Optimal Region Search with Submodular Maximization

Xuefeng Chen[1] , **Xin Cao**[1] , **Yifeng Zeng**[2] , **Yixiang Fang**[1] and **Bin Yao**[3]
[1] School of Computer Science and Engineering, University of New South Wales, Australia,
[2] Department of Computer & Information Sciences, Northumbria University, UK,
[3] Department of Computer Science and Engineering, Shanghai Jiaotong University, China,
{xuefeng.chen, xin.cao, yixiang.fang}@unsw.edu.au, yifeng.zeng.uk@gmail.com,
yaobin@cs.sjtu.edu.cn

Abstract

Region search is an important problem in location-based services due to its wide applications. In this paper, we study the problem of optimal region search with submodular maximization (ORS-SM). This problem considers a region as a connected subgraph. We compute an objective value over the locations in the region using a submodular function and a budget value by summing up the costs of edges in the region, and aim to search the region with the largest objective score under a budget value constraint. ORS-SM supports many applications such as the most diversified region search. We prove that the problem is NP-hard and develop two approximation algorithms with guaranteed error bounds. We conduct experiments on two applications using three real-world datasets. The results demonstrate that our algorithms can achieve high-quality solutions and are faster than a state-of-the-art method by orders of magnitude.

1 Introduction

With the proliferation of mobile devices and Location-Based Services (LBS) (e.g., Google Maps, Foursquare), a huge amount of geo-tagged data are being generated every day. The geo-tagged data has rich information including timestamp, geo-location and comments, which facilitates a large number of location-based applications. As an important problem in LBS, region search has wide applications such as user exploration in a city [Feng *et al.*, 2016; Cao *et al.*, 2014], similar regions query [Feng *et al.*, 2019a], region detection [Feng *et al.*, 2019b], etc.

Most existing studies consider a region as a rectangle with a fixed length and width [Nandy and Bhattacharya, 1995; Choi *et al.*, 2012; Tao *et al.*, 2013; Feng *et al.*, 2016]. It is not general in practice since the regions might be of arbitrary shapes. In order to address this limitation, Cao *et al.* [Cao *et al.*, 2014] define a region as a connected subgraph and propose the Length-Constrained Maximum-Sum Region (LCMSR) query. It considers two attributes for region search: an objective value of the region computed by summing up the weights of nodes in a region and a budget value by summing up the costs of edges in a region. The target of the LCMSR query is to find the region that has the largest objective value under a given budget constraint.

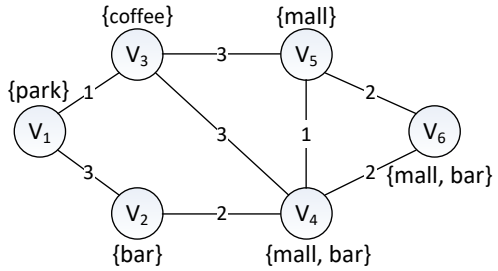


Figure 1: A toy road network with POIs.

A simple accumulative function (e.g., SUM) cannot measure the objective value in some applications (e.g., the influence or the diversity of a region [Feng *et al.*, 2016], the likelihood of information [Leskovec *et al.*, 2007], the entropy of locations [Zhang and Vorobeychik, 2016]). In this paper, we use a submodular function to compute the objective score of a region defined as a connected subgraph, which makes the problem more challenging. We denote this problem as optimal region search with submodular maximization (ORS-SM).

We use the example network shown in Figure 1 to illustrate the problem, which consists of 6 locations connected by 8 edges. Each location is associated with some keywords (e.g, mall, coffee) which represent its POI categories, and each edge has a travel time cost. Consider a user travels in a city. She would like to explore the most diversified region to experience the most types of POIs, and she does not want to spend too much time on the road. We can use ORS-SM to find solutions for the user by computing the objective score of a region as the number of different types of POIs contained in the region and setting a travel time limit Δ. Given $\Delta = 5$, LCMSR returns the region $R_1 = \{e_1 = (v_2, v_4), e_2 = (v_4, v_5), e_3 = (v_4, v_6)\}$, as R_1 has the most the number of keywords (not distinct!), but R_1 is not the most diversified region since it only contains two types of locations. By contrast, ORS-SM returns $R_2 = \{e_1 = (v_1, v_3), e_2 = (v_3, v_5), e_3 = (v_4, v_5)\}$ which has the most distinct keywords.

As the objective function we use is submodular, the ORS-SM problem is a type of submodular optimization [Krause and Golovin, 2014], which are extremely hard [Zeng *et al.*, 2015; Zhang and Vorobeychik, 2016; Stan *et al.*, 2017; Crawford, 2019]. We prove that solving the ORS-SM prob-

lem is NP-hard. To our best knowledge, only the Generalized Cost-Benefit Algorithm (GCB) [Zhang and Vorobeychik, 2016] for the problem of submodular optimization with routing constraints can be adapted to solve our ORS-SM problem. The algorithm requires a root node, which does not exist in our problem. Hence, we need to find a tree with the largest objective value by treating each node as root, and the best tree among them is returned as the result for ORS-SM. We denoted this method as GCBAll. GCBAll has a high complexity, so that it has poor efficiency and is not scalable.

To better solve ORS-SM, we propose a $\frac{1}{O(\sqrt{\Delta/b_{min}})}$-approximation algorithm AppORS, where b_{min} is the minimal cost on edges. We observe that the returned region is always a tree. The basic idea of this algorithm is to first find a set of nodes with large objective scores, and we guarantee that there exists a tree connecting them whose total cost is smaller than the budget limit Δ. Next, we find the best region by using these nodes. Next, we propose another algorithm with the same approximation ratio called IAppORS to further improve the effectiveness of AppORS. This algorithm can obtain regions of better quality.

In summary, the contribution of this paper is threefold. Firstly, we propose ORS-SM through a submodular optimization and prove its NP-hardness. Secondly, we develop two approximation algorithms AppORS and IAppORS for this problem. Finally we conduct experiments on two applications using three real-world datasets and demonstrate the efficiency and accuracy of the proposed algorithms.

