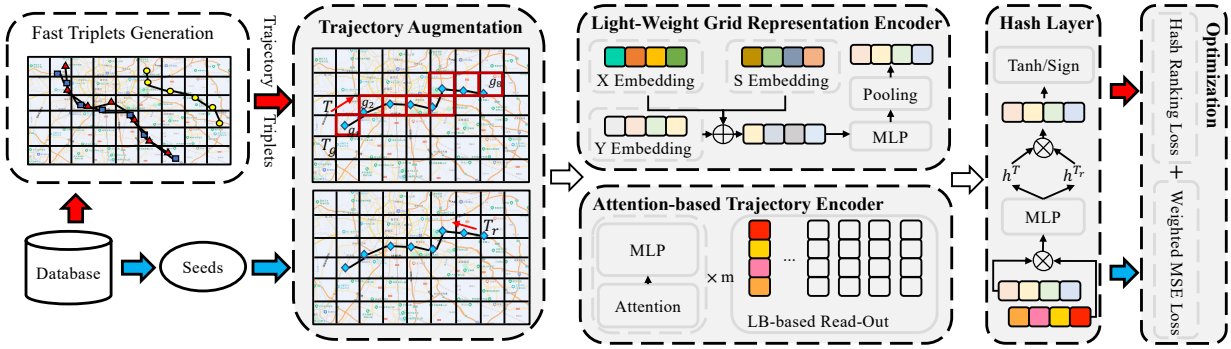


Fig. 2. The proposed framework.

embeddings (i.e., each grid has its independent embedding to be learned). In addition, the spatial proximity among grids can be naturally modeled with the decomposed grid representation to some extent. For instance, the embeddings of two neighbor grids $(3, 5)$ and $(3, 6)$ are similar even without training due to the sharing of the same coordinate embedding (i.e., x_3).

Moreover, to model the spatial relation among grids and relieve the training burden of *Traj2Hash*, noise contrastive estimation (NCE) is leveraged to pre-train the embedding of grids. In particular, we define the grids within r radius as the neighbor of g_i, i.e., $\mathcal{N}(g_i)$, and the others as noise data, denoted as $\mathcal{O}(g_i)$. The training objective of grids can be described as follows:

$$L_g = \sum_{i}^{N_g \times N_g} \sum_{g_i^p \in \mathcal{N}(g_i)}^{N_p} \sum_{g_i^n \in \mathcal{O}(g_i)}^{N_n} \boldsymbol{e}_{g_i}^T \boldsymbol{e}_{g_i^p} + \boldsymbol{e}_{g_i}^T \boldsymbol{e}_{g_i^n} \tag{6}$$

where $\boldsymbol{e}_{g_i} \in \mathsf{R}^{d \times 1}$ is the embedding of g_i according to Equation 5, N_p and N_n indicate the number of sampled neighbors and noises, in which the noise data is uniformly sampled from $\mathcal{O}(g_i)$ and the neighbor grids can be fastly sampled due to the decomposed representation of g_i as follows:

$$g_i^p = (x_i + x_s, y_i + y_s) \tag{7}$$

where x_s and y_s are uniformly sampled from $[1, r]$. After the pre-training procedure, the grid embeddings are frozen, which means that they are not updated in the later optimization component since the spatial information may be poisoned after updating.

Next, to obtain the representation of a grid trajectory efficiently, we devise a light-weight grid aggregation method. Specifically, we first adopt the positional encoding [46] to integrate the sequential information.

$$\begin{aligned} \boldsymbol{s}_i(2k) &= \sin(i/10000^{2k/d}) \\ \boldsymbol{s}_i(2k+1) &= \cos(i/10000^{2k/d}) \end{aligned} \tag{8}$$

where $\boldsymbol{s}_i$ presents the embedding of the i-th position in a sequence, and k indicates the k-th dimension of the latent vector. Then, the representation of i-th term in grid trajectory with positional encoding is obtained through the sum operation, i.e., $\tilde{\boldsymbol{e}}_{g_i} = \boldsymbol{e}_{g_i} + \boldsymbol{s}_i$. After that, the representation of the grid trajectory can be described as follows:

$$\begin{aligned} \boldsymbol{h}_{g_i} &= \mathit{MLP}_g(\mathsf{e}_{g_i}) \\ \boldsymbol{h}_g &= \mathit{Pooling}([\mathsf{h}_{g_1}, \quad, \mathsf{h}_{g_n}]) \end{aligned} \tag{9}$$

where MLP_g is a two-layer full-connected network with ReLU as activation and $\mathit{Pooling}$ is to aggregate all the embeddings in the grid trajectory. In this paper, we adopt $Mean$ as the pooling operation due to its simplification and the preservation of the scale of embeddings.

D. Attention-based Trajectory Encoder

Although the information from the grid trajectory can well distinguish the distant trajectories and the closed ones, the details of the trajectory are dismissed, which impedes the model to achieve a highly accurate approximation to the ground-truth distance. To deal with this, an attention-based trajectory encoder is developed to model the fine-grained information from the GPS trajectory. Specifically, we extract the location feature from two-dimensional GPS space, e.g., $l_i = (lat_i, lon_i)$, as follows:

$$\boldsymbol{e}_{l_i} = \mathit{MLP}_e(\mathit{Normalize}(\mathsf{lat}_i, \mathsf{lon}_i)) \tag{10}$$

where $\boldsymbol{e}_{l_i}$ represents the features of l_i, $\mathit{Normalize}$ is to normalize the features via mean and standard variance, i.e., gaussian normalization, and MLP_e is one-layer full-connected network. Similar to the light-weight grid representation encoder, we add the location features with the positional encoding, i.e., $\boldsymbol{e}_{l_i} = \boldsymbol{e}_{l_i} + \boldsymbol{p}_i$, due to the agnostic of sequentiality of the following blocks.

Then, we leverage m stacked blocks of Attention-MLP with residual connection to model the complex sequential information based on the location features. In summary, the k-th layer block is as follows:

$$\boldsymbol{e}_{l_1}^k, \quad, \boldsymbol{e}_{l_n}^k = \mathit{MLP}^k(\mathit{Attn}^k([\mathsf{e}_{l_1}^{k-1}, \quad, \mathsf{e}_{l_n}^{k-1}])) \tag{11}$$

where $\boldsymbol{e}_{l_i}^0$ equals to $\boldsymbol{e}_{l_i}$ in Equation 10, MLP^k is a two-layer full-connected network with $ReLU$ as activation, and Attn^k

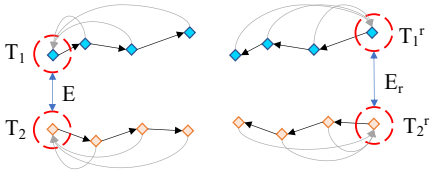


Fig. 3. An example of attention-based trajectory encoder with reversed augmentation

is the self-attention [46]. The details of this equation can be presented in Equation 12.

