Knowl Inf Syst (2018) 56:581–611
https://doi.org/10.1007/s10115-017-1106-5

REGULAR PAPER

On route planning by inferring visiting time, modeling user preferences, and mining representative trip patterns

Cheng-Te Li[1] · **Hsin-Yu Chen**[1] · **Ren-Hao Chen**[2] · **Hsun-Ping Hsieh**[2,3]

Received: 7 November 2015 / Revised: 18 April 2017 / Accepted: 21 September 2017 /
Published online: 5 October 2017

Abstract Location-based services allow users to perform check-in actions, which record the geo-spatial activities and provide a plentiful source to do more accurate and useful geographical recommendation. In this paper, we present a novel *Preferred Time-aware Route Planning* (PTRP) problem, which aims to recommend routes whose locations are not only representative but also need to satisfy users' preference. The central idea is that the goodness of visiting locations along a route is significantly affected by the visiting time and user preference, and each location has its own proper visiting time due to its category and population. We develop a four-stage preference-based time-aware route planning framework. First, since there is usually either noise time on existing locations or no visiting information on new locations, we devise an inference method, *LocTimeInf*, to predict the location visiting time on routes. Second, considering the geographical, social, and temporal information of users, we propose the *GST-Clus* method to group users with similar location visiting preferences. Third, we find the representative and popular time-aware location-transition behaviors by proposing *Time-aware Transit Pattern Mining* (TTPM) algorithm. Finally, based on the mined time-aware transit patterns, we develop a *Preferred Route Search* (PR-Search) algorithm to construct the final time-aware routes. Experiments on Gowalla and Foursquare check-in data exhibit the

✉ Cheng-Te Li
chengte@mail.ncku.edu.tw

Hsin-Yu Chen
r26054129@mail.ncku.edu.tw

Ren-Hao Chen
q36051114@mail.ncku.edu.tw

Hsun-Ping Hsieh
hphsieh@mail.ncku.edu.tw

1 Department of Statistics, National Cheng Kung University, Tainan, Taiwan

2 Institute of Computer and Communication Engineering, National Cheng Kung University, Tainan, Taiwan

3 Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan

promising effectiveness and efficiency of the proposed methods, comparing to a series of competitors.

Keywords Preferred routes · Transit patterns · Visiting time · Route planning · Check-in data

1 Introduction

Nowadays, location-based services (LBS), such as Foursquare and Gowalla, keep track of personal geospatial journeys through check-in actions. With smart phones, users can easily perform check-in actions, and the geographical information of locations with timestamps is stored in LBS. Eventually a large-scaled user-generated routes are derived. Such data not only collectively represent human geo-activities, but also serve as a handy resource to do location-based recommendation. Since check-in records reveal how people travel with rich spatial and temporal information, including longitude, latitude, and timestamp, one of the applications is to recommend travel routes. Existing work on trip planning [1–3] considers either geodesic distance or shortest time to recommend routes. *Location recommendation* [1, 3,4] is to recommend locations that users had never visited before, while *location prediction* [5–7] aims at predicting next existing locations that users had ever visited. Although there are a number of studies on route recommendation, the goodness of visiting locations in terms of time is not investigated yet.

This paper aims to exploit visiting time information of locations to recommend routes. The central idea is that the goodness of visiting places along a route is significantly diminished if arrived at the wrong time. Some places have a wider range of visiting time period while others are constrained to particular time periods. For example, most people do not want to visit a beach during the boiling hot noon, but rather arrive in the late afternoon to enjoy the sunset scene. In addition, different people may have different preference of visiting time to a particular location. One prefers to visit NYC Central Park in the morning for jobbing while another loves to enjoy the sunbathing in the afternoon. Therefore, we aim to combine both the goodness and preference of location visiting time for the task of route planning.

We propose *Preferred Time-aware Route Planning* (PTRP). The goal is to recommend *preferred time-aware routes*, which is not only representative and pleasant to visit in terms of time but also satisfy user preference considering her location visiting history, based on the user query consisting of a starting location, a starting time, and/or a destination location. We have two fundamental assumptions. First, if more people visit a particular location l at a certain time period t, then t is said to be good to visit location l. Second, the preference of location visiting time for a user can be collectively reflected by not only herself but also those users who possess similar location visiting histories. A time-aware route is a sequence of locations, in which each location is associated with a visiting time label that is supposed to be the most proper. All the locations visited at the given time along a desired route should be proper and satisfied by users. To achieve the PTRP, we develop a four-stage route planning framework to tackle diverse challenges. The first is to infer the visiting time labels of locations in routes. The second is to do user clustering to find the visiting time preference of users based on their historical geographical, social, and time information. The third is to mine the popular time-aware transit patterns, which can be regarded as the representative route segments generated by users. The fourth is to compose the desired route based on the query.

(*a*) *Inferring Visiting Time* Given locations in a route, we aim to infer their correct visiting time labels from check-in data. Acquiring the correct visiting time of locations can benefit many applications, such as location recommendation [4], location prediction [8], and measuring route quality [9]. However, it is difficult to have the complete and correct visiting time of locations. The reason is twofold. First, new locations or *Point-of-Interests* (POI) are rapidly generated due to new events, new buildings, attractions, even new developed areas, and so on. Such *cold-start* places have no or rare records of visiting time. What we have could be incomplete or less visiting time. Second, even if we have some location check-in records, the data tend to contain noise and abnormal check-in behaviors. For example, one might perform check-in actions using Web interfaces at home, instead of mobile phones *instantaneously*. The time recorded by the former manner might not be the immediate check-in time (could be wrong time), but it does in the latter manner. In other words, we need to do data correction or manually annotate the data before usages, which are expensive and time-consuming.

(*b*) *Geographical-Social-Temporal User Clustering* Given a collection of time-labeled routes derived from the inference of visiting time, we aim at grouping users who tend to have similar location visiting preferences. We consider users' visiting behaviors from three perspectives, geographical, social, and time. For the geographical part, users have higher potential to visit the same locations if their past visited locations are spatially close to each other. For the social part, those who are acquainted with each other tend to have the same preference since they have higher possibility to visit together. For the temporal part, while the location visiting time of users can reflect their temporal visiting behaviors, users with similar distributions of location visiting time tend to have similar temporal preference. Users in each cluster are collectively used to their visiting preference for recommending preferred time-aware routes.

(*c*) *Mining Time-aware Transit Patterns* Given a set of time-labeled routes for users in each cluster, we aim to mine the *Time-aware Transit Patterns* (TTP), which are considered as the representative and popular time-aware trip segments collectively created by a cluster of users. A TTP is a sequence of locations, in which each transit between locations is associated with visiting timestamps. An example TTP is *<(MMOA museum, Macy's store, 0, 3), (Macy's store, H&M store, 3, 5), (H&M store, Time Square, 5, 6)>*, which refers to that the sub-route consists of *MMOA museum, Macy's store, H&M store and Time Square* in order. This pattern also suggests that the staying time and the transportation time between MMOA museum and Macy's store is 3-0=3 hours, and so on.

(*d*) *Time-aware Route Planning* In the final stage, we aim to find the preferred time-aware routes based on the query requirement, the user clusters, and the mined TTPs. We allow users to input two kinds of queries: source query and source–destination query. Both include a starting location, a starting time, and/or the number of locations in the desired route while the latter needs an additional destination location.

The proposed approach is data-driven, which means that the parameters, time labels, and trip patterns can be automatically transplanted into the collections of users and check-ins in different cities. In addition, although we emphasized on the check-in data throughout the paper, in fact any kinds of geo-spatial route data can be employed under our framework. For example, the GPS trajectory data can be used if we can perform some pre-processing in advance to identify the points-of-interest (i.e., venues).