2 Related Work

Region search has been well studied and has a wide range of applications such as user exploration in a city [Feng *et al.*, 2016; Cao *et al.*, 2014], similar regions search [Feng *et al.*, 2019a], region detection [Feng *et al.*, 2019b], etc.

Most existing region search problems consider the region as size-fixed rectangles or radius-fixed circles. The Max-Enclosing Rectangle (MER) problem [Nandy and Bhattacharya, 1995] aims to find a rectangle with a given size $a \times b$ such that the rectangle encloses the maximum number of spatial objects. This problem is systematically studied as the maximizing range sum problem [Choi *et al.*, 2012; Tao *et al.*, 2013]. The similar region search [Feng *et al.*, 2019a], bursty region detection [Feng *et al.*, 2019b], and the best region search [Feng *et al.*, 2016] also consider regions as rectangles. However, as shown in the study of the LCMSR query proposed by Cao *et al.* [Cao *et al.*, 2014], in practice the regions are usually arbitrarily shaped. LCMSR defines a region as a connected subgraph with relevant objects. It uses a sum function to accumulate the weights of objects in the region, and it aims to find the optimal region that has the largest total weight and does not exceed a size limit. We use a submodular function to compute the objective value for a region and aim to maximize this value under a budget constraint, which makes the problem more challenging.

Our work is also related to submodular optimization which has breadth of applicability with applications including viral marketing [Kempe *et al.*, 2003], recommendation system [Chen *et al.*, 2015], information gathering [Leskovec *et al.*, 2007], deep neural network training [Joseph *et al.*, 2019], etc. A large amount of research considers submodular optimization under constraints, such as cardinality constraint [Kempe *et al.*, 2003], cost constraint [Qian *et al.*, 2017], routing constraint [Zeng *et al.*, 2015; Zhang and Vorobeychik, 2016], connectivity constraint [Kuo *et al.*, 2014], and so on. Submodular Optimization under Connectivity Constraint (SOCC) [Kuo *et al.*, 2014] is similar to our problem, but it considers the cost on nodes to compute the budget values. In contrast, the budget values are associated with edges in our problem, and thus the algorithms for SOCC is not feasible to solve ORS-SM.

3 Problem Formulation

In this section, we formally define the problem of *optimal region search with submodular maximization* (ORS-SM) and prove its hardness. The problem is defined over a spatial network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges. Each node $v \in \mathcal{V}$ represents a location, and each edge $\langle v_i, v_j \rangle \in \mathcal{E}$ represents an undirected path between two locations v_i and v_j in $\mathcal{V}$ associated with a cost $b(v_i, v_{i+1})$ (representing the distance, the travel time, etc.).

Definition 1. Region. *Given a spatial network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a region R is a connected subgraph of $\mathcal{G}$. We denote the nodes and edges in R as $R.\mathcal{V}$ and $R.\mathcal{E}$ respectively.*

In order to measure the quality of a region, we consider two values, namely the *budget value* and the *objective value*. The budget value is defined as the sum of the costs on all the edges in the region, which is computed as:

$$BS(R) = \sum_{(v_i,v_j)\in R.\mathcal{E}} b(v_i, v_j) \tag{1}$$

The objective value is computed by a submodular function $f()$ over the set of locations in a region (e.g., the number of distinct keywords, the social influence, etc.):

$$OS(R) = f(\cup_{v_i \in R.\mathcal{V}} v_i) \tag{2}$$

$OS(R)$ is submodular means that $OS(R_1 \bigcup v_i) - OS(R_1) \geq OS(R_2 \bigcup v_i) - OS(R_2)$, for all $R_1 \subseteq R_2$ and $v_i \notin R_1$, and $R_1(R_2) \bigcup v_i$ composes a new region.

For example, we consider a problem of finding the most diversified region in a network $\mathcal{G}$ as shown in Figure 1. In this example, the objective value is computed as $OS(R) = |\cup_{v_i \in R.\mathcal{V}} \mathcal{K}(v_i)|$, where $\mathcal{K}(v_i)$ is the set of distinct keywords on v_i. For R_2 in the example, we can get $OS(R_2) = |\{bar, coffee, park, mall\}| = 4$ and $BS(R_2) = 5$.

Formally, we define the ORS-SM problem as follows:

Definition 2. Optimal Region Search with Submodular Maximization (ORS-SM). *Given $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and a budget constraint Δ, we aim to find the region R such that*

$$\begin{array}{l} R = argmax_R \ OS(R) \\ subject\ to \ BS(R) \leq \Delta \end{array} \tag{3}$$

Theorem 1. *Solving the ORS-SM problem is NP-hard.*

Proof. The LCMSR query [Cao *et al.*, 2014] which is mentioned in the Introduction is a special case of ORS-SM, because the sum function also satisfies the submodular properties. As the LCMSR problem is proved to be NP-hard, the ORS-SM problem is NP-hard as well. ■

4 Approximation Algorithm AppORS

In this section, we propose a novel approximation algorithm called AppORS for solving the ORS-SM problem, and we also prove its error bound. For any subgraph region R_i, we can always find its minimum spanning tree T_i such that $BS(T_i) \leq BS(R_i)$ and $OS(T_i) = OS(R_i)$. Hence, the optimal region of the ORS-SM problem must be a tree, and we consider the tree search in the subsequent discussions directly. Before presenting the algorithm, we first introduce the following lemma which lays the foundation of our algorithm.

Lemma 1. *For any tree $T = (\mathcal{V}_T, \mathcal{E}_T)$ with budget value $BS(T)$, there always exist $n \leq max(10\lfloor\frac{BS(T)}{m}\rfloor, 2)$ subtrees $T_i = (\mathcal{V}_{T_i}, \mathcal{E}_{T_i})$, $1 \leq i \leq n$, such that $BS(T_i) \leq m$ if $|\mathcal{E}_{T_i}| > 1$ and $\cup_{i=1}^{n} \mathcal{V}_{T_i} = \mathcal{V}_T$.*

Proof. We utilize the **claim 3** of the Maximum Connected Submodular set function with Budget constraint problem (MCSB) [Kuo *et al.*, 2014], and we conduct the proof through the following steps:

1) Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, we can get its line graph $\mathcal{G}^L = (\mathcal{V}^L, \mathcal{E}^L)$. For a tree $T = (\mathcal{V}_T, \mathcal{E}_T)$ with the budget value $BS(T)$ in $\mathcal{G}$, we can get its corresponding subgraph G_T^L in $\mathcal{G}^L$. If G_T^L contains cycles, we break cycles by removing some edges. After that, we get a tree $T^L = (\mathcal{V}_{T^L}, \mathcal{E}_{T^L})$ spanning all nodes of G_T^L. Now each edge in T corresponds to a node in T^L.

