Efficient Similar Region Search with Deep Metric Learning

Yiding Liu
Nanyang Technological University
Singapore
liuy0130@e.ntu.edu.sg

Kaiqi Zhao
Nanyang Technological University
Singapore
kzhao002@e.ntu.edu.sg

Gao Cong
Nanyang Technological University
Singapore
gaocong@ntu.edu.sg

ABSTRACT

The popularization of mobile devices and location-based services enables us to understand different geographical regions (e.g., shopping areas) by leveraging rich geo-tagged data. However, the huge number of regions with complicated spatial information are expensive for people to explore. To solve this issue, we study the problem of searching similar regions given a user specified query region. However, the problem is challenging in both similarity definition and search efficiency. To tackle the two challenges, we propose a novel solution equipped by (1) a deep learning approach to learning the similarity that considers both object attributes and the relative locations between objects; and (2) an efficient branch and bound search algorithm for finding top-N similar regions. Moreover, we propose an approximation method to further improve the efficiency by slightly sacrificing the accuracy. Our experiments on three real world datasets demonstrate that our solution improves both the accuracy and search efficiency by a significant margin compared with the state-of-the-art methods.

CCS CONCEPTS

• **Information systems** → *Geographic information systems*; *Similarity measures*; *Data mining*;

KEYWORDS

Metric learning; Similarity search; Spatial data

ACM Reference Format:
Yiding Liu, Kaiqi Zhao, and Gao Cong. 2018. Efficient Similar Region Search with Deep Metric Learning. In *KDD 2018: 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/XXXXXX.XXXXXX

1 INTRODUCTION

With the popularization of mobile devices and location-based services (e.g., Foursquare and Yelp), huge amounts of geo-tagged objects, such as points of interest (POIs), are becoming available from numerous sources. Meanwhile, these objects are increasingly tagged with rich information, such as category and textual description. This offers us great opportunities to understand the information in a particular urban region. However, the explosive expansion and complication of cities make the information beyond people's ability to perceive. As a result, most people are only familiar with a small part of the city they live. Their knowledge are gained in this area and may not be applicable to new areas. Therefore, there is a need to *search similar regions* on geographical space, which enables people to apply their knowledge to explore similar new areas. Consider the following scenarios of searching similar regions:

- **Scenario 1: Air pollution control**. Suppose the government has developed a solution for solving the air pollution problem in a specific region. They may also want to find other regions with similar pollution sources (e.g., factories), where the solution can also be applied.
- **Scenario 2: Business site selection**. Suppose a restaurant is going to start a new branch, the owners may want to look for regions that are similar to their current surrounding, where they can reuse the existing strategies and experience on the new branch.
- **Scenario 3: Improving location-based services.** Location-based services often collect user's mobility/interests in different regions. Such information can be leveraged to suggest other similar regions for users to visit, and thus improving services like POI recommendation [19] and region recommendation [20].

Therefore, it is of great significance to search similar regions on geographical space, which is fundamental to support a variety of real-world applications.

However, searching similar regions faces two challenges. The first challenge is how to model region similarity. A straightforward method is to represent a region as the sum of the attribute vectors (e.g., categories, keywords) of the objects inside, and compute the similarity between the attribute vectors of two regions. However, this method completely misses the spatial information of the objects. According to Tobler's first law of geography [28], "everything is related to everything else, but near things are more related than distant things". The nearby objects often have underlying relationships and influence [7, 13, 35, 36], which are important for real-world applications. Let us consider the aforementioned Scenario 1 as an example. Assume there are two regions R_A and R_B and both of them contain the same number of factories and residential buildings. The factories in R_A are closed to the residential buildings, while those in R_B are not. The solution of tackling pollution in R_B may not be applicable to R_A, where we need to further consider the influence to the nearby residents. Therefore, the relative spatial locations between objects is an important dimension for measuring region similarity. In addition, spatial data is usually noisy. It is also necessary yet challenging to develop a metric that is robust to noise.

The second challenge is the efficiency of similar region search. Given a 2-d geographical space, there is a huge number of potential

KDD 2018, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/XXXXXX.XXXXXX

regions with various sizes and scales. Apparently, it is impossible to exhaustively compare a query to all the regions. Furthermore, after getting the search results, users may want to modify the location of the query region and re-initiate a new query interactively. This even poses a higher demand for the efficiency.

A recent study [25] tries to tackle the challenges. It proposes to measure the similarity between two regions as the cosine similarity of their spatial feature vectors, each of which is computed based on the distances of the objects to five predefined "reference points" in the region. It also proposes a search algorithm based on quadtree index. However, such hand-crafted spatial features do not consider the relative locations between objects, they are also not robust to noise. Besides, the search algorithm is very time-consuming to be applied to large-scale datasets.

In this paper, we propose a novel solution that addresses the challenges of similar region search. To address the first challenge, we leverage a deep metric learning model, namely triplet network, to learn a ranking metric for comparing regions. It extracts features using Convolutional Neural Network (CNN), which can inherently capture local relationships between objects for similarity learning. We propose a method to generate training data with hard negative example mining, which makes the similarity metric robust to small noise and object shift. We also propose a ratio-based training loss that is robust to highly skewed spatial data. To address the second challenge with the learned similarity metric, we first reduce the time cost of computing similarity by sharing the computation for all regions. We next propose a branch and bound algorithm, namely ExactSFRS, that greatly reduces the search space. In addition, we also propose an approximation method, namely ApproxSFRS, whose worst-case time complexity is adjustable. It can make trade-off between accuracy and efficiency to support different applications.

Overall, our contribution can be summarized as follows:

- We propose the first deep metric learning method to learn similarity between regions. The model not only considers the attributes of objects in a region, but also captures the relative locations between the objects.
- We propose an efficient search algorithm called ExactSFRS that solves the similar region search problem. We also propose an approximation method called ApproxSFRS that can trade off accuracy for efficiency to support different applications.
- We apply the proposed methods to three large-scale datasets. The effectiveness experiments demonstrate our deep metric learning method is significantly better than the baselines for measuring region similarity. The efficiency experiments show that ExactSFRS is more than $10^5\times$ faster than the exhaustive search method and over $45\times$ faster than the best baseline method. Furthermore, we demonstrate ApproxSFRS can largely reduce the runtime by slightly sacrificing the accuracy.

2 RELATED WORK

2.1 Measuring Region Similarity

There are several recent proposals trying to measure the similarity between regions. Sheng et al. [25] first studied the similar region search problem. They measure the similarity between two regions as the cosine similarity of their spatial feature vectors, which is defined as the average distances between the objects and five predefined reference points. Le Falher et al. [18] and Kafsi [12] et al. both studied the problem of finding similar neighborhoods across different cities. The neighborhoods are predefined areas within cities. Besides, Preoţiuc-Pietro [22] et al. studied a problem of measuring city similarity, considering a city as a large region and representing it as grids or predefined neighborhoods for comparison.

Our work differs from the existing studies [12, 18] in measuring similarity. They only consider the aggregated object attributes of a region, while our work measures the similarity considering both the spatial locations and attributes of objects. Sheng et al.'s work [25] is the most germane to our paper. It also considers the spatial locations of objects. However, its similarity modeling is very sensitive to noise. The similarity can easily be affected by adding, deleting or shifting few objects in the region, which are commonly-seen noise in spatial object databases. Worse still, it only considers the spatial distance of the objects to the center and boundary of the region, missing how different types of objects are co-located. This means the underlying influence and relationships between objects are not included in the region similarity.

