Top-k Route Search through Submodularity Modeling of Recurrent POI Features

Hongwei Liang
School of Computing Science
Simon Fraser University, Canada
hongweil@sfu.ca

Ke Wang
School of Computing Science
Simon Fraser University, Canada
wangk@cs.sfu.ca

ABSTRACT

We consider a practical top-k route search problem: given a collection of points of interest (POIs) with rated features and traveling costs between POIs, a user wants to find k routes from a source to a destination and limited in a cost budget, that maximally match her needs on feature preferences. One challenge is dealing with the personalized diversity requirement where users have various trade-off between quantity (the number of POIs with a specified feature) and variety (the coverage of specified features). Another challenge is the large scale of the POI map and the great many alternative routes to search. We model the personalized diversity requirement by the whole class of submodular functions, and present an optimal solution to the top-k route search problem through indices for retrieving relevant POIs in both feature and route spaces and various strategies for pruning the search space using user preferences and constraints. We also present promising heuristic solutions and evaluate all the solutions on real life data.

KEYWORDS

Location-based Search; Route Planning; Diversity Requirement

ACM Reference Format:
Hongwei Liang and Ke Wang. 2018. Top-k Route Search through Submodularity Modeling of Recurrent POI Features. In *SIGIR '18: The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, July 8–12, 2018, Ann Arbor, MI, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209978.3210038

1 INTRODUCTION

The dramatic growth of publicly accessible mobile/geo-tagged data has triggered a revolution in location based services [10]. An emerging thread is route planning, with pervasive applications in trip recommendation, intelligent navigation, ride-sharing, and augmented-reality gaming, etc. According to [23], the travel and tourism industry directly and indirectly contributed US$7.6 trillion to the global economy and supported 292 million jobs in 2016. The majority of current route planning systems yields shortest paths or explores popular POIs [28], or recommends routes based on users' historical records [15] or crowdsourced experience [20].

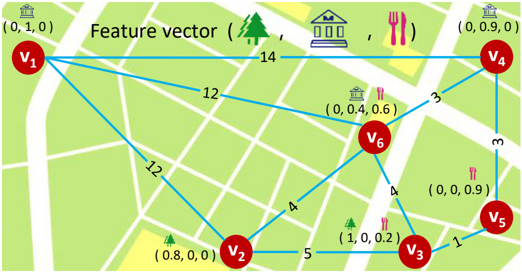


Figure 1: A sample POI map. Each node v_i represents a POI with 3 features (Park, Museum, Restaurant). Each feature has a numeric rating in the range $[0, 1]$, indicated by the vector aside the POI. Each edge has an associated traveling cost.

A practical problem that has not been well studied is that, a user wants to be suggested a small number of routes that not only satisfy her cost budget and spatial constraints, but also best meet her personalized requirements on diverse POI features. We instantiate this problem with a travel scenario. Consider that a new visitor to Rome wishes to be recommended a trip, starting from her hotel and ending at the airport, that allows her to visit museums, souvenir shops, and eat at some good Italian restaurants (not necessarily in this order) in the remaining hours before taking the flight. She values the variety over the number of places visited, e.g., a route consisting of one museum, one shop, and one Italian restaurant is preferred to a route consisting of two museums and two shops.

The above problem is actually generalizable to various route planning scenarios, and they illustrate some common structures and requirements. First, there is a *POI map* where POIs are connected by edges with traveling cost between POIs, and each POI has a location and is associated with a vector of features (e.g., museum) with numeric or binary ratings. The POI map can be created from Google Map, and features and ratings of POIs can be created from user rating and text tips available on location-based services such as Foursquare, or extracted from check-ins and user provided reviews [7]. Second, the user seeks to find *top-k routes* $\{\mathcal{P}_1, \cdots, \mathcal{P}_k\}$, from a specified source x to a specified destination y within a travel cost budget b, that have highest values of a certain gain function $Gain(\mathcal{P}_{iV})$ for the set of POIs $\mathcal{P}_{iV}$ on the routes $\mathcal{P}_i$. The user specifies her preference of routes through a weight vector $\mathbf{w}$ with $\mathbf{w}_h$ being the weight of a feature h, and a *route diversity requirement*, which specifies a trade-off between quantity (the number of POIs with a preferred feature) and variety (the coverage of preferred features) for the POIs on a route. The gain function has the form $Gain(\mathcal{P}_V) = \sum_h \mathbf{w}_h \Phi_h(\mathcal{P}_V)$, where Φ_h for each feature h aggregates the feature's scores of the POIs $\mathcal{P}_V$.

SIGIR'18, July 2018, Ann Arbor, MI, USA

ACM ISBN 978-1-4503-5657-2/18/07...$15.00
https://doi.org/10.1145/3209978.3210038

To better motivate the route diversity requirement, let us consider the POI map in Figure 1 and a user with the source v_1, destination v_5 and the budget $b = 18$. The user weights the features Park and Museum using the vector $\mathbf{w} = (0.4, 0.6, 0)$, and values *both* quantity and variety. If the sum aggregation Φ_h is used, the route $v_1 \rightarrow v_6 \rightarrow v_4 \rightarrow v_5$ will have the highest *Gain*. However, the user may not prefer this route because it does not include any park though it includes 3 museums. With the max aggregation used, the route $v_1 \rightarrow v_3 \rightarrow v_5$ has the highest *Gain* by including one top scored museum and one top scored park, but this route does not maximally use the entire budget available. Intuitively, the sum aggregation is "quantity minded" but ignores variety, whereas the max aggregation is the opposite; neither models a proper trade-off between quantity and variety as the user considered. The above user more prefers the route $v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_5$ that visits multiple highly scored museums and parks, which will better address both quantity and variety.

Solving the top-k route search problem faces two challenges.

Challenge I. One challenge is to design a general enough Φ_h that includes a large class of aggregation functions to model a *personalized* route diversity requirement where each user has her own quantity and variety trade-off. Our approach is treating the satisfaction by visiting each POI as the marginal utility and modeling the aggregation of such utilities of POIs with the diminishing marginal utility property by *submodular* set functions Φ_h. The intuition is that, as the user visits more POIs of the same, her marginal gain from such visits decreases gradually. Submodularity has been used for modeling user behaviors in many real world problems [14][11]. To the best of our knowledge, modeling user's diversity requirement on a route by submodularity has not been done previously.

Challenge II. The top-k route problem is NP-hard as it subsumes the NP-hard orienteering problem [4]. However, users typically demand the routes not only be in high quality, even optimal, but also be generated in a timely manner (seconds to minutes). Fortunately, the users' preferences and constraints on desired routes provide new opportunities to reduce the search space and find optimal top-k routes with fast responses. For example, for a user with only 6-hour time budget and preferring museums and parks on a route, all the POIs in other types or beyond the 6 hours limit will be irrelevant. The key of an exact algorithm is to prune, as early as possible, such irrelevant POIs as well as the routes that are unpromising to make into the top-k list due to a low gain $Gain(\mathcal{P}_V)$. However, this task is complicated by the incorporation of a generic submodular aggregation function Φ_h motivated above in our objective $Gain(\mathcal{P}_V)$, and designing a tight *upper bounding* strategy on $Gain(\mathcal{P}_V)$ for pruning unpromising routes is a major challenge.

Contributions. The main contributions of this paper are:

- We define the top-k route search problem with a new personalized route diversity requirement where the user can choose any submodular function Φ_h to model her desired level of diminishing return. As an instantiation, we show that the family of power-law functions is a sub-family of submodular functions and can model a spectrum of personalized diversity requirement. (Section 3)
- Our first step towards an efficient solution is to eliminate irrelevant POIs for a query, by proposing a novel structure for indexing the POI map on both features and travel costs. This index reduces the POI map to a small set of POIs for a query.(Section 4)
- Our second step towards an efficient solution is to prune unpromising routes, by proposing a novel optimal algorithm, PACER. The novelties of the algorithm include an elegant route enumeration strategy for a compact representation of search space and the reuse of computed results, a cost-based pruning for eliminating non-optimal routes, and a gain-based upper bound strategy for pruning routes that cannot make into the top-k list. The algorithm works for *any* submodular function Φ_h. (Section 5)
- To deal with the looser query constraints, we present two heuristic algorithms with a good efficiency-accuracy trade-off, by finding a good solution with far smaller search spaces. (Section 6)
- We evaluate our algorithms on real-world data sets. PACER provides optimal answers while being orders of magnitude faster than the baselines. The heuristic algorithms provide answers of a competitive quality and work efficiently for a larger POI map and/or a looser query constraint. (Section 7)

2 RELATED WORK

Route recommendation/planning that suggests a POI sequence or a path is related to our work. Works like [15] [6] learn from historical travel behaviors and recommend routes by either sequentially predicting the next location via a Markov model or globally constructing a route. These works rely on users' historical visit data, thus, cannot be applied to a new user with no visit data or a user with dynamically changed preferences. [1] interactively plans a route based on user feedback at each step. Our approach does not rely on user's previous visit data or interactive feedback, and works for any users by modeling the preferences through a query.

Several works recommend a route by maximizing user satisfaction under certain constraints. [8] assumes that each POI has a single type and searches for a route with POIs following a predetermined order of types. [25] allows the user to specify a minimum number of POI types, instead of exact types, in a route. [19] estimates temporal-based user preferences. [17] focuses on modeling the queuing time on POIs. [3] constructs an optimal route covering user-specified categories associated with locations. None of them considers a general route diversity requirement for modeling user's quantity and variety trade-off.

[24], perhaps most related to our work, adopts a keyword coverage function to measure the degree to which query keywords are covered by a route, similar to ours. Their pruning strategies are designed specifically for their specific keyword coverage function; thus, does not address the personalized route diversity requirement, where a different submodular function may be required. Our pruning strategies apply to any submodular function Φ_h. Finally [24] produces a single route, and its performance is only "2-3 times faster than the brute-force algorithm", as pointed in [24].