2) According to the claim 3 for MCSB, for T^L with budget value $BS(T^L)$, there always exist $n = max(5\lfloor\frac{BS(T^L)}{m}\rfloor, 1)$ subtrees $T_i^L = (\mathcal{V}_{T_i^L}, \mathcal{E}_{T_i^L})$, such that $BS(T_i^L) \leq m + BS(r_i^L)$ and $\cup_{i=1}^{n} \mathcal{V}_{T_i^L} = \mathcal{V}_{T^L}$, where r_i^L is the root of T_i^L for all $1 \leq i \leq n$. For each subtree T_i^L, if $\mathcal{V}_{T_i^L} > 1$, we can always select a node with degree 1 as r_i^L, and we can then divide it into two subtrees $T_i^{1,L}$ consisting of r_i^L and $T_i^{2,L}$ consisting of the remaining of $\mathcal{V}_{T_i^L}$. Hence, we know that $BS(T_i^{2,L}) \leq m$. After that, we can obtain $n \leq max(10\lfloor\frac{BS(T^L)}{m}\rfloor, 2)$ subtrees $T'^L_i = (\mathcal{V}_{T'^L_i}, \mathcal{E}_{T'^L_i})$, such that $BS(T'^L_i) \leq m$ if $|\mathcal{V}_{T'^L_i}| > 1$ and $\cup_{i=1}^{n} \mathcal{V}_{T'^L_i} = \mathcal{V}_{T^L}$.

3) For each subtree T'^L_i, we find its line graph G_i. As T'^L_i is obtained from the line graph of G, G_i must be a subgraph of the original tree T in the first step. Based on translated properties of a line graph, i.e., the the line graph of a connected graph is connected, G_i must be connected. Thus, each G_i must be a tree as well and we denote it as T_i, and we have $\cup_{i=1}^{n} \mathcal{V}_{T_i^L} = \mathcal{V}_{T^L} = \mathcal{E}_T$. We thus complete the proof. ■

Now we are ready to present our approximation algorithm AppORS. The basic idea of AppORS is to find a tree T^*, such that $OS(T^*)$ is larger than the objective score of any subtree T_i whose budget score is no larger than m and $BS(T^*) \leq \Delta$. We then can prove that the region R^* obtained based on T^* is an approximate feasible solution of the problem based on Lemma 1. In AppORS, Nemhauser's greedy algorithm [Nemhauser *et al.*, 1978] is used to to solve the Maximum Rooted Submodular set function (MRS) problem [Kuo *et al.*, 2014], which aims to find a node set containing a given root node and other $K-1$ nodes such that the score computed by a submodular function over the K nodes is maximum.

Algorithm 1: AppORS Algorithm

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Δ
Output: A region R^*

1 Initialize a node set $\mathcal{V}_{opt}^K \leftarrow \emptyset$, $v_{opt} \leftarrow \emptyset$;
2 $K \leftarrow \lceil\sqrt{\frac{\Delta}{b_{min}}}\rceil + 1$, $DisLim \leftarrow \sqrt{\Delta b_{min}}$;
3 **for** *each node* $v_i \in \mathcal{V}$ **do**
4 $\quad \mathcal{V}_{v_i} \leftarrow \{v_j | v_i \neq v_j, dis(v_i, v_j) \leq DisLim\}$;
5 $\quad$ Use Nemhauser's greedy algorithm to solve $MRS(OS(\cdot), \mathcal{V}_{v_i}, K, v_i)$, and get top-K node set $\mathcal{V}_{v_i}^K$;
6 $\quad$ **if** $OS(\mathcal{V}_{opt}^K) < OS(\mathcal{V}_{v_i}^K)$ **then**
7 $\quad\quad \mathcal{V}_{opt}^K \leftarrow \mathcal{V}_{v_i}^K$, $v_{opt} \leftarrow v_i$;
8 $T^* \leftarrow findOptTree(\mathcal{V}_{opt}^K, \mathcal{G}, \Delta, v_{opt})$;
9 $T' \leftarrow$ a single edge in $\mathcal{E}$ with the maximum OS value and $BS(T') \leq \Delta$;
10 $R^* \leftarrow T^*$ **if** $OS(T^*) \geq OS(T')$ **else** T';
11 **return** the region R^*;

As shown in Alg.1, AppORS starts with an initial node set $\mathcal{V}_{opt}^K$, a root node v_{opt}, two parameters K and $DisLim$ (lines 1-2), where $b_{min} = min_{(v_i,v_j)\in\mathcal{V}} b(v_i, v_j)$. After that, for each node $v_i \in \mathcal{V}$, AppORS first finds the candidate node set $\mathcal{V}_{v_i}$ in which each node has a distance to v_i smaller than $DisLim$ (lines 3-4). Then AppORS constructs a MRS problem instance $MRS(OS(\cdot), \mathcal{V}_{v_i}, K, v_i)$, and then uses the Nemhauser's greedy algorithm to solve the instance for getting a K-node set $\mathcal{V}_{v_i}^K$ (line 5). If $OS(\mathcal{V}_{opt}^K) < OS(\mathcal{V}_{v_i}^K)$, AppORS updates $\mathcal{V}_{opt}^K$ and v_{opt} (lines 6-7). Next, AppORS uses function $findOptTree(\mathcal{V}_{opt}^K, \mathcal{G}, \Delta, v_{opt})$ to find a tree T^* spanning all nodes in $\mathcal{V}_{opt}^K$, satisfying $BS(T^*) \leq \Delta$ (line 8). After that, AppORS finds T' consisting of a feasible edge by scanning all edges in $\mathcal{E}$ which has the maximum objective score (line 9). Finally, it returns the tree with the larger objective value as the region R^* (lines 10-11). One simple method of implementing $findOptTree(\mathcal{V}_{opt}^K, \mathcal{G}, \Delta, v_{opt})$ is to find all shortest paths from v_{opt} to all nodes in $\mathcal{V}_{opt}^K$, and then get a tree T^* by removing duplicate edges in the shortest paths. We will present a better method in Section 5.3.