There are also some work focus on modeling the relationships between regions [31, 32, 37], where the relationships (e.g., traffic flow) can be used to define similarity.

2.2 Similar Region Search

Many of the above-mentioned studies [12, 22, 31, 32, 37] aim at measuring the similarity between areas with predefined boundary (e.g., neighborhoods, cities). They are different from our proposal, which supports searching regions with various sizes and scales at an arbitrary location on the map. Another recent proposal [25] that supports searching similar regions proposes a quadtree-based search method. It stores the spatial objects using a quadtree and consider the tree nodes as candidate regions. When a query comes, it searches top-N similar tree nodes (i.e., regions) and then expand their sizes greedily to further improve the results. However, the region expansion is very time-consuming. It needs to recompute the similarity every time a region expands, while our proposed search method can directly localize the most similar regions without adjusting the size and recomputing the the similarity. Also, the quadtree based method gives no guarantee about the running time, while our proposed approximation method has a worst-case complexity guarantee.

There are also some studies in Computer Vision about finding regions on images, namely visual instance retrieval [23, 29]. They usually sample many candidate regions on an image and check them exhaustively. However, this is not applicable on large geographical space. It needs at least billions of samples to consider regions with various scales at arbitrary locations on the map. Besides, this method may miss the optimal results.

2.3 Metric Learning

For applications that rely on distances or similarities, such as information retrieval and clustering, learning a good metric is important [15]. Recently, Convolutional Neural Networks (CNNs) based metric learning has achieved great performance on many Computer Vision tasks, such as image classification [34] and human

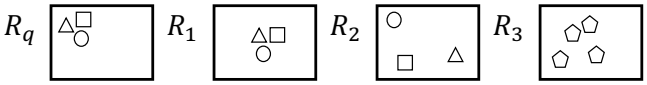


Figure 1: An example of defining $sim(\cdot)$.

re-identification [10]. The ability of capturing local spatial correlation makes CNN inherently applicable to geo-spatial data. Our model is based on a triplet architecture of CNNs, namely triplet network [11], which learns a ranking metric for region comparison. To apply triplet network on spatial data, we design a method to generate training data with hard negative example mining [30]. We also use a ratio-based training loss that is robust to highly skewed data.

3 PROBLEM DEFINITION

We consider a set of spatial objects O in a 2-dimensional rectangular geographical space P. Each object $o \in O$ is associated with an attribute vector $\mathbf{a}_o \in \mathbb{R}^{1\times d}$ and a geo-location $(o.x, o.y)$. The attributes of objects can be different in different applications, such as categories of POIs and vector representations of tweets. We define a region R as:

Definition 1 (Region). *A spatial rectangular subspace on P (i.e., $R \subset P$) bounded by (t, b, l, r) (i.e., top, bottom, left and right bound, respectively) that contains a set of objects $O_R \subset O$, where $\forall o \in O_R, b < o.x < t, l < o.y < r$.*

Note that we extend $\subset$ to denote the inclusion relation between two regions.

For any two regions R_1 and R_2, we define their similarity as $sim(R_1, R_2)$. The function $sim(\cdot)$ should consider both the attributes (i.e., $\mathbf{a}_o, \forall o \in O_R$) and the relative locations of the objects in a region. Figure 1 shows an example of the definition of $sim(\cdot)$. Suppose we have a query region R_q and three candidate regions R_1, R_2 and R_3. Different shapes represent different types of objects. In the three regions, R_3 is the most dissimilar region to R_q because it contains different types of objects to R_q. Both R_1 and R_2 contain the same number of objects from each type as R_q. However, the relative locations of the objects in R_1 are more similar to R_q, i.e., the three types of objects are co-located together, while the objects in R_2 are scattered. Thus, we have $sim(R_q, R_1) > sim(R_q, R_2) > sim(R_q, R_3)$.

We formulate the similar region search (SRS) problem as:

Definition 2 (SRS problem). *Given a geographical space P, a query region R_q and a similarity metric $sim(\cdot)$, the **similar region search (SRS) problem** is to retrieve a set of N regions (denoted as $\mathcal{R}$), such that $sim(R_q, R_i) \geq sim(R_q, R_j), \forall R_i \in \mathcal{R}, \forall R_j \notin \mathcal{R}$.*

In this paper, we take two steps to solve the *SRS* problem. We first propose a method to learn the similarity metric $sim(\cdot)$ in Section 4. Then, we develop efficient search algorithms based on the learned $sim(\cdot)$ in Section 5.

4 REGION SIMILARITY LEARNING

In this section, we prone to design $sim(\cdot)$ that (1) captures the information about the relative locations of objects, and (2) is robust to small noise and shift. Instead of designing a hand-crafted similarity metric, we present a deep metric learning method, namely triplet network, that directly learns $sim(\cdot)$ from data. Table 1 lists the notations for similarity learning.

Table 1: Notations for similarity learning.

Symbol	Description
R (or R_q)	a region (or a query region)
$sim(\cdot)$	the similarity metric
$Net(\cdot)$	the shared convolutional neural network
$g(\cdot)$	the feature aggregation function (max-pooling)
$\mathcal{X}_i$	the feature map of region R_i
$\mathbf{v}_i$	the aggregated feature vector or region R_i

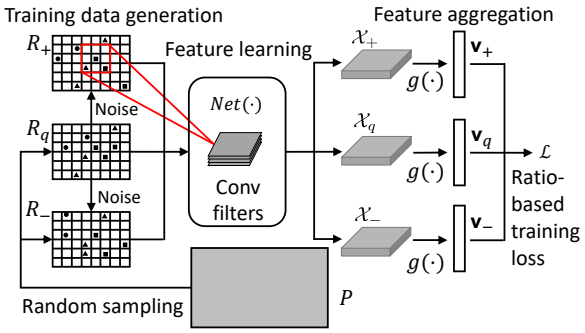


Figure 2: The workflow of region similarity learning.

4.1 Ranking Metric Learning Model

In *SRS*, it is hard to set a threshold to distinguish similar and dissimilar. However, given a query region and two candidate regions, it is easier to tell which one is more similar to the query than the other one. Therefore, we propose to learn a ranking metric for regions.

Triplet network. Triplet network [11] is widely-used in Computer Vision to learn ranking metrics between images. A triplet network comprises three instances of a shared convolutional neural network (CNN) [14] (denoted as $Net(\cdot)$). The input of a triplet network consists of: a query x_q, a positive example x_+ (i.e., similar) and a negative example x_- (i.e., dissimilar). When feeding the three examples, the triplet network will generate two Euclidean distances $d_+ = ||Net(x_q) - Net(x_+)||_2$ and $d_- = ||Net(x_q) - Net(x_-)||_2$, where $Net(x)$ is the feature map of x in the last layer of the CNN. Note that the fully-connected layer in the CNN is removed, as we are interested in a feature embedding only.

The goal of learning the triplet network is to train $Net(\cdot)$ such that the positive example has smaller distance than the negative example to the query [33]:

$$\mathcal{L} = \sum_{(x_q, x_+, x_-)} max\{0, d_+ - d_- + \delta\} + \lambda||Net(\cdot)||_2,$$

where δ is the gap parameters between two distances. $\lambda||Net(\cdot)||_2$ is a L_2 regularization term for $Net(\cdot)$, which is parameterized by λ.

The CNN in a triplet network is inherently appropriate for perceiving regions. In particular, convolutions can extract features of the local areas in a region, preserving the information about object attributes and how they are locally distributed. Moreover, the feature extraction is performed in a hierarchical manner, which is similar to how objects are organized in a region. Thus, it is helpful

to capture the relative locations between the objects at different levels (i.e., from local to global) of a region. Therefore, we apply triplet network to region similarity learning.

