A General Model for Out-of-town Region Recommendation

Tuan-Anh Nguyen Pham
School of Computer Science and Engineering
Nanyang Technological University, Singapore
pham0070@e.ntu.edu.sg

Xutao Li
Department of Computer Science
Shenzhen Graduate School, Harbin Institute of Technology
lixutao@hitsz.edu.cn

Gao Cong
School of Computer Science and Engineering
Nanyang Technological University, Singapore
gaocong@ntu.edu.sg

ABSTRACT

With the rapid growth of location-based social networks (LBSNs), it is now available to analyze and understand user mobility behavior in real world. Studies show that users usually visit nearby points of interest (POIs), located in small regions, especially when they travel out of their hometowns. However, previous out-of-town recommendation systems mainly focus on recommending individual POIs that may reside far from each other, which makes the recommendation results less useful. In this paper, we introduce a novel problem called Region Recommendation, which aims to recommend an out-of-town region of POIs that are likely to be visited by a user. The proximity characteristic of user mobility behavior implies that the probability of visiting one POI depends on those of nearby POIs. Thus, to make accurate region recommendation, our proposed model exploits the influence between POIs, instead of treating them individually. Moreover, to overcome the efficiency problem of searching the best region, we propose a sweeping line-based method, and subsequently an constant-bounded algorithm for better efficiency. Experiments on two real-world datasets demonstrate the improved effectiveness of our models over baseline methods and efficiency of the approximate algorithm.

Keywords

Location-based social networks; region recommendation; out-of-town recommendation

1. INTRODUCTION

Recent years have witnessed a rapid development of location-based social networks (LBSNs), such as Foursquare and Facebook Places. In these online services, users are able to share their locations and experiences, by checking in via mobile devices, when visiting points of interest (POIs), *e.g.*, restaurants, museums. As of December 2015, Foursquare had 55 million monthly active users and totally 7 billion check-ins. Being overwhelmed by the huge amount of data from LBSNs, users usually encounter difficulties in finding or searching interesting POIs. Motivated by the problem, recently, there is a growing interest in POI recommendation problem, such as [21, 22, 14, 13]. Success in POI recommendation is important as it helps users explore new locations, which benefits both users and LBSN services.

POI recommendation is particularly important when a user travels to a new place, *e.g.*, new city or country, where she has very little, or even no, knowledge about her destination. This leads to a new problem, namely out-of-town recommendation, which aims to find POIs that a given user may be interested in when she travels out of her hometown. Out-of-town recommendation has received little research interest. Existing work on this problem mainly focuses on improving the recommendation accuracy on individual POIs [1, 22, 20]. However, we argue that it is often more beneficial to recommend a set of nearby locations to users rather than just individual locations. This is because, when traveling to new cities, users usually have very limited time to visit places. Investigations on Foursquare data show that a large proportion of visitors visited multiple POIs resided in small regions, and this proportion is higher when the visiting duration is limited (details of these findings are presented in Section 3). Hence, in this paper, we introduce a new problem, namely Region Recommendation, to recommend out-of-town regions to users. Specifically, a region is preferred if it contains more POIs that the given user would like to visit. To the best of our knowledge, this is the first work on region recommendation for out-of-town users.

There are two challenges to solve the region recommendation problem. The first challenge is how to measure the attractiveness of a region to a user. One intuitive way is to compute the user's attractiveness score of each POI in the region and then aggregate them as the score of the region. However, this approach considers each POI independently and ignores the influence between POIs. In contrast, the user's decision to visit a place is also affected by how much she is interested in nearby POIs [16, 13, 2]. In other words, nearby POIs can reinforce each other to attract users. For example, when choosing hotels to stay, travelers usually prefer the ones that are near their interesting places, such as cinemas, restaurants and tourist attractions (museums, theaters, etc.). Considering POIs individually neglects such influences between POIs. As a result, it is necessary to consider the POI influences when searching the optimal region. As a result, in this paper, we propose a general procedure to estimate the region's attractiveness with the POI interaction being taken into account.

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052667

The second challenge is the efficiency of searching the optimal region. Given a set of POIs in a 2-dimension space, there are *infinite* number of ways to place a region of a given size. Obviously, we cannot check all the cases to find the optimal region, and hence a more efficient approach is required to find the result region quickly. In this paper, based on the sweeping algorithm [10, 18], we introduce a simple but efficient method to find the optimal region. Specifically, we convert the original problem into a geometric intersection problem, and then apply the sweeping algorithm. As a result, the region searching problem can be solved in $\mathcal{O}((N+M)\log N)$, where N and M are the number of POIs and their pair-wise interactions, respectively. Moreover, to achieve better efficiency, a constant-bounded approximate algorithm is developed for the region search problem.

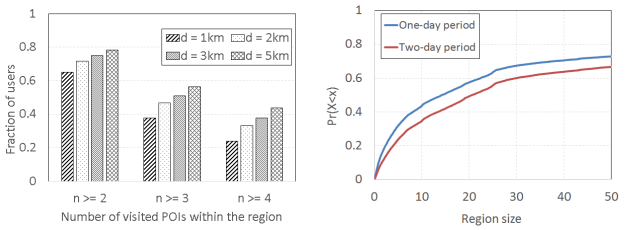
Overall, our contribution can be summarized as follows:

- We propose a novel region recommendation problem which aims to recommend regions to users who travel to new areas (cities or countries). According to users' movement behaviors, we introduce a scoring scheme for regions, which considers the influences between POIs.
- We develop a sweeping algorithm-based method to solve the problem of searching optimal region efficiently. Subsequently, a constant-bounded approximate algorithm is proposed to further improve the efficiency while still achieving similar accuracy of the exact solution.
- We demonstrate the effectiveness of our region recommendation method on two real-life datasets. Experiments also show the significant improvement from the proposed approximate algorithm in terms of efficiency.

2. RELATED WORK

Out-of-town recommendation is a special sub-problem of POI recommendation, which aims to recommend locations to users who travel out of their hometowns. This problem suffers from the cold-start issue a lot due to the insufficiency of out-of-town check-ins [6]. Therefore, previous out-of-town recommendation models focus on how to overcome the sparsity problem, usually by exploiting the content of POIs [22, 20] and experts' knowledge in destination cities [1, 20]. However, all the previous works on out-of-town recommendation focus on recommending individual POIs to users. As discussed in Section 1, it is more useful to recommend a region with several POIs than just singular POI when users travel out of their hometowns. Therefore, in this paper, we propose a novel region recommendation problem that aims to find regions interesting to a user.

Previous work on POI search and recommendations finds that an interesting POI whose nearby POIs are also interesting to a user is preferable when compared to a POI without interesting nearby POIs. In other words, nearby POIs reinforce each other to make them more attractive to users than separate ones. For example, when searching locations, Cao et al. [2] demonstrate that a relevant location is more interesting to users if it is surrounded by other relevant locations, and propose algorithms that propagate prestige scores among nearby spatial objects. In POI recommendation, Liu et al. [16] observe that nearby POIs tend to share more common visitors, and hence their proposed factorization model tries to learn similar preference for nearby POIs. Similarly,



(a) Fraction of users having a $d \times d$ region with at least n POIs.

(b) CDF of region sizes made by check-ins in one and two days, respectively.

Figure 1: Observations on out-of-town check-ins.