To achieve the PTRP, we develop a four-stage route planning framework. The first is to infer the visiting time labels of locations in routes. The second is to do user clustering to find the visiting time preference of users based on their historical geographical, social, and time information. The third is to mine the popular time-aware transit patterns, which can be

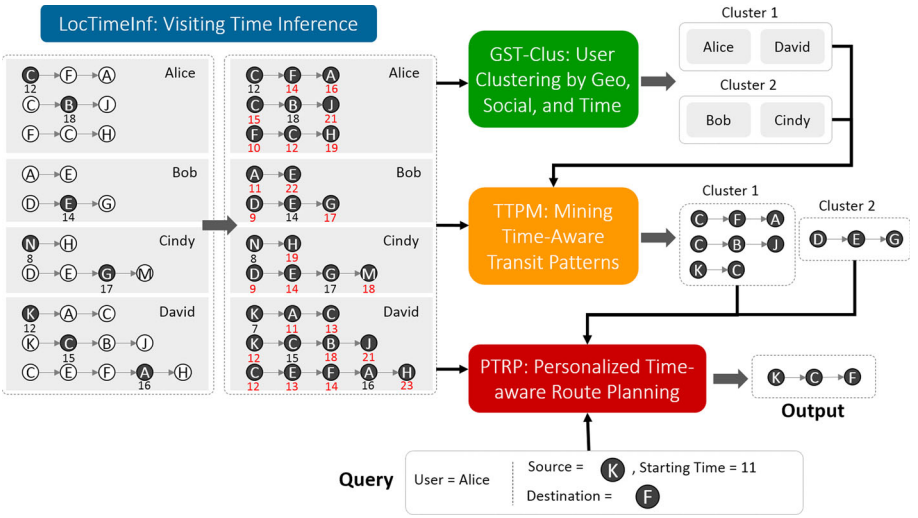

Fig. 1 The methodology overview of this work

regarded as the representative route segments generated by users. The fourth is to compose the desired route based on the query. In the following, we will elaborate the goals, ideas, and challenges of each stage using the methodology overview, as shown in Fig. 1.

We summarize the contributions of this paper in the following.

- We formulate the *Preferred Time-aware Route Planning* (PTRP) problem based on check-in data. A general framework is developed to tackle PTRP, which consists four parts, which corresponds to four sub-problems: (a) inferring visiting time of locations, (b) clustering users based on their preferences, (c) mining time-aware transit patterns, and (d) personalized time-aware location recommendation.
- We propose four algorithms to address these four sub-problems. A semi-supervised learning method, *LocTimeInf*, is proposed to infer the visiting time labels of locations in routes. A clustering method, *GST-Clus*, is proposed to combine geographical, social, and temporal information for user clustering. A pattern mining method, *TTPM*, is devised to discover the closed frequent time-aware transit patterns. And a proper route search algorithm, *PR-Search*, is proposed to find the preferred time-aware routes based on the inferred time labels, the discovered user clusters, and mined patterns.
- We conduct the experiments both Foursquare and Gowalla check-in data, and the results exhibit all of the four proposed methods can significantly outperform the corresponding sets of existing state-of-the-art methods. A set of parameters are extensively studied, and the results show the robustness of the proposed PTRP framework.
- We develop the PTRP framework as a real system. With the system, we invite a number of participants to conduct a user study. The results demonstrate that users can feel more satisfied for the routes recommended by our framework, compared with other well-known strategies.

The remaining of this paper is organized as below. We first review the relevant studies in Sect. 2 before the methodology, which consists of four Sects. 3–6. Section 3 delivers the *LocTimeInf* component to infer the location visiting time from check-in data. Section 4 presents the *GST-Clus* component to group users with similar visiting preferences. Section 5 describes

the *TTPM* algorithm that mines the time-aware transit patterns as the representative trip segments. Section 6 aims at generating the recommended routes based on user requirements. We exhibit the experimental studies in Sect. 7 and conclude this paper in Sect. 8.

2 Related work

Route Planning Recent work exploits complete time-labeled check-in data for location and trip recommendation. Yuan et al. develop a collaborative recommendation model [4] to recommend POIs for a given user at a specified time in a day. Hsieh et al. [9] develop a *TripRouter* system to construct time-sensitive routes, which consider location popularity, visiting order, proper visiting time, and proper transit time to model the goodness of a route. Wei et al. [2] develop a route inference to construct the popular routes passing through a given location sequence within a specified timespan. Chiang [10] consider the current time to predict locations. They construct a *Time-constrained Mobility Graph* that captures a user's moving behavior within a certain time interval and compute the *reachability* between locations to infer next one. Cheng et al. [11] present a preferred travel recommendation according to the automatically detected *demographic* attributes (e.g., gender, age, and race) from Flickr geo-tagged photos. Lu et al. [12] develop the *Photo2Trip* system, which integrates a series of traveling factors (including time duration, season, user preference, destination type, and popularity) to recommend trip itineraries. Kurashima et al. [13] develop a *probabilistic photographer behavior model* (combing topic model and Markov chain) to estimate the probability of visiting a landmark and iteratively construct routes based on user preference and present location. Sharifzadeh et al. [14] and Cao et al. [15] present the *optimal sequenced route* (OSR) problem and the *keyword-aware optimal route* (KOR) problem, respectively, which aim to find an optimal and popular route that covers a set of user-specified keywords/types (e.g., shopping mall, pub, and restaurant) of locations. Given a sequence of locations and a set of desired *labels* that represent the *activities* of locations, Zheng et al. [16] present the *activity trajectory similarity search* which returns k check-in trajectories that cover the activity labels and yield the shortest minimum *order-sensitive* match distance with respect to the query. Quercia et al. [17] recommend routes that are not only short but also emotionally pleasant in terms of three crowd-sourcing labels: *beautiful*, *quiet*, and *happy*. Tan et al. [18] consider the recognition of traffic signs for planning the routes of vehicles. To the best of our knowledge, we are the first to tackle time-aware route planning by location check-in data.

Location Recommendation and Prediction *Location recommendation* (LR) is to recommend new locations that the user had *never* visited before while *location prediction* (LP) aims at predicting next *existing* locations that the user had ever visited. For LR, Ye et al. [19] point out the importance of *geographical influence*, which refers to that people tend to visit (a) *nearby* locations and (b) may be interested in *farther* locations that they are in favor of. Liu et al. [20] do a *personalized* point-of-interest recommendation using matrix factorization to mine the transition of user preferences over location categories, to enhance the recommendation accuracy. By combining user preference, geographical influence, and historical trajectories, Liu et al. [21] develop a geographical probabilistic method to learn how such factors affect the performance of POI recommendation. Yuan et al. [4] propose a temporal POI recommendation to find suitable locations at a specified time in a day, using a temporal-enhanced collaborating filtering. For LP, Sadilek et al. [6] predict the most likely locations of an individual at any time, given historical trajectories of his/her friends. Monreale et al. [5] predict the next location of a moving object with a assumption: people tend to

follow common paths. Noulas et al. [22] predict the check-in venue that a mobile user will visit by formulating it as a *ranking* task: given a user with his/her current location, ranking all the venues so that the predicted one is at the highest position.

Another research direction is to recommendation location based on matrix and tensor factorization. GeoMF [23] is a weighted matrix factorization-based approach to POI recommendation by incorporating both user's visiting preferences and the correlation between venues in the geo-spatial neighborhood. WWO [24] is another POI recommendation system that jointly leverages the techniques of probabilistic graphical model and matrix factorization to model when and where a user will visit within a specific time period. Yao et al. [25] exploit non-negative tensor factorization to jointly model user's preference of location visiting and the social connections between users for POI recommendation. STELLAR [26] is a ranking-based pairwise tensor factorization-based POI recommender, which imposes time indexing schema into modeling among users, time, and POI. Rank-GeoFM [27] aims at simultaneously capturing users' preferences and embedding time-aware characteristics of POIs under the setting of tensor factorization.