$$\begin{aligned} \boldsymbol{e}_{l_i}^{k-1} &= \boldsymbol{e}_{l_i}^{k-1} + \sum_{j}^{n} w_{ij}(\boldsymbol{W}_v \boldsymbol{e}_{l_j}^{k-1}) \\ w_{ij} &= \mathit{softmax}_j((W_q e_{l_i}^{k-1})^T (W_k e_{l_j}^{k-1})) \\ \boldsymbol{e}_{l_i}^{k} &= \mathit{MLP}^k(e_{l_i}^{k-1}) + e_{l_i}^{k-1} \end{aligned} \tag{12}$$

where $\boldsymbol{W}_q$, $\boldsymbol{W}_k$, and $\boldsymbol{W}_v \in \mathsf{R}^{d \times d}$ are the to-be-learned parameters for map the query, key, value embeddings, respectively. We also use the multi-head strategy to improve the ability of expression following the previous studies [24], [46]. After throughout the stacked blocks, a read-out layer should be employed to summarize the sequence information. For example, BERT [47] inserts a *CLS* token at the beginning and uses its embedding after the stacked blocks as the sentence embedding. TrajGAT [24] uses mean pooling to aggregate the embeddings at all positions. However, these methods are not the best way to model trajectories for different distance functions, such as DTW and the Frechet distance, since the characteristics of distance functions are ignored. According to Lemma 1, we have known that the distance between the first points (or the last points) of the trajectories is the lower bound of the distance between these trajectories under the DTW and Frechet distance. Thus, we propose a simple but effective read-out layer, i.e., directly using the embedding of the first point as the embedding of the input trajectory.

$$\boldsymbol{h}_l = \boldsymbol{e}_{l_1}^m \tag{13}$$

where $\boldsymbol{h}_l$ is the embedding of input GPS trajectory, and $\boldsymbol{e}_{l_1}^m$ is the embedding of the first GPS point after m blocks. We provide a brief illustration of an attention-based trajectory encoder with reversed augmentation in Figure 3, where the grey lines present the weighted aggregations in the attention layer, and the red dash circles are the lower-bound induced read-out operation. In a nutshell, the trajectory embedding induced from GPS representation is obtained by treating the first point as an anchor to aggregate information from other points. Besides, due to the reversed augmentation, the predicted distance between T_1 and T_2 comes from two parts (i.e., E and E_r), which means that the lower bounds from both sides (i.e., the first and the last points) are simultaneously considered in the approximation process.

E. Hash Layer

Different from the previous studies, we aim to encode the trajectories into Euclidean space and Hamming space to structure them for an efficiently searching process. Thus, a hash layer is created upon the grid and GPS trajectory encoders. Firstly, the representations of T from the two encoders are concatenated and pass a fusing layer as follows:

$$\boldsymbol{h} = \mathit{MLP}_f([h_l, h_g]) \tag{14}$$

Then, we encode the reversed version of T, i.e., T_r, in the same way and obtain $\boldsymbol{h}_r$. After that, we project and concatenate the two embeddings in the metric space where the loss function works, which aims to comply with the reverse symmetric property of the trajectory distance function as stated in Lemma 3.

$$\boldsymbol{h}_f^T = [\boldsymbol{W}_p \boldsymbol{h}, \boldsymbol{W}_p \boldsymbol{h}_r] \tag{15}$$

where $\boldsymbol{W_p} \in \mathsf{R}^{\frac{d}{2} \times d}$ is the parameter of the projector, the bias term is ignored for simplicity, and $\boldsymbol{h}_f^T$ is the final representation of trajectory T. To hash $\boldsymbol{h}_f^T$, we use a sign function to achieve this.

$$\boldsymbol{z}^T = \mathit{sign}(h_f^T) \tag{16}$$

where $\mathit{sign}(x) = 1$ if $x > 1$, otherwise $\mathit{sign}(x) = -1$.

F. Optimization

To make the model have the ability to be aware of the trajectory distance and binarize the representations simultaneously, we combine a WMSE [22], [24] and a ranking-based hashing objective to train the model parameters. In the WMSE part, given the distance matrix D containing the pair-wise distances of trajectories in τ, we follow NeuTraj to transform D into a similarity matrix S, i.e., $S_{ij} = exp(-\theta * D_{ij})/max(exp(-\theta * D))$, and use S as supervision signal, in which θ is a tunable hyper-parameter to smooth the similarity distribution. Then, the objective function of WMSE can be defined as follows:

$$L_s = \sum_{i}^{N} \sum_{j}^{M} r_j \quad (g(T_i, T_j) \quad f(T_i, T_j))^2 \tag{17}$$

where $g(T_i, T_j) = \exp(-\mathit{Euclidean}(h_f^{T_i}, h_f^{T_j}))$, M is the number of samples for each trajectory, r_j is the sample weight computed according to the ranking order of the sample [22]. Despite the effectiveness of WMSE in approximating the distance function [22], [24] in Euclidean distance, it may not be suitable to operate the latent vector in Hamming space since the binary process will cause the undistinguishable representations and instability in training. To solve this problem, we adopt a ranking-based hashing objective working in Hamming space to maximize the distance between the similar and the noisy ones.

$$L_r = \sum_{i}^{N} \sum_{j}^{M/2} [H(\boldsymbol{z}^{T_i}, \boldsymbol{z}^{T_j^p}) \quad H(\boldsymbol{z}^{T_i}, \boldsymbol{z}^{T_j^n}) + \alpha]_+ \tag{18}$$

where we group the M samples into $M/2$ pairs, in which T_j^p is more similar to T_i than T_j^n according to the similarity matrix S, $[x]_+ = 0$ if $x \leq 0$, otherwise $[x]_+ = x$, α is a threshold that indicates the extend of the margin between the similar and the dissimilar pairs, and $H(\cdot, \cdot)$ is the Hamming distance. Note that there is a nice relationship between Hamming distance

and inner product, where $H(\boldsymbol{z}^{T_i}, \boldsymbol{z}^{T_j^n}) = \frac{1}{2}(d_h - \boldsymbol{z}^{T_i}\boldsymbol{z}^{T_j^p})$. The above objective can be rewritten as:

$$\mathcal{L}_r = \frac{1}{2}\sum_{i}^{N}\sum_{j}^{M/2}[\ \ \boldsymbol{z}^{T_i}\boldsymbol{z}^{T_j^p} + \boldsymbol{z}^{T_i}\boldsymbol{z}^{T_j^n} + \alpha]_+ \tag{19}$$

Until now, the model cannot be trained due to the gradient untraceable of *sign* function in Equation 16. To circumvent this, we follow HashNet [37] to relax the hard *sign* function to $\textit{tanh}(\beta *)$ in the training phase, in which β, a scaling parameter, is initialized to 1 and increased after each training iteration.