To feed a region into $Net(\cdot)$, we partition it into grids with a predefined granularity. We associate each grid with a vector by summing the attribute vectors of the objects in the grid cell. We associate empty grids with $\mathbf{0}$. A region is therefore represented as a 3-d tensor, i.e., $R \in \mathbb{R}^{w \times h \times d}$, where the first two dimensions are the spatial dimensions of the grid cells and the last dimension represents the attributes. We denote the output feature map of $Net(R)$ as $\mathcal{X} \in \mathbb{R}^{w^* \times h^* \times K}$ (i.e., $\mathcal{X} = Net(R)$), where K is the number of dimensions of the output features. Each of the K dimensions is a latent feature that contains spatial information. To compare feature maps with different sizes, we further apply a feature aggregation layer (denoted as $g(\cdot)$) after $Net(\cdot)$ to aggregate each feature map to a K-dimensional feature vector, i.e., $\mathbf{v} = g(\mathcal{X}), \mathbf{v} \in \mathbb{R}^{1 \times K}$. In this paper, we use a global max-pooling layer as $g(\cdot)$. Hence, the output distances are $d_+ = ||\mathbf{v}_q - \mathbf{v}_+||_2$ and $d_- = ||\mathbf{v}_q - \mathbf{v}_-||_2$. By using the learned metric, we define the similarity between two regions R_1 and R_2 as

$$sim(R_1, R_2) = \frac{1}{1 + ||\mathbf{v}_1 - \mathbf{v}_2||_2}.$$

4.2 Training Robust Triplet Network

We next propose to train a triplet network on regions that is robust to small noise and shift. In solving visual recognition tasks, the robustness of a triplet network is usually achieved by learning from semantic labels of data. However, there is no labeled data for region similarity. To solve this problem, we propose a self-supervised learning scheme that directly mines robustness by learning self-similarity, where the "labels" are free to obtain.

Training data generation. The robustness of $sim(\cdot)$ can also be interpreted in another way: regions that differ in a small amount of noise and shift should be considered as similar. Taking a leaf from this intuition, we propose to generate the query region R_q, the negative input R_- and the positive input R_+ as follows:

We randomly sample regions with various size at arbitrary locations as R_q. For each R_q, we first generate R_+ by adding noise and shift to R_q, including (1) randomly removing objects, (2) adding random objects at random locations, and (3) randomly shifting the locations of the objects. We control the generation of noise and shift by a *noise rate* and a *shift rate*, respectively. In particular, setting the noise rate as 0.1 means randomly deleting 10% of the objects then adding the same number random objects at random locations in the region. Setting the shift rate as 0.1 means moving each object along a random direction by a random distance, which is <10% of the region's height and width. The specific values of noise rate and shift rate used in the experiments are introduced in the parameter setting of Section 6.

We next generate R_- for each R_q. Specifically, we randomly sample regions on the map that does not overlap with R_q. However, some of the sampled R_- may be too dissimilar to R_q. In this case, the trivial task of distinguishing R_+ and R_- leads the model to converge on a trivial solution. To solve this problem, we also generate R_- by applying slightly more noise to R_q than the noise applied for generating R_+. By doing this, R_+ and R_- are both noisy versions of R_q, while R_+ contains less noise and thus more similar to R_q. We put a small portion of such hard examples in the training data. It pushes the model to learn more discriminative features to correctly distinguish R_+ and R_- and thus minimize the loss. Using the generated data, the model can learn features that are robust to small noise and random shift.

Ratio-based training loss. The numbers of objects in different regions are very different, which makes the triplet loss highly skewed. Some training tuples with very large loss values would overshadow other training data. A common way to solve this is to apply normalization to the input. However, this would loss the information about the number of objects in the regions, e.g., a region with 100 restaurants would be falsely considered similar to a region with 5 restaurants. Therefore, we resort to adjusting the loss computation. In particular, we propose a ratio-based training loss:

$$\mathcal{L} = \sum_{(R_q, R_+, R_-)} max\{0, \frac{d_+}{d_+ + d_-} - \delta\} + \lambda ||Net(\cdot)||_2,$$

The ratio-based loss for every training tuple is bounded in $[0, 1-\delta]$, and the extreme training instances would not excessively affect $\mathcal{L}$.

Figure 2 summarizes the process of training triplet network for regions. We propose to generate training tuples, feed them into a triplet network and train the model based on a ratio-based loss. The model can be optimized by commonly-used optimization methods, such as Stochastic Gradient Descent (SGD) and Adaptive Subgradient Methods (AdaGrad) [6].

Table 2: Notations for the search algorithms.

Symbol	Description
R_c	a candidate region
$\mathcal{X}_P$	the feature map of the geographical space P
$\mathcal{S}$	a search space
(t^*, b^*, l^*, r^*)	the location of a feature region on $\mathcal{X}_P$
$\hat{f}(\cdot)$	the distance lower bound function

5 REGION SEARCH ALGORITHMS

In this section, we propose efficient search algorithms to search similar region with the learned $sim(\cdot)$. For ease of presentation, Table 2 lists the notations for the search algorithms.

The geographical space P is a 2-dimensional continuous space, which contains an infinite number of regions and is infeasible to search. Therefore, we first divide P into grids with the same grid granularity used in the metric learning. A region is thus represented as a rectangular sub-area of grids in P.

Given a query R_q and the learned similarity metric $sim(\cdot)$, a naïve search method is to compare R_q with every candidate region $R_c \subset P$, i.e., computing $sim(R_q, R_c), \forall R_c \subset P$. However, this is very time-consuming due to two reasons:

- **Computing $sim(\cdot)$ is inefficient**. In particular, the main computational cost of $sim(R_q, R_c), \forall R_c \subset P$ lies in computing $\mathcal{X}_c = Net(R_c), \forall R_c \subset P$, which contains a lot of convolutions.
- **The number of candidate regions is numerous**. For a 30km × 30km city with a grid size of 10m×10m, the number of candidate regions is over 10^{13}. Even if we only search for regions whose height and width are within 500m to 3km, the number is still larger than 100 billion. Such number of data instances are

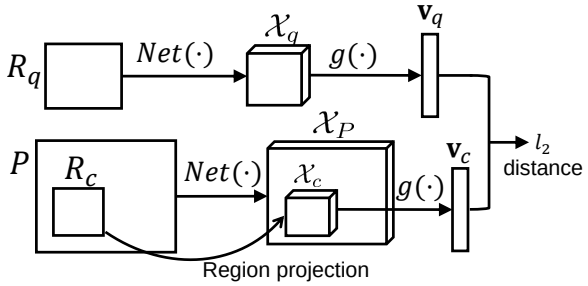


Figure 3: Sharing $\mathcal{X}_P$ for computing $sim(R_q, R_c)$.

impossible for exhaustive search and unmanageable for indexing methods like Locality Sensitive Hashing [5] and Kd-tree [2].

Therefore, to answer *SRS* queries efficiently, we prone to solve the above-mentioned issues in this section. In Section 5.1, we first share the computation for $Net(R_c), \forall R_c \subset P$, transforming the *SRS* problem to a *Similar Feature Region Search (SFRS)* problem. Section 5.2 and Section 5.3 present our exact and approximate solutions for the *SFRS* problem, respectively.