We need to point out the differences between our problem setting and theirs, which make matrix/tensor-based approaches infeasible and impractical. First, location recommendation tasks in existing studies aim to predict a set of locations that users will visit in the future. Their settings do not consider the sequential property of "routes", which is our target. In other words, existing location recommendation considers locations "separately." Our work instead models how users transit from one location by discovering the time-aware transit patterns from routes. Second, existing tensor-based approaches assume that time information is always available so that the tensor can be constructed correspondingly. However, in real-world scenarios, time information of each geo-tagged postings can be either missing or abnormal. Under such kind of scenarios, it is impossible to construct the tensor for location recommendation. Therefore, the proposed method is developed to be able to not only infer the visiting time of locations, but also to recommend locations in routes.

Temporal Pattern Mining When considering temporal pattern mining, sequential pattern mining is one of the traditional methods to mine frequently occurring temporal events or subsequences. The sequential pattern mining problem was first introduced by Agrawal et al. [28], who adopt a generate-and-test approach based on the Apriori method to mine frequent sequential patterns. SPAM [29] exploits a vertical bitmap structure to count supports efficiently. PrefixSpan [30] uses the projected database to mine the complete set of patterns and to reduce the efforts of candidate subsequence generation. A frequent pattern is closed if there does not exist any super-pattern with the same support. Yan et al. [31] propose CloSpan to mine closed sequential patterns by candidate maintenance-and-test paradigm to prune the search space and check if a newly found frequent sequence is promising to be closed. Wang et al. [32] present the BIDE algorithm to improve the performance of CloSpan without keeping track of any frequent closed patterns (or candidates) for closure checking. SPADE [7] uses a vertical data format and a divide-and-conquer strategy to reduce the search space and the number of database scans. Though a number of sequential pattern mining techniques are proposed, they cannot directly apply to mine the proposed time-aware transit patterns (TTP). The reason is twofold. First, TTP considers the starting and end timestamps of nearby locations. That says, patterns should frequent in terms of not only sequential order but also time period. Second, the transition behaviors of users are essentially represented in a graph form with constraints. Therefore, we need to devise an effective and efficient method to mine TTPs.

3 LocTimeInf: visiting time inference

3.1 Problem statement

A (time-labeled) route is a sequence of time-labeled locations, denoted by $r =< (l_1, t_1), (l_2, t_2), \ldots, (l_k, t_k) >$, in which l_i is a location and t_i is the corresponding time label in hour (i.e., 0, 1, 2, …, 23). A time-unlabeled route is a route, in which the locations do not necessarily have the time labels associated. We also define the *time-label ratio* (denoted by τ) as the number of time-labeled routes divided by the number of time-unlabeled routes. Given a collection of routes with a certain time-label ratio, our goal aims to infer the visiting time labels for locations in such routes. Note that the time-label ratio τ will be set to be less than 1% and be varied for the evaluation. Also note that we choose an hour as the time granularity because people usually use "hour" as the basic unit to schedule their time [33].

To tackle the proposed time label inference problem, we devise an inference model, *LocTimeInf*, which is a semi-supervised learning, consisting of three steps. (1) We construct a *Route Correlation Graph* (*RCG*) for each location to capture the temporal correlation of a location across routes. (2) We propagate and infer the probability of a particular visiting time label for each location in a time-unlabeled route. In other words, our method propagates time labels of labeled instances to unlabeled instances such that nearby instances in the route correlation graph tend to share similar probability distributions of time labels. (3) We refine the inferred time labels considering the proper visiting order of locations.

3.2 Route correlation graph

For each location l_i that appears in all n_i routes (including both labeled and unlabeled), we construct a *Route Correlation Graph* (RCG) to capture the temporal correlation for l_i in different routes.

Definition 1 (*Route correlation graph*) A route correlation graph RCG_{li} for a certain location l_i is a *weighted connected* graph constructed from a collection of routes that contain l_i, in which a node is a route with location l_i and an edge is constructed if two routes are similar to some extent (see below for edge construction). In a RCG_{li}, some nodes have time labels (derived from time-labeled routes) while other nodes do not. Edge weights depict the similarity between all pairs of routes containing l_i.

The central idea of *RCG* is that if two routes are more similar in both geographical and contextual aspects, their time labels of location l_i tend to be more correlated. Therefore, we give higher edge weights if two routes are more similar. The proposed route similarity consists of three parts: (a) *Location overlapping* is the *Jaccard coefficient* on the location sets of two routes. If two routes share more common locations, the corresponding users tend to have similar visiting preferences, and thus derive a higher similarity score. (b) *Position difference* is the reciprocal of the maximum position difference of location l_i between routes, smoothed by adding one. If a location is visited at a relatively-close position on two routes, which reflects similar visiting orders, the edge weight gets higher. (c) *Geographical proximity* is the average distance in geography over locations between routes. The geographical proximity is useful to model the spatial range of travels, which also implicitly reflects the difference of transportation time between routes. Higher proximity indicates shorter transition time duration between locations in routes. After normalization, we calculate the geometric mean of such three similarity scores and regard the value as the edge weight between routes of location l_i in RCG. Note that the RCG of location l_i is a complete graph, which would lead

to high computational complexity. Therefore, for each node in a RCG we use a parameter π to retain π edges with top-π highest weights so that the efficiency of the following inference can be boosted. The parameter π will be evaluated. Note that the edge weights are computed based on the route similarity, they are not what we aim to infer. We aim to infer time labels of locations.

RCG for Cold-Started New Locations In the real-world cases, there might be new locations established in location-based services. The new locations could either have no accurate visiting time labels, or be rarely visited by the routes. The cold-start problem then happens because no sufficient time-labeled nodes can help the time-label inference in RCG. We discover *top-k auxiliary locations* to deal with such case. The idea of selecting auxiliary locations for a new location l_i is based on the route similarity scores between l_i and the routes of candidate time-labeled locations. The detailed procedure of selecting top-k auxiliary locations consist three steps. First, we scan the database to find all the locations l_c that are geographically close to l_i within a certain geo-distance δ. Second, we enumerate all the routes involved by l_c and compute their route similarity to l_i. Finally, while k-RCGs are constructed from top-k auxiliary locations, we impose the RCG of l_i into the k-RCGs by constructed edges from each l_i to all nodes in the k-RCGs, in which edge weights are associated as well. Note that the parameter π is also applied to remove insignificant edges here. A new *auxiliary RCG* of l_i is then derived for the following inference.

3.3 Time label inference

Based on the constructed RCG, we learn the visiting time label for each location in time-unlabeled routes, by devising a graph-based semi-supervised learning mechanism. The fundamental idea is to exploit Gaussian random fields together with harmonic functions to relax the Boltzmann machines. Our goal is to optimize the loss function on the RCG such that the labeled data are clamped, where unlabeled nodes that are nearby (i.e., connected by higher edge weights) would be given similar time labels. We design a quadratic loss function $E(f) = \sum_{i,j} w_{ij} \left(f(S_i) - f\left(S_j\right)\right)^2$, where S_i and S_j is the sets of labeled and unlabeled nodes, respectively. We design the objective function $f = argmin_L E(f)$ such that it is harmonic. Therefore, the harmonic property concludes that the value of function f at each unlabeled node is the average value over its neighboring nodes: $f\left(S_j\right) = \frac{1}{d_j} \sum_{i \sim j} w_{ij} (f(S_i))$. The intuition behind this approach is to do the propagation of time labels from labeled instances to unlabeled instances such that nearby instances in the route correlation graph tend to share similar probability distributions of time labels.

We can solve the optimization problem using *combinatorial Laplacian* Δ, which can be calculated through $\Delta = D - W$, where D is the diagonal matrix, where $d_i = \sum_j w_{ij}$, and W is the weighted matrix. Since the harmonic solution process goes beyond the scope of this paper, please refer to Zhu et al.'s work [34] for details. Assume the RCG contains N nodes, in which N_l are labeled, our goal is to infer the probabilities that the location was visited for 24 h units. The harmonic solution would generate $(N - N_l)$*24 label matrices for the unlabeled instances, in which each row is for an unlabeled node while each column is for a class label (i.e., hour). The time label (hour) with the highest probability value is regarded as the predicted time label for each unlabeled instance.