Moreover, we argue that the limited number of trajectories in the seed set cannot structure the Hamming space well (cf. Section V-D). To deal with this, a straight way is to enlarge the seed set. However, it is time-consuming due to the quadratic computation complexity of the distance function. Fortunately, the ranking-based hashing objective does not require precise distance between trajectories. Therefore, a fast triplet generation method with less effort is adopted to enrich the training corpus to strengthen the regularization on the Hamming space. Specifically, given a corpus of trajectories τ_u, we convert the GPS trajectories into grid trajectories with relatively large grid sizes, such as 500m×500m used in this paper. Then, we organize the GPS trajectories as the same cluster if they share the same grid trajectory. As shown in the left part of Figure 2, the two trajectories colored blue and red will be organized in the same cluster. Based on the clusters, the triplets, e.g., (T_a, T_p, T_n), can be generated, in which T_a and T_p are sampled from the same cluster and T_n is randomly sampled from the rest. The idea behind it is that the trajectories within the same cluster are actually bounded by the grid size. For example, in the Frechet distance, the distance between the trajectories within the same cluster will not exceed the grid size. We denote the generated triplets as τ_t. The ranking objective on τ_t can be discribed as follows:

$$\mathcal{L}_t = \sum_{(T_a,T_p,T_n)\in\tau_t}[\ \ \boldsymbol{z}^{T_a}\boldsymbol{z}^{T_p} + \boldsymbol{z}^{T_i}\boldsymbol{z}^{T_n} + \alpha]_+ \tag{20}$$

The overall objective is the combination of the three objectives above with a balanced weight γ as follows:

$$\mathcal{L} = \mathcal{L}_s + \gamma\ \ (\mathcal{L}_r + \mathcal{L}_t) \tag{21}$$

We train all the parameters in an end-to-end way through the back-propagation algorithm to minimize $\mathcal{L}$ and employ the Adam optimizer for the update of parameters.

V. Experiments

A. Experimental Settings

1) Datasets: We evaluate the performance on two publicly available datasets: ChengDu[2] and Porto[3]. The first dataset is from DiDi Inc, which consists of around 1.2 million taxi trajectories located in ChengDu, China. The second dataset is collected from 2013 to 2014 in Porto, Portugal, which contains over 1.7 million trajectories. We follow NeuTraj [22] to preprocess the original data. Specifically, we discretize the area into 50m × 50m grid cells. Then, we remove the trajectories with less than 10 records. After preprocessing, we obtain around 1 and 1.5 million trajectories in ChengDu and Porto, respectively.

2) Experimental Protocal: To evaluate the performance of *Traj2Hash*, we conduct experiments on top-k similarity search tasks under different trajectory measurements, i.e., the Frechet distance, the Hausdorff distance, and DTW. The ground-truth of this task is the exact of top-k similar trajectories. Due to the high computational costs of trajectory measurements, it is impractical to generate all the labels for both datasets. Following previous studies [22]–[24], we randomly sample 10K trajectories to compute the similarity, in which 20% of them are used as seeds in Equation 17 and 80% are used for validation. Except that, 200K trajectories are randomly sampled as the corpus to generate the triplets. To obtain reliable evaluation results, we construct a large test set, in which 10K and 100K trajectories are randomly sampled from the rest as query and database, respectively.

3) Baselines: Since our model can obtain dense vectors in Euclidean space (e.g., $\boldsymbol{h}_f^T$) and hash codes in hamming space (e.g., $\boldsymbol{z}^T$) at the same time, we will separately evaluate the performance of the two representations. For the studies of the performance of searching in Euclidean space, we compare *Traj2Hash* with six neural-based similarity computation methods as follows.

- t2vec [42]: is a sequential auto-encoder, and adopts novel embedding techniques to capture spatial information and improve the robustness of embeddings.
- CL-TSim [43]: adopts recurrent neural network to encode trajectories into vectors and adopts contrastive learning equipped with two trajectory augmentations to train.
- NeuTraj [22]: is a metric learning approach, in which a novel spatial attention memory module and a distance-weighted ranking loss are proposed.
- NT-No-SAM [22]: is an ablation version of NeuTraj by removing the spatial attention memory module.
- Transformer [46]: alternately stacks dot-product attention and feed-forward layer to perform sequence encoding, which shows strong competitiveness in approximating trajectory distance [24].
- TrajGAT [24]: is designed for long trajectory similarity computation, which represents trajectory as a graph through PR quadtree and develops GraphTransformer to convert the graph into vectors.

Besides, for the studies of the performance of searching in Hamming space, we leverage the proposed ranking-based hashing objective (c.f. Equation 18) with a extra trainable linear layer to convert the dense vectors from baselines above into hash codes. After that, the top-k trajectories are returned according to the hamming distance to the query trajectory. In addition, the following baseline will be added when searching in hamming space.

[2]https://gaia.didichuxing.com

[3]http://www.geolink.pt/ecmlpkdd2015-challenge

- Fresh [18]: is a locality sensitive hash for curves, which firstly maps the trajectories into randomly shifted grids and adopts multiply-shift hashing to convert a sequence of grids into an integer.

4) Evaluation Metrics: For evaluation of the performance of top-k similarity search, we employ three metrics as in [22]–[24], i.e., HR@10, HR@50, and R10@50. The HR@k presents the hitting ratio, which is the overlap between returned top-k results and ground-truth. The R10@50 is the top-50 recall for top-10 ground-truth, which indicates the extent that the returned top-50 trajectories cover top-10 ground-truth. The larger the three metrics are, the stronger the model performs.

5) Parameter Settings: In the grids pre-training phase, the number of sampled neighbors and noise is set to 1. The neighbor range r is set to 5. We use standard gradient descent with Adam optimizer to update the grid embedding. In *Traj2Hash*, the number of blocks m and the number of heads in the attention-based trajectory encoder are set to 2 and 4, respectively. We set the default value of the latent dimension d and d_h to 64. In the training objective, the value of α and γ are separately searched from [0, 25] and [0, 12] on the validation set, where the default value of them are set to 5 and 6, respectively. The sample size M and the training batch size for WMSE objective are set to 10 and 20, which is aligned in all the evaluation models for fair comparison. The training batch size on generated triplets corpus is set to 500. We set the maximal training epoch to 100 and use the model parameters with the highest HR@10 on validation set for testing. The learning rate of *Traj2Hash* is set to 1e-3. For the fast triplets generation algorithm, we set the size of grid at 500m$\times$500m, which totally generates 700K and 6M triplets on Porto and ChengDu datasets, respectively. In Fresh, the resolution is set to 1 kilometer. The number of LSH repetitions and of LSH concatenations are set to 4 and 1, in which each hash function maps the trajectories into a 16 bits integer for aligning the length of hash codes. For the neural-based approximation approaches, we set the latent dimension with ours for fair comparison. For t2vec and CL-TSim, we set the distorting and dropping rate are [0, 0.2, 0.4, 0.6]. For Transformer and TrajGAT, we keep the same number of head and number of layers with ours.

6) Evaluation Platform: Our method is implemented in Python and Pytorch, and trained using a GeForce GTX 1080 Ti GPU. The platform runs the Ubuntu 16.04 operating system with 48-cores Intel(R) CPU E5-2650 v4 @ 2.20GHz 256GB RAM.