5.1 Problem Transformation via Sharing Computation

Sharing computation. Computing $\mathcal{X}_c = Net(R_c), \forall R_c \subset P$ is slow because it performs forward pass through $Net(\cdot)$ for each region, without sharing computation. Inspired by the idea of sharing computation in visual object recognition methods [8, 9, 24], we precompute the feature map of the entire geographical space, i.e., $\mathcal{X}_P = Net(P)$, and share it for computing $\mathcal{X}_c$. Figure 3 shows the computation of $sim(R_q, R_c)$ via sharing $\mathcal{X}_P$. When a query comes, we first compute $\mathbf{v}_q = g(Net(R_q))$. Next, we project the location of R_c on P (i.e., (t, b, l, r)) to the location of $\mathcal{X}_c$ on $\mathcal{X}_P$ (denoted as (t^*, b^*, l^*, r^*)). We can get $\mathcal{X}_c$ directly from the projected location on $\mathcal{X}_P$. The time complexity of region projection is $O(1)$ and the time cost of computing $Net(R_c)$ is therefore eliminated.

Problem transformation. By sharing computation, the *SRS* problem is equivalent to searching feature regions (i.e., $\mathcal{X}_c$) on $\mathcal{X}_P$ that are similar to $\mathcal{X}_q$. We call $\mathcal{X}_P$ the feature space and $\mathcal{X}_c$ a feature region in the rest of this paper. We formulate the problem as a *Similar Feature Region Search (SFRS)* problem:

Definition 3 (SFRS problem). *Given $\mathcal{X}_q$ and $\mathcal{X}_P$, searching a set of N feature regions (denoted as $\mathcal{F}$) on $\mathcal{X}_P$, such that*

$$||g(\mathcal{X}_q) - g(\mathcal{X}_i)||_2 \leq ||g(\mathcal{X}_q) - g(\mathcal{X}_j)||_2, \forall \mathcal{X}_i \in \mathcal{F}, \forall \mathcal{X}_j \notin \mathcal{F},$$

After finding the top-N similar feature regions, we can project their locations on $\mathcal{X}_P$ back to the locations of their corresponding regions on P. Hence, the top-N similar regions are found.

5.2 Exact Algorithm for *SFRS*

In *SFRS* problem, the second challenge still remains, i.e., the number of candidate feature regions is numerous. In this subsection, we propose an efficient search algorithm, namely ExactSFRS, that exactly solves the *SFRS* problem.

Representing search space. We apply a method for representing image patches [16] to represent a search space in *SFRS* problem.

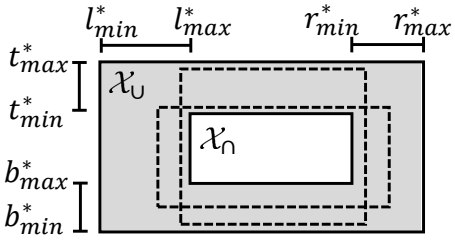


Figure 4: An example of search space.

In particular, the feature space $\mathcal{X}_P \in \mathbb{R}^{W^* \times H^* \times K}$ is a 3-d tensor, where W^* and H^* are its width and height, respectively. A candidate feature region $\mathcal{X}_c \in \mathbb{R}^{w^* \times h^* \times K}$ is a sub-tensor on $\mathcal{X}_P$ that has the same number of feature dimensions (i.e., K) as $\mathcal{X}_P$. Its location is parameterized by a tuple (t^*, b^*, l^*, r^*), where $1 \leq l^* < r^* \leq H^*$ and $1 \leq b^* < t^* \leq W^*$. By indicating the ranges of the four location parameters, a search space can be represented by four intervals:

Definition 4 (Search space). *A set of feature region locations $\mathcal{S} = \bar{t} \times \bar{b} \times \bar{l} \times \bar{r}$, where $\bar{t} = [t^*_{min}, t^*_{max}]$, $\bar{b} = [b^*_{min}, b^*_{max}]$, $\bar{l} = [l^*_{min}, l^*_{max}]$ and $\bar{r} = [r^*_{min}, r^*_{max}]$, such that $\forall (t^*, b^*, l^*, r^*) \in \mathcal{S}$ satisfies $t^* \in \bar{t}$, $b^* \in \bar{b}$, $l^* \in \bar{l}$ and $r^* \in \bar{r}$.*

Here, $\times$ denotes the Cartesian product of intervals. Figure 4 shows an intuitive example of a search space. In this case, the four intervals form an area (i.e. the shadowed area) between a large feature region $\mathcal{X}_\cup$ (i.e., $(t^*_{max}, b^*_{min}, l^*_{min}, r^*_{max})$) and a small feature region $\mathcal{X}_\cap$ (i.e., $(t^*_{min}, b^*_{max}, l^*_{max}, r^*_{min})$). This search space $\mathcal{S}$ therefore represents a set of all the feature regions that lie in this area, i.e., between $\mathcal{X}_\cap$ and $\mathcal{X}_\cup$ (i.e., $\mathcal{X}_\cap \subset \mathcal{X}_c \subset \mathcal{X}_\cup, \forall \mathcal{X}_c \in \mathcal{S}$). The dashed rectangles are two examples of the feature regions in $\mathcal{S}$. Using this representation, the entire search space can be denoted as $\mathcal{S}_\forall = [1, H^*] \times [1, H^*] \times [1, W^*] \times [1, W^*]$.

Branch and bound search. Instead of exhaustively search in $\mathcal{S}_\forall$, we use a branch and bound search scheme. The intuition is that the objects on a geographical space are not randomly distributed. Some search subspaces are more probably to contain the optimal results.

To perform a branch and bound search, we first define the distance lower bound of $\mathcal{S}$ given a query $\mathcal{X}_q$ as $\hat{f}(\mathcal{S}|\mathcal{X}_q) \leq ||g(\mathcal{X}_q) - g(\mathcal{X}_c)||_2, \forall \mathcal{X}_c \in \mathcal{S}$. The function $\hat{f}(\mathcal{S}|\mathcal{X}_q)$ estimates the minimum distance that $\exists \mathcal{X}_c \in \mathcal{S}$ could have compared to $\mathcal{X}_q$. Intuitively, lower $\hat{f}(\mathcal{S}|\mathcal{X}_q)$ means a higher chance that $\exists \mathcal{X}_c \in \mathcal{S}$ is a top-N similar feature region of $\mathcal{X}_q$. To construct $\hat{f}(\mathcal{S}|\mathcal{X}_q)$, we first reexamine the example of search space in Figure 4, where we have $\mathcal{X}_\cap \subset \mathcal{X}_c \subset \mathcal{X}_\cup, \forall \mathcal{X}_c \in \mathcal{S}$. Moreover, because $g(\cdot)$ (i.e., max-pooling) is a monotonically increasing function, we can conclude that $g(\mathcal{X}_\cap) \leq g(\mathcal{X}_c) \leq g(\mathcal{X}_\cup)$, i.e., $\mathbf{v}_\cap[k] \leq \mathbf{v}_c[k] \leq \mathbf{v}_\cup[k], \forall k \in [1, K]$, where $\mathbf{v}[k]$ denotes the k-th dimension of $\mathbf{v}$. We reformulate the distance computation between $\mathcal{X}_c$ and $\mathcal{X}_q$ as $\sqrt{\sum_{k \in [1,K]} (\mathbf{v}_q[k] - \mathbf{v}_c[k])^2}$, where each of the K terms $(\mathbf{v}_q[k] - \mathbf{v}_c[k])^2$ can be bounded using $\mathbf{v}_\cap[k] \leq \mathbf{v}_c[k] \leq \mathbf{v}_\cup[k]$ as:

(i) $(\mathbf{v}_q[k] - \mathbf{v}_c[k])^2 \geq (\mathbf{v}_q[k] - \mathbf{v}_\cap[k])^2$, if $\mathbf{v}_\cap[k] \geq \mathbf{v}_q[k]$.
(ii) $(\mathbf{v}_q[k] - \mathbf{v}_c[k])^2 \geq (\mathbf{v}_q[k] - \mathbf{v}_\cup[k])^2$, if $\mathbf{v}_\cup[k] \leq \mathbf{v}_q[k]$.