Li et al. [13] exploit the neighborhood effect when recommending POIs to users. The signal of this effect is even stronger when users travel as they usually have limited time to visit POIs in a small region, making it beneficial for region recommendation. Inspired by these studies, in this paper, we propose the problem of recommending regions to users that takes the POI influences into consideration. Different from previous work, our model considers the neighborhood effect collectively on a group of POIs, rather than individual ones. Studies [9, 24] aim to recommend a tour route to users, by considering time and distances, where the orders of POIs are important. In our model, we recommend an out-of-town region for users to explore, where we emphasize locality of POIs rather than the orders of POIs. The work [23] is the most relevant to our work, but different from us, their recommended regions (neighborhoods) are from a fixed set, and users' content information (tweets) is needed for recommendation.

When making region recommendation, one critical issue is how to locate the user's optimal region, which has the highest satisfaction score. This problem is related to the maximum object enclosing rectangles (MaxOER) problem that aims to find the rectangle that covers most number of points in 2-dimension space. To solve this problem, two line sweeping algorithms are proposed by Imai et al. [10] and Nandy et al. [18], respectively, both of which have time complexity $\mathcal{O}(N \log N)$, where N is the total number of points. Solutions for MaxOER in approximate scheme [19, 12, 5] are also proposed. However, all the algorithms can only solve the original MaxOER problem, and cannot be directly applied in our region search problem where the interactions between points (*i.e.*, POIs) exist. We modify and improve the MaxOER solution to derive an algorithm for our problem, which will be described in Section 4.2. Moreover, to further enhance the efficiency, we also propose an approximate algorithm with guaranteed bounds.

3. OBSERVATIONS

We present some observations from real-world Foursquare dataset [8] that motivate the region recommendation problem. We first infer home locations of all users, and keep users and locations located in the US. For each user, the region within $r = 100km$ around her home location is considered as her hometown. Details of the preprocessing step are described in Section 6.1.

First, we investigate the users' distribution of out-of-town check-ins as follows. Given a length of d kilometers and a number n, we compute the fraction of users who have one

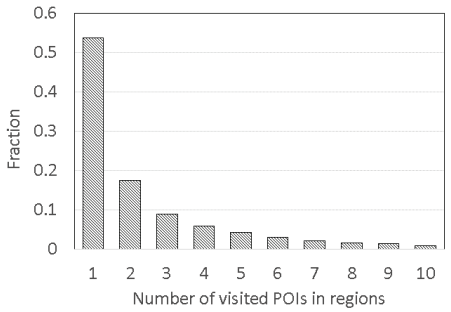


Figure 2: Fraction of regions with different number of visited POIs.

region of size $d\times d$ with at least n visited POIs, and then plot Fig. 1a. The figure shows that a large proportion of users have multiple check-ins in a small region. For example, more than 60% of users have visited at least 2 POIs in a $1km\times1km$ region. When the size of region grows, the fraction of those users obviously increases, which is at nearly 80% when $d = 5km$. For larger value of n, the user ratio is decreased but still at the high number. For example, there are still 25% of users who have at least 4 visited POIs located in an $1km\times1km$ region. This implies that users tend to go to locations within a small region when they go outside their hometowns. To investigate the reason, we compute the sizes of regions containing check-ins of same users in t consecutive days and plot their cumulative distribution functions (CDF) for $t = 1$ and $t = 2$ in Fig. 1b. The figure shows that 10% of one-day check-ins occur in a region smaller than $1km\times1km$, and this number is about 45% when $d = 10km$. When $t = 2$, the CDF line is lower, which means that users tend to visit larger regions when the duration is longer. This confirms our belief that users often visit small regions due to limited time. Overall, these observations motivate us to consider the problem of recommending regions to users.

Definition 1. Given a user, a set of POIs in spatial space and a query size of $d\times d$, the **Region Recommendation (RegRS) Problem** aims to find a $d\times d$ region that the user is most likely to visit. Here, we consider a region is visited by a user when she visits at least one POI in that region.

Instead of recommending individual POIs, the region recommendation problem aims to recommend a group of POIs that are located within a region.

In previous work [16, 4], regions are predetermined by using some spatial partitioning algorithms. Although our recommendation model can be easily applied for such set of fixed regions, we do not divide the space into fixed regions but instead consider any rectangle spatial area as a region. This is more general and flexible as we are able to find meaningful and crucial regions that can lie inside, contain or intersect those fixed regions. Moreover, we assume that a region has the square shape $d\times d$ for simplicity. One can define a rectangle of size $a\times b$ as a region. However, it does not lose the generality because we can easily scale the space to make the region become a square. Also, the size of the region is given by the user, and different users can issue different query sizes depending on their distance and time constraints.

4. REGION RECOMMENDATION

4.1 Proposed model

In this section, we present our model for recommending regions to users. Consider a region R containing multiple POIs: $R = \{p_i\}$. Let $S_{i_1\dots i_k}$ be the satisfaction of user u when she visits POIs $p_{i_1}\dots p_{i_k}$ in R. Let $P_u(p_{i_1},\dots,p_{i_k}|k)$ be the probability that user u visits POIs $p_{i_1},\dots,p_{i_k}$ in R given she will visit k distinct POIs. For convenience, we drop the notation u unless necessary, *i.e.*, we simply use $P(p_{i_1},\dots,p_{i_k}|k)$. Then, the total expected satisfaction of u, S_{total}, when she visits the region R can be written as

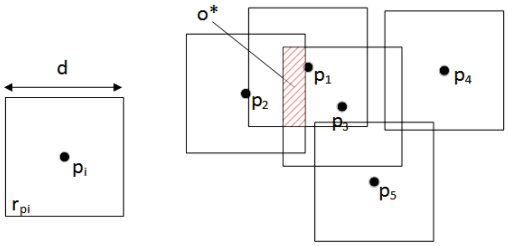
$$\begin{aligned}S_{total} &= \left(\sum_{p_{i_1}\in R} S_{i_1}P(p_{i_1}|k=1)\right)P(k=1)\\ &+\left(\sum_{p_{i_1},p_{i_2}\in R} S_{i_1i_2}P(p_{i_1},p_{i_2}|k=2)\right)P(k=2)+\dots\\ &+\left(\sum_{p_{i_1},\dots,p_{i_{|R|}}\in R} S_{i_1\dots i_{|R|}}P(p_{i_1},\dots,p_{i_{|R|}}|k=|R|)\right)P(k=|R|).\end{aligned} \tag{1}$$

Equation (1) covers all the cases where user u may visit from 1 to $|R|$ POIs in R. However, we observe that, although a user may have a region with many visited POIs (see Section 3), most of her visited regions have only 1 or 2 POIs. To demonstrate this, given a user, we generate randomly some regions with the size of $4km\times4km$ such that each region covers at least one of her visited POIs. Then, we compute the proportion of regions with different number of visited POIs, and plot the distribution in Fig. 2. From the figure, we can see that most of the regions have less than 3 POIs visited by a user. Therefore, it is sufficient to consider only the first- and second-order terms, and ignore other higher order terms of S_{total}:

$$\begin{aligned}S_{total} \approx &\left(\sum_{p_{i_1}\in R} S_{i_1}P(p_{i_1}|k=1)\right)P(k=1)\\ &+\left(\sum_{p_{i_1},p_{i_2}\in R} S_{i_1i_2}P(p_{i_1},p_{i_2}|k=2)\right)P(k=2).\end{aligned} \tag{2}$$