We analyze the error bound of AppORS as below.

Lemma 2. *AppORS returns a feasible solution of the ORS-SM problem.*

Proof. In AppORS, $BS(T') \leq \Delta$. And $\mathcal{V}_{opt}^K$ contains K nodes, the largest distance between v_{opt} and $v_j \in \mathcal{V}_{opt}^K$ is $DisLim$. Thus, in the tree T^* is achieved by combining all shortest paths from v_{opt} to all nodes in $\mathcal{V}_{opt}^K$, $BS(T^*) \leq (K-1) * DisLim = \Delta$. ■

Lemma 3. $OS(R_{opt}) \leq O(\sqrt{\Delta/b_{min}})OS(R^*)$, *where R_{opt} denotes the optimal region of the ORS-SM problem.*

Proof. We denote the optimal solution of $MRS(OS(\cdot), \mathcal{V}_{v_i}, K, v_i)$ as $\mathcal{V}_{opt}^{MRS}$, the optimal tree rooted at v_i whose budget value under $DisLim$ as $T^{v_i,DL}$, and $T^{v_i,DL}$ with the maximal objective value as $T_{opt}^{v_i,DL}$. We can get that $T_{opt}^{v_i,DL}$ has at most $\lfloor\frac{DisLim}{b_{min}}\rfloor$ nodes except v_i,

and these nodes are in $\{v_j|dis(v_i,v_j) \leq DisLim\}$. And $\lfloor \frac{DisLim}{b_{min}} \rfloor \leq \lfloor \sqrt{\frac{\Delta}{b_{min}}} \rfloor \leq K-1$. For any node $v_j \in T_{opt}^{v_i,DL}$, $v_j \in \mathcal{V}_{v_i}$. Thus, $OS(\mathcal{V}_{opt}^{MRS}) \geq OS(T_{opt}^{v_i,DL})$. Meanwhile, in AppORS, $OS(T^*) \geq OS(\mathcal{V}_{opt}^K)$. And based on Lemma 2 in [Kuo *et al.*, 2014], Nemhauser's greedy algorithm can get a $\frac{e-1}{e}$-approximate solution for the MRS problem, we have $OS(\mathcal{V}_{opt}^K) \geq \frac{e-1}{e} OS(\mathcal{V}_{opt}^{MRS})$. Hence we can get $OS(T^*) \geq \frac{e-1}{e} OS(T_{opt}^{v_i,DL})$.

As R_{opt} must be a tree, we use T_{opt} to represent it. According to Lemma 1, for $T_{opt} = (\mathcal{V}_{T_{opt}}, \mathcal{E}_{T_{opt}})$ with the budget value T_{opt}, there always exist $n \leq max(10\lfloor \frac{\Delta}{DisLim} \rfloor, 2)$ subtrees $T_i = (\mathcal{V}_{T_i}, \mathcal{E}_{T_i})$, where $BS(T_i) \leq DisLim$ if $|\mathcal{E}_{T_i}| > 1$, such that $\cup_{i=1}^n \mathcal{V}_{T_i} = \mathcal{V}_{T_{opt}}$. Hence, we have:

$$
\begin{aligned}
&OS(R_{opt}) = OS(T_{opt}) = OS(\mathcal{V}_{T_{opt}}) = OS(\cup_{i=1}^n \mathcal{V}_{T_i}) \\
&\leq max(10\lfloor \frac{\Delta}{DisLim} \rfloor, 2) * max(OS(T_{opt}^{v_i,DL}), OS(T')) \\
&\leq O(\sqrt{\Delta/b_{min}}) * max(\frac{e}{e-1} OS(T^*), OS(T')) \\
&= O(\sqrt{\Delta/b_{min}}) OS(R^*)
\end{aligned}
$$

where the first inequality is derived from the property of the submodular function $OS(\cdot)$. ■

Based on Lemma 2 and 3, we can obtain Theorem 3:

Theorem 2. *AppORS is a $\frac{1}{O(\sqrt{\Delta/b_{min}})}$-approximation algorithm for the ORS-SM problem.*

Proof. It is obvious since AppORS returns a feasible solution whose objective score is no smaller than $\frac{1}{O(\sqrt{\Delta/b_{min}})}$ times of the optimal value. ■

5 Improved AppORS

As AppORS considers the worst case to compute K nodes under the budget constraint and only searches within $\mathcal{V}_{v_i} = \{v_j|dis(v_i,v_j) \leq DisLim\}$, it cannot achieve high-quality solutions in practice (as shown in the experimental study). To improve the solution quality of AppORS, we present an Improved version of AppORS (IAppORS). Meanwhile, we develop two heuristic methods to implement a key function of IAppORS.

5.1 An Improved Version of AppORS

We abbreviate lines 3-7 of Alg. 1 as: for each node $v_i \in \mathcal{V}$ do $update(v_{opt}, \mathcal{V}_{opt}^K)$. In the improved version of AppORS, the details of the new $update(v_{opt}, \mathcal{V}_{opt}^K)$ function are shown in Alg. 2. Different to the old function, the new one improves the solution quality by using a method $findFeaNodeSet(v_i, \mathcal{G}, \Delta)$ to find another **feasible** node set $\mathcal{V}_{v_i}^{2,K}$ (lines 4-6). Note that, iff $\mathcal{V}_{v_i}^{2,K}$ has a tree $T_{v_i}^{2,K}$ spanning its all nodes, and $BS(T_{v_i}^{2,K}) \leq \Delta$, $\mathcal{V}_{v_i}^{2,K}$ is a feasible node set.