We denotes the k values where $\mathbf{v}_\cap[k] > \mathbf{v}_q[k]$ as k_1 and those satisfy $\mathbf{v}_\cup[k] < \mathbf{v}_q[k]$ as k_2. Therefore, the distance lower bound is

Algorithm 1: ExactSFRS

Input: Initial search space $\mathcal{S}_\forall$, query feature region $\mathcal{X}_q$, distance lower bound $\hat{f}(\cdot)$
Output: top-N similar feature regions $\mathcal{F}$

```
1  begin
2      F ← ∅; Q ← ∅;
3      Q.Insert(S_∀);
4      repeat
5          repeat
6              S' ← Q.RetrieveTop() ;
7              split S → S_1 ∪ S_2;
8              Q.Insert((f̂(S_1|X_q), S_1)) ;
9              Q.Insert((f̂(S_2|X_q), S_2)) ;
10         until |S'| = 1;
11         F ← F ∪ S';
12     until |F| = N;
13 end
```

given by

$$\hat{f}(\mathcal{S}|\mathcal{X}_q) = \sqrt{\sum_{k_1}(\mathbf{v}_q[k_1] - \mathbf{v}_\cap[k_1])^2 + \sum_{k_2}(\mathbf{v}_q[k_2] - \mathbf{v}_\cup[k_2])^2}.$$

Note that when $t^*_{min} < b^*_{max}$ or $r^*_{min} < l^*_{max}$, the feature region $\mathcal{X}_\cap$ does not exist. In this case, we use $\hat{f}(\mathcal{S}|\mathcal{X}_q) = \sqrt{\sum_{k_2}(\mathbf{v}_q[k_2] - \mathbf{v}_\cup[k_2])^2}$ to bound the search space.

We next present the branch and bound search scheme of ExactSFRS in Algorithm 1, which contains the following steps:

- *Initialization* step (line 2-3): We define an empty result set $\mathcal{F}$ and a priority queue Q. We put the initial search space $\mathcal{S}_\forall$ in Q.
- *Branch* step (line 6-7): We retrieve from Q the top search space $\mathcal{S}'$, which has the smallest distance lower bound in the current Q, i.e., $\hat{f}(\mathcal{S}'|\mathcal{X}_q) \le \hat{f}(\mathcal{S}|\mathcal{X}_q), \forall \mathcal{S} \in Q$. Next, we split $\mathcal{S}'$ into two disjoint subspace $\mathcal{S}_1$ and $\mathcal{S}_2$ by equally dividing the largest interval of its $\bar{t}$, $\bar{b}$, $\bar{l}$ and $\bar{r}$, while the other three keep unchanged.
- *Bound* step (line 8-9): We compute $\hat{f}(\mathcal{S}_1|\mathcal{X}_q)$ and $\hat{f}(\mathcal{S}_2|\mathcal{X}_q)$, and then put them into Q with $\mathcal{S}_1$ and $\mathcal{S}_2$, respectively.

We repeat the branch and bound steps until $|\mathcal{S}'| = 1$, i.e., the top-retrieved space only contains 1 feature region (denoted as $\mathcal{X}'$). This only happens when the intervals of $\mathcal{S}'$ satisfy $t^*_{max} = t^*_{min}$, $b^*_{max} = b^*_{min}$, $l^*_{max} = l^*_{min}$ and $r^*_{max} = r^*_{min}$. For $\mathcal{S}'$ in this case, we have $\mathcal{X}_\cap = \mathcal{X}_\cup = \mathcal{X}'$ and $k_1 \cup k_2 = [1, K]$, which indicates $\hat{f}(\mathcal{S}'|\mathcal{X}_q) = ||\mathbf{v}_q - \mathbf{v}'||_2$. Therefore, $\mathcal{X}'$ can be guaranteed to be the top-N similar feature regions of $\mathcal{X}_q$ by

$$||\mathbf{v}_q - \mathbf{v}'||_2 \le \hat{f}(\mathcal{S}|\mathcal{X}_q) \le ||\mathbf{v}_q - \mathbf{v}_c||_2, \forall \mathcal{S} \in Q, \forall \mathcal{X}_c \in \mathcal{S}.$$

We put $\mathcal{X}'$ into $\mathcal{F}$ and continue the search process until N results are found. The feature regions in $\mathcal{F}$ are the top-N similar feature regions, which can be project back to find the top-N similar regions on P. Note that the method of constructing distance lower bound is also applicable to other $g(\cdot)$ and L_p distance, as long as $g(\cdot)$ is monotonically increasing, such as max-pooling and sum-pooling.

Time complexity analysis. In the branch step, splitting a search space takes $O(1)$ time; In the bound step, we use integral histogram technique [21, 29] to compute $\mathbf{v}_\cap$ and $\mathbf{v}_\cup$ and the total complexity of the bound step is $O(K)$. In the worst case, the search process needs to check totally $2|\mathcal{S}_\forall| - 1$ search spaces and takes $O(\log|\mathcal{S}_\forall|)$ time to maintain Q, where $|\mathcal{S}_\forall|$ denotes the total number of candidate feature regions. Therefore, the total complexity is $O(K|\mathcal{S}_\forall|\log|\mathcal{S}_\forall|)$. Although the worst-case complexity is very high, ExactSFRS is empirically efficient as shown in the experiments.

Table 3: Spatial object datasets.

Dataset	SG	NYC	US
object type	POI	POI	Tweet
#objects	244,266	435,210	1,980,943
width (W)	47.7km	61.2km	6078.3km
height (H)	28.7km	48.1km	2608.4km
#attr.	10	10	50
attr. type	Category	Category	Topic

5.3 Approximate Algorithm for *SFRS*

In an exploratory search environment, the users usually need very fast response and can endure a slightly decrease of the accuracy. For example, social media websites need to efficiently response the queries launched by thousands of users at the same time. Moreover, the queries can be submitted interactively in an exploratory manner. To adapt our model to this kind of scenarios, we propose an approximation method, namely ApproxSFRS, that can further improve the efficiency by slightly sacrificing the accuracy.

ApproxSFRS uses the same branch and bound search scheme as ExactSFRS, except that we limit the size of Q by a constant M, where $M \le |\mathcal{S}_\forall|$. When $Q.size > M$ during the search process, the worst search space (i.e., has the largest $\hat{f}(\mathcal{S}|\mathcal{X}_q)$) in Q will be discarded. To support popping the worst search space out of Q, we re-introduce Q as a min-max heap, whose time complexity of retrieving min/max value and insertion are $O(\log M)$ in our case.

The intuition behind ApproxSFRS is to reduce the unnecessary branch and bound for the bad search spaces. The reason is that branching a bad search space may generate two worse search spaces, who are also unlikely to contain similar regions. Therefore, we choose to discard those bad search space to prevent wasting computation on its subspaces.