Less related to our work is the next POI recommendation [26] that aims to recommend the POI to be visited next, and the travel package recommendation [18] that aims to recommend a set of POIs. They are quite different from our goal of finding a route as a sequence of featured POIs. Trajectory search either retrieves *existing* (segments of) trajectories that match certain similarity query [27] from a database, or constructs a route based on the retrieved trajectories [5]. These works assume the existence of a trajectory database, instead of a POI map for route construction.

Table 1: Nomenclature

Notation	Interpretation
$\mathbf{F} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times \lvert\mathcal{H}\rvert}$	POI-feature matrix **F** with POI set $\mathcal{V}$ and feature set $\mathcal{H}$
$\mathbf{F}_{i,h}$	the rating on feature $h \in \mathcal{H}$ for POI $i \in \mathcal{V}$
s_i	staying cost on POI i
$t_{i,j}$	the traveling cost on edge $e_{i,j} \in \mathcal{E}$
$T_{i,j}$	the least traveling cost from any POI i to any POI j
$\mathcal{P}$, $\mathcal{P}_V$	route $\mathcal{P}$ with the included POI set $\mathcal{P}_V$
$Q =$ $(x, y, b,$ $\mathbf{w}, \boldsymbol{\theta}, \Phi)$	user query with parameters: x and y – source and destination location b – travel cost budget $\mathbf{w} \in \mathbb{R}^{\lvert\mathcal{H}\rvert}$ – feature preference vector $\boldsymbol{\theta} \in \mathbb{R}^{\lvert\mathcal{H}\rvert}$ – filtering vector on feature ratings Φ – feature aggregation functions
$\mathcal{V}_Q$, n	POI candidates set $\mathcal{V}_Q$ retrieved by Q with its size n
$\tilde{\mathbf{F}}_{i,h}$	$\mathbf{F}_{i,h}$ after filtered by $\boldsymbol{\theta}$
$Gain(\mathcal{P}_V, Q)$	*gain* of a route $\mathcal{P}$ given query Q

The classical Orienteering Problem (OP), such as [4], studied in operational research on theoretical level, finds a path, limited in length, that visits some nodes and maximizes a global reward collected from the nodes on the path. No POI feature or route diversity requirement is considered in OP.

3 PRELIMINARY

Table 1 summarizes the notations frequently used throughout the paper. The variables in **bold-face** are vectors or matrices.

3.1 Problem Statement

Definition 1. ***[A POI Map]*** *A POI map $G = (\mathcal{V}, \mathcal{E})$ is a directed/undirected and connected graph, where $\mathcal{V}$ is a set of geo-tagged POI nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges between nodes (i, j), $i, j \in \mathcal{V}$. $\mathcal{H}$ is a set of features on POIs. $\mathbf{F} \in \mathbb{R}^{\lvert\mathcal{V}\rvert \times \lvert\mathcal{H}\rvert}$ denotes the POI-feature matrix, where $\mathbf{F}_{i,h} \in [0, \beta]$ is the rating on a feature h for the POI i. Each POI $i \in \mathcal{V}$ is associated with a staying cost s_i. Each edge $e_{i,j} \in \mathcal{E}$ has a travel cost $t_{i,j}$.* □

The choices of s_i and $t_{i,j}$ can be time, expenses, or other costs.

Definition 2. ***[Routes]*** *A route $\mathcal{P}$ is a path $x \longrightarrow \cdots i \cdots \longrightarrow y$ in G from the origin x to the destination y through a sequence of non-repeating POIs i except possibly $x = y$. $\mathcal{P}_V$ denotes the set of POIs on $\mathcal{P}$. $T_{i,j}$ denotes the least traveling cost from i to the next visited j, where i, j are not necessarily adjacent in G. The cost of $\mathcal{P}$ is*

$$cost(\mathcal{P}) = \sum\nolimits_{i \in \mathcal{P}_V} s_i + \sum\nolimits_{i \to j \in \mathcal{P}} T_{i,j}. \quad \square \qquad (1)$$

A route $\mathcal{P}$ includes only the POIs i that the user actually "visits" by staying at i with $s_i > 0$. Each $i \longrightarrow j$ on a route is a path from i to j with the least traveling cost $T_{i,j}$. The intermediate POIs between i, j on path $i \longrightarrow j$ are not included in $\mathcal{P}$. The staying times at x and/or y can be either considered or ignored depending on the user choice. The latter case can be modeled by setting $s_x = s_y = 0$.

At the minimum, the user has an origin x and a destination y for a route, not necessarily distinct, and a budget b on the cost of the route. In addition, the user may want the POIs to have certain features specified by a $\lvert\mathcal{H}\rvert$-dimensional weight vector $\mathbf{w}$ with each element $\mathbf{w}_h \in [0, 1]$ and $\Sigma_h \mathbf{w}_h = 1$. The user can also specify a filtering vector $\boldsymbol{\theta}$ so that $\mathbf{F}_{i,h}$ is set to 0 if it is less than $\boldsymbol{\theta}_h$. $\tilde{\mathbf{F}}_{i,h}$ denotes $\mathbf{F}_{i,h}$ after this filtering. Finally, the user may specify a route diversity requirement through a feature aggregation function vector Φ, with Φ_h for each feature h. $\Phi_h(\mathcal{P}_V)$ aggregates the rating on feature h over the POIs in $\mathcal{P}_V$. See more details in Section 3.2.

Definition 3. ***[Query and Gain]*** *A query Q is a 6-tuple $(x, y, b, \mathbf{w}, \boldsymbol{\theta}, \Phi)$. A route $\mathcal{P}$ is* valid *if $cost(\mathcal{P}) \le b$. The* gain *of $\mathcal{P}$ w.r.t. Q is*

$$Gain(\mathcal{P}_V, Q) = \sum\nolimits_h \mathbf{w}_h \Phi_h(\mathcal{P}_V). \quad \square \qquad (2)$$

Note that only the specification of x, y, b is required; if the specification of $\mathbf{w}, \boldsymbol{\theta}, \Phi$ is not provided by a user, their default choices can be used, or can be learned from users' travel records if such data are available (not the focus of this paper). $Gain(\mathcal{P}_V, Q)$ is a set function and all routes $\mathcal{P}$ that differ only in the order of POIs have the same $Gain$, and the order of POIs affects only $cost(\mathcal{P})$.

[Top-k route search problem] Given a query Q and an integer k, the goal is to find k valid routes $\mathcal{P}$ that have different POI sets $\mathcal{P}_V$ (among all routes having the same $\mathcal{P}_V$, we consider only the route with the smallest $cost(\mathcal{P})$) and the highest $Gain(\mathcal{P}_V, Q)$ (if ties, ranked by $cost(\mathcal{P})$). The k routes are denoted by $topK$.

In the rest of the paper, we use $Gain(\mathcal{P}_V)$ for $Gain(\mathcal{P}_V, Q)$.

3.2 Modeling Route Diversity Requirement

To address the personalized route diversity requirement, we consider a submodular Φ_h to model the diminishing marginal utility as more POIs with feature h are added to a route. A set function $f : 2^V \longrightarrow \mathbb{R}$ is *submodular* if for every $X \subseteq Y \subseteq V$ and $v \in V \setminus Y$, $f(X \cup \{v\}) - f(X) \ge f(Y \cup \{v\}) - f(Y)$, and is *monotone* if for every $X \subseteq Y \subseteq V$, $f(X) \le f(Y)$. The next theorem follows from [13] and the fact that $Gain(\mathcal{P}_V)$ is a nonnegative linear combination of Φ_h.

Theorem 1. *If for every feature h, $\Phi_h(\mathcal{P}_V)$ is nonnegative, monotone and submodular, so is $Gain(\mathcal{P}_V)$.*

We aim to provide a general solution to the top-k route search problem for any nonnegative, monotone and submodular Φ_h, which model various personalized route diversity requirement. To illustrate the modeling power of such Φ_h, for example, consider Φ_h defined by the power law function

$$\Phi_h(\mathcal{P}_V) = \sum\nolimits_{i \in \mathcal{P}_V} R_h(i)^{-\boldsymbol{\alpha}_h} \tilde{\mathbf{F}}_{i,h}, \qquad (3)$$

where $R_h(i)$ is the rank of POI i on the rating of feature h among all the POIs in $\mathcal{P}_V$ (the largest value ranks the first), rather than the order that i is added to $\mathcal{P}$, and $\boldsymbol{\alpha}_h \in [0, +\infty)$ is the power law exponent for feature h. $R_h(i)^{-\boldsymbol{\alpha}_h}$ is non-increasing as $R_h(i)$ increases. For a sample route $\mathcal{P} = A(3) \to B(5)$ with the ratings of feature h for each POI in the brackets, and $\boldsymbol{\alpha}_h = 1$, the ranks for A and B on h are $R_h(A) = 2$ and $R_h(B) = 1$. Thus, $\Phi_h(\mathcal{P}_V) = 2^{-1} \times 3 + 1^{-1} \times 5$, with a diminishing factor 2^{-1} for the secondly ranked A. If we use a larger $\boldsymbol{\alpha}_h = 2$, $\Phi_h(\mathcal{P}_V) = 2^{-2} \times 3 + 1^{-2} \times 5$ has a larger diminishing factor for A.

In general, a larger $\boldsymbol{\alpha}_h$ means a faster diminishing factor for the ratings $\tilde{\mathbf{F}}_{i,h}$ on the recurrent feature, i.e., a diminishing marginal value on h. Note that the sum aggregation ($\boldsymbol{\alpha}_h = 0$) and the max aggregation ($\boldsymbol{\alpha}_h = \infty$) are the special cases. Hence, Eqn. (3) supports a spectrum of diversity requirement through the setting of $\boldsymbol{\alpha}_h$.