3.4 Refining time labels in unlabeled routes

We have derived the time labels for each location l_i in each time-unlabeled route. The next step is to refine the time label with the probability for each l_i, considering the *visiting order* of locations in a route. The idea is that users usually tend to visit locations with the most proper time and order. Therefore, given a time-unlabeled route with location sequence $< l_1, l_2, \ldots, l_k >$, we aim to find the corresponding time labels with probabilities p_1, p_2, …, and p_k such that $\prod_{i=1..k} p_i$ is maximized. Such goal is constrained by the fact that the visiting time of the (i-1)th location in a route should not be later than the visiting time of ith one. We devise a simple but effective three-stage method to find the refined time labels. First, we use the inferred probability values to sort the possible time labels of each location in a descending order. Second, for each possible time label of the source location, we search for the next location's time label that not only possesses the highest probability but also satisfies the time order constraint. Each search will stop when reaching the destination location. Totally there are 24 sequences of time labels. Third, we compute the value of $\prod_{i=1..k} p_i$ for each sequence of time labels. The time-label sequence with the highest $\prod_{i=1..k} p_i$ is reported.

4 GST-Clus: geographical–social–temporal user clustering

The goal of user clustering is to group users who share similar location visiting preferences in terms of geographical, social, and temporal perspectives using the inferred time labels on locations. Users in each cluster can collectively represent their visiting behaviors. Such collective representation of user visiting preferences via clustering can bring us three benefits. First, it can help deal with the data sparsity problem, i.e., some users might have rare check-in records. Second, time-labeled routes of a user are suitable being used to recommend routes for other users in the same cluster since they share similar visiting preferences. Third, to do the clustering, a user is allowed to either choose one of the three options, geographical, social, and temporal, or use our automatic weighting combination mechanism, and thus her decision can be made from diverse planned routes. Note that the novelty of this part lies in developing a reasonable measure to estimate the distance between two users. We devise the geographical–social–temporal distance, which is a weighting combination of geographical distance, social distance, and temporal distance defined in the following.

In fact, each perspective has its own physical meaning. User clusters obtained from using the geographical coordinates of their visited locations can reflect that those had ever visited the same of nearby locations tend to have similar taste of locations. User clusters based on their social connections assume that friends can drive their preferences, i.e., one may favor the locations visited by her friends. Clusters derived from the location visiting time of users reveal their habits of visiting locations along a day, e.g., some users may prefer to get up in the afternoon to start a traveling day and end up in the mid night. To perform user clustering, in the following, we first define three distance measures.

Definition 2 (*Geographical distance*) Given two users u_i and u_j with their visited location sets L_{u_i} and L_{u_j}, the geographical distance between u_i and u_j is defined by:

$$d_g\left(u_i, u_j\right) = \left(\sum\nolimits_{l_i \in L_{u_i}} \sum\nolimits_{l_j \in L_{u_j}} \frac{ED\left(l_i, l_j\right)}{\max ED}\right) / \left|L_{u_i}\right| \left|L_{u_j}\right|,$$

where $ED\left(l_i, l_j\right)$ is the spatial Euclidean distance between locations l_i and l_j, and $\max ED$ is a normalization term that is the largest spatial Euclidean distance between locations.

Note that one might think that the distance derived by the road network can reflect more realistic transportation cost between two locations. In this work, we aim to model the users' similarities based on the idea: two individuals possess similar location visiting behaviors if they visit similar geo-spatial activity areas. Therefore, as indicated by an existing study [35], it is reasonable to adopt the Euclidean distance between locations to approximate the real road network-based distance.

Definition 3 (*Social distance*) Given two users u_i and u_j with their friend sets F_{ui} and F_{uj}, the social distance between u_i and u_j is defined using Katz similarity [36], given by:

$$d_s\left(u_i, u_j\right) = 1 - \sum\nolimits_{\text{len}=1}^{\infty} \beta^{\text{len}} \cdot \left|\text{paths}_{u_i,u_j}^{\text{len}}\right|,$$

where $\text{paths}_{u_i,u_j}^{\text{len}}$ is the set of all paths with length len between u_i and u_j in the social graph, and the parameter is an exponential damping factor, which is typically set to 0.05.

Before we define the temporal distance, we need to first define the *visiting time distribution* (VTD) of a user u_i and the distance between two VTDs.

Definition 4 (*Visiting time distribution*) A *Visiting Time Distribution* (VTD) of user u_i is a probability distribution over time labels in hour, given by $\text{VTD}_{u_i}(t) =< (t_0, p_0), (t_1, p_1), \ldots, (t_{23}, p_{23}) >$, where $p_0 + p_1 + ... + p_{23} = 1.0$, where $p_t = \text{freq}_t/\text{freq}_{\text{all}}$, freq_t is the number of check-ins at time t by user u_i, and freq_{all} is her total number of check-ins (i.e., sum over all time t, $t = 0, 1, \ldots, 23$).

Given the time-aware preference of a user is represented by the visiting time distribution, the idea is that the distance between two users is shorter if they share higher overlap between their VTDs. While the *symmetric Kullback–Leibler* (KL) *Divergence* is a good choice to estimate the degree of overlap between two distributions, we adopt KL Divergence here. A smaller KL value indicates that two individuals tend to match the visiting time distribution derived from data and are supposed to be better in the same cluster.

Definition 5 (*Time distribution distance*) Given two VTDs, $\text{VTD}_{u_i}(t)$ and $\text{VTD}_{u_j}(t)$, the time distribution distance is defined using their *symmetric Kullback–Leibler (KL) Divergence*, given by:

$$D_{\text{KL}}(\text{VTD}_{u_i}(t) \,||\text{VTD}_{u_j}(t)) = \sum_t \text{VTD}_{u_i}(t) \log \frac{\text{VTD}_{u_i}(t)}{\text{VTD}_{u_j}(t)} + \sum_t \text{VTD}_{u_j}(t) \log \frac{\text{VTD}_{u_j}(t)}{\text{VTD}_{u_i}(t)}$$

Definition 6 (*Temporal distance*) Given two users u_i and u_j with visiting time distributions $\text{VTD}_{u_i}(t)$ and $\text{VTD}_{u_j}(t)$, their temporal distance is the time distribution distance $d_t\left(u_i, u_j\right) = D_{\text{KL}}(\text{VTD}_{u_i}(t) \,||\text{VTD}_{u_j}(t))$.

Finally, the geographical–social–temporal distance $d_{\text{gst}}\left(u_i, u_j\right)$ between u_i and u_j is defined by the weighted sum of $d_g\left(u_i, u_j\right)$, $d_s\left(u_i, u_j\right)$, and $d_t\left(u_i, u_j\right)$, i.e., $d_{gst}\left(u_i, u_j\right) = \omega_g \cdot d_g\left(u_i, u_j\right) + \omega_s \cdot d_s\left(u_i, u_j\right) + \omega_t \cdot d_t\left(u_i, u_j\right)$, where $\omega_g, \omega_s, \omega_t \in [0, 1]$ and $\omega_g + \omega_s + \omega_t = 1$.

With $d_{\text{gst}}\left(u_i, u_j\right)$, to group users, we take advantage of the *DBSCAN* clustering model [37], which is a density-based clustering algorithm. It should be noted that there are two decisive parameters in DBSCAN, ϵ and $MinPts$. ϵ is to determine the neighborhood of a

user u_i (i.e., geographical–social–temporal ϵ-neighborhood) while $MinPts$ is the minimum number of users in the neighborhood of a *core* user. We will empirically show how ϵ and $MinPts$ affect the results of user clusters and the performance of PTRP.