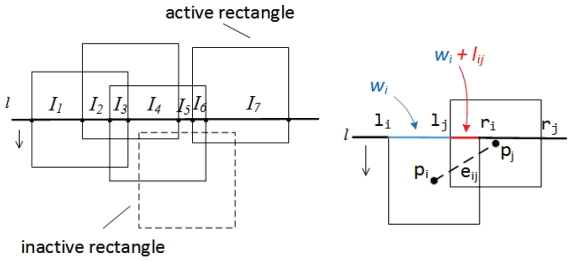
Next, we will define two probabilities: $P(p_{i_1}|k=1)$ and $P(p_{i_1},p_{i_2}|k=2)$. The probability $P(p_{i_1}|k=1)$ represents the probability that user u visits p_{i_1}, *i.e.*, $P(p_{i_1}|k=1) = P(p_{i_1}|u)$, which can be computed by using any recommendation model such as memory-based collaborative filtering [21] or matrix factorization [16, 14, 13]. For the probability $P(p_{i_1},p_{i_2}|k=2)$, one simple way is to define this probability as $P(p_{i_1},p_{i_2}|k=2) = P(p_{i_1}|u)\cdot P(p_{i_2}|u)$, assuming that the user's decisions on visiting different POIs are independent. However, as discussed above, the user's interest on a POI could be affected by other nearby POIs, which means that the influences between POIs cannot be ignored. Hence, in our model, we define the probability as $P(p_{i_1},p_{i_2}|k=2) = \frac{1}{2}\cdot P(p_{i_2}|p_{i_1})\cdot P(p_{i_1}|u)+\frac{1}{2}\cdot P(p_{i_1}|p_{i_2})\cdot P(p_{i_2}|u)$, where $P(p_{i_1}|p_{i_2})$ is the transition probability from p_{i_2} to p_{i_1}, which can be computed from historical check-in data as follows:

$$P(p_{i_1}|p_{i_2}) = \frac{\#\text{users who visited both } p_{i_1} \text{ and } p_{i_2}}{\#\text{users who visited } p_{i_2}}. \tag{3}$$



(a) Constructed rectangle for a point p_i (b) GI problem (o^* can be any point in the crossed area)

Figure 3: Reduction to GI problem.



(a) Solving the RegRS problem (b) Updating weights of intervals

Figure 4: The illustration of the sweeping algorithm.

Probabilities $P(p_{i_1}|k=1)$ and $P(p_{i_1}, p_{i_2}|k=2)$ are defined simply based on users' check-in history. However, more sophisticated definitions of those probabilities can be given by integrating additional information. For example, we can boost up (or diminish) the probability of visiting two POIs if we know they are complementary (or substitutable) for each other [17]. Moreover, other information, *e.g.*, categorical [25] or textual information [7], can be utilized to compute the visiting probability. In this paper, we keep the simple definitions for those probabilities and leave any enhancement for future work.

Next, we define the satisfaction $S_{i_1 i_2,\ldots,i_k}$. In general, the more POIs a user visited, the more satisfaction the user obtains. Hence, we set $S_{i_1} = 1$, $S_{i_1 i_2} = 2$, ..., and $S_{i_1 i_2,\ldots,i_k} = k$ in this paper. We note that other satisfaction function can be defined in terms of different application needs. As a result, the expected satisfaction S_{total} can be rewritten as

$$S_{total} \approx \left(\beta \cdot \sum_{p_i \in R} P(p_i|u) + \sum_{\langle p_i, p_j\rangle \in O_R} P(p_i|p_j) \cdot P(p_j|u)\right) P(k=2), \tag{4}$$

where O_R is all the ordered pairs $\langle p_i, p_j\rangle$ in R, and $\beta = \frac{P(k=1)}{P(k=2)}$ is a user-dependent parameter. However, it is difficult to set β for each user, and hence, in our model, we set β to 1 for all users, since it gives our model the best performance.

Finally, since the goal of RegRS problem is to find the region that gives the maximum satisfaction to a user, it can be formulated as the following objective function:

$$R^* = \arg\max_{R \in \mathcal{R}} \sum_{p_i \in R} P(p_i|u) + \sum_{\langle p_i, p_j\rangle \in O_R} P(p_i|p_j) \cdot P(p_j|u). \tag{5}$$

Here, $\mathcal{R}$ is the set of all possible regions, and the term $P(k=2)$ is dropped as it is a constant given a user.

4.2 Searching maximum region

The goal of RegRS problem is to find the region R^* according to Eq. (5). In other words, we aim to locate the position of the maximum region, which satisfies the objective function Eq. (5). However, there are *infinite* possible ways to place such region in the space. Therefore it is impossible to check every region to find the solution.

In this section, we present our algorithm, namely RegRS algorithm, to efficiently find the maximum region. First, we define some notations. Besides N points $\mathcal{P} = \{p_i \,|\, 1 \le i \le N\}$, each of which represents a POI in $\mathbb{R}^2$, we define a set of M edges $\mathcal{E} = \{e_{ij}, 1 \le i, j \le N\}$, where each of edges connects two points $\langle p_i, p_j\rangle$ in P. Each point p_i and edge e_{ij} have positive weights $w_i = P(p_i|u)$ and $l_{ij} = q_{ij} \cdot w_i + q_{ji} \cdot w_j$, respectively, where $q_{ij} = P(p_j|p_i)$. Our problem is to find the maximum rectangle R^* of given size $d \times d$ that maximizes the objective function Eq. (5), which can be rewritten as:

$$s(R) = \sum_{p_i \in R} w_i + \sum_{\substack{p_i, p_j \in R, \\ e_{ij} \in \mathcal{E}}} l_{ij}. \tag{6}$$

The main idea of the solution is to reduce the original problem to the geometric intersection problem, and then a space *sweeping* algorithm is performed to solve the new problem. Our method is based on the technique to solve the maximum object enclosing rectangles problem [10, 18].

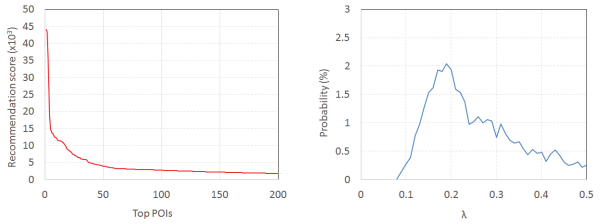
4.2.1 Geometric intersection problem

The first step of the algorithm is to reduce the RegRS problem to the geometric intersection (GI) problem [10, 18]. The reduction is performed as follows: for each point $p_i \in P$, we draw an $d \times d$ rectangle centered at p_i, and denoted by r_{p_i}. The N constructed rectangles $\{r_{p_i}\}$ intersect each other, and thus create multiple disjoint regions, each region is the intersection of a unique set of rectangles. For each disjoint region D, let D_R the set of points p_i, whose rectangles r_{p_i} form the region D. For example, in Fig. 3, the crossed area is a disjoint region created by rectangles r_{p_1}, r_{p_2} and r_{p_3}, and hence $D_R = \{p_1, p_2, p_3\}$. Given any point o in a disjoint region D, the $d \times d$ rectangle r_o centered at o will cover all the points in D_R. This is because if a point o is covered by a rectangle r_{p_i}, then r_o also covers p_i. Let the weight of D_R be calculated by Eq. (6), where $R = D_R$. As a result, the goal of the GI problem is to find the disjoint region D_R^* with the maximum weight. Then, any point o^* in the D_R^* can be returned as the center of the maximum region R^* for the RegRS problem.