Note that $R_h(i)$ for an existing POI i may decrease when a new POI j is added to $\mathcal{P}$, so it is incorrect to compute the new $\Phi_h(\mathcal{P}_V)$ by simply adding the marginal brought by j to existing value of $\Phi_h(\mathcal{P}_V)$. For ease of presentation, we assume $\boldsymbol{\alpha}_h$ has same value for all h and use $\boldsymbol{\alpha}$ for $\boldsymbol{\alpha}_h$ in the rest of the paper.

THEOREM 2. *$\Phi_h(\mathcal{P}_V)$ defined in Eqn. (3) is nonnegative, monotone and submodular.*

PROOF. The nonnegativity and monotonicity of $\Phi_h(\mathcal{P}_V)$ in Eqn. (3) is straightforward. For its submodularity, we omit the mathematical proof due to limited space and only present an intuitive idea. Let X and Y be the set of POIs in two routes, $X \subseteq Y$. Intuitively, for every $i \in X$, there must be $i \in Y$ and i's rank in X is not lower than that in Y. Let $v \in \mathcal{V} \setminus Y$ so that $X' = X \cup \{v\}$ and $Y' = Y \cup \{v\}$. Similarly, v' rank in X' is not lower than that in Y', thus the increment brought by v to X is not less than that to Y, which means $\Phi_h(X') - \Phi_h(X) \geq \Phi_h(Y') - \Phi_h(Y)$. Hence, it is submodular. □

The user can also personalize her diversity requirement by specifying any other submodular Φ_h, such as a log utility function $\Phi_h(\mathcal{P}_V) = log(1+\sum_{i\in\mathcal{P}_V} \tilde{\mathbf{F}}_{i,h})$ and the coverage function $\Phi_h(\mathcal{P}_V) = 1-\prod_{i\in\mathcal{P}_V}[1-\tilde{\mathbf{F}}_{i,h}]$. Our approach only depends on the submodularity of Φ_h, but is independent of the exact choices of such functions.

Our problem subsumes two NP-hard problems, i.e., the submodular maximization problem [13] and the orienteering problem [4].

3.3 Framework Overview

To efficiently deal with the high computational complexity of this problem, we divide the overall framework into the offline component and the online component. Before processing any query, the offline component carefully indexes the POI map on feature and cost dimensions for speeding up future POI selection and travel cost computation. The online component responds to the user query Q with *Sub-index Retrieval* that extracts the sub-indices relevant to Q, and *Routes Search* that finds the top-k routes using the sub-indices. For routes search, as motivated in Section 1, we consider both the exact algorithm with novel pruning strategies, and heuristic algorithms to deal with the worst case of less constrained Q.

We first consider an indexing strategy in Section 4, and then consider routes search algorithms in Section 5 and Section 6.

4 INDEXING

In this section, we explain the offline indexing component and the Sub-index Retrieval of the online component.

4.1 Offline Index Building

The POI map data is stored on disk. To answer user queries rapidly with low I/O access and speed up travel cost computation, we build two indices, FI and HI stored on disk.

FI is an inverted index mapping each feature h to a list of POIs having non-zero rating on h. An entry $(v_i, \mathbf{F}_{i,h})$ indicates the feature rating $\mathbf{F}_{i,h}$ for POI v_i, sorted in descending order of $\mathbf{F}_{i,h}$. **FI** helps retrieving the POIs related to the features specified by a query.

The least traveling cost $T_{i,j}$ between two arbitrary POIs i and j is frequently required in the online component. To compute $T_{i,j}$ efficiently, we employ the 2-hop labeling [9] for point-to-point shortest distance querying on weighted graphs. [9] shows scalable results for finding 2-hop labels for both unweighted and weighted graphs, and the constructed labels return *exact* shortest distance queries. Our HI index is built using the 2-hop labeling method.

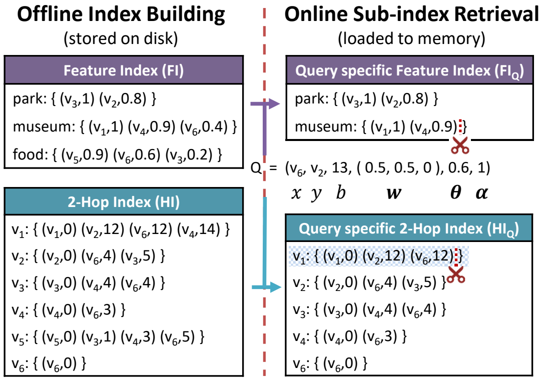

Offline Index Building
(stored on disk)
Online Sub-index Retrieval
(loaded to memory)
Feature Index (FI)
park: { (v_3,1) (v_2,0.8) }
museum: { (v_1,1) (v_4,0.9) (v_6,0.4) }
food: { (v_5,0.9) (v_6,0.6) (v_3,0.2) }
Query specific Feature Index (FI_Q)
park: { (v_3,1) (v_2,0.8) }
museum: { (v_1,1) (v_4,0.9) }
Q = (v_6, v_2, 13, (0.5, 0.5, 0), 0.6, 1)
x y b $\mathbf{w}$ $\boldsymbol{\theta}$ $\boldsymbol{\alpha}$
2-Hop Index (HI)
v_1: { (v_1,0) (v_2,12) (v_6,12) (v_4,14) }
v_2: { (v_2,0) (v_6,4) (v_3,5) }
v_3: { (v_3,0) (v_4,4) (v_6,4) }
v_4: { (v_4,0) (v_6,3) }
v_5: { (v_5,0) (v_3,1) (v_4,3) (v_6,5) }
v_6: { (v_6,0) }
Query specific 2-Hop Index (HI_Q)
v_1: { (v_1,0) (v_2,12) (v_6,12) }
v_2: { (v_2,0) (v_6,4) (v_3,5) }
v_3: { (v_3,0) (v_4,4) (v_6,4) }
v_4: { (v_4,0) (v_6,3) }
v_6: { (v_6,0) }

Figure 2: Left part: FI and HI built from the POI map in Figure 1. Right Part: Given a query Q, retrieve POI candidates $\mathcal{V}_Q$ by retrieving the subindices FI_Q and HI_Q from FI and HI.

HI. For an undirected graph, there is one list of labels for each node v_i, where each label (u, d) contains a node $u \in \mathcal{V}$, called *pivot*, and the least traveling cost d between v_i and u. $HI(v_i)$ denotes the list of labels for v_i, sorted in the ascending order of d. According to [9], $T_{i,j}$ between v_i and v_j is computed by

$$T_{i,j} = \min_{(u,d_1)\in HI(v_i) \cap (u,d_2)\in HI(v_j)} (d_1 + d_2). \tag{4}$$

Figure 2 (left part) shows the FI and HI for the POI map in Figure 1. For example, to compute $T_{2,5}$, we search for the common pivot nodes u from the pivot label lists of v_2 and v_5 and find that v_3 minimizes the traveling cost between v_2 and v_5, so $T_{2,5} = 5 + 1 = 6$.

In the case of a directed graph, each POI v_i will have two lists of labels in HI, $HI(v_i^{out})$ for v_i as the source, and $HI(v_i^{in})$ for v_i as the destination. And we simply replace v_i with v_i^{out} and v_j with v_j^{in} in Eqn. (4) to compute $T_{i,j}$.

4.2 Online Sub-index Retrieval

Given a query Q, the first thing is to retrieve the POI candidates $\mathcal{V}_Q$ that are likely to be used in the routes search part. In particular, the POIs that do not contain any feature in the preference vector $\mathbf{w}$ or do not pass any threshold in $\boldsymbol{\theta}$ will never be used, nor the ones that cannot be visited on the way from the source x to the destination y within the budget b. This is implemented by retrieving the query specific sub-indices FI_Q from FI and HI_Q from HI.

Figure 2 (right part) shows how the retrieval works for a query $Q = (x = v_6, y = v_2, b = 13, \mathbf{w} = (0.5, 0.5, 0), \boldsymbol{\theta} = 0.6, \boldsymbol{\alpha} = 1)$, where the weights in $\mathbf{w}$ are for (Park, Museum, Food), and $\boldsymbol{\alpha}$ is the power law exponent in Eqn. (3). Here the elements in each vector $\boldsymbol{\theta}$ and $\boldsymbol{\alpha}$ have the same value for all features.

FI_Q, a sub-index of FI, is retrieved using $\mathbf{w}$ and $\boldsymbol{\theta}$. $\mathbf{w}$ directly locates the lists for the user preferred (with $\mathbf{w}_h > 0$) features. $\boldsymbol{\theta}$ is used to cut off lower rated POIs on the sorted lists indicated by red scissors. $\mathcal{V}_Q = \{v_1, v_2, v_3, v_4\}$ contains the remaining POIs.

$\mathbf{HI_Q}$, a sub-index of HI, is then formed by retrieving the lists for each POI in $\mathcal{V}_Q$ and also those for x and y, and b is used to cut off the sorted lists, indicated by red scissors. We also check whether a POI i in current $\mathcal{V}_Q$ is actually reachable by checking the single-point visit cost: if $s_x + T_{x,i} + s_i + T_{i,y} + s_y > b$, we remove i from $\mathcal{V}_Q$ and remove its list from $\mathrm{HI_Q}$, as indicated by the blue shading. Then we get the final POI candidates $\mathcal{V}_Q$. Typically, $|\mathcal{V}_Q| \ll |\mathcal{V}|$.

$\mathbf{FI_Q}$ and $\mathbf{HI_Q}$ are retrieved only once and kept in memory.

5 OPTIMAL ROUTES SEARCH

With POI candidate set $\mathcal{V}_Q$ and the sub-indices extracted, the next step is the Routes Search phase. We present an optimal routes search algorithm in this section. Considering the complexity and generality of the problem, a standard tree search or a traditional algorithm for the orienteering problem does not work. An ideal algorithm design should meet the following goals: i. search all promising routes in a smart manner without any redundancy; ii. prune unpromising routes as aggressively as possible while preserving the optimality of the top-k answers; iii. ensure that the search and pruning strategies are applicable to any nonnegative, monotone and submodular aggregation functions Φ_h. To this end, we propose a novel algorithm, **P**refix b**A**sed **C**ompact stat**E**s g**R**owth (**PACER**), that incorporates the idea of dynamic programming and fuses a cost-based pruning strategy and a gain-based pruning strategy in an unified way.