B. Performance Comparison on Euclidean Space

Table I shows the performance of different methods on Euclidean space. From the results on both datasets, we can observe that *Traj2Hash* significantly outperforms the baselines on all metrics. Taking the Frechet distance on Porto as an example. Compared with the best-performing baseline, i.e., NeuTraj, *Traj2Hash* gains about 11% performance improvements on HR@10 and about 7% on R10@50. Most surprisedly, the improvement of *Traj2Hash* under the non-metric distance DTW on Porto is about 13% on HR@10, which demonstrates the universality of *Traj2Hash* for different situations. Although most of the baselines employ deep neural networks to approximate the ground-truths, *Traj2Hash* has three advantages to achieve the best performance. First, the properties of different distances are considered such as reverse symmetric property and the lower-bound of DTW, and the Frechet distance, which relieve the burden in model training. Second, the grid representations are profitable to enrich the spatiality information. For example, these representations can be used to easily distinguish the far-away samples. Lastly, the ranking-based objective with external triplets helps the model to regularize the metric space, in which the similarity between trajectories is well ranked.

Except that, we also have the following observations: 1) The general trajectory encoding methods, i.e., t2vec and CL-TSim, perform the worst since these methods are distance-agnostic. Actually, the goal of these methods is to define a brand new distance that is robust to the noise in trajectory rather than approximate the existing distances. 2) Transformer-based models, i.e., Transformer and TrajGAT, are better at approximating the Hausdorff distance rather than the others, which may be illustrated that the global read-out, i.e., *CLS* in Transformer and *MeanPooling* in TrajGAT, is a good choice for the sets matching distance. 3) On the contrary, the variants of NeuTraj are better at approximating the Frechet distance and DTW, in which stacked RNN are used to encode trajectories and the last hidden state is read-outed. We conjecture the different preferences of these methods come from the different ways of read-out, in which the last hidden state read-out in NeuTraj implicitly achieves the lower-bound reduced read-out for DTW and the Frechet distance (c.f. Equation 1) while limits the representative ability in approximating the Hausdorff distance. We will further study the effect of the read-out layer in the later section.

C. Performance Comparison on Hamming Space

We compare the performance of different methods on Hamming space in Table II. From the results, we observe that all the neural-based methods have a large performance drop comparing the searching in Euclidean space due to the information loss caused by the binarization of dense vectors, among which *Traj2Hash* achieves the least decreases since 1) the light-weight grid representation with pre-trained decomposed embeddings provide the spatial similarity between trajectories. Actually, the grid trajectory can be seen as a hash sequence of the corresponding GPS trajectories while the grid trajectory encoding can be seen as a re-hash process to preserve the ranking of Hamming distance between hash codes; 2) the well approximating to the ground distance achieved through modeling the properties of different distance functions in Euclidean space also benefits the similarity modeling in Hamming space; 3) the fast triplets generation produce enriches the training data for the ranking-based objective, which is necessary to well structure the trajectories in Hamming space.

TABLE I
PERFORMANCE COMPARISON FOR DIFFERENT METHODS IN EUCLIDEAN SPACE ON FRECHET, HAUSDORFF, AND DTW DISTANCES.

Dataset	Method	Frechet			Hausdorff			DTW		
		HR@10	HR@50	R10@50	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50
Porto	t2vec	0.2761	0.3606	0.5218	0.2684	0.3279	0.5437	0.2762	0.3355	0.5492
	CL-TSim	0.3107	0.3370	0.5764	0.2801	0.2860	0.5289	0.2961	0.3909	0.5848
	NT-No-SAM	0.4982	0.5820	0.8124	0.3502	0.4241	0.7357	0.4619	0.5025	0.7584
	NeuTraj	0.5053	0.5953	0.8157	0.3834	0.4460	0.7410	0.4711	0.5329	0.7885
	Transformer	0.4290	0.5238	0.7392	0.4389	0.5098	0.7761	0.3576	0.4424	0.6887
	TrajGAT	0.4737	0.5699	0.7905	0.4594	0.5174	0.7839	0.4535	0.5178	0.7649
	Traj2Hash	**0.5652**	**0.6162**	**0.8755**	**0.4640**	**0.5307**	**0.8021**	**0.5327**	**0.5822**	**0.8565**
ChengDu	t2vec	0.3329	0.4254	0.5709	0.3453	0.3790	0.5428	0.3256	0.3572	0.5781
	CL-TSim	0.3513	0.3844	0.5980	0.3011	0.3258	0.5892	0.3401	0.3576	0.6292
	NT-No-SAM	0.6903	0.7509	0.9403	0.5393	0.6498	0.8350	0.5229	0.5815	0.8836
	NeuTraj	0.6936	0.7551	0.9421	0.5802	0.6593	0.8511	0.5391	0.5990	0.8905
	Transformer	0.6455	0.6997	0.9303	0.6593	0.7212	0.9279	0.5519	0.5803	0.7649
	TrajGAT	0.6832	0.7345	0.9337	0.6764	0.7395	0.9385	0.6288	0.6937	0.9350
	Traj2Hash	**0.7297**	**0.7818**	**0.9572**	**0.6838**	**0.7415**	**0.9591**	**0.6796**	**0.7278**	**0.9507**

TABLE II
PERFORMANCE COMPARISON FOR DIFFERENT METHODS IN HAMMING SPACE ON FRECHET, HAUSDORFF, AND DTW DISTANCES.

Dataset	Method	Frechet			Hausdorff			DTW		
		HR@10	HR@50	R10@50	HR@10	HR@50	R10@50	HR@10	HR@50	R10@50
Porto	t2vec	0.0236	0.0357	0.0488	0.0129	0.0254	0.0355	0.0186	0.0214	0.0383
	CL-TSim	0.0138	0.0165	0.0240	0.0147	0.0158	0.0247	0.0232	0.0243	0.0409
	NT-No-SAM	0.0479	0.0956	0.1201	0.0345	0.0710	0.0821	0.0235	0.0572	0.0728
	NeuTraj	0.0525	0.1128	0.1378	0.0270	0.0622	0.0768	0.0278	0.0613	0.0799
	Transformer	0.0412	0.0811	0.1000	0.0680	0.1467	0.1838	0.0174	0.0390	0.0482
	TrajGAT	0.0457	0.0921	0.1175	0.0794	0.1543	0.2037	0.0201	0.0567	0.0833
	Fresh	0.1322	0.1382	0.2784	0.1092	0.1234	0.2418	0.1303	0.1371	0.2726
	Traj2Hash	**0.3072**	**0.3966**	**0.6117**	**0.2204**	**0.2994**	**0.4677**	**0.2931**	**0.3881**	**0.5948**
ChengDu	t2vec	0.0319	0.0443	0.0625	0.0094	0.0147	0.0295	0.0257	0.0530	0.0684
	CL-TSim	0.0346	0.0491	0.0683	0.0101	0.0134	0.0273	0.0359	0.0597	0.0763
	NT-No-SAM	0.0426	0.1088	0.1220	0.0189	0.0442	0.0548	0.0858	0.1439	0.1894
	NeuTraj	0.0417	0.0941	0.1079	0.0241	0.0557	0.0634	0.0945	0.1635	0.2151
	Transformer	0.0706	0.1387	0.1695	0.0991	0.2047	0.2520	0.0049	0.0164	0.0175
	TrajGAT	0.0874	0.1543	0.1730	0.1020	0.2111	0.2683	0.0132	0.0248	0.0533
	Fresh	0.2694	0.2955	0.5483	0.2330	0.2339	0.4608	0.2715	0.2952	0.5454
	Traj2Hash	**0.3743**	**0.4733**	**0.6945**	**0.2596**	**0.3499**	**0.5102**	**0.4065**	**0.4964**	**0.7324**