5 Mining time-aware transit patterns

Equipped with the inferred time labels on locations, we have complete and precise routes. At this stage, we mine *time-aware transit patterns* (TTP) for users in each cluster, which are regarded as the representative location transitions of a certain user preference, from the route database. A time-aware transit pattern mining (TTPM) algorithm is devised. First, we construct a collection of *location transit graph* (LTG) from time-labeled routes, in which a LTG can be considered as the location-transition behaviors of users *within one day*. Second, we mine all frequent patterns of length one (denoted as 1-patterns) from the LTG database. For each frequent k-pattern ($k \geq 1$), denoted by P, we build a projected database. Then we scan its projected database to find the local frequent 1-patterns e which is connected with P. For each e, we concatenate P with e to form a frequent (k+1)-pattern. The concatenations are recursively performed in a depth-first search manner until no more frequent closed patterns can be found. We also devise closure checking and pruning strategies to reduce unnecessary candidates so that the closed frequent time-aware transit patterns can be efficiently mined.

5.1 Definitions and notations

We represent the transition behaviors of users between locations within a day by *Location Transit Graph* (LTG), with a series of definitions that are used to mine TTPs.

Definition 7 (*Location transit graph*) A *location transit graph* LTG $g = (V, E)$ is a directed labeled graph, in which each node in the vertex set V is a location, and each edge (l_i, l_j) in the edge set E is a directed edge that represents a transit from location l_i to l_j and is associated with timestamps. An edge is represented by a 4-tuple, (v_s, v_d, t_s, t_e), where v_s and v_d are nodes, t_s is the starting time of the source v_s, and t_e is the arrival time of the destination v_d. For example, a transit from location L_1 to L_2 with time duration [30,32] (unit: hour) is (L_1, L_2, 14, 16).

Given a set of time-labeled routes which represents the location transitions *within one day*, we can construct a LTG g_i. A graph database $DB = \{g_1, g_2, \ldots, g_n\}$ containing n LTGs can be derived, where $1 \leq i \leq n$, and n is the total number of days. We sort the edges by time labels so that a LTG graph can be represented as an edge sequence, based on *edge order*: let $a = (s_1, d_1, t_{s1}, t_{e1})$ and $b = (s_2, d_2, t_{s2}, t_{e2})$ be two edges in a graph. $\alpha < b$ if (1) $t_{s1} < t_{s2}$, (2) $t_{s1} = t_{s2}$ and $t_{e1} < t_{e2}$, (3) $s_1 < s_2$, $t_{s1} = t_{s2}$, and $t_{e1} = t_{e2}$, or (4) $d_1 < d_2$, $s_1 = s_2$, $t_{s1} = t_{s2}$, and $t_{e1} = t_{e2}$.

Definition 8 (*Time-aware transit pattern*) A *Time-aware Transit Pattern* (TTP) is defined as $< (s_1, d_1, t_{s1}, t_{e1}), (s_2, d_2, t_{s2}, t_{e2}), \ldots, (s_h, d_h, t_{sh}, t_{eh}) >$, where $t_{s1} = 0$, and all the edges in the pattern are sorted in an ascending order. A TTP should follow *route connected constraint*, which is satisfied if $\forall\ s_i \in \{s_2, s_3, \ldots, s_h\}$ we can always find an edge (i, j) whose $i < j$, and $d_j = s_i$ or $s_j = s_1$. That says, such route is required to be represented as a connected graph.

Definition 9 (*Pattern existence*) A pattern $< (ps_1, pd_1, pt_{s1}, pt_{e1}), (ps_2, pd_2, pt_{s2}, pt_{e2}), \ldots, (ps_m, pd_m, pt_{sm}, pt_{em}) >$ is contained by a graph $< (gs_1, gd_1, gt_{s1}, gt_{e1}), (gs_2, gd_2, gt_{s2},$

$gt_{e2}), \ldots, (gs_n, gd_n, gt_{sn}, gt_{en}) >$ if there exists a sequence of integers $j_1 < j_2 < \ldots < j_n$ such that $pu_i = gu_{ji}, pl_i = gl_{ji}, pv_i = gv_{ji}, pt_{si} \geq gt_{sji} - gt_{sj1}$, and $pt_{ei} \leq gt_{eji} - gt_{sj1}$, $i = 1, 2, \ldots, n$.

A pattern may contain many edges, which could lead to inefficiency and less usefulness for route planning. Therefore, we define constraints to reduce unnecessary ones. For example, $P\langle(L_4, L_3, 0, 2), (L_3, L_6, 9, 10)\rangle$ is not helpful since the timespan between two edges are too large. The user may only be interested in such patterns with large time gaps. We use a maximum timespan threshold *maxgap* to rule out such redundant patterns. Let the timespan between edges $(s_i, d_i, t_{si}, t_{ei})$ and $(s_k, d_k, t_{sk}, t_{ek})$ be $|t_{sk} - t_{ei}|$. We mine the patterns which follow the *maxgap* constraint: for each kth edge e_k ($k \geq 2$) in the pattern P, we must find another edge e_i from e_1 to e_{k-1} that makes $|t_{sk} - t_{ei}| \leq maxgap$, where *maxgap* is a user-specified threshold. Note that t_{ei} is sometimes larger than t_{sj} due to the flexible route construction.

Definition 10 (*Super-pattern*) A pattern $< (ps_1, pd_1, pt_{s1}, pt_{e1}), (ps_2, pd_2, pt_{s2}, pt_{e2}), \ldots, (ps_m, pd_m, pt_{sm}, pt_{em}) >$ is a super-pattern of another pattern $< (qs_1, qd_1, qt_{s1}, qt_{e1}), (qs_2, qd_2, qt_{s2}, qt_{e2}), \ldots, (qs_n, qd_n, qt_{sn}, qt_{en}) >$ if there is a sequence of integers $j_1 < j_2 < \ldots < j_n$ so that $ps_i = qs_j, pd_i = qd_{ji}, qt_{si} \geq pt_{sji} - pt_{sj1}$, and $qt_{ei} \leq pt_{eji} - pt_{sj1}, i = 1, \ldots, n$.

For example, $P = < (L_1, L_2, 0, 2), (L_2, L_4, 2, 3), (L_4, L_5, 4, 5), (L_5, L_6, 6, 9) >$ is a super-pattern of $Q_1 = < (L_1, L_2, 0, 2), (L_2, L_4, 2, 3), (L_4, L_5, 4, 5) >$, where $j_1 = 1$, $j_2 = 2$ and $j_2 = 3$. Moreover, P is a super-pattern of $Q_2 = < (L_1, L_2, 0, 2), (L_2, L_4, 2, 3), (L_4, L_5, 4, 5), (L_5, L_6, 6, 7) >$, where $j_1 = 1$, $j_2 = 2$, $j_3 = 3$ and $j_4 = 4$.

Definition 11 (*Frequent pattern and closed pattern*) A pattern P is *frequent* if the *support* value $sup(P)$, defined by the number of LTG graphs containing P in the database, is not less than *minsup*, where *minsup* is a user-specified minimum support threshold. A frequent pattern P is *closed* if there is no super-pattern of P with the same support. For example, assume $P = < (L_1, L_2, 0, 2), (L_2, L_4, 2, 3), (L_4, L_5, 4, 5), (L_5, L_6, 6, 9) >$ is a super-pattern of $Q = < (L_1, L_2, 0, 2), (L_2, L_4, 2, 3), (L_4, L_5, 4, 5) >$, and their supports are the same. Then Q is not closed.

Definition 12 (*Prefix and postfix*) Given a pattern $P = < (S_1, D_1, t_{s1}, t_{e1}), (S_2, D_2, t_{s2}, t_{e2}), \ldots, (S_m, D_m, t_{sm}, t_{em}) >, Q = < (S_1, D_1, t_{s1}, t_{e1}), (S_2, D_2, t_{s2}, t_{e2}), \ldots, (S_i, D_i, t_{si}, t_{ei}) >$ is called a prefix of P, and $R = < (S_{i+1}, D_{i+1}, t_{si+1}, t_{ei+1}), (S_{i+2}, D_{i+2}, t_{si+2}, t_{ei+2}), \ldots, (S_m, D_m, t_{sm}, t_{em}) >$ is called the postfix of P, $1 \leq i \leq m$.