Next, we present our enumeration and pruning strategies, followed by the detailed algorithm and the complexity analysis.

5.1 Prefix-based Compact State Growth

A route $\mathcal{P}$ is associated with several variables: $\mathcal{P}_V$, $Gain(\mathcal{P}_V)$, the ending POI $end(\mathcal{P})$, and $cost(\mathcal{P})$. If x is not visited, s_x and $\tilde{\mathbf{F}}_{x,h}$ for every h are set to 0; the same is applied to y. A POI sequence is an *open route* if it starts from x and visits several POIs other than y; it is a *closed route* if it starts from x and ends at y. The initial open route includes only x. An open route $\mathcal{P}$ is *feasible* if its closed form $\mathcal{P} \rightarrow y$ satisfies $cost(\mathcal{P} \rightarrow y) \leq b$. In the following discussion, $\mathcal{P}$ denotes either an open route or a closed route. An open route $\mathcal{P}^-$ with $end(\mathcal{P}^-) = i$ can be extended into a longer open route $\mathcal{P} = \mathcal{P}^- \rightarrow j$ by a POI $j \notin \mathcal{P}^-_V \cup \{y\}$. The variables for $\mathcal{P}$ are

$$\begin{cases} \mathcal{P}_V = \mathcal{P}^-_V \cup \{j\} \\ Gain(\mathcal{P}_V) = \sum_h \mathbf{w}_h \Phi_h(\mathcal{P}_V) \\ end(\mathcal{P}) = j \\ cost(\mathcal{P}) = cost(\mathcal{P}^-) + T_{i,j} + s_j. \end{cases} \tag{5}$$

Compact states $\mathbb{C}$. $\mathcal{P}_V$ and $Gain(\mathcal{P}_V)$ depend on the POI set of the route $\mathcal{P}$ but are independent of how the POIs are ordered. Hence, we group all open routes sharing the same $\mathcal{P}_V$ as a *compact state*, denoted as $\mathbb{C}$, and let $\mathbb{C}_L$ denote the list of open routes having $\mathbb{C}$ as the POI set. $\mathbb{C}$ is associated with the following fields:

$$\begin{cases} Gain(\mathbb{C}) : \text{the gain of routes grouped by } \mathbb{C} \\ \mathbb{C}_L : \forall \mathcal{P} \in \mathbb{C}_L, end(\mathcal{P}), cost(\mathcal{P}). \end{cases} \tag{6}$$

These information is cached in a hash map with $\mathbb{C}$ as the key.

We assume that the POIs in $\mathcal{V}_Q$ are arranged in the lexicographical order of POI IDs. The compact states are enumerated as the subsets of $\mathcal{V}_Q$. x is included in every compact state, so we omit x.

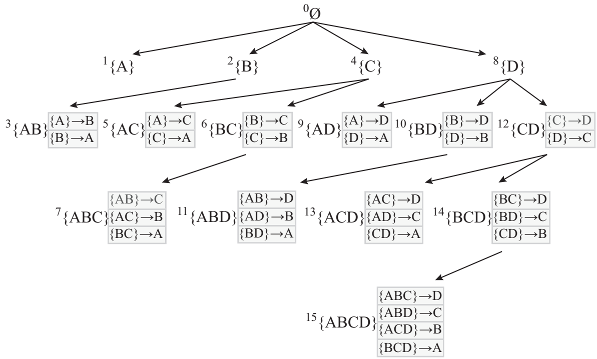

0 Ø
1 {A}
2 {B}
4 {C}
8 {D}
3 {AB} {A}→B {B}→A
5 {AC} {A}→C {C}→A
6 {BC} {B}→C {C}→B
9 {AD} {A}→D {D}→A
10 {BD} {B}→D {D}→B
12 {CD} {C}→D {D}→C
7 {ABC} {AB}→C {AC}→B {BC}→A
11 {ABD} {AB}→D {AD}→B {BD}→A
13 {ACD} {AC}→D {AD}→C {CD}→A
14 {BCD} {BC}→D {BD}→C {CD}→B
15 {ABCD} {ABC}→D {ABD}→C {ACD}→B {BCD}→A

Figure 3: The compact state enumeration tree for PACER. The number indicates the order of enumeration.

Figure 3 shows a compact state enumeration tree for $\mathcal{V}_Q = \{A, B, C, D\}$, excluding x and y. Each capital letter represents a POI, each node represents a compact state. We define the set of POIs that precede i, in the above order, in a POI set as the *prefix* of a POI i, e.g., prefix of C is $\{AB\}$. The compact states are generated in a specific *prefix-first depth-first* manner so that longer open routes are extended from earlier computed shorter ones. Initially, the root is the empty set $\emptyset$. A child node $\mathbb{C}$ of the current node $\mathbb{C}^-$ is generated by appending a POI i that precedes any POIs in $\mathbb{C}^-$ to the front of $\mathbb{C}^-$, and all child nodes are arranged by the order of i. For example, Node 7 $\{ABC\}$ is generated as a child node of Node 6 $\{BC\}$ by appending A to the front of $\{BC\}$ because A precedes B and C.

At node $\mathbb{C}$, the open routes in $\mathbb{C}_L$ are generated by extending the cached routes in every compact state $\mathbb{C}^{-j} = \mathbb{C} \setminus \{j\}$ where $j \in \mathbb{C}$. There are $|\mathbb{C}|$ such $\mathbb{C}^{-j}$. We generate each route $\mathcal{P} = \mathcal{P}^- \rightarrow j$ by selecting the routes $\mathcal{P}^-$ from each $\mathbb{C}^{-j}_L$ and append j at the end, and compute the gain and cost of $\mathcal{P}$ based on the accessed information for $\mathbb{C}^{-j}$ from the hash map. $\mathcal{P}$ is kept in $\mathbb{C}_L$ if it is feasible.

For example, to generate the open routes at the node $\{ABC\}$, we access the cached open routes at nodes $\{AB\}$, $\{AC\}$ and $\{BC\}$ and append the missing POI. $\{AB\} \rightarrow C$ represents all the open routes ended with C and the first two POIs in any order, i.e., $x \rightarrow A \rightarrow B \rightarrow C$ and $x \rightarrow B \rightarrow A \rightarrow C$. Note that it materializes only the current expanded branch of the tree, instead of the entire tree.

A closed route $\mathcal{P} \rightarrow y$ for each $\mathcal{P} \in \mathbb{C}_L$ is used to update the top-k routes $topK$. If $\mathbb{C}_L$ is empty, this compact state is not kept. If no compact state is expandable, we stops the enumeration and yield the final $topK$. Note that each $\mathbb{C}_L$ can include $|\mathbb{C}|!$ open routes and enumerating all the routes can be very expensive. We present two strategies to prune unpromising routes.

5.2 Cost-based Pruning Strategy

Consider two feasible open routes $\mathcal{P}$ and $\mathcal{P}'$. We say $\mathcal{P}$ *dominates* $\mathcal{P}'$ if $\mathcal{P}_V = \mathcal{P}'_V$, $end(\mathcal{P}) = end(\mathcal{P}')$ and $cost(\mathcal{P}) \leq cost(\mathcal{P}')$. Because, if a route $\mathcal{P}' \rightarrow \hat{\mathcal{P}}$ is feasible, $\mathcal{P} \rightarrow \hat{\mathcal{P}}$ with the same extension $\hat{\mathcal{P}}$ must be also feasible and $cost(\mathcal{P} \rightarrow \hat{\mathcal{P}}) \leq cost(\mathcal{P}' \rightarrow \hat{\mathcal{P}})$.

Pruning-1: cost dominance pruning. Leveraging the above dominance relationship, at the compact state $\mathbb{C}$, when generating $\mathcal{P} = \mathcal{P}^- \rightarrow j$ for a given j, we only select the open route $\mathcal{P}^-$ from $\mathbb{C}^{-j}_L$ such that $\mathcal{P}$ is feasible and has the least cost, thus, dominates all other routes $\mathcal{P}'^- \rightarrow j$ with $\mathcal{P}'^-$ from $\mathbb{C}^{-j}_L$. This reduces $|\mathbb{C}|!$

open routes to at most $|\mathbb{C}|$ dominating open routes at the compact state $\mathbb{C}$, one for each j in $\mathbb{C}$, without affecting the optimality. We call this strategy *cost dominance pruning*.

For example, $\{AB\} \to C$ on node 7 $\{ABC\}$ now represents only one open route with the least cost chosen from $A \to B \to C$ and $B \to A \to C$. Note that Pruning-1 is a subtree pruning, e.g., if $A \to B$ on node 3 is pruned, all the open routes starting with $A \to B$, such as $A \to B \to C$ on node 7 and $A \to B \to D$ on node 11, will never be considered.

Though all dominated open routes are pruned, many of the remaining dominating open routes are still unpromising to lead to the top-k closed routes. This further motivates our next strategy.