Moreover, we observe that 1) Compared with the neural-based baselines, the traditional method, i.e., Fresh, performs the best in most cases. This is because the performance of hashing of neural-based methods relies heavily on the size of training data. With only the seed set of trajectories, the learned hash codes in Hamming space cannot be well organized, such as the dissimilar trajectories are wrongly fitted into the same bucket. 2) The similar trend of neural-based baselines is also observed, i.e., Transformer-based methods prefer approximating Hausdorff distance while the variants of NeuTraj perform better in DTW and the Frechet distance. 3) *Traj2Hash* achieves superior performance among all the competitors. On the Porto dataset, *Traj2Hash* gains around two times performance improvement compared with Fresh. For example, *Traj2Hash* improves HR@50 of Frechet from 0.1382 to 0.3966. On the other dataset, *Traj2Hash* also outperforms the best-performing baseline with a large margin, such as HR@10 of DTW from 0.2715 to 0.4065. These observations indicate the effectiveness of *Traj2Hash* in approximately retrieving k most similar trajectories via Hamming distance.

D. Ablation Study

To deeply understand the effect of each component, we sequentially ablate the main components in *Traj2Hash*, i.e., -Grids, -RevAug, and -Triplets, and evaluate the top-k similar search in Euclidean space and Hamming space, respectively.

- -Grids: removes the light-weight grid representation encoder from *Traj2Hash*. Only the GPS trajectory encoder is used to encode trajectory.
- -RevAug: removes the reverse augmentation from *Traj2Hash*, which means this variant does not hold the reverse symmetric property.
- -Triplets: removes the fast triplets generation from *Traj2Hash*, i.e., L_t in Equation 21 is eliminated. The model is simplified into Transformer with lower-bound read-out.

It should be noted that the ablated component in the former variant is also eliminated in the latter, e.g., in -RevAug, the light-weight grid representation encoder is eliminated. Table III shows the performance of each variant. We first look at the results evaluated in Euclidean space. We can observe that 1) Compared with *Traj2Hash*, the performance of -Grids decreases, especially on the Porto dataset, e.g., HR@10 decreases from 0.5652 to 0.5466 under the Frechet distance, which demonstrates the effectiveness of the light-weight grid representation encoder; 2) By preserving the reverse symmetric property, -Grids achieves remarkable performance improve-

TABLE III
ABLATION STUDIES ON BOTH DATASETS

Dataset	Metrics		Porto				ChengDu			
			Traj2Hash	-Grids	-RevAug	-Triplets	Traj2Hash	-Grids	-RevAug	-Triplets
Frechet	Euclidean	HR@10	**0.5652**	0.5466	0.5018	0.4699	**0.7297**	0.7231	0.6749	0.6508
		HR@50	**0.6162**	0.6087	0.5692	0.5644	**0.7818**	0.7782	0.7280	0.7084
		R10@50	**0.8755**	0.8331	0.7980	0.7798	**0.9572**	0.9476	0.9364	0.9161
	Hamming	HR@10	**0.3072**	0.3011	0.2970	0.0349	**0.3743**	0.3604	0.3528	0.0374
		HR@50	**0.3966**	0.3841	0.3805	0.0748	**0.4733**	0.4694	0.4515	0.0890
		R10@50	**0.6117**	0.6043	0.5886	0.0866	**0.6945**	0.6892	0.6613	0.1040
DTW	Euclidean	HR@10	**0.5327**	0.4967	0.4714	0.3646	**0.6796**	0.6542	0.6224	0.6043
		HR@50	**0.5822**	0.5470	0.5401	0.4520	**0.7278**	0.7138	0.6759	0.6572
		R10@50	**0.8565**	0.8051	0.7923	0.7017	**0.9507**	0.9272	0.9194	0.9102
	Hamming	HR@10	**0.2931**	0.2717	0.2555	0.0176	**0.4065**	0.3783	0.3760	0.0216
		HR@50	**0.3881**	0.3763	0.3491	0.0498	**0.4964**	0.4737	0.4733	0.0537
		R10@50	**0.5948**	0.5675	0.5220	0.0827	**0.7324**	0.6975	0.6933	0.0816

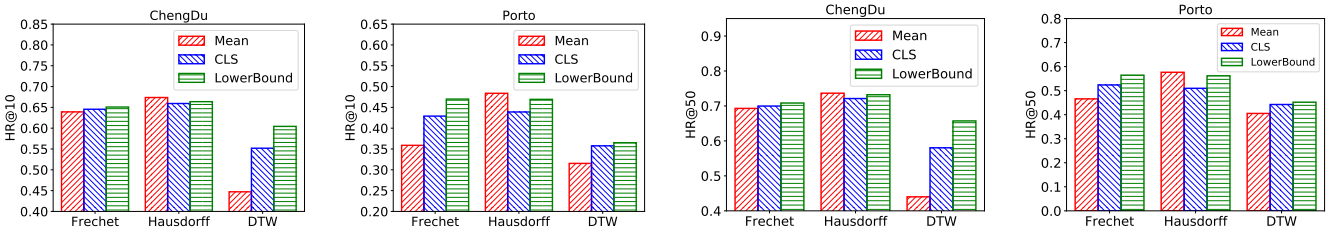


Fig. 4. The effect of different read-out layers.

ments compared with -RevAug. For example, on ChengDu dataset under the Frechet distance, -Grids gains around 7% improvements. 3) The importance of the generated triplets with ranking-based hashing objective is verified by comparing the results of -RevAug and -Triplets. For example, on the Porto dataset under the Frechet distance, -RevAug outperforms -Triplets with a large margin; 4) *Traj2Hash* achieves the best performance compared with all the variants, which demonstrates the effectiveness of the proposed techniques.

Then, looking at the results evaluated in Hamming space, we have similar observations in Euclidean space except for the importance of the fast triplets generation. We can see a sharp decrease from -RevAug to -Triplets on both datasets, which demonstrates that the limited seed set of trajectories is not enough to structure the vectors in Hamming space and the proposed fast triplets generation with ranking-based hashing objective provides abundant supervised signals to help the model to learn the expressive hash codes.

Moreover, we also conduct an extra experiment on both datasets to verify the effectiveness of the proposed lower-bound read-out layer. Specifically, we use the Transformer as the network backbone to encode the GPS trajectory and denote three variants, i.e., *Mean*, *CLS*, and *LowerBound*, according to the type of read-out layer, in which other techniques in this paper are eliminated such as the light-weight grid representation encoder, the reverse augmentation, and the fast triplets generation. Figure 4 shows the performance of three variants searching in Euclidean space. We can observe that 1) Compared with *Mean* and *CLS*, the proposed lower-bound read-out layer, i.e., *LowerBound*, achieves the best on both datasets under DTW and the Frechet distance. It is expected that the proposed read-out layer provide more discriminative information to approximate the ground-truth distances; 2) *Mean* outperforms the others under the Hausdorff distance, which may be because that the sequential information is not considered in the Hausdorff distance and the mean operator is better at fitting this situation. 3) *CLS* is not recommended to be used in this problem since its performance varies a lot under different distances and is always dominated by *LowerBound*.