The reduction gives us a new way to solve the RegRS problem. Finding maximum disjoint region D_R^* can be done by extending sweeping algorithm [10, 18].

4.2.2 Sweeping algorithm

The main idea of the algorithm is to use a horizontal line l to scan the space from top to bottom. The process begins when the sweeping line l is at the upmost line and starts sweeping down. When l encounters and goes inside a rectangle r_{p_i}, r_{p_i} cuts l at two intersection points, $\mathtt{l}_i$ and $\mathtt{r}_i$, by its two vertical edges, and r_{p_i} is said to be active. When l exits r_{p_i}, two corresponding intersection points are removed, and r_{p_i} becomes inactive. At any moment, if there are currently k active rectangles, *i.e.*, l is inside k rectangles, there are $2k$



(a) Recommendation scores of top 200 POIs for a random user (b) Distribution of values of λ

Figure 5: Some observations

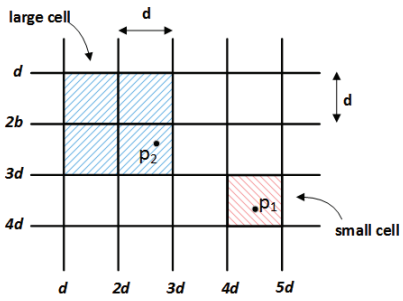


Figure 6: Dividing space by grid G_d

intersection points on l, and those points create $2k-1$ disjoint intervals on l. Figure 4a illustrates one moment when the sweeping line lies across 4 rectangles, and 7 intervals, I_1 to I_7, are created accordingly.

An interval I_k is said to be covered by an active rectangle r_{p_i} if it is inside the window $[\mathtt{l}_i, \mathtt{r}_i]$ created by r_{p_i}. Then we can define weight $s(\mathcal{T}_k)$ for an interval I_k, where $s(\mathcal{T}_k)$ is computed by Eq. (6), and $\mathcal{T}_k$ is the set of points whose rectangles cover I_k. When the sweeping line l encounters a rectangle r_{p_i} (Fig. 4b), not only new intervals are created, but also weights of all intervals covered by r_{p_i} are increased. The update includes two steps corresponding to two terms of Eq. (6). First, all the intervals covered by r_{p_i} (*i.e.*, the ones inside the window $[\mathtt{l}_i, \mathtt{r}_i]$) are added up by the weight w_i of p_i. Second, if p_i links to any point p_j (*i.e.*, $e_{ij} \in \mathcal{E}$) and two rectangles r_{p_i} and r_{p_j} overlap each other, all the intervals in the overlapping window, which is $[\mathtt{l}_j, \mathtt{r}_i]$ in Fig. 4b, are added up by l_{ij}. Similarly, when l exits r_{p_i}, some intervals disappear, and weights of all intervals covered by r_p are decreased.

The sweeping process finishes when l has crossed all the rectangles. If I_k is the interval with highest weight in the whole process, then then we can return I_k as the result of the GI problem as it is inside the maximum disjoint region. For example, in Fig. 4a, the interval I_3 is the one with the largest weight and is returned as the result.

Complexity. In order to efficiently maintain intervals (*i.e.*, creating, updating and deleting intervals) during the scanning process, a special tree-based data structure, Interval tree [18], is used in our implementation. It takes $\mathcal{O}(N \log N)$ to construct the Interval tree, and $\mathcal{O}(\log N)$ to handle intervals created by a point or an edge. Note that only edges that can be covered by a $d \times d$ rectangle are processed. Overall, the time complexity of RegRS algorithm is $\mathcal{O}((N+\overline{M}) \log N)$, where $\overline{M}$ is the number of processed edges.

5. APPROXIMATE SOLUTION

Based on the time complexity analyzed above, runtime performance of RegRS degrades significantly when the size of query rectangle increases (which is also demonstrated in experiments in Section 6.5). This is because a larger query rectangle can cover more edges, and hence, more edges will be processed, which will degrade the efficiency of the searching algorithm. In this section, we present an approximate algorithm to answer the RegRS problem efficiently. Firstly, we introduce the ϵ-approximate RegRS problem.

Definition 2. Given an input $I = \langle \mathcal{P}, \mathcal{E}, w, l, d, \epsilon \rangle$, where $\mathcal{P}$, $\mathcal{E}$, w, l, d are the input of the RegRS problem and ϵ is a real number such that $0<\epsilon<1$, the **ϵ-approximate RegRS problem** aims to return a $d \times d$ rectangle R_{appx} whose weight $s(R_{appx})$ is at least $\epsilon s(R^*)$, *i.e.*, $s(R_{appx}) \geq \epsilon s(R^*)$, where R^* is the maximum region.

5.1 Main idea

The approximate algorithm is inspired by the following observation. Figure 5a shows the top 200 recommendation scores, $p(p_i|u)$, for one randomly picked user. It is clear that the distribution of recommendation scores follows a long tail distribution, *i.e.*, most of them have very small values, which is a common phenomenon in recommendation systems. Since small-weighted POIs contribute little to the total weight of the region, intuitively if we ignore those points and keep only large-weighted ones, we can save searching time significantly without sacrificing much the accuracy. Moreover, we also need to consider edges as they also contribute to weights of regions. Particularly, since the edge e_{ij}'s weight is computed by $l_{ij} = q_{ij} \cdot w_i + q_{ji} \cdot w_j$, l_{ij} is likely to be large if w_i or w_j is large. Therefore, it is necessary to preserve all the edges of large-weighted points. As a result, neighbors of large-weighted points should be kept as well.

5.2 Solving approximate RegRS problem

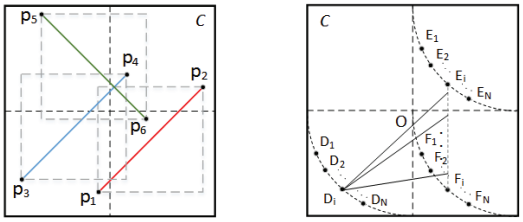
Our approach is, instead of handling the whole space, we divide the space into $d \times d$ cells and use cells as spatial units to select points for approximate problem. Let us first define some notations used in our algorithms.

Definition 3. Grid G_d is the set of vertical and horizontal lines that are defined as follows:

$$G_d = \{(x, y) \in \mathbb{R}^2 \mid x = k_1 \cdot d \wedge y = k_2 \cdot d, k_1, k_2 \in \mathbb{Z}\}$$

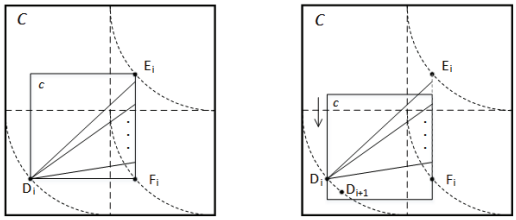
Apparently, G_d partitions the space into $d \times d$ cells, which are called *small cells*. Each small cell is uniquely identified by the pair of real numbers (k_1, k_2), called the identity of the small cell, which are coefficients corresponding the left vertical line and top horizontal line of the cell, respectively. Subsequently, we define a *large cell* to be a $2d \times 2d$ cell constructed by 4 adjacent small cells as in Fig. 6. Large cells also have unique identity which is the identity of its top-left small cell. For any point $p = (x, y)$, it is easy to find the small cell with the identity $\left(\lfloor \frac{x}{d} \rfloor, \lfloor \frac{y}{d} \rfloor\right)$, and hence 4 large cells, that it falls in. Hereinafter, we will refer term cell to large cell, if there is no explanation.