5.3 Gain-based Pruning Strategy

We can extend a dominating open route $\mathcal{P}$ step by step using the remaining budget $\Delta b = b - cost(\mathcal{P})$ into a closed route $\mathcal{P} \to \hat{\mathcal{P}}$. The POIs used for extension at each step should be reachable from the current $end(\mathcal{P})$, therefore, chosen from the set

$$\mathcal{U} = \{i | T_{end(\mathcal{P}), i} + s_i + T_{i,y} + s_y \leq \Delta b\}, \tag{7}$$

where i is an unvisited POI other than y. $T_{end(\mathcal{P}),i}$ and $T_{i,y}$ can be computed through HI_Q. $\mathcal{P} \to \hat{\mathcal{P}}$ has gain $Gain(\mathcal{P}_V \cup \hat{\mathcal{P}}_V)$. Then the *marginal gain* by concatenating $\hat{\mathcal{P}}$ to the existing $\mathcal{P}$ is

$$\Delta Gain(\hat{\mathcal{P}}_V | \mathcal{P}_V) = Gain(\mathcal{P}_V \cup \hat{\mathcal{P}}_V) - Gain(\mathcal{P}_V). \tag{8}$$

Let $\mathcal{P} \to \hat{\mathcal{P}}^*$ denote the $\mathcal{P} \to \hat{\mathcal{P}}$ with the highest gain. If $\mathcal{P} \to \hat{\mathcal{P}}^*$ ranks lower than the current k-th top routes $topK[k]$, $\mathcal{P}$ is not promising and all the open routes extended from $\mathcal{P}$ can be pruned.

Pruning-2: marginal gain upper bound pruning. However, finding $\hat{\mathcal{P}}^*$ is as hard as finding an optimal route from scratch, so we seek to estimate an *upper bound UP* of the marginal gain $\Delta Gain(\hat{\mathcal{P}}_V | \mathcal{P}_V)$, such that if $Gain(\mathcal{P}_V) + UP$ is less than the gain of $topK[k]$, $\mathcal{P}$ is not promising, thus, $\mathcal{P}$ and all its extensions can be pruned without affecting the optimality. We call this *marginal gain upper bound pruning*. As more routes are enumerated, the gain of $topK[k]$ increases and this pruning becomes more powerful.

The challenge of estimating UP is to estimate the cost of the extended part $\hat{\mathcal{P}}$ without knowing the order of the POIs. Because $\Delta Gain(\hat{\mathcal{P}}_V | \mathcal{P}_V)$ is independent of the POIs' order, we can ignore the order and approximate the "route cost" by a "set cost", i.e., the sum of some cost $c(i)$ of each POI $i \in \hat{\mathcal{P}}_V$, where $c(i)$ is no larger than i's actual cost when it is included in $\hat{\mathcal{P}}$. We define $c(i)$ as:

$$c(i) = s_i + min(t_{j,i})/2 + min(t_{i,k})/2, \tag{9}$$

where $t_{j,i}$ is the cost on an in-edge $e_{j,i}$ and $t_{i,k}$ is the cost on an out-edge $e_{i,k}$. As the order of POIs is ignored, it is easy to verify that *min* ensures the above property of $c(i)$. The destination y is "one-sided", i.e., $c(y) = s_y + min(t_{j,y})/2$. To make a tighter cost approximation, we also count the half out-edge cost $min(t_{end(\mathcal{P}),k})/2$ for $end(\mathcal{P})$.

Then, UP is exact the solution, i.e., the maximum $\Delta Gain(S^* | \mathcal{P}_V)$, to the following optimization problem:

$$\max_{S \subseteq \mathcal{U} \cup \{y\}} \Delta Gain(S | \mathcal{P}_V) \ s.t. \sum\nolimits_{i \in S} c(i) \leq B, \tag{10}$$

where $\mathcal{U}$ is defined in Eqn. (7) and $B = \Delta b - min(t_{end(\mathcal{P}),k})/2$. Note that S should include y because $end(\hat{\mathcal{P}}) = y$. As $c(i)$ and $c(end(\mathcal{P}))$ are no larger than their actual costs, $\Delta Gain(S^* | \mathcal{P}_V) \geq \Delta Gain(\hat{\mathcal{P}}_V | \mathcal{P}_V)$ for any $\hat{\mathcal{P}}$. Thus, using $\Delta Gain(S^* | \mathcal{P}_V)$ as UP never loses the optimality. To solve Eqn. (10), we first show the properties of the marginal gain function $\Delta Gain$.

Theorem 3. *The marginal gain function $\Delta Gain$ as defined in Eqn. (8) is nonnegative, monotone and submodular.*

Proof. We only show that $\Delta Gain$ is submodular. According to [13], if a set function $g : 2^V \to \mathbb{R}$ is submodular, and $X, Y \subset V$ are disjoint, the *residual* function $f : 2^Y \to \mathbb{R}$ defined as $f(S) = g(X \cup S) - g(X)$ is also submodular. Since *Gain* is submodular (Theorem 1) and since $\mathcal{P}_V, \mathcal{U} \subset \mathcal{V}$ are disjoint, $\Delta Gain(\hat{\mathcal{P}}_V | \mathcal{P}_V) = Gain(\mathcal{P}_V \cup \hat{\mathcal{P}}_V) - Gain(\mathcal{P}_V)$ is residual on $\hat{\mathcal{P}}_V$, thus, is submodular. □

Apparently, Eqn. (10) is a submodular maximization problem subject to a knapsack constraint, which unfortunately is also NP-hard [13]. Computing $\Delta Gain(S^* | \mathcal{P}_V)$ is costly, thus, we consider estimating its upper bound.

One approach, according to [22], is to run a $\Omega(B|\mathcal{U}|^4)$ time (B is defined in Eqn. (10)) greedy algorithm in [12] to obtain an approximate solution $\Delta Gain(S' | \mathcal{P}_V)$ for the above problem with approximation ratio of $1 - e^{-1}$, then the upper bound of $\Delta Gain(S^* | \mathcal{P}_V)$ is achieved by $\Delta Gain(S' | \mathcal{P}_V)/(1 - e^{-1})$. A less costly version of this algorithm runs in $O(B|\mathcal{U}|)$ but its approximation ratio is $\frac{1}{2}(1 - e^{-1})$.

Compared with the above mentioned *offline* bounds, i.e., $1 - e^{-1}$ and $\frac{1}{2}(1 - e^{-1})$ that are stated in advance before running the actual algorithm, the next theorem states that we can instead use the submodularity to acquire a much tighter *online* bound.

Theorem 4. *For each POI $i \in \mathcal{U} \cup \{y\}$, let $\delta_i = \Delta Gain(\{i\} | \mathcal{P}_V)$. Let $r_i = \delta_i / c(i)$, and let $i_1, \cdots, i_m$ be the sequence of these POIs with r_i in decreasing order. Let l be such that $C = \sum_{j=1}^{l-1} c(i_j) \leq B$ and $\sum_{j=1}^{l} c(i_j) > B$. Let $\lambda = (B - C)/c(i_l)$. Then*

$$UP = \sum\nolimits_{j=1}^{l-1} \delta_{i_j} + \lambda \delta_{i_l} \geq \Delta Gain(S^* | \mathcal{P}_V). \tag{11}$$

Proof. [16] showed a theorem that a tight online bound for arbitrary given solution $\hat{\mathcal{A}}$ (obtained using any algorithm) to a constrained submodular maximization problem can be got to measure how far $\hat{\mathcal{A}}$ is from the optimal solution. By applying [16] to the problem in Eqn. (10) and let $\hat{\mathcal{A}} = \emptyset$, Theorem 4 is deduced. □

By this means, UP is computed without running a greedy algorithm. We also empirically proved that this online bound in Eqn. (11) outperforms the offline bounds on both tightness and computational cost. Thus, we finally choose the online bound.

5.4 Algorithm

Algorithm 1 incorporates the above enumeration and pruning strategies. Given the global variables, $\mathtt{PACER}(\mathbb{C}^-, I)$ recursively enumerates the subtree at the current compact state $\mathbb{C}^-$ with the POI set I available for extending $\mathbb{C}^-$, and finally return the k best routes in *topK*. The initial call is $\mathtt{PACER}(\emptyset, \mathcal{V}_Q)$, when only x is included.

As explained in Section 5.1, Line 1 - 3 extends $\mathbb{C}^-$ by each i in the set I in order, creating the child node $\mathbb{C}$ and computing $Gain(\mathbb{C})$. Lines 4 - 11 generate the dominating and promising open routes $\mathbb{C}_L$. Specifically, for each $j \in \mathbb{C}$ selected as the ending POI, Line 5 - 6 find the dominating route $\mathcal{P}^-$ from the previously computed $\mathbb{C}_L^{-j}$.

Algorithm 1: PACER($\mathbb{C}^{-}, I$) (Recursive funcion)

```
Globals     : $Q = (x, y, b, \mathbf{w}, \boldsymbol{\theta}, \Phi)$, $\mathcal{V}_Q$, $\mathsf{FI}_Q$ and $\mathsf{HI}_Q$ to
              compute $Gain(\mathbb{C})$ and $cost(\mathcal{P})$, and $k$
Parameters  : compact state $\mathbb{C}^{-}$ and the set of POIs $I$
Output      : a priority queue $topK$
1  forall $POI\ i\ in\ set\ I\ in\ order$ do
2      $\mathbb{C} \leftarrow \{i\} \cup \mathbb{C}^{-}$;
3      compute $Gain(\mathbb{C})$;
4      forall $POI\ j\ in\ \mathbb{C}$ do
5          $\mathbb{C}^{-j} \leftarrow \mathbb{C} \setminus \{j\}$;
6          $\mathcal{P}^{-} \leftarrow$ the dominating route in $\mathbb{C}_L^{-j}$ such that
             $cost(\mathcal{P}^{-} \rightarrow j)$ is minimum; // prune-1
7          $\mathcal{P} \leftarrow \mathcal{P}^{-} \rightarrow j$;
8          if $cost(\mathcal{P} \rightarrow y) \leq b$ then
9              Compute $UP$ using Eqn. (11);
10             if $Gain(\mathbb{C}) + UP \geq Gain(topK[k])$ then
11                 insert route $\mathcal{P}$ into $\mathbb{C}_L$; // prune-2
12     UpdateTopK($\mathbb{C}_L$, $topK$);
13     PACER($\mathbb{C}$, prefix of $i$ in $I$);
```

This corresponds to Pruning-1. Only when the new open route $\mathcal{P}$ is feasible, Pruning-2 is applied to check if $\mathcal{P}$ has a promising gain, and if so, $\mathcal{P}$ is inserted into $\mathbb{C}_L$ (Lines 8 - 11). After $\mathbb{C}_L$ is finalized, it selects an open route $\mathcal{P}$ in $\mathbb{C}_L$ such that $\mathcal{P} \rightarrow y$ has the least cost to update *topK* (Line 12). The information of the new compact state $\mathbb{C}$, as in Eqn. (6), is added to the hash map. At last, $\mathbb{C}$ is extended recursively with the POIs in the prefix of i in current I (Line 13).