Definition 13 (*Projected database*) The P-projected database, denoted as $DB|_p$, contains all postfixes of the graph possessing P in database DB, where P is a frequent pattern in DB.

Note that there is a fundamental difference between time-aware transit patterns and existing work. In our problem, the input database of *Location Transit Graphs* is constructed from the inferred time-labeled routes, and each edge in the location transit graph is represented by a 4-tuple, (v_s, v_d, t_s, t_e), where v_s and v_d are nodes, t_s is the starting time of the source v_s, and t_e is the arrival time of the destination v_d. Such input is quite different from existing work. Comparing to traditional pattern mining work, we further consider the time information t_s and t_e, to make the patterns meaningful since locations in each trip route are usually associated with time information which inspires us to propose the PTRP. We not only propose the new definitions of patterns, super-patterns, and prefix & postfix, but also introduce the

route connected constraint and the *maxgap* constraint. These changes make our problem significantly different from what the traditional techniques are to mine in the database of general labeled graphs or sequential patterns.

5.2 Frequent patterns enumeration

We construct a *TTP pattern tree* to enumerate frequent patterns where each node represents a frequent TTP pattern, and the level at which the node is located represents the length of the frequent pattern, which is the number of edges. A pattern of length k is called a k-pattern. For example, the frequent 1-patterns are recorded at level 1 and the frequent 2-patterns are at level 2. Moreover, a pattern at level k is derived from the pattern of its parent node at level k-*1* ($k \geq 2$). The root of the tree is labeled by $\varnothing$. To generate all frequent patterns, we scan the database once to find all frequent 1-patterns and build the corresponding projected database. Then the frequent 1-patterns are added to the level 1 of the *TTP pattern tree*.

We recursively extend a frequent k-pattern P ($k \geq 1$) at level k to get its frequent super $(k+1)$-patterns in a depth-first search. To find the frequent super-patterns of P, we scan the projected database of P and find local frequent 1-patterns which are connected with P (i.e., the destination of P's kth edge is same as the source location of 1-patterns), and the timespan between P and each frequent 1-pattern found is not greater than *maxgap*. For each frequent 1-pattern q, we concatenate P with q to form a frequent $(k+1)$-pattern. In the meanwhile, we adopt lemma 1 to remove the redundant patterns.

Lemma 1 (Same projected database removal) *If P_1 is a super-pattern of P_2 and both share the same projected database, P_2 can be removed from TTP pattern tree.*

Proof Since P_1 is a super-pattern of P_2 and both share the same projected database, every pattern derived from P_2 is contained by a pattern derived from P_1, and both derived patterns have same support. Thus, P_2 and the patterns generated from P_2 should be pruned because they are not closed. □

5.3 Closure checking and pruning strategies

We have generated frequent TTP patterns. However, some patterns might not be closed. To eliminate to redundant patterns, we devise two closure checking and pruning strategies to check if a pattern is closed.

Lemma 2 (Forward redundant checking scheme) *A pattern P is not closed if there exists a frequent 1-pattern e in P's projected database, whose support is equal to P's support, and e is connected with P.*

Proof If there is a frequent 1-pattern e in P's projected database whose support is equal to P's support, it indicates we can always find another frequent pattern which is formed by concatenating P and e and its support is equal to P's support. Thus, P must not be closed. □

We illustrate *forward redundant checking scheme* as follows. Assume we have a new frequent pattern $< (L_1, L_2, 0, 2), (L_2, L_3, 2, 3) >$, if we find a 1-pattern $e\langle(L_2, L_6, 3, 4)\rangle$ which occurs in every graph in the projected database, we can ensure that P is not closed. Since P is the sub-pattern of the pattern formed by concatenating P and e, which has the same support as P, P is not closed.

Lemma 3 (Backward Redundant Checking Scheme) *A pattern P is not needed to grow if there is a frequent 1-pattern e before P, whose support is equal to P's support, the timespan between e and P is not greater than maxgap, and e is connected with P.*

Proof If there is a frequent 1-pattern e before P, whose support is equal to P's support and the timespan between e and P is not greater than *maxgap*, it means that we can always find another frequent pattern which is formed by concatenating e and P and its support is equal to P's support. Each pattern generated from P is contained by the pattern generated from concatenating P and e and both patterns have the same support. Thus, P does not need to be grown and it is not closed. □

Assume $P\langle(L_2, L_3, 2, 3), (L_2, L_6, 3, 4)\rangle$ is a frequent pattern. If we find a pattern (say, $< (L_1, L_2, 0, 2) >$, before $< (L_2, L_3 2, 3) >$) in every graph containing P and the timespan between $< (L_1, L_2, 0, 2) >$ and P is not greater than *maxgap*, we can conclude that P needs no to grow. It is because of that there must exist another frequent pattern formed by concatenating $< (L_1, L_2, 0, 2) >$ and P.

Algorithm 1. TTPM algorithm

Input: a LTG database *DB*, a maximum timespan *maxgap*, and a minimum support *minsup*.
Output: all closed frequent patterns *TTP*.

1: Scan the database *DB* once to find all frequent 1-patterns and build the projected database for each frequent 1-pattern found;
2: *TTP* = ∅;
3: **for each** 1-pattern found *P* **do:**
4: **if** (*P* passes *Backward Redundant Checking Scheme*) **then:**
5: **if** (*P* passes *Forward Redundant Checking Scheme*) **then:**
6: *TTP* = *TTP* ∪ *P*;
7: *TTP* = *TTP-Growth*(*P*, $D|_p$, *minsup*, *maxgap* , *TTP*);
8: **return** TTP;

Algorithm 2. TTP-Growth sub-procedure

Input: a prefix pattern *P*, a projected database $D|_p$, a minimum support *minsup*, and a maximum timespan *maxgap*.
Output: all closed frequent patterns *TTP*.

1: Scan $D|_p$ once to find all frequent 1-patterns in the projected database, where the timespan between *P* and each frequent 1-pattern found is not greater than *maxgap*;
2: **for each** frequent 1-pattern found q **do:**
3: **if** (q is connected with any edge of *P*) **then:**
4: Let $R = P \oplus q$ and build the projected database of R ;
5: **if**(*R* passes *Backward Redundant Checking Scheme*) **then:**
6: **if**(*R* passes *Forward Redundant Checking Scheme*) **then:**
7: *TTP*=*TTP* ∪ *P*;
8: *TTP* = *TTP-Growth*(*P*, $D|_p$, *minsup*, *maxgap* , *TTP*);
9: **return** TTP;

5.4 The TTPM algorithm

We find all frequent 1-patterns in the projected database, where the timespan between P and each frequent 1-pattern found is not greater than *maxgap*. We describe the TTPM algorithm with a sub-procedure, TTP-Growth in the following.

TTPM first scans the graph database once to find all frequent 1-patterns and builds the projected database. For each frequent 1-pattern P, if the frequent 1-pattern P passes the *forward redundant checking scheme*, we add P to *TTP* in steps 4–6. We use the *backward redundant checking scheme* strategy to check whether a frequent pattern needs to be grown or not. If this is the case, in step 7, we call the TTP-Growth sub-procedure, which grows

a pattern P to find all closed super-patterns of P. In TTP-Growth sub-procedure, from the projected database of P, we find all frequent 1-patterns in step 1. For each frequent 1-pattern q, if q is structurally connected with P, we concatenate P and q (denoted by $P \oplus q$) to form a new pattern R and build its projected database in steps 3–4. If R passes forward and backward redundant checkings, we add R to *TTP* in steps 5–7. In step 8, we recursively use this sub-procedure to generate all the closed patterns.