It can be easily seen that any $d \times d$ rectangle, including the maximum rectangle R^*, is covered by a cell. Hence, our approach is to process each cell one by one. For a cell C, we *remove points from C, together with their edges, such*



(a) Case 1 (b) Case 2

Figure 7: Special point and edge arrangements for cell C.



(a) (b)

Figure 8: Two possible positions of a rectangle c in case 2.

that the maximum $d\times d$ rectangle returned by the RegRS algorithm on remaining points (and edges) is the answer of the ϵ-approximate RegRS problem on C. Particularly, let c_{max} be the maximum $d\times d$ rectangle in C. Then, let S_C and S'_C be the weights of cell C before and after removing points respectively; and s_{max} and s'_{max} are defined similarly but for c_{max}. Then we want that

$$\frac{s'_{max}}{s_{max}} \geq \epsilon \tag{7}$$

Set $\Delta_s = s_{max} - s'_{max}$ and $\Delta_S = S_C - S'_C$. Since removed points (and edges) can be outside c_{max}, it is obvious that $\Delta_s \leq \Delta_S$. We have:

$$\frac{s'_{max}}{s_{max}} = \frac{s_{max} - \Delta_s}{s_{max}} = 1 - \frac{\Delta_s}{s_{max}} \geq 1 - \frac{\Delta_S}{s_{max}} \tag{8}$$

Now, we assume that $\frac{s_{max}}{S_C} \geq \lambda$ where λ is a constant in $(0, 1]$, then

$$\frac{s'_{max}}{s_{max}} \geq 1 - \frac{\Delta_S}{\lambda S_C} = 1 - \frac{S_C - S'_C}{\lambda S_C} = \frac{1}{\lambda}(\lambda - 1 + \frac{S'_C}{S_C}) \tag{9}$$

From Eq. (9), the sufficient condition for (7) is:

$$\frac{1}{\lambda}(\lambda - 1 + \frac{S'_C}{S_C}) \geq \epsilon \Rightarrow \frac{S'_C}{S_C} \geq 1 - (1-\epsilon)\lambda \tag{10}$$

The condition Eq. (10) means that, when removing points (and edges) from C, as long as S'_C is still not less than $\big(1 - (1-\epsilon)\lambda\big)S_C$, the condition in Eq. (7) is always guaranteed.

5.2.1 Finding λ

To complete the condition (10), we need to find the value of λ, which is the lower bound of the ratio $\frac{s_{max}}{S_C}$. It is easy to prove that if cell C contains only points without edges, the lower bound is $\lambda = 1/4$. Unfortunately, by the following theorem, a constant lower bound λ does not exist in the presence of edges.

THEOREM 1. *When C contains both positively weighted points and edges, there does not exist a lower bound of the ratio λ. In other words, the ratio can be arbitrarily small.*

PROOF. Below, we use the square-shaped 2×2 rectangle C for the ease of proof, but the statement is still true for the rectangles of any shape.

To prove Theorem 1, we consider two cases: 1) edge weights are *not bounded*, and 2) edge weights are *bounded by their two points*. In both cases, we show special examples when the lower bound of the ratio λ can be arbitrarily small.

Case 1: weights of edges are not bounded by their two ending points. In this case, we set up a special instance of cell C as follows: *every edge is a diagonal of one* 1×1 *rectangle* c (Fig. 7a). As a result, a rectangle c can cover two edges at most, which are its two diagonals. Note that, we can place *infinite* number of such edges in C. Assume that all edges have equal weights of 1, and c_{max} contains two edges, then

$$\frac{s_{max}}{S_C} = \frac{s^{points}_{max} + 2}{S^{points}_C + E}$$

where s^{points}_{max} and S^{points}_C are total weight of points in c_{max} and C, respectively, and E is total number of edges in C. Now, if we set weights of points very close to zero, then edge weights will dominate point weights in rectangles, *i.e.*, $r^{points}_{max} \ll 2$ and $S^{points}_C \ll E$. This leads to $\frac{s_{max}}{S_C} \approx \frac{2}{E}$. This means that, if we add more edges into C, the ratio decreases accordingly. When $E \to \infty$, then $\frac{s_{max}}{S_C} \to 0$. In other words, λ can be arbitrarily small.

Case 2: weights of edges are bounded by their two ending points. The above proof is only valid when edge weights are arbitrarily large. However, in our region recommendation problem, we know that $l_{ij} = q_{ij} \cdot w_i + q_{ji} \cdot w_j \leq w_i + w_j$, because $q_{ij} = P(p_j|p_i) \leq 1, \forall i, j$. In other words, *the edge weight cannot be larger than sum of weights of its two points.* For the completeness of the proof, we need to consider this case as well.

The setup of the special instance is little more complicated. First, we place N points in the lower-left 1×1 rectangle of C such that those points form a $1/4$ arc of the circle with same center O of C. We call set of those points $\mathbf{D} = \{D_1, D_2, ..., D_N\}$. For each point $D_i \in \mathbf{D}$, we create two more points E_i and F_i in the same positions but in the upper-right and lower-right 1×1 corner rectangles, respectively. Those points form two other point sets, namely $\mathbf{E}$ and $\mathbf{F}$. Obviously, $\mathbf{E}$ and $\mathbf{F}$ also create two $1/4$ arcs of the circles as in Fig. 7b. Then, for each triple $\langle D_i, E_i, F_i\rangle$, we create $N-1$ edges from D_i to some points in the line $\overline{E_iF_i}$ as in Fig. 7b. It is easy to see that a rectangle c covers an edge only when the point D_i of the edge is in the left edge of c. Finally, we set the weight of each D_i be one and other points be (very close to) zero. All the edge weights are one. Obviously the edge weights satisfy the above constraint.

There are 3 possibilities of the position of a rectangle c: 1) c's lower-left corner is a $D_i \in \mathbf{D}$ (Fig. 8a); and hence the upper-right corner is the corresponding point E_i. In this position, c's weight is N (1 from D_i and $N-1$ from $N-1$ edges); 2) the second position is obtained from the first case by shifting c vertically down (Fig. 8b). When being shifted down, c contains some more points D_k but loses some edges. Since we are free to choose positions of

Algorithm 1: Approximate algorithm

Input: $\langle \mathcal{P}, \mathcal{E}, w, l, d, \epsilon \rangle$
Output: Approximate optimal rectangle R_{appx}

1 Divide space by G_d;
2 Compute the upper bound weight S_C^U of each cell;
3 $S'_C \leftarrow 0, \forall C$ //initial new weights of cells
4 $\mathcal{P}' = \{\}$;
5 **foreach** *cell C from largest to smallest* S^U **do**
6 **if** $S_C^U < \max\{S'_C\} \times \lambda$ **then**
  break;
7 **foreach** *point* $p_i \in C$ *from largest to smallest* w_i **do**
8 **if** $p_i \in \mathcal{P}'$ **then**
  continue;
9 $\mathcal{P}' \leftarrow \mathcal{P}' \cup \{p_i\} \cup \text{neigh}(p_i)$;
10 Update S'_C;
11 **if** $S'_C/S_C^U \geq 1 - (1-\epsilon)\lambda$ **then**
  break;
12 Run sweeping algorithm on $\langle \mathcal{P}', \mathcal{E}', w, l, d \rangle$ and return R_{appx};

edges, edges are created so that whenever c receives a point D_k when shifting down, it loses one edge. Therefore, the weight of c is still N; 3) other positions. For those positions, c covers no edges and at most N points D_i, hence it has a weight of N at most.