Summary of the properties of PACER. (1) PACER works for **any** nonnegative, monotone, and submodular *Gain* function so as to deal with the personalized diversity requirement. (2) Open routes are enumerated as compact states in a prefix-first depth-first order to construct open routes incrementally, i.e., **dynamic programming**. (3) With **Pruning-1**, we compute at most $|\mathbb{C}|$ dominating feasible open routes at each compact state $\mathbb{C}$, instead of $|\mathbb{C}|!$ routes. (4) **Pruning-2** further weeds out the dominating feasible open routes not having a promising estimated maximum gain.

5.5 Complexity Analysis

We measure the computational complexity by the number of routes examined. Two main factors affecting this measure are the size of the POI candidate set, i.e., $|\mathcal{V}_Q|$ denoted by n, and the maximum length of routes examined (excluding x and y), i.e., the maximum $|\mathcal{P}|$ denoted by p. $p \ll n$. We analyze PACER relatively to the brute-force search and a state-of-the-art approximation solution.

PACER. The compact states on the l-th level of the enumeration tree (Figure 3) compute the routes containing l POIs; thus, there are at most $\binom{n}{l}$ compact states on level l. And thanks to Pruning-1, each compact state represents at most l dominating open routes. There are n dominating open routes with single POI on level $l = 1$. Starting from $l = 2$, to generate each dominating open route on level l, we need to examine $(l-1)$ sub-routes having the same set of POIs as the prefix and add the same ending to find the dominating one, according to the cost dominance pruning. Hence, with $p \ll n$ and the Pascal's rule [2], the number of routes examined is at most

$$n + \sum_{l=2}^{p} l(l-1)\binom{n}{l} = n + n(n-1)\sum_{l=2}^{p}\binom{n-2}{l-2} \approx n(n-1)(\binom{n-2}{p-2} + \binom{n-2}{p-3}) = n(n-1)\binom{n-1}{p-2} = \frac{n-1}{(n-p+1)(p-2)!}\frac{n!}{(n-p)!}. \tag{12}$$

Therefore, the computation cost of PACER is $O(\frac{1}{(p-2)!}\frac{n!}{(n-p)!})$ with $p \ll n$. If Pruning-2 is also enabled and it prunes the γ percent of the routes examined by PACER with Pruning-1, the computation cost of PACER is $O((1-\gamma)\frac{1}{(p-2)!}\frac{n!}{(n-p)!})$.

Brute-force algorithm (BF). The brute-force algorithm based on the breadth-first expansion examines $O(\frac{n!}{(n-p)!})$ routes.

Approximation algorithm (AP). [4] proposed a *quasi-polynomial time* approximation algorithm for the Orienteering Problem. We modified AP to solve our problem. It uses a recursive binary search to produce a single route with the approximation ratio $\lceil \log p \rceil + 1$ and runs in $O((n \cdot OPT \cdot \log b)^{\log p})$, where OPT and b are the numbers of discrete value for an estimated optimal Gain and for the budget, respectively. The cost is expensive if b or OPT has many discrete values. For example, for $b = 512$ minutes, $n = 50$, $p = 8$ and $OPT = 10.0$ (100 discrete values with the single decimal point precision), the computation cost is $(50 \cdot 100 \cdot \log 512)^{\log 8} = 9.11 \times 10^{13}$. Lower precision leads to smaller computation cost, but also lower accuracy. [21] noted that AP took more than 10^4 seconds for a small graph with 22 nodes. Compared with AP, the computation cost of PACER with Pruning-1 given by Eqn. (12) is only $50 \times 49 \times \binom{49}{6} = 3.43 \times 10^{10}$. This cost is further reduced by Pruning-2. PACER finds the optimal top-k routes whereas AP only finds single approximate solution. We will experimentally compare PACER with AP.

6 HEURISTIC SOLUTIONS

PACER remains expensive for a large budget b and a large POI candidate set $\mathcal{V}_Q$. The above approximation algorithm is not scalable. Hence, we design two heuristics when such extreme cases arise.

State collapse heuristic. The cost dominance pruning in PACER keeps at most l open routes for a compact state representing a set of l POIs (excluding x and y). A more aggressive pruning is to keep only one open route having the least cost at each compact state, with the heuristic that this route likely visits more POIs. We denote this heuristic algorithm by **PACER-SC**, where SC stands for "State Collapsing". Clearly, PACER-SC trades optimality for efficiency, but it inherits many nice properties from PACER and Section 7.2 will show that it usually produces k routes with quite good quality.

Analogous to the complexity analysis for PACER in Section 5.5, with $p \ll n$, PACER-SC examines no more than $\sum_{l=1}^{p} l\binom{n}{l} \approx n\binom{n}{p-1}$ routes, which is around $1/p$ of that for PACER.

Greedy algorithm. PACER-SC's computation complexity remains exponential in the route length p. Our next greedy algorithm runs in polynomial time. It starts with the initial route $x \rightarrow y$ and iteratively inserts an unvisited POI i to the current route to maximize the marginal gain/cost ratio

$$\frac{Gain(\{i\} \cup \mathbb{C}) - Gain(\mathbb{C})}{s_i + T_{x,i} + T_{i,y}}, \tag{13}$$

where $\mathbb{C}$ denotes the set of POIs on the current route. It inserts i between two adjacent POIs in the current route so that the total cost of the resulting route is minimized. The term $T_{x,i} + T_{i,y}$ constrains the selected POIs i to be those not too far away from the two end points. The expansion process is repeated until the budget b is used up. The algorithm only produces a single route and examines $O(pn)$ routes because each insertion will consider at most n unvisited POIs.

7 EXPERIMENTAL EVALUATION

All algorithms were implemented in C++ and were run on Ubuntu 16.04.1 LTS with Intel i7-3770 CPU @ 3.40 GHz and 16G of RAM.

7.1 Experimental Setup

7.1.1 Datasets. We use two real-world datasets from [24]. *Singapore* denotes the Foursquare check-in data collected in Singapore, and *Austin* denotes the Gowalla check-in data collected in Austin. *Singapore* has 189,306 check-ins at 5,412 locations by 2,321 users, and *Austin* has 201,525 check-ins at 6,176 locations by 4,630 users. Same as suggested in [3, 24], we built an edge between two locations if they were visited on the same date by the same user. The locations not connected by edges were ignored. We filled in the edge costs $t_{i,j}$ by querying the traveling time in minute using Google Maps API under driving mode. The staying time s_i were generated following the Gaussian distribution, $s_i \sim \mathcal{N}(\mu, \sigma^2)$, with $\mu = 90$ minutes and $\sigma = 15$. The features are extracted based on the user mentioned keywords at check-ins, same to [24]. We obtain the rating of a feature h on POI i by

$$\mathbf{F}_{i,h} = \min\{\frac{NC_h(i)}{1/|S_h| \times \sum_{j \in S_h} NC_h(j)} \times \frac{\beta}{2}, \beta\}, \qquad (14)$$

where $NC_h(i)$ is the number of check-ins at POI i containing the feature h, S_h is the set of POIs containing h, β is the maximum feature rating and is set to $\beta = 5$ for both data sets. The calculation scales the middle value $\frac{\beta}{2}$ by the ratio of a POI's check-in count to the average check-in count on h. Table 2 shows the descriptive statistics of the datasets after the above preprocessing.

Table 2: Dataset statistics

	# POI	# Edges	Average $t_{i,j}$	# Features
Singapore	1,625	24,969	16.24 minutes	202
Austin	2,609	34,340	11.12 minutes	252

Both datasets were used in [24], which also studied a route planning problem. The datasets are not small considering the scenario for a daily trip in a city where the user has a limited cost budget. Even with 150 POIs to choose from, the number of possible routes consisting of 5 POIs can reach 70 billions. Compared to our work, [17] evaluated its itinerary recommendation methods using theme park data, where each park contains only 20 to 30 attractions.

7.1.2 Algorithms. We compared the following algorithms. **BF** is the brute-force method (Section 5.5). **PACER+1** is our proposed optimal algorithm with only Pruning-1 enabled. **PACER+2** enables both Pruning-1 and Pruning-2. **PACER-SC** is the state collapse algorithm and **GR** is the greedy algorithm in Section 6. **AP** is the approximation algorithm proposed by [4] (see Section 5.5). **A*** is the A* algorithm proposed by [24]. Since A* works only for its specific keyword coverage function, it is not compared until Section 7.3 where we adapt their coverage function in our method. To be fair, all algorithms use the indices in Section 4 to speed up. Note that BF, PACER+1, PACER+2 and A* are exact algorithms, while PACER-SC, GR, and AP are greedy or approximation algorithms.

7.1.3 Queries. A query Q has the six parameters $x, y, b, \mathbf{w}, \boldsymbol{\theta}, \Phi$. For concreteness, we choose Φ_h in Eqn. (3) with $\boldsymbol{\alpha}$ controlling the diversity of POIs on a desired route. We assume $\boldsymbol{\theta}_h$ and $\boldsymbol{\alpha}_h$ are the same for all features h. For *Singapore*, we set x as Singapore Zoo and y as Nanyang Technological University; and for *Austin*, we set x as UT Austin and y as Four Seasons Hotel Austin.