In addition, we further study the effect of the grid representation by comparing the proposed decomposed representation with direct training on grid id using a widely-used network embedding method Node2vec [48], in which the walk length, number of walks, window size, return parameter, and in-out parameter are set to 80, 10, 10, 1, and 1, respectively. We report HR10 and R10@50 on Porto as shown in Figure 7. From the results, we can observe that the decomposed representation with the proposed pre-training method achieves the best, which may demonstrate that although Node2vec has a higher degree of freedom since each grid has its own representation, decomposed representation has a stronger ability in modeling spatial information. Besides, -Grids perform the worst, which empirically proves the effectiveness of the proposed light-weight grid representation again. Moreover, we also notice a large improvement in pre-training efficiency[4], i.e., the decomposed representation only costs around 80 seconds while Node2vec costs more than 2 hours in training 1100×1100 grid space since Node2vec needs to sample a series of random walks starting from each grid, which is so time-consuming when the cardinal of grid space becomes large.

E. Efficiency Study

We study the searching efficiency on both datasets with DTW and the Frechet distance. Specifically, we denote

[4]This is not formally presented due to the space limitation.

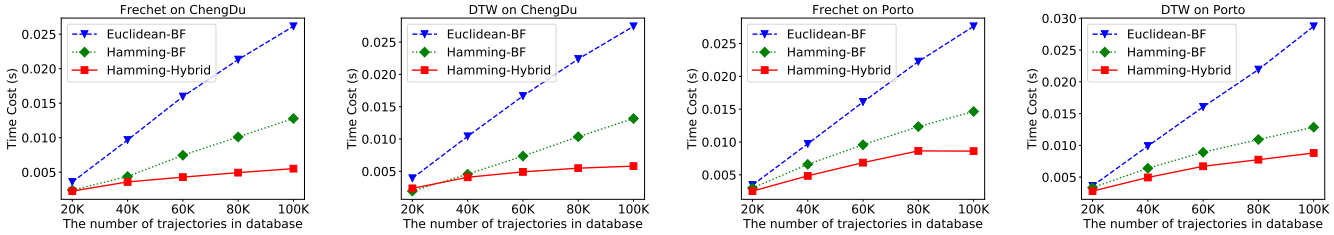


Fig. 5. Time cost of different searching stratagies with the various number of trajectories in database.

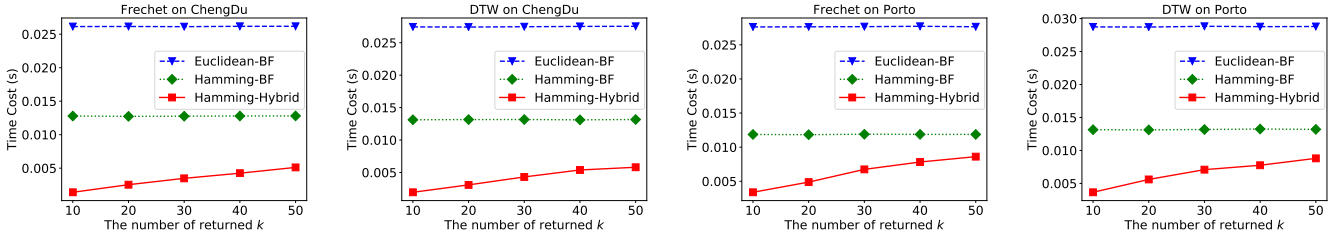


Fig. 6. Time cost of different searching stratagies with the various number of returned k

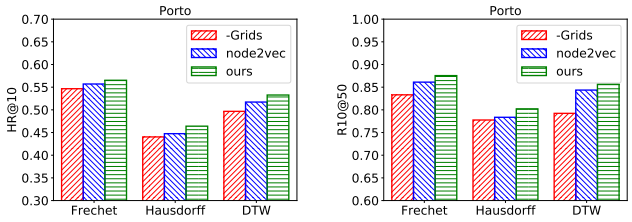


Fig. 7. The effect of different grids representation

three searching strategies: Euclidean-BF, Hamming-BF, and Hamming-Hybrid, in which Euclidean-BF and Hamming-BF compute the distance between the query and all the candidates in Euclidean space and Hamming space and then sort all the results to return k trajectories with minimal distance, and Hamming-Hybrid searches the trajectories through Hamming distance within 2 radii if the number of returned trajectories exceeds k, otherwise brute-forcibly searches the trajectories as Hamming-BF. It should be noted that we do not conduct pure table-lookup searching in Hamming space since we notice that there are lots of empty buckets in this space and the pure neighbor expansion is inefficient to search k trajectories[5].

Figure 5 shows the time cost of the three strategies with different database sizes. Specifically, we randomly sample five subsets with sizes 20K, 40K, 60K, 80K, and 100K, respectively. Then we perform a top-50 similarity search with 10K query trajectories and report the average time cost for processing one query. From this figure, we can see that Hamming-BF always costs less time than Euclidean-BF due to the fast distance calculation and the sorting process in Hamming space. Besides, Hamming-Hybrid consistently outperforms the others, which is because searching in Hamming space within 2 radii can be implemented via table-lookup. The computation complexity via table-lookup is constant, which can save the time costs of distance computation and sorting between query and database. In addition, we can observe that the time costs of the three searching strategies increase as the database size varies from 20K to 100K, in which Hamming-Hybrid increases slower than the others. The reasons behind this are two folds. First, for the query trajectories whose number of neighbor trajectories within radius 2 is increased to exceed 50, the time cost will be reduced via table-lookup. Second, for the other queries that do always have not 50 neighbors within radius 2, the time cost will be added due to the expansion of the search space. The slower time cost increase can be interpreted as the latter factor dominates the time costs but the former factor makes it not increase too much. Moreover, Hamming-Hybrid is more prominent when the database size increases.

Except that, we also show the time cost of the three strategies with different returned k in Figure 6. Specifically, we fix the database size as 100K and vary the returned k from 10 to 50 with 10K query trajectories. The reported results are the average time cost for processing one query. From the results, we can observe that Hamming-Hybrid can achieve around 3x speedup compared with Euclidean-BF, such as searching 10 similar trajectories on Porto dataset under the Frechet distance, which shows the efficiency of encoding trajectories into hash vectors. Besides, the time costs of two brute-force strategies, i.e., Euclidean-BF and Hamming-BF, are rather stable than those of the Hamming-Hybrid strategy. This is because when k is relatively small such as 10, most queries searching through Hamming-Hybrid can be done via table-lookup, while as the increases of k the Hamming-Hybrid strategy will gradually degenerate into Hamming-BF.

[5]Suppose we have 100K trajectories in database, which will cause 100K non-empty buckets at most while the Hamming space with 64 dimensions will have 2^{64} buckets. Thus, lots of buckets are empty. Considering a trajectory that is far away from the others, searching its neighbors in Hamming space will unnecessarily scan plenty of empty buckets, which is inefficient even than computing the Euclidean distance and then returning the k nearest trajectories.