Lemma 4 *All TTPM-generated patterns are frequent and closed.*

Proof Every 1-pattern must be frequent in step 1 of TTPM. By concatenating k-pattern P and 1-pattern q in step 4 of TTP-Growth, every k-pattern ($k \geq 2$) must be frequent. Then we can conclude that every pattern generated by TTPM will be frequent. We also apply the forward and backward redundant checking in steps 5–6 of TTP-Growth. Thus, all the frequent patterns kept in *TTP* must be closed. Therefore, we can conclude all patterns mined by TTPM are frequent and closed. □

Lemma 5 *TTPM can find all the frequent closed patterns.*

Proof By scanning the database, we can find all the frequent 1-patterns. The concatenated operations of k-patterns in step 4 of TTP-Growth can be used to find all frequent (k+1)-patterns, where $k \geq 2$. By the closure checking and pruning strategies, we can prune the nodes in a TTP-tree whose patterns are not closed without missing any closed frequent patterns. Thus, all the patterns kept in *TTP* must be frequent and closed. Therefore, we can conclude that *TTPM* finds all frequent closed patterns. □

Theorem 1 *The time complexity of the TTPM algorithm is $O(n \times l \times p)$, where n is the number of LTGs in the database, l the average length in a LTG, and p the number of frequent TTP patterns.*

Proof In the worst case, every pattern generated needs to do $(n \times l)$ traces to count the local frequent edges to concatenate new patterns of next level. Assume the total pattern number is p in our database. Thus, the total size is no more than $n \times l \times p$. However, we must check each new pattern produced is closed or not. Assume $|pa|$ is the average lengths of all patterns, all closure checking execution can be computed in time $p \times |pa|$. Therefore, the total time of *TTPM* can be finished in $O(n \times l \times p + p \times |pa|)$. However, it is obvious that the execution time of $p \times |pa|$ is extremely less than $n \times l \times p$. Thus, the time complexity of *TTPM* is $O(n \times l \times p)$. □

6 Preferred time-aware route planning

Equipped with the inferred time labels on locations as well as the mined time-aware transit patterns for each user cluster, we can plan the preferred time-aware routes. We provide two general route-planning queries for users, *source query* and *source–destination query*.

Definition 14 (*Source query*) The input consists of (a) the source/starting location $Q_s = (l_s, t_s)$, where l_s contains the longitude and the latitude of such location in check-in data, and t_s is the starting timestamp (e.g., 8AM), and/or (b) the number k of locations in the final route.

Definition 15 (*Source–destination query*) The input consists of (a) the source/starting location $Q_d = (l_s, t_s, l_d)$, where l_s contains the longitude and the latitude of such location in

check-in data, t_s is the starting timestamp (e.g., 8AM), and l_d is the destination/end location, and/or (b) the number k of locations in the final route. Note that the number k of locations can be either specified or not.

We develop a *Preferred Time-aware Route Planning* (PTRP) method, which consists of three parts. First, we construct a *Routable Map*, which models the time-aware transitions between locations, in Sect. 6.1. Second, we propose a time-aware measure, *Route Visiting Goodness* (RVG), to determine whether the route with time-labeled locations is proper to visit, in Sect. 6.2. Third, we develop the *Preferred Route Search* (PR-Search) algorithm, to generate final routes in Sect. 6.3.

6.1 Routable map construction

We construct the *routable map* to model the representative transition behaviors between locations for users in the same cluster. Routable map will be used in the proposed *Preferred Route Search* to guide the generation of final routes. When recommending routes for a particular user, the routable map constructed from users belonging to the same user cluster will be utilized. Technically, the routable map is a directed graph $H = \left(V^H, E^H\right)$, in which each node $v \in V^H$ is a location and each edge $e \in E^H$ is a transition between locations. In addition, each node is associated with a set of time labels which represents when users had ever visited here. We use the mined time-aware transit patterns of a user cluster to construct a routable map. Given a set of TTP patterns mined from time-labeled routes of a user cluster, the construction of routable map consists of three steps. First, we create nodes for each location that appear in TTPs. Second, for each transition in each TTP, we construct a directed edge that connects the source to the destination location in the routable map. Third, for each location in the routable map, we find a set of the corresponding time labels occurring in the transitions of TTPs, and associate such set with the location.

6.2 Route visiting goodness

We devise a novel time-aware measure, *Route Visiting Goodness* (RVG), to determine whether it is proper to visit places along a route with given visiting times. The RVG consists of two parts: (a) measuring the goodness of visiting a location at a given time, and (b) measuring the extent of goodness when using a given transition time between two locations. While Definition 4 defines the visiting time distribution at a user, *VTD* can be also defined at a location l_i by computing the check-in probability of that location at each time (i.e., hour). Using the same idea, we define the transition time distribution between locations, which can be derived using the inferred time labels in those routes containing l_i.

Definition 16 (*Transition time distribution*) We define the *Transition Time Distribution* (TTD) between location l_i and l_j as the probability distribution over time duration Δ in hour, $TTD_{li,lj}(\Delta) =< (\Delta_1, p_1), (\Delta_2, p_2), \ldots, (\Delta_{23}, p_{23}) >$, where $p_1 + p_2 + ... + p_{23} = 1.0$, $p_t = \text{freq}_{\Delta_t}/\text{freq}_{\text{all}}$, freq_{Δ_t} is the number of transits using Δ_t time from l_i to l_j, and freq_{all} is the total number of transits from l_i to l_j (i.e., sum over all users and over all time Δ_t, $\Delta_t = 0, 1, \ldots, 23$).

We elaborate how to use *VTD* of a location l_i to measure the goodness when visiting l_i at time t. Note that the goodness measure of *TTD* follows the same idea. Assume we want to know how well a decision is to visit a place at time t, given the location's *VTD*, we propose to first generate a thin Gaussian distribution $G\left(t; \mu, \sigma^2\right)$ whose mean value μ is at time t

with a very small variance σ^2 (e.g., standard deviation is 1). And then we can transform the original task into measuring the difference between the Gaussian distribution with the learnt *VTD* of such location. We use the time distribution distance (Definition 5) between $G\left(t;\mu,\sigma^2\right)$ and $\mathrm{VTD}_l\left(t\right)$ to represent the fitness of the assignment. Smaller time distribution distance indicates better match between the assignment and the distribution learned from data. Consequently, we formally define the Route Visiting Goodness score, $RVG(s)$, of a route $r\langle(l_1,t_1),(l_2,t_2),\ldots,(l_n,t_n)\rangle$, as a combination of the popularity of places together with the fitness of each location over time:

$$\mathrm{RVG}\left(r\right)=\beta\times\left(\prod_{i=1}^{n}D_{\mathrm{KL}}\left(G\left(t;t_i,\sigma^2\right)||\mathrm{VTD}_{l_i}\left(t\right)\right)\times\frac{1}{\mathrm{pop}\left(l_i\right)}\right)^{\frac{-1}{n}}$$
$$+\left(1-\beta\right)\times\left(\prod_{i=1}^{n-1}D_{\mathrm{KL}}\left(G\left(t;\Delta_{i,i+1};\sigma^2\right)||\mathrm{TTD}_{l_il_j}\left(\Delta\right)\right)\right)^{\frac{-1}{n-1}}$$

where $\mathrm{pop}\left(l_i\right)=N\left(l_i\right)/N_{\max}$, $N\left(l_i\right)$ is the number of recording actions performed on location l_i, and $N_{\max}$ is the maximum number of actions among all the locations. The parameter β is used to control the preference or importance on either proper visiting time or proper transition time. Higher β values refer to prefer the visiting quality on locations, while lower β values indicate the route should be strictly planned based on regular transportations. If places in a route s are visited during the proper time period, the $RVG\left(r\right)$ value would be higher.