For each dataset, we generated 50 weight vectors $\mathbf{w}$ to model the feature preferences of 50 users as follows. For each $\mathbf{w}$, we draw m features, where m is a random integer in $[1, 4]$, and the probability of selecting each feature h is $\mathbf{Pr}(h) = \frac{\sum_{i \in S_h} NC_h(i)}{\sum_{h \in \mathcal{H}} \sum_{i \in S_h} NC_h(i)}$. $NC_h(i)$ and S_h are defined in Eqn. (14). Let $\mathcal{H}_Q$ be the set of selected features. For each $h \in \mathcal{H}_Q$, $\mathbf{w}_h = \frac{\sum_{i \in S_h} NC_h(i)}{\sum_{h \in \mathcal{H}_Q} \sum_{i \in S_h} NC_h(i)}$.

Finally, we consider $b \in \{4, 5, \mathbf{6}, 7, 8, 9\}$ in hours, $\boldsymbol{\theta} \in \{0, 1.25, \mathbf{2.5}, 3.75\}$, and $\boldsymbol{\alpha} \in \{0, \mathbf{0.5}, 1, 2\}$ with the default settings in bold face. For each setting of $b, \boldsymbol{\theta}, \boldsymbol{\alpha}$, we generated 50 queries $Q = (x, y, b, \mathbf{w}, \boldsymbol{\theta}, \boldsymbol{\alpha})$ using the 50 vectors $\mathbf{w}$ above. All costs are in minutes, therefore, $b = 5$ specifies the budget of 300 minutes.

We first evaluate the performance of our proposed algorithms (Section 7.2), then we compare with the A* algorithm (Section 7.3).

7.2 Performance Study

Evaluation metrics. As we solve an optimization problem, we evaluate *Gain* for effectiveness, *CPU runtime* and *search space* (in the number of examined open routes) for efficiency.

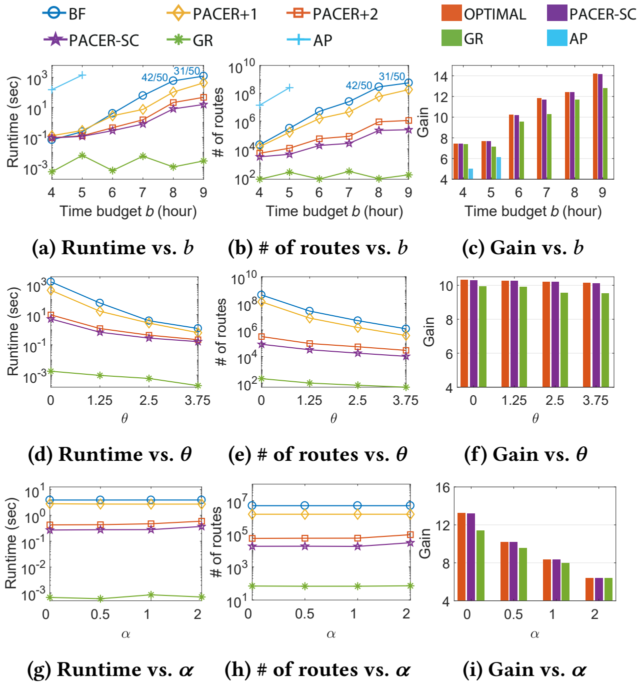
For every algorithm, we evaluate the three metrics for processing a query, and report the average for the 50 queries (i.e., vectors $\mathbf{w}$) under each setting of $(b, \boldsymbol{\theta}, \boldsymbol{\alpha})$ chosen from the above ranges. GR and AP only find single route, thus, we first set $k = 1$ to compare all algorithms, and discuss the impact of larger k in Section 7.2.4.

Figures 4 and 5 report the experiments for *Singapore* and *Austin*, respectively. Each row corresponds to various settings of one of $b, \boldsymbol{\theta}, \boldsymbol{\alpha}$ while fixing the other two at the default settings. OPTIMAL denotes the same optimal gain of PACER+2, PACER+1 and BF. We terminated an algorithm for a given query after it runs for 1 hour or runs out of memory, and used the label beside a data point to indicate the percentage of finished queries. If more than a half of the queries were terminated, no data point is shown.

7.2.1 Impact of budget b (Figure 4a - 4c and 5a - 5c). b affects the length of routes (the number of POIs included).

AP is the worst. This is consistent with the analysis in Section 5.5 that AP suffers from a high complexity when b and OPT have many discrete values. $b = 6$ has 360 discrete values in minute, a majority of the queries cannot finish. The efficiency of BF drops dramatically as b increases, since the number of open routes becomes huger and processing them is both time and memory consuming.

PACER+1's search space is two orders of magnitude smaller than that of BF, thanks to the compact state enumeration and the cost dominance pruning. PACER+2 is the best among all the exact

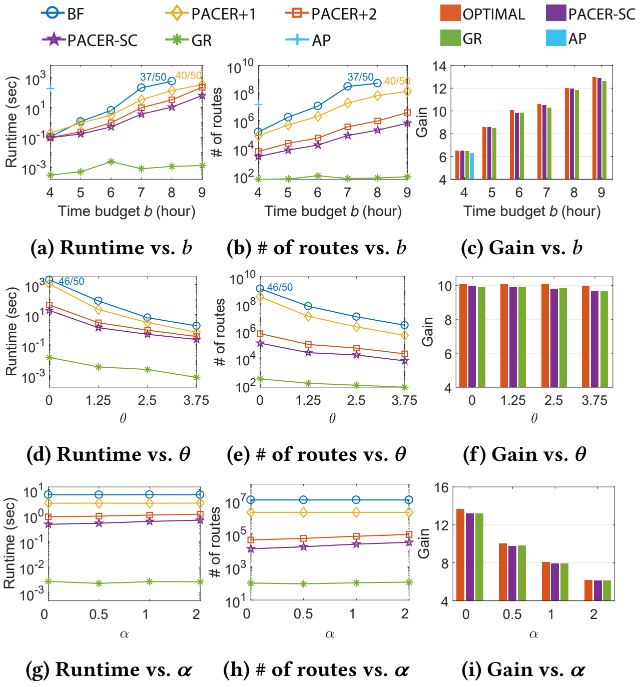


(a) Runtime vs. b (b) # of routes vs. b (c) Gain vs. b

(d) Runtime vs. θ (e) # of routes vs. θ (f) Gain vs. θ

(g) Runtime vs. α (h) # of routes vs. α (i) Gain vs. α

Figure 4: Experimental results for *Singapore*. Run time and search space (# of routes) are in logarithmic scale. The labels beside data points indicate the ratio of queries successfully responded by the algorithm under the parameter setting. No label if no query fail. Data point or bar is not drawn if more than half fail. AP can only respond queries with small b.



(a) Runtime vs. b (b) # of routes vs. b (c) Gain vs. b

(d) Runtime vs. θ (e) # of routes vs. θ (f) Gain vs. θ

(g) Runtime vs. α (h) # of routes vs. α (i) Gain vs. α

Figure 5: Experimental results for *Austin*

algorithms. Compared with PACER+1, the one order of magnitude speedup in runtime and two orders of magnitude reduction in search space clearly demonstrates the additional pruning power of the Gain based upper bound pruning. PACER-SC trades optimality for efficiency. Surprisingly, as shown in Figure 4c and 5c, PACER-SC performs quite well with Gain being close to that of OPTIMAL.

GR always finishes in less than 10^{-2} seconds. For *Singapore*, the achieved gain is far worse than that of OPTIMAL, compared with the difference for *Austin*. This is because x and y for *Singapore* are relatively remote to the central city. GR will greedily select a POI i not too far away from x and y (Eqn. (13)), thus, many POIs with possibly higher feature ratings located in the central city are less likely to be chosen. In contrast, x and y for *Austin* are in the downtown area and this situation is avoided in most cases.

7.2.2 *Impact of of filtering threshold $\boldsymbol{\theta}$.*

In Figure 4d - 4f and 5d - 5f, as $\boldsymbol{\theta}$ increases, the POI candidate set becomes smaller and all the algorithms run faster. The majority of the queries for AP cannot finish and its results are not shown. The study suggests that a reasonable value of $\boldsymbol{\theta}$, e.g., 2.5, reduces the searching cost greatly while having little loss on the quality of the found routes.

7.2.3 *Impact of route diversity parameter $\boldsymbol{\alpha}$ (Figure 4g - 4i and 5g - 5i).*

PACER+2 and PACER-SC are slightly affected when $\boldsymbol{\alpha}$ varies. As $\boldsymbol{\alpha}$ increases, the marginal return diminishes faster and Φ_h behaves more towards the max aggregation. In this case, Pruning-2 becomes less effective. When $\boldsymbol{\alpha} = 0$ (the sum aggregation), both *Gain* and the difference between OPTIMAL and GR reach the maximum.

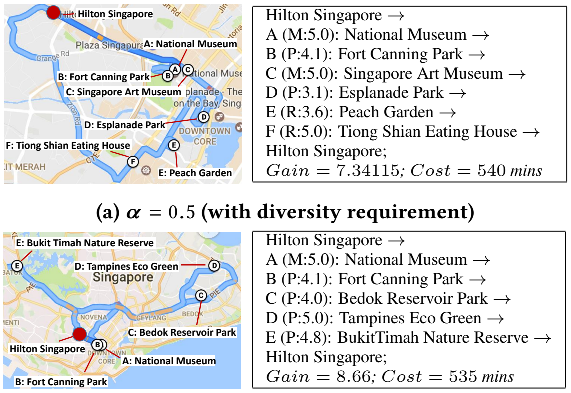
Figure 6 illustrates the effectiveness of our power law function in Eqn. (3) for modeling the personalized route diversity requirement. We run two queries on *Singapore*, one with $\boldsymbol{\alpha} = 0.5$, which specifies a diversity requirement, and one with $\boldsymbol{\alpha} = 0$, which specifies the usual sum aggregation. The other query parameters are the same. The figures show the best routes found for each query, with the POIs on a route labeled sequentially as A, B $\cdots$. The red dots represent the source x and destination y. The route for $\boldsymbol{\alpha} = 0.5$ covers all specified features, i.e., two POIs for each feature, while maximizing the total *Gain*. While the route for $\boldsymbol{\alpha} = 0$ has four parks out of five POIs due to the higher weight of Park in $\mathbf{w}$; thus, it is less preferred by a user who values diversity. In fact, the second route's *Gain* value when evaluated using $\boldsymbol{\alpha} = 0.5$ is only 6.60.