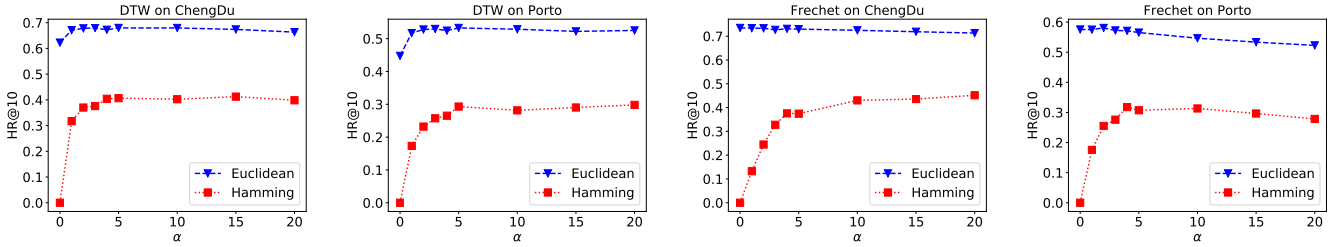


Fig. 8. The performance changes with α.

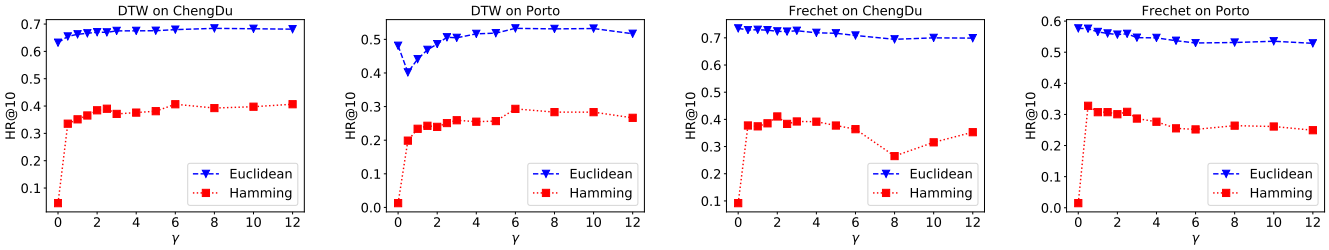


Fig. 9. The performance changes with γ.

F. Parameter Analysis

In this subsection, we evaluate the effect of main hyper-parameters in *Traj2Hash*, such as α (cf. Equation 20) and γ (cf. Equation 21). We set the other hyper-parameters as default and report the HR@10 on both datasets.

1) The effect of the margin α: We first study the effect of the margin α in the ranking-based hashing objective. Figure 8 shows the results in different spaces with the varies of α from 0 to 25. From this figure, we can see that under DTW, the performance increases at the beginning and then maintains for a while before slightly decreasing in both spaces. Under the Frechet distance, the performance does not change too much when searching in Euclidean space. Compared with HR@10 of searching in Euclidean space, the performance in Hamming space is heavily affected by α. We conjecture that when α is relatively small, the hash codes learned by the model are intensive due to the weak strength to expel the dissimilar pairs, which causes hardly distinguishable hash vectors for similar trajectories. When α varies from 0 to 5, the performance in Hamming space gradually increases and achieves around the best. With the further increasing α, there are no gains or harmful in terms of the searching accuracy.

2) The effect of the balanced weight γ: We proceed to demonstrate the effect of the balanced weight γ in Equation 21. Figure 9 shows the results in different spaces with the changes of γ, in which γ is unevenly tuned from 0 to 12. From this figure, we can observe that when *Traj2Hash* is to approximate DTW, the searching performance in Euclidean space increases firstly and then tend to be gentle. Whereas for the Frechet distance, the curve of performance is stable and gradually decreases. However, we can see a performance decreases in ablation study after removing the generated triplets (from -RevAug to -Triplets), which may be concluded that the architecture of *Traj2Hash* can be trained easier in Euclidean space with limited training samples. When searching in Hamming space, the performance varies a lot as γ increases from 0 to 12. Specifically, when γ equals 0, the performance is extremely poor, which shows that the limited seed set of trajectories is hardly to well regularize the vectors in Hamming space. Then, as γ increases, the performance gets better and then reaches the best, in which for DTW and the Frechet distance, the best γ is around 6 and 1, respectively.

VI. Conclusion

In this paper, we propose the first learning to hash algorithm, *Traj2Hash*, to encode trajectories into Euclidean space and Hamming space simultaneously for accurate similarity computation and efficient top-k similar search. Specifically, we preserve the reverse symmetric property and obtain the informative embeddings through reverse augmentation and the lower-bound induced read-out layer, respectively. We also propose a decomposed grid representation to improve training efficiency and reduce memory consumption. In the model training phase, we combine a weighted mean squared error loss and a ranking-based hashing loss to enable the model similarity-aware and representations well self-structured, in which a fast triplets generation is adopted to deal with the limitation of seed set of trajectories in regularizing vectors in Hamming space. Experiments conducted on two real-world datasets demonstrate that *Traj2Hash* can approximate different trajectory distances while achieving high efficiency in searching process compared with state-of-the-art baselines.

VII. Acknowledgement

This work is partially supported by NSFC (No. 61972069, 61836007 and 61832017), Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021), and Municipal Government of Quzhou under Grant (No. 2022D037, 2023D044), and Key Laboratory of Data Intelligence and Cognitive Computing, Longhua District, Shenzhen.

REFERENCES

[1] D. Yao, C. Zhang, Z. Zhu, J. Huang, and J. Bi, "Trajectory clustering via deep representation learning," *IJCNN*, pp. 3880–3887, 2017.

[2] Y. Xu, J. Xu, J. Zhao, K. Zheng, A. Liu, L. Zhao, and X. Zhou, "Metaptp: An adaptive meta-optimized model for personalized spatial trajectory prediction," *SIGKDD*, 2022.

[3] Y. Liu, X. Ao, L. Dong, C. Zhang, J. Wang, and Q. He, "Spatiotemporal activity modeling via hierarchical cross-modal embedding," *ICDE*, vol. 34, pp. 462–474, 2020.

[4] D. Yang, B. Fankhauser, P. Rosso, and P. Cudré-Mauroux, "Location prediction over sparse user mobility traces using rnns: Flashback in hidden states!" in *IJCAI*, 2020.

[5] R. Laxhammar and G. Falkman, "Online learning and sequential anomaly detection in trajectories," *TPAMI*, vol. 36, pp. 1158–1173, 2014.

[6] M. Chen, Y. Zhao, Y. Liu, X. Yu, and K. Zheng, "Modeling spatial trajectories with attribute representation learning," *TKDE*, 2020.

[7] Y. Zhao, S. Shang, Y. Wang, B. Zheng, Q. V. H. Nguyen, and K. Zheng, "Rest: A reference-based framework for spatio-temporal trajectory compression," in *SIGKDD*, 2018, pp. 2797–2806.

[8] K. Zheng, Y. Zhao, D. Lian, B. Zheng, G. Liu, and X. Zhou, "Reference-based framework for spatio-temporal trajectory compression and query processing," *TKDE*, vol. 32, no. 11, pp. 12 227–2240, 2020.