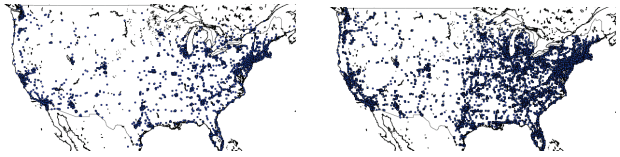
Overall, the weight of c_{max} is $s_{max} = N$. Meanwhile, the weight of C is $S_C = N+N(N-1) = N^2$, since there are N points D_i and $N-1$ edges for each D_i. Therefore, $\frac{s_{max}}{S_C} = \frac{N}{N^2} = \frac{1}{N}$. This means that, if we add more points following this arrangement, the ratio decreases, and when $N \to \infty$, then $\frac{s_{max}}{S_C} \to 0$.

Overall, in any case, we do not have a constant lower bound λ for the ratio $\frac{s_{max}}{S_C}$. □

Theorem 1 tells us that we do not have a fixed value for the lower bound λ. However, in practice, when points and edges are randomly located in C, we find that the ratio cannot be too small. Particularly, from the Foursquare dataset used in our experiment (see Section 6), we pick randomly some users and randomly some $2d \times 2d$ regions for each user ($d = 1km, 2km, 5km \ldots$), then find λ for each test case and plot the distribution of λ in Fig. 5b. From the figure, we can see that the minimum value of λ is around 0.08. Note that, we also obtain the same observation for another dataset used in Section 6. This means that, in practice, we can set λ to any number below 0.08. In our implementation, λ is set to 0.05.

5.2.2 Computing upper bound of S_C

For each cell C, in order to keep track the ratio S'_C/S_C, we need to compute the weight S_C, which is very time-consuming. We know that the weight S_C is computed as $S_C = s(C) = \sum_{p_i \in C} w_i + \sum_{p_i \in C} \sum_{p_j \in C} q_{ij} \cdot w_i = \sum_{p_i \in C} (1 + \sum_{p_j \in C} q_{ij}) \cdot w_i$. Since q_{ij} is the transition between two POIs, it is independent of the query user and can be computed offline. Therefore, to approximate S_C, for each point p_i, we maintain a list of tuples $\langle d_j, q_{ij}^{ub} \rangle$. Here, d_j is the distance from p_i to a neighbor p_j (*i.e.*, point that has an edge to p_i), and q_{ij}^{ub} is computed as $q_{ij}^{ub} = \sum_{p_k \in \text{neigh}(p_i, d_j)} q_{ik}$, where $\text{neigh}(p_i, d_j)$ is the set of p_i's neighbors within the distance d_j from p_i. The list is sorted by d_j. Then, given the size $d \times d$ of C, for each p_i, we use binary search to find $\langle d_j, q_{ij}^{ub} \rangle$ where $d \leq d_j$, and finally the upper bound of S_C is computed as $S_C^U = \sum_{p_i \in C} (1 + q_{ij}^{ub}) \cdot w_i$. This step will take $\mathcal{O}(\log N_i^{nb})$ for each p_i, where $N_i^{nb} = |\text{neigh}(p_i)|$. Then, S_C^U can be used to replace S_C in the condition Eq. (10).



(a) Foursquare (b) Twitter

Figure 9: POI distribution of two datasets.

5.3 Approximate algorithm

Now, we are ready to describe our approximate algorithm (Algorithm 1). We firstly divide space by G_d (line 1), and then check each cell C in decreasing order of upper bounds S_C^U (line 5). In each cell C, we iterate points in decreasing order of w_i, and add it, with its neighbors, into the selection set $\mathcal{P}'$ (line 9) until current weight S'_C of C satisfies condition Eq. (10). The selection step terminates when unchecked cells cannot give the maximum rectangle (line 6). Finally, the sweeping algorithm is performed on the new set $\mathcal{P}'$ and R_{appx} is returned (line 12).

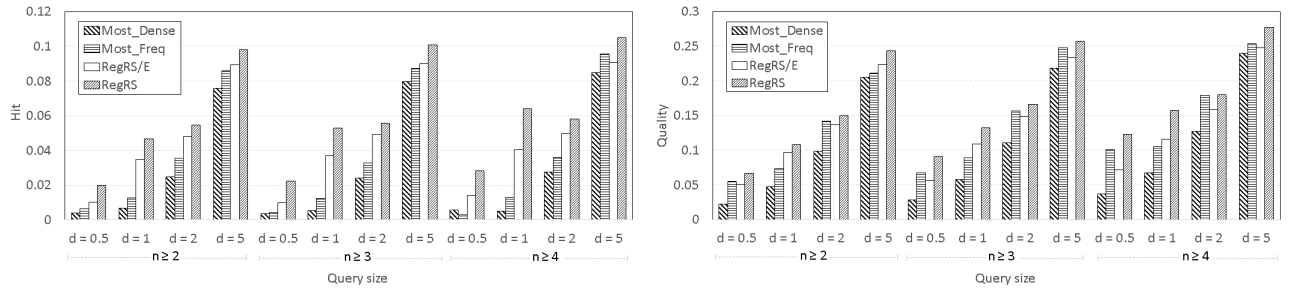
Time complexity. The most costly parts of the algorithm 1 are computing upper bounds of all cells and running sweeping algorithm on $\mathcal{P}'$. Overall, its complexity is $\mathcal{O}(N \log \overline{N}^{nb} + (N' + M') \log N')$, where $\overline{N}^{nb}$ is the average number of neighbors of all points, N' and M' are the number of points in $\mathcal{P}'$ and their edges, respectively. Considering $\overline{N}^{nb} \ll N$, $N' \ll N$ and $M' \ll \overline{M}$, we can see that the approximate algorithm has a better time complexity than the exact RegRS algorithm.

6. EXPERIMENTS

6.1 Experimental settings

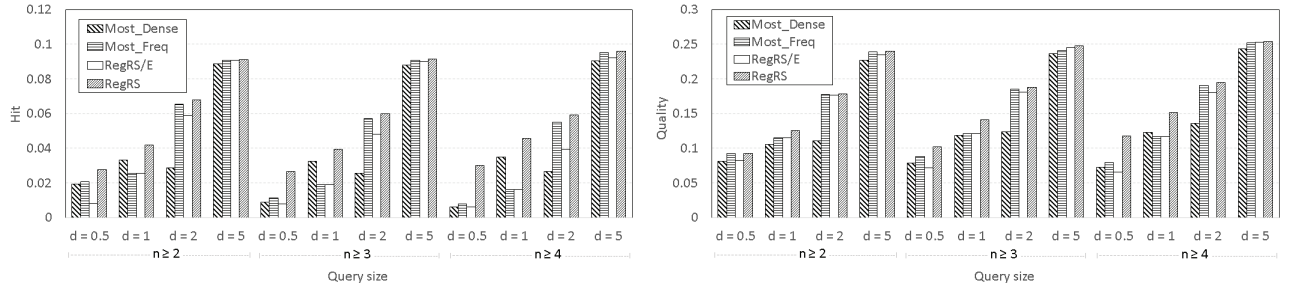
In our experiment, we use two public datasets, **Foursquare** [8] and **Twitter** [3]. For both datasets, we first infer users' home locations by following the strategy in [3]. Then, for each dataset, we extract the users and check-ins located in the US, and keep users with at least 5 visited POIs and POIs with at least 5 users. After preprocessing, the Foursquare dataset contains 8,368 users and 19,241 POIs with 332,952 check-ins; and the Twitter dataset contains 31,378 users, 21,951 POIs and 456,578 check-ins. The POI distributions of two datasets is shown in Fig. 9.