7.2.4 *Impact of k.*

We vary k in range $[1, 100]$ while fixing $b, \boldsymbol{\theta}, \boldsymbol{\alpha}$ at the default values and run the algorithms, except GR and AP, on both datasets. As k only influences the gain-based pruning, the performance of BF and PACER+1 are unchanged. For PACER+2 and PACER-SC, the change is limited (less than 25% slower for $k = 100$). Because when k is small, the Gain of the k-th best route is usually not far away to that of the best route, thus, the marginal gain upper bound pruning is not seriously influenced. We omit the figures due to limited space.

7.3 Comparison with A*

A* [24] only works for their keyword coverage function: $\Phi_h(\mathcal{P}_V) = 1 - \prod_{i\in\mathcal{P}_V}[1 - \tilde{\mathbf{F}}_{i,h}]$, and finds single route. In [24], $\tilde{\mathbf{F}}_{i,h}$ is in the range $[0, 1]$ and it is set to 1 if the number of check-ins on POI i for



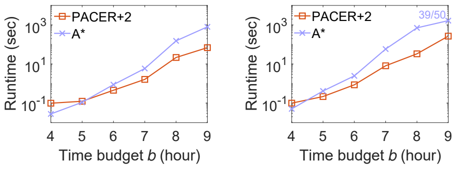
Hilton Singapore →
A (M:5.0): National Museum →
B (P:4.1): Fort Canning Park →
C (M:5.0): Singapore Art Museum →
D (P:3.1): Esplanade Park →
E (R:3.6): Peach Garden →
F (R:5.0): Tiong Shian Eating House →
Hilton Singapore;
$Gain = 7.34115; Cost = 540$ *mins*

(a) $\alpha = 0.5$ (with diversity requirement)

Hilton Singapore →
A (M:5.0): National Museum →
B (P:4.1): Fort Canning Park →
C (P:4.0): Bedok Reservoir Park →
D (P:5.0): Tampines Eco Green →
E (P:4.8): BukitTimah Nature Reserve →
Hilton Singapore;
$Gain = 8.66; Cost = 535$ *mins*

(b) $\alpha = 0$ (without diversity requirement)

Figure 6: Two routes found from *Singapore* by PACER+2 for the query $Q = (x, y, b = 9, \mathbf{w} = (P : 0.4, M : 0.3, R : 0.3), \theta = 2.5, \alpha)$, where x and y are Hilton Singapore, and P, M and R represent Park, Museum, and Chinese Restaurant.



(a) Runtime - *Singapore* **(b) Runtime - *Austin***

Figure 7: PACER+2 vs. A* (logarithmic scale).

feature h is above average. In this case, the single POI in $\mathcal{P}$ yields the maximum $\Phi_h(\mathcal{P}_V)$ value; the feature h of other POIs will be ignored. For a fair comparison, we set $\beta = 0.5$ in Eqn. (14) for both algorithms, we also leverage our indices to speed up A*. Note that the maximum b in [24] is 15 kilometers in their efficiency study, which is about 20 minutes by Google Maps under driving mode.

Figure 7 shows the comparison between PACER+2 and the modified A* on both datasets. The report of *Gain* is omitted as they are both exact algorithms. We also omit the comparison of search space due to page limit (PACER+2 searches one to two orders of magnitude less than A*). Apparently, PACER+2 outperforms A*, especially for a large b. A few queries of A* on *Austin* even failed for $b = 9$. Although A* has a pruning strategy specifically for their keyword coverage function, the search strategy itself is a bottleneck. Besides, their pruning based on the greedy algorithm in [12] has a bound looser than ours. In fact, the experiments in [24] showed that A* is just 2-3 times faster than the brute-force algorithm.

8 CONCLUSION

We considered a personalized top-k route search problem. The large scale of POI maps and the combination of search in feature space and path space make this problem computationally hard. The personalized route diversity requirement further demands a solution that works for any reasonable route diversity specification. We presented an exact search algorithm with multiple pruning strategies to address these challenges. We also presented high-performance heuristic solutions. The analytical evaluation suggested that our solutions significantly outperform the state-of-the-art algorithms.

Acknowledgments. Ke Wang's work was partially supported by a discovery grant from The Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] Senjuti Basu Roy, Gautam Das, Sihem Amer-Yahia, and Cong Yu. 2011. Interactive itinerary planning. In *ICDE*. IEEE, 15–26.

[2] David M Burton. 2006. *Elementary number theory*. Tata McGraw-Hill Education.

[3] Xin Cao, Lisi Chen, Gao Cong, and Xiaokui Xiao. 2012. Keyword-aware optimal route search. *VLDB Endowment* 5, 11 (2012), 1136–1147.

[4] Chandra Chekuri and Martin Pal. 2005. A recursive greedy algorithm for walks in directed graphs. In *FOCS*. IEEE, 245–253.

[5] Jian Dai, Bin Yang, Chenjuan Guo, and Zhiming Ding. 2015. Personalized route recommendation using big trajectory data. In *ICDE*. IEEE, 543–554.

[6] Munmun De Choudhury, Moran Feldman, Sihem Amer-Yahia, Nadav Golbandi, Ronny Lempel, and Cong Yu. 2010. Automatic construction of travel itineraries using social breadcrumbs. In *ACM Hypertext and Hypermedia*. ACM, 35–44.

[7] Qiming Diao, Minghui Qiu, Chao-Yuan Wu, Alexander J Smola, Jing Jiang, and Chong Wang. 2014. Jointly modeling aspects, ratings and sentiments for movie recommendation (jmars). In *the 20th ACM SIGKDD*. ACM, 193–202.

[8] Aristides Gionis, Theodoros Lappas, Konstantinos Pelechrinis, and Evimaria Terzi. 2014. Customized tour recommendations in urban areas. In *WSDM*. 313–322.

[9] Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Yanyan Xu. 2014. Hop doubling label indexing for point-to-point distance querying on scale-free networks. *VLDB Endowment* 7, 12 (2014), 1203–1214.

[10] Iris A Junglas and Richard T Watson. 2008. Location-based services. *Commun. ACM* 51, 3 (2008), 65–69.

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

[12] Samir Khuller, Anna Moss, and Joseph Seffi Naor. 1999. The budgeted maximum coverage problem. *Inform. Process. Lett.* 70, 1 (1999), 39–45.

[13] Andreas Krause and Daniel Golovin. 2012. Submodular function maximization. *Tractability: Practical Approaches to Hard Problems* 3, 19 (2012), 8.

[14] Andreas Krause and Carlos Guestrin. 2008. Beyond convexity: Submodularity in machine learning. *ICML Tutorials* (2008).

[15] Takeshi Kurashima, Tomoharu Iwata, Go Irie, and Ko Fujimura. 2010. Travel route recommendation using geotags in photo sharing sites. In *CIKM*. 579–588.

[16] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. 2007. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD*. ACM, 420–429.

[17] Kwan Hui Lim, Jeffrey Chan, Shanika Karunasekera, and Christopher Leckie. 2017. Personalized itinerary recommendation with queuing time awareness. In *Proceedings of the 40th ACM SIGIR*. ACM, 325–334.

[18] Qi Liu, Yong Ge, Zhongmou Li, Enhong Chen, and Hui Xiong. 2011. Personalized travel package recommendation. In *ICDM*. IEEE, 407–416.

[19] Eric Hsueh-Chan Lu, Ching-Yu Chen, and Vincent S Tseng. 2012. Personalized trip recommendation with multiple constraints by mining user check-in behaviors. In *SIGSPATIAL*. ACM, 209–218.

[20] Daniele Quercia, Rossano Schifanella, and Luca Maria Aiello. 2014. The shortest path to happiness: Recommending beautiful, quiet, and happy routes in the city. In *ACM Hypertext and Social media*. ACM, 116–125.

[21] Amarjeet Singh, Andreas Krause, Carlos Guestrin, William J Kaiser, and Maxim A Batalin. 2007. Efficient Planning of Informative Paths for Multiple Robots.. In *IJCAI*, Vol. 7. 2204–2211.

[22] Maxim Sviridenko. 2004. A note on maximizing a submodular set function subject to a knapsack constraint. *Operations Research Letters* 32, 1 (2004), 41–43.

[23] World Travel and Tourism Council. 2017. Travel and Tourism Global Economic Impact and Issues 2017. *https://www.wttc.org/* (2017).

[24] Yifeng Zeng, Xuefeng Chen, Xin Cao, Shengchao Qin, Marc Cavazza, and Yanping Xiang. 2015. Optimal Route Search with the Coverage of Users' Preferences. In *24th IJCAI*. AAAI Press, 2118–2124.

[25] Chenyi Zhang, Hongwei Liang, and Ke Wang. 2016. Trip recommendation meets real-world constraints: POI availability, diversity, and traveling time uncertainty. *ACM TOIS* 35, 1 (2016), 5.

[26] Wei Zhang and Jianyong Wang. 2015. Location and Time Aware Social Collaborative Retrieval for New Successive Point-of-Interest Recommendation. In *Proceedings of the 24th ACM CIKM*. ACM, 1221–1230.

[27] Bolong Zheng, Nicholas Jing Yuan, Kai Zheng, Xing Xie, Shazia Sadiq, and Xiaofang Zhou. 2015. Approximate keyword search in semantic trajectory database. In *31st ICDE*. IEEE, 975–986.

[28] Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. 2009. Mining interesting locations and travel sequences from GPS trajectories. In *WWW*. 791–800.