Time complexity analysis. The time complexity of ApproxSFRS is $O(KM\log M(1 + \log|\mathcal{S}_\forall| - \log M))$ (see Appendix for the proof[1]). M can be varied to adjust the time complexity for different applications. Particularly, when $M = |\mathcal{S}_\forall|$, the complexity becomes the same as ExactSFRS, i.e., $O(K|\mathcal{S}_\forall|\log|\mathcal{S}_\forall|)$; When $M = 1$, the complexity becomes $O(K\log|\mathcal{S}_\forall|)$, which is equivalent to a binary search in $\mathcal{S}_\forall$. Therefore, ApproxSFRS is essentially a trade-off between ExactSFRS and a binary search method.

6 EXPERIMENTS

6.1 Experimental Setup

Datasets. Our experiments are conducted on three spatial object datasets. Table 3 presents the statistics of the datasets. In particular, we collect two POI datasets from Singapore (SG) and New York area

[1] http://www.ntu.edu.sg/home/kqzhao/srs_appendix.pdf

(NYC), respectively. The attribute of a POI is its category (e.g., hotel), which is represented as an one-hot vector. To evaluate the efficiency of our search algorithms, we further introduce a large dataset US, which contains around 2 million tweets from United States (except Alaska and Hawaii). We train a Latent Dirichlet Allocation (LDA) [3] model using these tweets, each of which is associated with a topic distribution vector as its attributes. We partition the regions and the geographical space of SG and NYC into 10m×10m grids. We use 500m×500m grids for US.

Test data generation. To evaluate the effectiveness of the methods, we use a similar method as described in Section 4.2 to generate test data. In particular, we randomly sample 2000 regions that contains more than 50 objects as the test queries, denoted as D_q. The height and width of a region vary from 640m to 3km. For an $R_q \in D$, we create a similar region R_+ as the ground truth by applying a small amount of noise and shift to R_q. We create a candidate region database as $D_c = (D_q/R_q) \cup R_+$ (also excluding those have overlap with R_q), considering other regions in D_q as dissimilar regions to R_q. We use R_q to retrieve its similar regions from D_c. Ideally R_+ is ranked at the top. We vary the noise rate and shift distance of generating R_+ in the effectiveness experiments.

To evaluate the efficiency, we reuse the 2000 random sampled R_q to search on P and record the average runtime for each method. To prevent finding too large or too small regions that are usually uninformative to users, we only consider the candidate regions R_c whose width and height are constrained by $0.5w_q \le w_c \le 2w_q$ and $0.5h_q \le h_c \le 2h_q$.

Baselines. To evaluate the effectiveness of our deep metric learning method (denoted as Triplet), we compare it with the following baseline similarity metrics.

- SPM [17]. This method is originally proposed for image categorization. SPM recursively partitions a region into four fine-grained grids and forms a pyramid structure. The l-th pyramid level has $2^{l-1} \times 2^{l-1}$ grids, where $l = 1, 2, 3 \cdots$. SPM computes the average grid-wise similarity for each level and sum up all levels with different weights. We use SPM3, SPM4 and SPM5 to represent partitioning a region into a pyramid with 3, 4 and 5 levels. We compare our method with SPM3, SPM4 and SPM5.
- Grid. This method is equivalent to comparing a specific level of two region pyramids in SPM. We denote a level with 4×4 grids as Grid4. We use Grid4, Grid8 and Grid16 as baselines.
- SVSM [25]. This method defines the four corners and the center of a region as its "reference points", and compute the average distance of each category of objects to each reference point. To compare two regions, it computes the cosine similarity between their distance vectors.

For the efficiency evaluation, we compare ExactSFRS with the following search methods:

- Quad [25]. This is a quadtree based search method developed for SVSM. It builds a quadtree for the geographical space and searches the tree nodes that are similar to the query in a top-down manner. The optimal tree nodes are further expanded greedily as the final similar regions.
- BruteSFRS. This is a brute-force method that solves the *SFRS* problem by comparing $\mathcal{X}_q$ with every candidate feature region, i.e., $\forall \mathcal{X}_c \in \mathcal{S}_\forall$.

Parameter setting. For all the experiments, we use a 3-layer CNN as $Net(\cdot)$. Note that all the layers are convolutional layers with 2×2 stride (instead of using pooling layers). A ReLU non-linearity is applied between two consecutive layers. The filter sizes and feature map dimensions (ordered from input to output) are {11,9,7} and {128, 64, 32}, respectively. Note that we use large filters in the convolutional layers, which is helpful for capturing the spatial relations between sparsely distributed objects. For the regularization, we set $\lambda = 0.00005$ and further apply dropout [27] to the output of $Net(\cdot)$. The dropout rate is set to 0.1. The gap parameter for training loss is $\delta = 0.3$. We train our model on 10k generated training tuples. For generating each R_+, we randomly set the noise rate and the shift rate between 0.1 to 0.3.

Evaluation metrics. We use two metrics to evaluate the effectiveness. (1) *HR@10*. Given the top-10 returned regions $\mathcal{R}$ for a query, HR@10 is defined as $HR@10 = 1$ if $R_+ \in \mathcal{R}$, $HR@10 = 0$ otherwise. We report the average HR@10 over all queries in the experiments. (2) *MRR*. MRR is a rank-based metric defined as $MRR = \frac{1}{rank}$, where $rank$ denotes the rank of R_+ in D_c for a query R_q. We report the average MRR over all queries in the experiments.

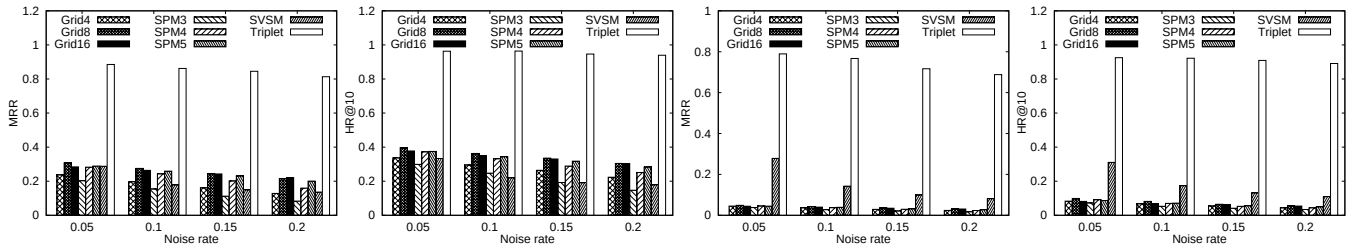
Experimental environment. All the methods are implemented in Python 3.5. The implementation of Triplet is based on Keras [4] (v2.0.6) with tensorflow [1] backend. The training can be done within several hours on a single NVIDIA Tesla P100 GPU (16GB memory). All the experiments are conducted on a server with 20-Cores Intel(R) Xeon(R) CPU E5-2680 v2 @2.8GHz and 64GB RAM.

6.2 Effectiveness of Triplet

Overall effectiveness. We first study the effectiveness of Triplet. We fix the shift distance as 50m (i.e., shifting the objects along a random direction by 50m) and vary the noise rate from 0.05 to 0.2. Figure 5 shows the results of different methods on both SG and NYC datasets with respect to the two metrics. We can see that Triplet is significantly better than the baselines on both datasets in terms of both HR@10 and MRR. For example, when applying only 5% noise, Triplet outperforms SVSM by 180% and 200% on SG and NYC, respectively, in terms of MRR. Grid and SPM perform worst in most of the cases, because they are developed for object categorization in images and cannot adapt to spatial object data very well. Moreover, Grid and SPM perform worse on NYC than on SG. This is because NYC has larger area and more spatial objects. Grid and SPM may easily fail on such massive and noisy data. In contrast, Triplet is very robust to noise. When increasing the noise rate from 0.05 to 0.2 on NYC data, the MRR and HR@10 of Triplet only decrease by 3%–12%, while those values of SVSM decline by 64%–70%.