Theorem 3. *IAppORS returns a $\frac{1}{O(\sqrt{\Delta/b_{min}})}$-approximation solution of the ORS-SM problem.*

Proof. As both of $\mathcal{V}_{v_i}^{1,K}$ and $\mathcal{V}_{v_i}^{2,K}$ are feasible node sets, IAppORS can find a feasible solution for the ORS-SM problem. Meanwhile, the solution quality of IAppORS is not worse than that of AppORS. This leads to the final conclusion. ■

Algorithm 2: New $update(v_{opt}, \mathcal{V}_{opt}^K)$

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Δ, K, $DisLim$, v_i, v_{opt}, $\mathcal{V}_{opt}^K$
Output: Updated v_{opt}, $\mathcal{V}_{opt}^K$

1 $\mathcal{V}_{v_i}^1 \leftarrow \{v_j|v_i \neq v_j, dis(v_i,v_j) \leq DisLim\}$;
2 Use Nemhauser's greedy algorithm to solve $MRS(OS(\cdot), \mathcal{V}_{v_i}^1, K, v_i)$, and get topK node set $\mathcal{V}_{v_i}^{1,K}$;
3 $\mathcal{V}_{v_i}^K \leftarrow \mathcal{V}_{v_i}^{1,K}$;
4 $\mathcal{V}_{v_i}^{2,K} \leftarrow findFeaNodeSet(v_i, \mathcal{G}, \Delta)$;
5 **if** $OS(\mathcal{V}_{v_i}^K) < OS(\mathcal{V}_{v_i}^{2,K})$ **then**
6 $\quad \mathcal{V}_{v_i}^K \leftarrow OS(\mathcal{V}_{v_i}^{2,topK})$;
7 **if** $OS(\mathcal{V}_{opt}^K) < OS(\mathcal{V}_{v_i}^K)$ **then**
8 $\quad \mathcal{V}_{opt}^K \leftarrow \mathcal{V}_{v_i}^K$, $v_{opt} \leftarrow v_i$;
9 **return** v_{opt}, $\mathcal{V}_{opt}^K$;

5.2 Two Heuristic Methods for IAppORS

To implement $findFeaNodeSet(v_i, \mathcal{G}, \Delta)$ in IAppORS, we present two heuristic methods. At first, we introduce the function $compute(BS(\mathcal{V}_i))$ which is used to compute the minimal budget value of the tree spanning all nodes in the node set $\mathcal{V}_i$. $compute(BS(\mathcal{V}_i))$ is a Steiner tree problem which is proved to be NP-hard [Vazirani, 2013]. Rather than computing accurate $BS(\mathcal{V}_i)$, we use an approximation algorithm to get an estimated value $\hat{BS}(\mathcal{V}_i)$. Here, we use Kruskal's algorithm to compute a Minimum-cost Spanning Tree (MST) for spanning all nodes in $BS(\mathcal{V}_i)$, as we can utilize the MST to obtain a 2-approximation solution for the Steiner tree problem [Vazirani, 2013], and Kruskal's algorithm is efficient. We denote this method as $compute(\hat{BS}(\mathcal{V}_i))$.

Next, we present the first implementation of $findFeaNodeSet_1(v_i, \mathcal{G}, \Delta)$. As shown in Alg. 3, it begins with initializing node sets $\mathcal{V}_{v_i}^{2,K}$ and $\mathcal{V}_{v_i}^2$ (lines 1). In each *while* loop, the method selects a node v_{max} with the maximum marginal objective value for the current $\mathcal{V}_{v_i}^{2,K}$ from $\mathcal{V}_{v_i}^2$. After that, the method removes v_{max} from $\mathcal{V}_{v_i}^2$ and computes $\hat{BS}(\mathcal{V}_{v_i}^{2,K} \cup v_{max})$ by utilizing the Kruskal's algorithm. If the estimated budget value is not larger than Δ, v_{max} would be inserted into $\mathcal{V}_{v_i}^{2,K}$. The process is terminated when $\mathcal{V}_{v_i}^2 = \emptyset$ (lines 2-7). At the last step, the method returns a node set $\mathcal{V}_{v_i}^{2,K}$.

Algorithm 3: Function $findFeaNodeSet_1(v_i, \mathcal{G}, \Delta)$

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, Δ, v_i
Output: $\mathcal{V}_{v_i}^{2,K}$

1 $\mathcal{V}_{v_i}^{2,K} \leftarrow \{v_i\}$, $\mathcal{V}_{v_i}^2 \leftarrow \{v_j|v_i \neq v_j, dis(v_i,v_j) \leq \Delta\}$;
2 **while** $\mathcal{V}_{v_i}^2 \neq \emptyset$ **do**
3 $\quad v_{max} \leftarrow argmaxOS_{v_j \in \mathcal{V}_{v_i}^2}(\mathcal{V}_{v_i}^{2,K} \cup v_j)$;
4 $\quad \mathcal{V}_{v_i}^2 \leftarrow \mathcal{V}_{v_i}^2 \setminus v_{max}$;
5 $\quad \hat{BS}(\mathcal{V}_{v_i}^{2,K}) \leftarrow compute(\hat{BS}(\mathcal{V}_{v_i}^{2,K} \cup v_{max}))$;
6 $\quad$ **if** $\hat{BS}(\mathcal{V}_{v_i}^{2,K} \cup v_{max}) \leq \Delta$ **then**
7 $\quad\quad \mathcal{V}_{v_i}^{2,K} \leftarrow \mathcal{V}_{v_i}^{2,K} \cup v_{max}$;
8 **return** $\mathcal{V}_{v_i}^{2,K}$;

Note that, as $BS(\mathcal{V}_{v_i}^{2,K} \cup v_{max}) \leq \hat{BS}(\mathcal{V}_{v_i}^{2,K} \cup v_{max})$, and thus we have $BS(\mathcal{V}_{v_i}^{2,K} \cup v_{max}) \leq \Delta$. So that $\mathcal{V}_{v_i}^{2,K}$ is a feasible solution of the ORS-SM problem. In addition, we can speed up the process of $findFeaNodeSet_1(v_i, \mathcal{G}, \Delta)$ using an optimization technique called Lazy Evaluations [Leskovec *et al.*, 2007]. We denote IAppORS with $findFeaNodeSet_1(v_i, \mathcal{G}, \Delta)$ as IAppORSHeu1.