Following [20], we consider the region that is $100km$ around a user's home location is her hometown. Then, we create the training and testing data as follows. Since not all users have out-of-town check-ins, a user is in testing data if she has at least 5 visited POIs in her hometown, and at least two of her out-of-town POIs are covered by a $d \times d$ region ($d = 0.5km$). For test users, their hometown POIs are put into training data, while their out-of-town POIs are added into testing data. For remaining users, all their visited POIs (both hometown and out-of-town ones) are included in training data. The intuition behind this setup is that we want to test our model in the extreme case when target users do not have any out-of-town check-ins, *i.e.*, cold-start problem.



(a) Hit (b) Quality

Figure 10: Performance of region recommendation methods on Foursquare dataset.



(a) Hit (b) Quality

Figure 11: Performance of region recommendation methods on Twitter dataset.

6.2 Evaluation metrics

The goal of the region recommendation problem is to find the region that a given user is most likely to visit. Since from the data there is no indication of the region that the user likes the most, we consider the one that covers the most number of POIs visited by the user is her groundtruth region. Next, we propose two metrics to evaluate methods for recommending region as follows.

Hit. For each test user, if a groundtruth region is recommended, the method gets a score of 1, otherwise 0. The final score is the average score of all test users[1].

$$Hit = \frac{\#\text{test cases returning groundtruth regions}}{\#\text{all test cases}}$$

Quality. In the previous metric, for a test case, the method gets score only when the recommended region covers all POIs in a groundtruth region, which is quite strict. Hence, we introduce another metric to better evaluate the performance of recommendation methods by considering partial coverage of set of visited POIs. In particular, for each test case t, the quality score of a method is computed as follows,

$$QScore(t) = \frac{\#\text{POIs in the returned region}}{\#\text{POIs in a groundtruth region}}$$

And the final *Quality* is computed by averaging the quality score of all test cases, *i.e.*

$$Quality = \frac{1}{|T|}\sum_{t}^{T} QScore(t).$$

[1]Since only one region is returned by methods, Hit metric is equivalent to Precision metric. Recall metric cannot be used because there are *infinite* groundtruth regions.

6.3 Baseline methods

Since there are no previous methods for region recommendation, we introduce two baselines to compare with our proposed method.

- Most_dense. This method always returns the region with most number of POIs. The intuition is that the more POIs a region contains, the higher chance users visit the region.
- Most_Freq. This method returns the region which is most frequently visited by users.
- RegRS/E. This method returns the region with the highest total visiting probability of its POIs. This method is similar to our proposed method without considering the influence between POIs.
- RegRS. This is our proposed method which considers the interaction between POIs.

Both RegRS/E and RegRS require the visiting probabilities of POIs, $P(p_i|u)$, as the input. Those probabilities can be obtained by using any POI recommendation model. We adopt the Non-negative Matrix Factorization method in [11] which performs well on POI recommendation [15]. Let rs_i^u be the recommendation score of a POI p_i w.r.t. a user u achieved by NMF, and then the vising probability of u to p_i is $P(p_i|u) = \frac{rs_i^u}{\sum_j rs_j^u}$.

6.4 Recommendation results

To test the performance of different region recommendation methods, we use query regions with different sizes $d \times d$, where $d = 0.5km, 1km, 2km$ and $5km$. Note that groundtruth regions of users have at least 2 POIs ($n \geq 2$). We also consider cases when users have groundtruth regions with at least 3 and 4 POIs ($n \geq 3$ and $n \geq 4$), respectively.

6.4.1 Performance on Foursquare dataset

Figure 10 shows the recommendation results of the four methods on Foursquare dataset. We make the following observations: Firstly, RegRS always outperforms other two baseline methods in all cases, especially when the query size is small. For example, when $d = 0.5km$, RegRS outperforms Most_dense by 400% and 300% in terms of Hit and Quality, respectively. Moreover, the Most_dense method is the worst because it recommends regions to users only based on the distribution of POIs but ignores the user's personalized preference on POIs. On the other hand, RegRS always achieves better results than RegRS/E (for example, by 93% and 30.7% in terms of Hit and Quality when $d = 0.5km$). This demonstrates the usefulness of considering the influence between POIs in region recommendation. The Most_Freq method has promising performance in terms of Quality, and this indicates that POIs that are frequently visited by users tend to locate nearby. When the query size increases, the performance of all the methods improves, and the disparity between RegRS and other methods becomes less significant. When $d = 5km$, RegRS achieves better results than Most_dense by only 29% (Hit) and 18.5% (Quality). This is because when the query size is large, there is a higher chance that a region covers many visited POIs, and thus it is easier to find a groundtruth region even by the simple method.

Figure 10 also shows the recommendation accuracy of three methods on users whose the number n of visited POIs in groundtruth regions is at least 3 and 4, respectively. We observe that, compared to the case $n \geq 2$, the performance of all methods increases but RegRS improves better than the other two methods. For example, compare to $n \geq 2$, at $n \geq 4$, the Quality of RegRS rises up nearly twice, while RegRS/E only improves by 41% when $d = 0.5km$. For large regions, the Most_dense baseline achieves the best relative improvement but is still the worst method. Overall, RegRS gives better recommendation for different values of n.

6.4.2 Performance on Twitter dataset

Figures 11 presents the recommendation results on Twitter dataset. Similar to Foursquare dataset, RegRS still performs better than the other methods. For example, when $d = 0.5km$, RegRS beats the Most_dense baseline by 43% and 13% in terms of Hit and Quality, respectively. Most_dense has better results than RegRS/E when query region is small. It is possibly because the distribution of POIs in Twitter dataset is more spread than in Foursquare dataset (Fig. 9), and thus more small region candidates are created, most of which are noisy regions. This makes the performance of RegRS/E significantly worse. However, RegRS does not suffer this problem since it can filter out bad regions in which POIs have little influence on each other.

6.5 Approximate algorithm

In this section, we evaluate the efficiency of RegRS and RegRS with approximation method (denoted by Appr-RegRS). We evaluate Appr-RegRS with two approximation ratio values, $\epsilon = 0.5$ and $\epsilon = 0.7$. Table 1 shows the results of both RegRS and Appr-RegRS in terms of Hit, Quality and running time on both datasets. From the table, it is observed that when the query region enlarges, the running time of RegRS increases. It is because when the query region size is larger, regions cover more edges and hence more edges are processed. Since the time complexity of RegRS is linear to the number of edges, its performance degrades when more edges are considered. On the other hand, Appr-RegRS is always faster than RegRS for all query sizes. For example, when $d = 0.5km$, the Appr-RegRS is nearly twice as fast as RegRS on Foursquare dataset. Moreover, the Hit scores achieved by Appr-RegRS are always at least 90% of those of RegRS. Overall, our proposed approximate method significantly improves the efficiency of RegRS while still achieves similar accuracy.