[9] Z. Fang, Y. Du, X. Zhu, D. Hu, L. Chen, Y. Gao, and C. S. Jensen, "Spatio-temporal trajectory similarity learning in road networks," in *SIGKDD*. ACM, 2022, pp. 347–356.

[10] L. Chen, Y. Gao, Z. Fang, X. Miao, C. S. Jensen, and C. Guo, "Real-time distributed co-movement pattern detection on streaming trajectories," *Proc. VLDB Endow.*, vol. 12, no. 10, pp. 1208–1220, 2019.

[11] C. Luo, Q. Liu, Y. Gao, L. Chen, Z. Wei, and C. Ge, "TASK: an efficient framework for instant error-tolerant spatial keyword queries on road networks," *Proc. VLDB Endow.*, vol. 16, no. 10, pp. 2418–2430, 2023.

[12] J. Zhao, Y. Gao, G. Chen, and R. Chen, "Towards efficient framework for time-aware spatial keyword queries on road networks," *ACM Trans. Inf. Syst.*, vol. 36, no. 3, pp. 24:1–24:48, 2018.

[13] K. Zheng, Y. Zheng, N. J. Yuan, and S. Shang, "On discovery of gathering patterns from trajectories," *ICDE*, pp. 242–253, 2013.

[14] F. Jin, W. Hua, T. Zhou, J. Xu, M. Francia, M. E. Orlowska, and X. Zhou, "Trajectory-based spatiotemporal entity linking," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, pp. 4499–4513, 2020.

[15] B.-K. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," *ICDE*, pp. 201–208, 1998.

[16] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *Int. J. Comput. Geom. Appl.*, vol. 5, pp. 75–91, 1995.

[17] L. Chen and R. T. Ng, "On the marriage of lp-norms and edit distance," in *VLDB*, 2004.

[18] M. Ceccarello, A. Driemel, and F. Silvestri, "Fresh: Fréchet similarity with hashing," *arXiv: Computational Geometry*, 2018.

[19] A. Driemel and F. Silvestri, "Locality-sensitive hashing of curves," in *International Symposium on Computational Geometry*, 2017.

[20] S. Zhang, J. Huang, R. Xiao, X. Du, P. Gong, and X. Lin, "Toward more efficient locality-sensitive hashing via constructing novel hash function cluster," *Concurrency and Computation: Practice and Experience*, vol. 33, 2021.

[21] S. Kanda, K. Takeuchi, K. Fujii, and Y. Tabei, "Succinct trit-array trie for scalable trajectory similarity search," *SIGSPATIAL*, 2020.

[22] D. Yao, G. Cong, C. Zhang, and J. Bi, "Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach," *ICDE*, pp. 1358–1369, 2019.

[23] H. Zhang, X. Zhang, Q. Jiang, B. Zheng, Z. Sun, W. Sun, and C. Wang, "Trajectory similarity learning with auxiliary supervision and optimal matching," in *IJCAI*, 2020.

[24] D. Yao, H. Hu, L. Du, G. Cong, S. Han, and J. Bi, "Trajgat: A graph-based long-term dependency modeling approach for trajectory similarity computation," *SIGKDD*, 2022.

[25] P. Yang, H. Wang, D. Lian, Y. Zhang, L. Qin, and W. Zhang, "Tmn: Trajectory matching networks for predicting similarity," *ICDE*, pp. 1700–1713, 2022.

[26] C. W. Tan, M. Herrmann, and G. I. Webb, "Ultra fast warping window optimization for dynamic time warping," *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 589–598, 2021.

[27] C. W. Tan, M. Herrmann, G. Forestier, G. I. Webb, and F. Petitjean, "Efficient search of the best warping window for dynamic time warping," in *SDM*, 2018.

[28] H. A. Dau, D. F. Silva, F. Petitjean, G. Forestier, A. Bagnall, A. A. Mueen, and E. J. Keogh, "Optimizing dynamic time warping's window width for time series data mining applications," *Data Mining and Knowledge Discovery*, vol. 32, pp. 1074–1120, 2018.

[29] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *VLDB*, 1999.

[30] B. Zheng, X. Zhao, L. Weng, N. Q. V. Hung, H. Liu, and C. S. Jensen, "PM-LSH: A fast and accurate LSH framework for high-dimensional approximate NN search," *Proc. VLDB Endow.*, vol. 13, no. 5, pp. 643–655, 2020.

[31] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *SCG '04*, 2004.

[32] Q. Lv, W. K. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe lsh: Efficient indexing for high-dimensional similarity search," in *VLDB*, 2007.

[33] Y. Tian, X. Zhao, and X. Zhou, "Db-lsh: Locality-sensitive hashing with query-based dynamic bucketing," *ICDE*, pp. 2250–2262, 2022.

[34] J. Gan, J. Feng, Q. Fang, and W. Ng, "Locality-sensitive hashing scheme based on dynamic collision counting," *SIGMOD*, 2012.

[35] T. Skopal, J. Pokorný, and V. Snásel, "Nearest neighbours search using the pm-tree," in *International Conference on Database Systems for Advanced Applications*, 2005.

[36] Q. Li, Z. Sun, R. He, and T. Tan, "Deep supervised discrete hashing," *NIPS*, vol. 30, 2017.

[37] Z. Cao, M. Long, J. Wang, and P. S. Yu, "Hashnet: Deep learning to hash by continuation," *ICCV*, pp. 5609–5618, 2017.

[38] K. Zhao, H. Lu, and J. Mei, "Locality preserving hashing," in *AAAI*, C. E. Brodley and P. Stone, Eds. AAAI Press, 2014, pp. 2874–2881.

[39] F. Zhao, Y. Huang, L. Wang, and T. Tan, "Deep semantic ranking based hashing for multi-label image retrieval," in *CVPR*. IEEE Computer Society, 2015, pp. 1556–1564.

[40] Q. Tan, N. Liu, X. Zhao, H. Yang, J. Zhou, and X. Hu, "Learning to hash with graph neural networks for recommender systems," in *WWW*, 2020, pp. 1988–1998.

[41] C.-C. M. Yeh, M. Gu, Y. luan Zheng, H. Chen, J. Ebrahimi, Z. Zhuang, J. Wang, L. Wang, and W. Zhang, "Embedding compression with hashing for efficient representation learning in large-scale graph," *SIGKDD*, 2022.

[42] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," *ICDE*, pp. 617–628, 2018.

[43] L. Deng, Y. Zhao, Z. Fu, H. Sun, S. Liu, and K. Zheng, "Efficient trajectory similarity computation with contrastive learning," *CIKM*, 2022.

[44] S.-W. Kim, S. Park, and W. W. Chu, "An index-based approach for similarity search supporting time warping in large sequence databases," *ICDE*, pp. 607–614, 2001.

[45] T. Rakthanmanon, B. J. L. Campana, A. A. Mueen, G. E. A. P. A. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," *SIGKDD*, vol. 2012, pp. 262 – 270, 2012.

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017, pp. 5998–6008.

[47] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *NAACL*, J. Burstein, C. Doran, and T. Solorio, Eds., 2019, pp. 4171–4186.

[48] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," *SIGKDD*, 2016.