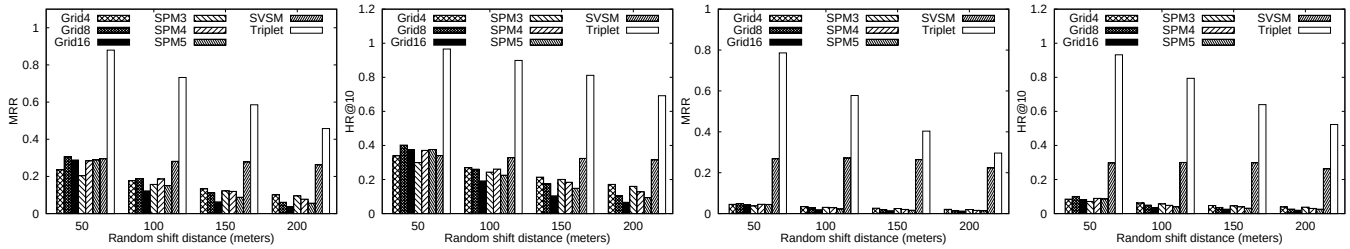
We next fix the noise rate as 0.05 and vary the shift distance from 50m to 200m. Figure 6 shows the results. Triplet still performs the best on different datasets and in terms of both metrics. For example, it outperforms SVSM by more than 55% on NYC in terms of HR@10. We also note that the performance of Triplet decreases when applying larger shift distance. This is because largely shifting objects may change the form of their relative locations, which affects the similarity. The method that does not consider their relative locations do not show this property.

Ablation study. We also study our improvements on triplet network for spatial data, i.e., using ratio-based training loss (denoted



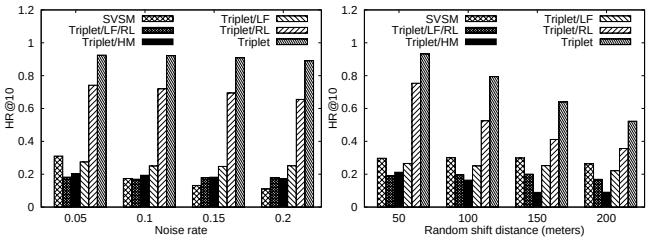
(a) MRR-SG (b) HR@10-SG (c) MRR-NYC (d) HR@10-NYC

Figure 5: Varying noise rate.



(a) MRR-SG (b) HR@10-SG (c) MRR-NYC (d) HR@10-NYC

Figure 6: Varying shift distance.



(a) Noise-NYC (b) Random shift-NYC

Figure 7: Comparing different triplet networks.

as RL) and applying hard negative example mining in training data generation (denoted as HM). We also include using large filters in CNN (denoted as LF) as an improvement. Because small filters are usually applied to image data [26]. We denote our triplet network without RL, HM and LF as Triplet/RL, Triplet/HM and Triplet/LF, respectively. Figure 7 compares the different versions of Triplet with SVSM on NYC in terms of HR@10. The results on SG and for MRR are similar and thus omitted. We can see that LF and HM are the most important for training triplet network on regions. For example, in Figure 7(b), the performance of Triplet/LF and Triplet/HM are even worse than SVSM by more than 10% and 40%, respectively. In addition, the results also show that RL can further improve the effectiveness. On NYC data, the improvement of RL on HR@10 is around 25%–50%.

6.3 Efficiency of ExactSFRS and ApproxSFRS

Overall efficiency. Figure 8 presents the average runtime of the methods for top-1 similar region search on three datasets. The runtime of BruteSFRS is too long (more than one day per query). Thus, we report its estimated runtime, which is computed by the time cost for checking one candidate region times the total number of candidate regions. Quad runs out of memory on US data and thus we do not show the result. We can see from the figure that ExactSFRS outperforms the two baselines by a significant margin. Particularly, ExactSFRS is over $10^5\times$ faster than BruteSFRS on all datasets, and more than 130× and 45× faster that Quad on SG and NYC, respectively. Moreover, even for a very large dataset like US, the runtime of ExactSFRS is still less than 100 seconds. These show us that ExactSFRS can greatly reduce the search space and is very efficient for similar region search.

Scalability. We demonstrate the scalability of ExactSFRS with respect to N (i.e., top-N similar region search). Figure 9 shows the runtime of ExactSFRS when varying N from 1 to 100. We can see that ExactSFRS scales well to N. For example, when N increases from 1 to 100, the runtime on US only increases by 5%.

We further investigate the scalability of ExactSFRS with respect to regions with different numbers of objects. We divide the query regions into five groups based on their numbers of objects: "<100", "100–500", "500–1k", "1k–2k" and "2k<", each of which indicates the range for the numbers of objects in a group. Table 4 compares the average runtime of ExactSFRS and Quad for different query groups on SG. The results show that Quad takes longer to answer for queries that contain more objects. Particularly, it costs around 50% more time for the last group of queries (2k<) than for the first group of queries (<100). In contrast, ExactSFRS is more efficient to answer queries that contain more objects. In particular, its runtime for the last group (2k<) drops by over 66% compared with its runtime for the first group (<100). The reasons are twofold: (1) After sharing the entire feature space $\mathcal{X}_P$, the time complexity of computing similarity between two feature regions is $O(K)$, which is irrelevant to the numbers of objects in the regions; (2) More objects can

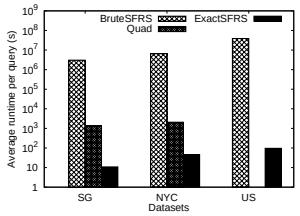


Figure 8: Runtime on different datasets.

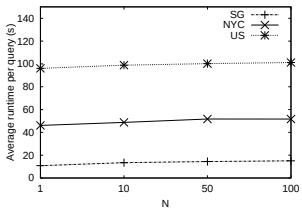
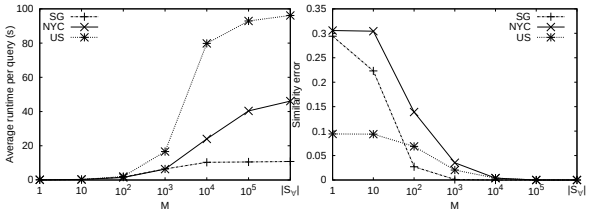


Figure 9: Runtime of ExactSFRS by varying N.



(a) Runtime (b) Similarity error

Figure 10: Trade-off of ApproxSFRS by varying M.

Table 4: Average runtime (sec.) for query regions with different numbers of objects on SG.

#obj.	<100	100–500	500–1k	1k–2k	2k<
Quad	1077.5	1281.31	1326.13	1549.77	1606.60
ExactSFRS	20.23	12.50	8.01	7.13	6.85

Table 5: Average runtime (sec.) for query regions with different areas (km^2) on SG.

Area (km^2)	<2	2–3	3–4	4–5	5<
Quad	1103.89	1098.22	1303.07	1421.82	1893.99
ExactSFRS	8.28	9.36	11.14	11.43	14.32

provide more information to characterize the spatial features in a region, which can be leveraged by ExactSFRS to reduce the search space even more effectively.