Next, we propose another heuristic method to implement $findFeaNodeSet(v_i, \mathcal{G}, \Delta)$ which is faster than the first one. Intuitively, for two nodes v_1 and v_2, when $dis(v_1, v_2)$ is smaller, the difference between $\mathcal{V}_{v_1}^{2,K}$ and $\mathcal{V}_{v_2}^{2,K}$ is smaller. Thus, we can use $\mathcal{V}_{v_1}^{2,K}$ to approximate $\mathcal{V}_{v_2}^{2,K}$. Based on this observation, we present the second method $findFeaNodeSet_2(v_i, \mathcal{G}, \Delta)$. The most steps are similar to Alg. 3, but the line 4 is replaced by $\mathcal{V}_{v_i}^{nei} \leftarrow \{v_j | dis(v_i, v_j) \leq \gamma\Delta\}$, $\mathcal{V}_{v_i}^{2} \leftarrow \mathcal{V}_{v_i}^{2} \setminus \{\mathcal{V}_{v_i}^{nei} \cup v_{max}\}$, where $\gamma \in (0, 1]$ is a parameter, and $\mathcal{V}_{v_i}^{nei}$ is the neighbor node set of v_i. In this way, we use the seconde feasible node set of v_i as those of v_i's neighbor nodes, and reduce the related computations. We denote IAppORS with $findFeaNodeSet_2(v_i, \mathcal{G}, \Delta)$ as IAppORSHeu2, and $\mathcal{V}_{v_i}^{2,K}$ in IAppORSHeu2 is also a feasible node set.

5.3 A Better $findOptTree(\mathcal{V}_{opt}^{K}, \mathcal{G}, \Delta, v_{opt})$

In this section, we present a better method to implement the function $findOptTree(\mathcal{V}_{opt}^{K}, \mathcal{G}, \Delta, v_{opt})$ in line 8 of Alg. 1.

We first compute the shortest distances between each pair of nodes using some existing methods. Then, we use the Kruskal's algorithm to find a MST (denoted as MST_1) spanning all nodes in $\mathcal{V}_{opt}^{K}$, and MST_1 contains unknown shortest paths between some pairs of nodes. Next, we find the real shortest paths by inserting the node set $\mathcal{V}_{mid} = \{v_i | v_i \in \mathcal{V}_{v_{opt}}, \hat{BS}(\mathcal{V}_{opt}^{K} \cup v_i) = \hat{BS}(\mathcal{V}_{opt}^{K})\}$ into $\mathcal{V}_{opt}^{K}$, and finding a new MST (denoted as MST_2) spanning all nodes in MST_2, where $\mathcal{V}_{v_{opt}} \leftarrow \{v_i | v_i \notin \mathcal{V}_{opt}^{K}, dis(v_{opt}, v_i) \leq \Delta\}$.

MST_2 is a feasible solution for the ORS-SM problem, but its budget value may be far less than Δ. We further improve MST_2 by extending it greedily. In each iteration, we select a node which is connected to MST_2 and has the maximum marginal objective value, and insert the node into MST_2. The process is repeated until no more nodes can be added into MST_2. Finally, we can get a better region MST_2 from $\mathcal{V}_{opt}^{K}$. Thus, we use this method to do $findOptTree(\mathcal{V}_{opt}^{K}, \mathcal{G}, \Delta, v_{opt})$ for all proposed algorithms by default.

5.4 Time Complexity of the Proposed Algorithms

In this section, we analyze the complexity of all the proposed algorithms AppORS, IAppORSHeu1 and IAppORSHeu2. The three algorithms have two main steps. Step1 computes a feasible node set $\mathcal{V}_{opt}^{K}$, and Step 2 searches a feasible spanning tree T^* from $\mathcal{V}_{opt}^{K}$ and T' consisting of one edge. We denote the time cost of computing $dis(v_i, v_j)$ as t_{dis}. AppORS needs $|\mathcal{V}|$ loops in Step 1, and in each loop, it takes $O(|\mathcal{V}|t_{dis})$ time to get the candidate node set $\mathcal{V}_{v_i}$, and costs $O(K|\mathcal{V}_{v_i}|)$ time to get top K node set and update $\mathcal{V}_{opt}^{K}$. The method in Section 5.3 is the default method used in Step 2. It first takes $O(|\mathcal{V}_{opt}^{K}|^2 log|\mathcal{V}_{opt}^{K}|)$ time to find MST_1. After that, it costs $O(|\mathcal{V}|t_{dis})$ time to get the candidate node set $\mathcal{V}_{v_{opt}}$, and then costs $O(|\mathcal{V}_{v_{opt}}||\mathcal{V}_{opt}^{K}|^2 log|\mathcal{V}_{opt}^{K}|)$ time to get MST_2. Next, it requires $O(|\mathcal{V}_{v_{opt}}||\mathcal{V}_{opt}^{K}|)$ time to extend MST_2. Finally, it takes $O(|\mathcal{E}|)$ time to find T'. Thus, the time complexity of AppORS is $O(|\mathcal{V}|^2 t_{dis} + |\mathcal{V}_{v_{opt}}||\mathcal{V}_{opt}^{K}|^2 log|\mathcal{V}_{opt}^{K}|)$.

IAppORSHeu1 also needs $|\mathcal{V}|$ loops in Step 1. In each loop, it first takes $O(|\mathcal{V}|t_{dis})$ time to get $\mathcal{V}_{opt}^{1,K}$, and then costs $O(|\mathcal{V}|t_{dis} + |\mathcal{V}_{opt}^{2,K}|^2 log|\mathcal{V}_{opt}^{2,K}|)$ time to get $\mathcal{V}_{opt}^{2,K}$. Step 2 of IAppORSHeu1 is the same to that of AppORS, and the worst case of IAppORSHeu2 is IAppORSHeu1. Consequently the time complexity of IAppORSHeu1 and IAppORSHeu2 is $O(|\mathcal{V}|^2 t_{dis} + |\mathcal{V}||\mathcal{V}_{opt}^{2,K}|^2 log|\mathcal{V}_{opt}^{2,K}|)$.