Table 1: Performance comparison between RegRS and Appr-RegRS in terms of Hit, Quality and running time (in seconds).

	Foursquare			Twitter		
	d = 0.5	d = 1	d = 5	d = 0.5	d = 1	d = 5
RegRS	0.0201	0.0468	0.0980	0.0275	0.0420	0.0912
	0.0666	0.1081	0.2433	0.0922	0.1255	0.2401
	2098 s	3282 s	9847 s	1160 s	1645 s	4216 s
Appr-RegRS	0.0184	0.0421	0.0923	0.0206	0.0373	0.0904
(ϵ=0.5)	0.0649	0.1000	0.2391	0.0905	0.1233	0.2345
	1152 s	2014 s	8539 s	400 s	459 s	571 s
Appr-RegRS	0.0189	0.0438	0.0947	0.0206	0.0373	0.0909
(ϵ=0.7)	0.0650	0.1028	0.2394	0.0901	0.1240	0.2348
	1164 s	2112 s	8891 s	434 s	488 s	607 s

It is interesting that the running time improvement of Appr-RegRS over RegRS on Twitter dataset is much better than on Foursquare dataset. This might be due to the difference between POI distributions of two datasets (Fig. 9), where the POIs in Twitter dataset are distributed more evenly than those in Foursquare dataset. As a result, most of POIs concentrate in few cells in Foursquare dataset, while in Twitter dataset there are many cells with similar number of points. Therefore, more POIs are removed on Twitter dataset when we ignore small-weighted cells, which leads to the better efficiency improvement than on Foursquare dataset.

7. CONCLUSIONS

In this paper, we introduce a novel problem of recommending regions for out-of-town users. We subsequently propose a general framework to solve the region recommendation problem. Instead of considering POIs separately, our framework utilizes interaction between POIs to achieve better recommendation accuracy. On the other hand, to overcome the issue of finding the optimal region, we propose an efficient searching algorithm based on the sweeping-based algorithm. We also develop a constant-bounded approximate algorithm to further improve efficiency. Experiments on two real datasets demonstrate the our model significantly outperforms baseline methods in region recommendation.

8. ACKNOWLEDGMENTS

This work is supported in part by a Tier-1 grant (RG22/15) and a Tier-2 grant (MOE-2016-T2-1-137) awarded by Ministry of Education Singapore, and a grant awarded by Microsoft, NSFC under Grant Nos. 61602132 and 61572158, and Shenzhen Science and Technology Program under Grant Nos. JSGG20150512145714247 and JCYJ20160330163900579.

9. REFERENCES

[1] J. Bao, Y. Zheng, and M. F. Mokbel. Location-based and preference-aware recommendation using sparse geo-social networking data. In *SIGSPATIAL*, pages 199–208. ACM, 2012.

[2] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *VLDB*, 3(1-2):373–384, 2010.

[3] Z. Cheng, J. Caverlee, K. Lee, and D. Z. Sui. Exploring millions of footprints in location sharing services. *ICWSM*, 2011:81–88, 2011.

[4] J. Cranshaw, R. Schwartz, J. I. Hong, and N. Sadeh. The livehoods project: Utilizing social media to understand the dynamics of a city. In *AAAI*, page 58, 2012.

[5] K. Feng, G. Cong, S. S. Bhowmick, W.-C. Peng, and C. Miao. Towards best region search for data exploration. In *SIGMOD*, pages 1055–1070. ACM, 2016.

[6] G. Ference, M. Ye, and W.-C. Lee. Location recommendation for out-of-town users in location-based social networks. In *CIKM*, pages 721–726. ACM, 2013.

[7] H. Gao, J. Tang, X. Hu, and H. Liu. Content-aware point of interest recommendation on location-based social networks. In *AAAI*, pages 1721–1727. Citeseer, 2015.

[8] H. Gao, J. Tang, and H. Liu. gscorr: Modeling geo-social correlations for new check-ins on location-based social networks. In *CIKM*, pages 1582–1586. ACM, 2012.

[9] A. Gionis, T. Lappas, K. Pelechrinis, and E. Terzi. Customized tour recommendations in urban areas. In *WSDM*, pages 313–322. ACM, 2014.

[10] H. Imai and T. Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of algorithms*, 4(4):310–323, 1983.

[11] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. In *NIPS*, pages 556–562, 2001.

[12] J. Li, H. Wang, B. Zhang, and N. Zhang. Linear time approximation schemes for geometric maximum coverage. In *COCOON*, pages 559–571. Springer, 2015.

[13] X. Li, G. Cong, X.-L. Li, T.-A. N. Pham, and S. Krishnaswamy. Rank-geofm: a ranking based geographical factorization method for point of interest recommendation. In *SIGIR*, pages 433–442. ACM, 2015.

[14] D. Lian, C. Zhao, X. Xie, G. Sun, E. Chen, and Y. Rui. Geomf: joint geographical modeling and matrix factorization for point-of-interest recommendation. In *SIGKDD*, pages 831–840. ACM, 2014.

[15] B. Liu, Y. Fu, Z. Yao, and H. Xiong. Learning geographical preferences for point-of-interest recommendation. In *SIGKDD*, pages 1043–1051. ACM, 2013.

[16] Y. Liu, W. Wei, A. Sun, and C. Miao. Exploiting geographical neighborhood characteristics for location recommendation. In *CIKM*, pages 739–748. ACM, 2014.

[17] J. McAuley, R. Pandey, and J. Leskovec. Inferring networks of substitutable and complementary products. In *SIGKDD*, pages 785–794. ACM, 2015.

[18] S. C. Nandy and B. B. Bhattacharya. A unified algorithm for finding maximum and minimum object enclosing rectangles and cuboids. *Computers & Mathematics with Applications*, 29(8):45–61, 1995.

[19] Y. Tao, X. Hu, D.-W. Choi, and C.-W. Chung. Approximate maxrs in spatial databases. *VLDB*, 6(13):1546–1557, 2013.

[20] W. Wang, H. Yin, L. Chen, Y. Sun, S. Sadiq, and X. Zhou. Geo-sage: a geographical sparse additive generative model for spatial item recommendation. In *SIGKDD*, pages 1255–1264. ACM, 2015.

[21] M. Ye, P. Yin, W.-C. Lee, and D.-L. Lee. Exploiting geographical influence for collaborative point-of-interest recommendation. In *SIGIR*, pages 325–334. ACM, 2011.

[22] H. Yin, Y. Sun, B. Cui, Z. Hu, and L. Chen. Lcars: a location-content-aware recommender system. In *SIGKDD*, pages 221–229. ACM, 2013.

[23] A. X. Zhang, A. Noulas, S. Scellato, and C. Mascolo. Hoodsquare: Modeling and recommending neighborhoods in location-based social networks. In *SocialCom*, pages 69–74. IEEE, 2013.

[24] C. Zhang, H. Liang, K. Wang, and J. Sun. Personalized trip recommendation with poi availability and uncertain traveling time. In *CIKM*, pages 911–920. ACM, 2015.

[25] J.-D. Zhang and C.-Y. Chow. Geosoca: Exploiting geographical, social and categorical correlations for point-of-interest recommendations. In *SIGIR*, pages 443–452. ACM, 2015.