In addition, we also illustrate the scalability of ExactSFRS w.r.t. different query region sizes, in which we divide the query regions into five groups based on their area (km^2): "<2", "2–3", "3–4", "4–5" and "5<". Table 5 shows the runtime of Quad and ExactSFRS for the five query groups on SG. We observe that both Quad and ExactSFRS scale linearly with respect to the area of query region.

Trade-off between efficiency and accuracy. We vary the maximum queue size M in ApproxSFRS from 1 to $|\mathcal{S}_\forall|$. Figure 10(a) and 10(b) present the average runtime and similarity error of ApproxSFRS, respectively, for top-1 similar region search. The similarity error is defined as the difference between the similarity values produced by ExactSFRS and ApproxSFRS. The results show that when M is small, ApproxSFRS is extremely efficient but the error is large. For example, when $M \leq 10$, the similarity difference is larger than 0.2 on SG and NYC. In contrast, when M becomes larger, ApproxSFRS tends to find the accurate results, while the runtime also increases largely. We find that when $M = 10^3$, ApproxSFRS can achieve a good trade-off, where the runtime is reduced by 40.31%, 86.23% and 82.72% on SG, NYC and US, respectively, compared to $M = \mathcal{S}_\forall$ (i.e., ExactSFRS). Meanwhile, the similarity error is no larger than 0.05. Thus, ApproxSFRS can further improve the search efficiency by slightly sacrificing the accuracy.

7 CONCLUSION

In this paper, we study a similar region search problem on geographical space. We leverage a deep metric learning method to learn a similarity metric for comparing regions. The method can capture both the attributes and the relative locations between objects in regions for measuring similarity, and is robust to small noise and shift. To search similar regions efficiently, we propose a branch and bound search algorithm that can greatly reduce the search space. We also develop an approximation method to further improve efficiency by slightly trading off the accuracy. Experiments on three datasets demonstrate our proposed methods outperform baseline methods for similar region search.

Acknowledgements.This research was carried out at the Rapid-Rich Object Search (ROSE) Lab at the Nanyang Technological University, Singapore. The ROSE Lab is supported by the National Research Foundation, Singapore, and the Infocomm Media Development Authority, Singapore. This work is also supported by the MOE Tier-2 grant (MOE2016-T2-1-137) awarded by Ministry of Education Singapore. We thank Prof. Guosheng Lin (Nanyang Technological University) for valuable discussion.

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.
[2] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.
[3] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *JMLR* 3, Jan (2003), 993–1022.
[4] François Chollet et al. 2015. Keras. (2015).
[5] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *SoCG*. ACM, 253–262.
[6] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR* 12, Jul (2011), 2121–2159.
[7] M Erlebach, P Klapka, M Halás, and P Tonev. 2014. Inner structure of functional region: theoretical aspects. In *17th International Colloquim on Regional Science, Conference Proceedings*. Masaryk University Brno, 722–727.
[8] Ross Girshick. 2015. Fast r-cnn. In *ICCV*. 1440–1448.
[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2014. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*. Springer, 346–361.
[10] Alexander Hermans, Lucas Beyer, and Bastian Leibe. 2017. In Defense of the Triplet Loss for Person Re-Identification. *arXiv preprint arXiv:1703.07737* (2017).
[11] Elad Hoffer and Nir Ailon. 2015. Deep metric learning using triplet network. In *ICLR Workshop*. Springer, 84–92.
[12] Mohamed Kafsi, Henriette Cramer, Bart Thomee, and David A Shamma. 2015. Describing and understanding neighborhood characteristics through online social media. In *WWW*. International World Wide Web Conferences Steering Committee, 549–559.
[13] P Klapka, M Halás, and P Tonev. 2013. Functional regions: concept and types. In *16th International Colloquium on Regional Sciences, Conference Proceedings. Brno, Masaryk University*. 94–101.
[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*. 1097–1105.
[15] Brian Kulis et al. 2013. Metric learning: A survey. *Foundations and Trends® in Machine Learning* 5, 4 (2013), 287–364.
[16] Christoph H Lampert, Matthew B Blaschko, and Thomas Hofmann. 2009. Efficient subwindow search: A branch and bound framework for object localization. *IEEE*

TPAMI 31, 12 (2009), 2129–2142.
[17] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. 2006. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, Vol. 2. IEEE, 2169–2178.
[18] Géraud Le Falher, Aristides Gionis, and Michael Mathioudakis. 2015. Where Is the Soho of Rome? Measures and Algorithms for Finding Similar Neighborhoods in Cities.. In *ICWSM*. 228–237.
[19] Yiding Liu, Tuan-Anh Nguyen Pham, Gao Cong, and Quan Yuan. 2017. An experimental evaluation of point-of-interest recommendation in location-based social networks. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1010–1021.
[20] Tuan-Anh Nguyen Pham, Xutao Li, and Gao Cong. 2017. A General Model for Out-of-town Region Recommendation. In *WWW*. IW3C2, 401–410.
[21] Fatih Porikli. 2005. Integral histogram: A fast way to extract histograms in cartesian spaces. In *CVPR*, Vol. 1. IEEE, 829–836.
[22] Daniel Preoţiuc-Pietro, Justin Cranshaw, and Tae Yano. 2013. Exploring venue-based city-to-city similarity measures. In *KDD Workshop on Urban Computing*. ACM, 16.
[23] Ali Sharif Razavian, Josephine Sullivan, Stefan Carlsson, and Atsuto Maki. 2014. Visual Instance Retrieval with Deep Convolutional Networks. *arXiv preprint arXiv:1412.6574* (2014).
[24] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*. 91–99.
[25] Chang Sheng, Yu Zheng, Wynne Hsu, Mong Li Lee, and Xing Xie. 2010. Answering top-k similar region queries. In *DSAA*. Springer, 186–201.
[26] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).
[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *JMLR* 15, 1 (2014), 1929–1958.
[28] Waldo R Tobler. 1970. A computer movie simulating urban growth in the Detroit region. *Economic geography* 46, sup1 (1970), 234–240.
[29] Giorgos Tolias, Ronan Sicre, and Hervé Jégou. 2015. Particular object retrieval with integral max-pooling of CNN activations. *arXiv preprint arXiv:1511.05879* (2015).
[30] Chong Wang, Xue Zhang, and Xipeng Lan. 2017. How to Train Triplet Networks with 100K Identities? *arXiv preprint arXiv:1709.02940* (2017).
[31] Hongjian Wang, Daniel Kifer, Corina Graif, and Zhenhui Li. 2016. Crime rate inference with big data. In *KDD*. ACM, 635–644.
[32] Hongjian Wang and Zhenhui Li. 2017. Region Representation Learning via Mobility Flow. In *CIKM*. ACM, 237–246.
[33] Xiaolong Wang and Abhinav Gupta. 2015. Unsupervised learning of visual representations using videos. In *ICCV*. 2794–2802.
[34] Xiaolong Wang, Kaiming He, and Abhinav Gupta. 2017. Transitive Invariance for Self-supervised Visual Representation Learning. *arXiv preprint arXiv:1708.02901* (2017).
[35] Ying Wei, Yu Zheng, and Qiang Yang. 2016. Transfer Knowledge between Cities.. In *KDD*. 1905–1914.
[36] Jing Yuan, Yu Zheng, and Xing Xie. 2012. Discovering regions of different functions in a city using human mobility and POIs. In *KDD*. ACM, 186–194.
[37] Xiangyu Zhao and Jiliang Tang. 2017. Modeling Temporal-Spatial Correlations for Crime Prediction. In *CIKM*. ACM, 497–506.