6 Experimental Study

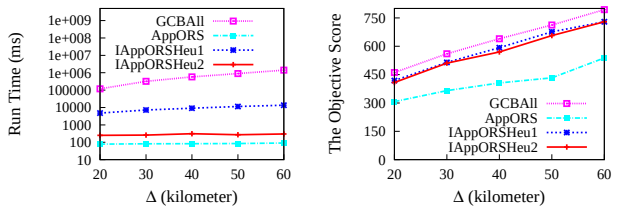
We compare the proposed algorithms to the state-of-the-art algorithm GCBAll in two applications on three real-world datasets. As discussed in Section 1, GCBAll is the adapted version of GCB [Zhang and Vorobeychik, 2016] for solving our ORS-SM problem. In GCBAll, to compute an approximation of the minimal budget value of a tree spanning all the given nodes, we adopt the 2-approximation algorithm for the Steiner tree problem [Vazirani, 2013] by utilizing the minimum spanning tree. We implement all the algorithms in JAVA on Windows 10, and run on a server with an Intel(R)Xeon(R) W-2155 3.3GHz CPU and 256 GB memory.

6.1 Case Study 1: Most Influential Region

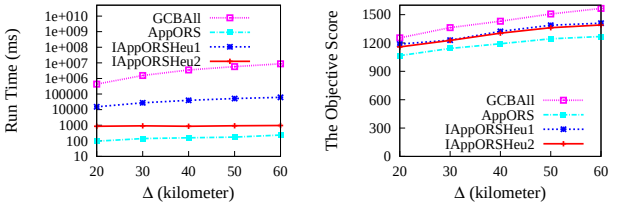
Problem Definition and Datasets.

We first investigate the Most Influential Region Search (MIRS) which aims to find an optimal region under a budget constraint such that the number of affected users is maximal in a geo-social network. This problem is useful in many scenarios, e.g., a company wants to find a region for marketing its products, the government intends to find a region for building some public facilities, etc. We model the geo-social network as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and compute the probability that a user u_i visits a region R as $P_{u_i}^{R} = 1 - \prod_{v_j \in R.\mathcal{V}}(1 - P_{u_i}^{v_j})$, where $P_{u_i}^{v_j}$ is the probability that u_i visits the location v_j, and it is computed as $P_{u_i}^{v_j} = \frac{\#\ of\ check{-}ins\ in\ v_j\ of\ u_i}{\#\ of\ check{-}ins\ of\ u_i}$. Thus, the objective value of R is the number of users expected to be affected by R, and it is computed as a submodular function: $OS(R) = \sum_{u_i \in \mathcal{U}} P_{u_i}^{R}$, where $\mathcal{U}$ represents all users.

In this problem, we use two real-world datasets SG and AS crawled from *FourSquare* (also used in the work [Zeng *et al.*, 2015]), in which SG has 189,306 check-ins made by 2,321 users at 5,412 locations in Singapore, and AS contains 201,525 check-ins made by 4,630 users at 6,176 locations in Austin. Following the work [Zeng *et al.*, 2015], we make an edge between two locations which were visited continuously in one day by the same user, and use the Euclidean distance as the budget value for each edge. We also removed the check-ins made by users who checked in a location less than 3 times.

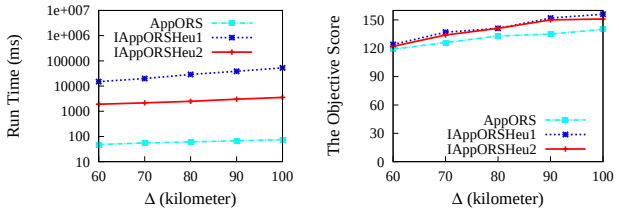


(a) SG (b) SG



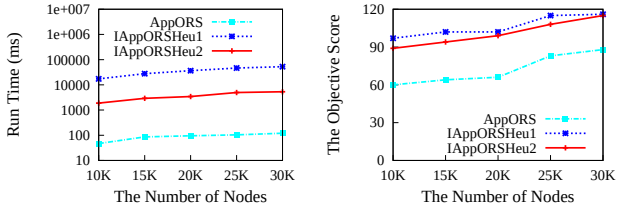
(c) AS (d) AS

Figure 2: Comparison of algorithms in MIRS.



(a) Efficiency (b) Solution Quality

Figure 3: Performances of the algorithms in MDRS on CA.



(a) Efficiency (b) Solution Quality

Figure 4: Performance of the algorithms in MDRS on SY.

Performance of Our Methods.
We compare three proposed algorithms with GCBAll in terms of efficiency (the run time) and effectiveness (the objective score) by varying Δ from $20km$ to $60km$. Fig. 2 shows that GCBAll is rather time-consuming when Δ is large, as it calls $compute(\hat{BS}(\mathcal{V}_i))$ too many times and cannot use Lazy Evaluations optimization [Leskovec *et al.*, 2007] to find v_{max} quickly. All the proposed algorithms are faster than GCBAll more than one order of magnitude. Meanwhile, the objective scores of IAppORSHeu1 and IAppORSHeu2 are relatively high (more than 90% of GCBAll), which is also consistent with our theoretical analysis of the solution quality. The results demonstrate that two presented heuristic methods can improve the solution quality of AppORS significantly. The experimental results also show that the solution quality of our algorithms is better than that of GCBAll within a run time limit (e.g., 1s, 5s, 10s). We omit these results here due to the space limitation.

6.2 Case Study 2: Most Diversified Region

Problem Definition and Datasets.
The second application Most Diversified Region Search (MDRS) is to find the most diversified region under a budget constraint. We consider MDRS on road networks which contains a set of locations associated with a set of keywords (e.g., mall and bar). We measure the diversity of a region using the number of different keywords, and thus the objective function is $OS(R) = |\cup_{v_i \in R.\mathcal{V}} \mathcal{K}(v_i)|$, where $\mathcal{K}(v_i)$ is the set of keywords on v_i.

We use the road network in California (CA) from a public website[1]. We then utilized the Foursquare APIs to fill in the missing keywords for nodes (categories of locations)[2]. CA contains 21,048 nodes and 22,830 edges [Li *et al.*, 2005].

Performance of Our Methods.
As GCBAll cannot find results under $\Delta = 30km$ on AS within $1,000s$ and CA is even larger than AS, we only compare the performance of the three proposed algorithms on a larger Δ using CA. As shown in Fig. 3, the objective scores of IAppORSHeu1 and IAppORSHeu2 are competitive, and higher than that of AppORS. Meanwhile, IAppORSHeu2 is faster than IAppORSHeu1 by about one order of magnitude, and can solve the problem with 4s when $\Delta = 100km$.

In addition, to compare the performances of the three algorithms by varying the dataset size, we generate a synthetic dataset (denoted as SY) based on the structure of the CA dataset. Next, we run the three algorithms on SY by varying the number of nodes from 10,000 to 30,000, and set $\Delta = 60km$. Fig. 4 shows that the run time of IAppORSHeu2 grows slowly with the increasing number of nodes, and the solution quality of IAppORSHeu2 is close to that of IAppORSHeu1, and better than that of AppORS.

7 Conclusion

We propose the ORS-SM problem which aims to find the optimal region such that it maximizes the objective score of a the region computed by a submodular function subject to a given budget score constraint. To efficiently solve the ORS-SM problem, we propose two approximation algorithms and further improve them with some optimization techniques. The results of empirical studies on two applications using three real-world datasets demonstrate the efficiency and the solution quality of our proposed algorithms. In future work, we would like to design an efficient index to improve the efficiency and scalability of the proposed methods.

Acknowledgements

Xin Cao is supported by ARC DE190100663. Yifeng Zeng is supported by EPSRC-EP/S011609/1. Bin Yao is supported by NSFC (No. U1636210, 61729202, 61922054, 61872235, 61832017), The National Key Research and Development Program of China (2018YFC1504504). Xin Cao is the corresponding author of this paper.

[1]http://www.cs.utah.edu/ lifeifei/SpatialDataset.htm
[2]https://developer.foursquare.com/docs/api

References

[Cao *et al.*, 2014] Xin Cao, Gao Cong, Christian S Jensen, and Man Lung Yiu. Retrieving regions of interest for user exploration. *Proceedings of the VLDB Endowment*, 7(9):733–744, 2014.

[Chen *et al.*, 2015] Xuefeng Chen, Yifeng Zeng, Gao Cong, Shengchao Qin, Yanping Xiang, and Yuanshun Dai. On information coverage for location category based point-of-interest recommendation. In *AAAI*, pages 37–43, 2015.

[Choi *et al.*, 2012] Dong-Wan Choi, Chin-Wan Chung, and Yufei Tao. A scalable algorithm for maximizing range sum in spatial databases. *Proceedings of the VLDB Endowment*, 5(11):1088–1099, 2012.

[Crawford, 2019] Victoria G Crawford. An efficient evolutionary algorithm for minimum cost submodular cover. In *IJCAI*, pages 1227–1233, 2019.

[Feng *et al.*, 2016] Kaiyu Feng, Gao Cong, Sourav S Bhowmick, Wen-Chih Peng, and Chunyan Miao. Towards best region search for data exploration. In *SIGMOD*, pages 1055–1070. ACM, 2016.

[Feng *et al.*, 2019a] Kaiyu Feng, Gao Cong, Christian S Jensen, and Tao Guo. Finding attribute-aware similar regions for data analysis. *Proceedings of the VLDB Endowment*, 12(11):1414–1426, 2019.

[Feng *et al.*, 2019b] Kaiyu Feng, Tao Guo, Gao Cong, Sourav S Bhowmick, and Shuai Ma. Surge: Continuous detection of bursty regions over a stream of spatial objects. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[Joseph *et al.*, 2019] KJ Joseph, Krishnakant Singh, and Vineeth N Balasubramanian. Submodular batch selection for training deep neural networks. In *IJCAI*, pages 2677–2683, 2019.

[Kempe *et al.*, 2003] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *KDD*, pages 137–146. ACM, 2003.

[Krause and Golovin, 2014] Andreas Krause and Daniel Golovin. Submodular function maximization., 2014.

[Kuo *et al.*, 2014] Tung-Wei Kuo, Kate Ching-Ju Lin, and Ming-Jer Tsai. Maximizing submodular set function with connectivity constraint: Theory and application to networks. *IEEE/ACM Transactions on Networking*, 23(2):533–546, 2014.

[Leskovec *et al.*, 2007] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. Cost-effective outbreak detection in networks. In *SIGKDD*, pages 420–429, 2007.

[Li *et al.*, 2005] Feifei Li, Dihan Cheng, Marios Hadjieleftheriou, George Kollios, and Shang-Hua Teng. On trip planning queries in spatial databases. In *SSTD*, pages 273–290. Springer, 2005.

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

[Nemhauser *et al.*, 1978] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functions. *Mathematical programming*, 14(1):265–294, 1978.

[Qian *et al.*, 2017] Chao Qian, Jing-Cheng Shi, Yang Yu, and Ke Tang. On subset selection with general cost constraints. In *IJCAI*, volume 17, pages 2613–2619, 2017.

[Stan *et al.*, 2017] Serban Stan, Morteza Zadimoghaddam, Andreas Krause, and Amin Karbasi. Probabilistic submodular maximization in sub-linear time. In *ICML*, volume 70, pages 3241–3250. JMLR. org, 2017.

[Tao *et al.*, 2013] Yufei Tao, Xiaocheng Hu, Dong-Wan Choi, and Chin-Wan Chung. Approximate maxrs in spatial databases. *Proceedings of the VLDB Endowment*, 6(13):1546–1557, 2013.

[Vazirani, 2013] Vijay V Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

[Zeng *et al.*, 2015] Yifeng Zeng, Xuefeng Chen, Xin Cao, Shengchao Qin, Marc Cavazza, and Yanping Xiang. Optimal route search with the coverage of users' preferences. In *IJCAI*, pages 2118–2124, 2015.

[Zhang and Vorobeychik, 2016] Haifeng Zhang and Yevgeniy Vorobeychik. Submodular optimization with routing constraints. In *AAAI*, pages 819–825, 2016.