6.3 Preferred route search

Given either the source query $Q_s=(l_s,t_s)$ or the source–destination query $Q_d=(l_s,t_s,l_d)$ with the preferred number of locations k if any, we propose *Preferred Route Search* (*PR-Search*) algorithm to generate the final routes for recommendation. The PR-Search is based on the routable map and the route visiting goodness. The fundamental idea is to find a directed path $r\langle(l_1=l_s,t_1=t_s),(l_2,t_2),\ldots,(l_k,t_k)\rangle$ in the routable map such that the value of $RVG\left(r\right)$ is maximized.

The PR-Search consists of three steps, and the detailed algorithm is described in Algorithm 3. We first construct the initial route r_0 by including the starting location l_s (line 1). A $PriorityQueue$ is used to maintain the route with the highest RVG score (line 2). Each element in $PriorityQueue$ consists of a route s and the corresponding RVG score. $PriorityQueue$ automatically sorts its elements according to their RVG scores. We add r_0 to initialize the $PriorityQueue$. After setting the final route r as the initial one r_0 (line 3), we perform the iterative expansion search process until the route r is constructed up to length k (line 5–13). For each iteration, the last location l_{last} in the route r with the highest RVG score is identified (line 6 and line 13) and each possible next visiting location l_{next} from the routable map H is put into a candidate set C (line 7). Then for each candidate next location l_c, and for each time label t_c in the time label set T_{l_c} of l_c, we can derive the score $\mathrm{RVG}\left(r_{\mathrm{tmp}}=r+\langle(l_c,t_c)\rangle\right)$. We put $\mathrm{RVG}\left(r_{\mathrm{tmp}}\right)$ together with the corresponding route r_{tmp} into $PriorityQueue$ (line 8–12). The $PriorityQueue$ will then pick the next best route and location to conduct the further expansions (line 13). Finally (line 14), the route r is reported as the final generated route for recommendation.

Algorithm 3. Preferred Route Search (PR-Search)

Input: (a) the constructed routable map; (b) the route planning query $Q_d = (l_s, t_s, l_d)$, if $l_d = \emptyset$: the source query, if $l_d \neq \emptyset$: the source-destination query; (c) k: the number of locations.

Output: a time-aware route $r = \langle (l_1, t_1), (l_2, t_2), \ldots (l_d, t_k) \rangle$

$r_0 = \langle (l_1 = l_s, t_1 = t_s) \rangle$.
$PriorityQueue = \{(r_0, 0)\}$.
$r = r_0$.
$l_{last} = null$.
while $|r| < k$ or $l_{last} \neq l_d$ **do:**
    $l_{last} = r.endLocation$.
    $C = \{l_{next} | l_{last} \rightarrow l_{next}$ in routable map $H\}$.
    for each $l_c \in C$ **do:**
        for each $t_c \in T_{l_c}$ **do:**
            $r_{tmp} = r + \langle (l_c, t_c) \rangle$.
            $score = RVG(r_{tmp})$.
            $PriorityQueue.Insert((s_{tmp}, score))$.
    $r = PriorityQueue.Pull()$.
return r.

The time complexity of *PR-Search* depends on three parts: the number of search expansion N, the number of candidates of next locations $|C|$, the number of time labels of the next location $\left|T_{l_c}\right|$, and the operation of the $PriorityQueue$ $O(\log(|E|))$, where E is the set of elements in the $PriorityQueue$. We can simply write down the complexity as $O\left(N \times |C| \times \left|T_{l_c}\right| \times \log(|E|)\right)$. In the worst case, both the candidate set C_i of each next location and the number of search expansion could be all the locations in the routable map, while $\left|T_{l_c}\right|$ is a constant (24 hours). By denoting the set of all locations as L, the worst complexity is $O(N \times |L| \times \log(|E|))$. Note that since the first two stages, inferring visiting time and mining time-aware transit patterns, can be done offline while the planning is the online stage, we focus on discussing the complexity of such online part.

7 Experiments

We use a large-scale check-in data from Gowalla [8] for the experiments. The dataset contains 6,442,890 check-in records from February 2009 to October 2010. The total number of check-in locations is 1,280,969. By constraining a route as a sequence of check-in locations of a user within a day, we have 1,136,737 routes whose lengths are more than one and the average route length is 4.09. We extract two check-in subsets falling into cities of New York and San Francisco because the data of such two cities are the more plentiful. Some statistics are reported in Table 1.

The experiments consist of three evaluation plans. The first is the evaluation on the inference of visiting time labels in Sect. 7.1. We aim to show the accuracy of the proposed *LocTimeInf* method under the setting of partially time-labeled locations and the cold-start new locations. In Sect. 7.2, we evaluate the time efficiency of proposed TTPM algorithm,

Table 1 The statistics of the two check-in data subsets

	Total number of check-ins	Avg. route length	Number of locations
New York	103,174	4.46	21,973
San Francisco	187,568	4.09	15,406

comparing to two modified state-of-the-art methods. The third plan is for the time-aware route planning, as reported in Sect. 7.3. We aim to show the accuracy of the proposed PTRP model for a route prediction task, comparing to a series of competitors. The effects of a series of parameters used in *LocTimeInf*, TTPM, and PTRP are discussed. Note that the following experiments are also conducted on the Foursquare check-in data [38], and derive similar results as Gowalla data. However, due to the page limit, here we do not report the results on Foursquare.

7.1 Evaluation on visiting time inference

We design two evaluation tasks to evaluate the effectiveness of *LocTimeInf*. The first is to test if the visiting time distribution (VTD), which can be derived from the inferred visiting time labels, on each location can be accurately inferred. We use the *symmetric Kullback–Leibler* (*KL*)*Divergence*, to measure the difference between the ground-truth *VTD* and the inferred *VTD*, the smaller the better. The average *KL* Divergence value over all unlabeled locations will be reported. In short, given a set of routes with a specified time-label ratio, we will show the *KL* difference between the ground-truth VTD(l_i) and the inferred VTD'(l_i) for each location l_i in the routes. The second task is to compute the accuracy of the inferred time labels. We define an accuracy measure Hit Rate, HitRate $=$ PL/TL, where PL is the number of successfully predicted check-in locations and TL is the number of unlabeled check-in locations over all time-unlabeled routes. Higher hit rate indicates better inference quality. We set $\pi = 7$. By varying the time-label ratio, we randomly select the time-labeled routes while the time labels of the remaining routes are removed (as the testing set) and regarded as the ground truths. The experiment is conducted on two settings: the first is partially time-labeled data (i.e., only few time labels), and the second is the cold-start new locations (i.e., no visiting time labels).

Partially labeled data In this experiment, for *KL*, we develop two baseline methods: (a) *normal distribution* generates thin Gaussian distributions whose means locate at the most frequent check-in time in the labeled data. (b) We generate *VTD*s from the partially *labeled data*, which would serve as a strong baseline. *Normal distribution* assumes that every location tends to have the same and regular visiting behaviors over users, and the regularity is determined by the most frequent check-in time at each location. *Partially labeled data* assume that even using rare check-in records on a location is capable of representing such location's visiting time behaviors of users. For hit rate, the baseline chooses the most frequent check-in time label as the predicted one from partially time-labeled routes. Figures 2 and 3 show the results. We can find *LocTimeInf* significantly outperforms the competitors in both cases, even when the time-label ratio is very low (e.g., 0.05%). As the time-label ratio increases, it is natural that the performance of the competitors is improved, but *LocTimeInf* still has the best results. We think it is because of that two competitors might embody some bias, i.e., not all time labels follow normal distribution while partially labeled data cannot fully capture the time-aware behaviors of users. In contrast, *LocTimeInf* considers both geographical and contextual features so that the accuracy can be boosted.

In constructing RCGs, we retain π edges with top-π highest weights so that the efficiency can be boosted. We use the partially labeled data to show the effect of π on the effectiveness and time efficiency. We report the average inference time (in second). Figure 4 shows that the average inference time grows as π becomes larger, because higher π will include more edges in RCG and lead to higher complexity. We can also find that the KL values are not affected by π, unless RCG is a complete graph that might include noise links in the inference.

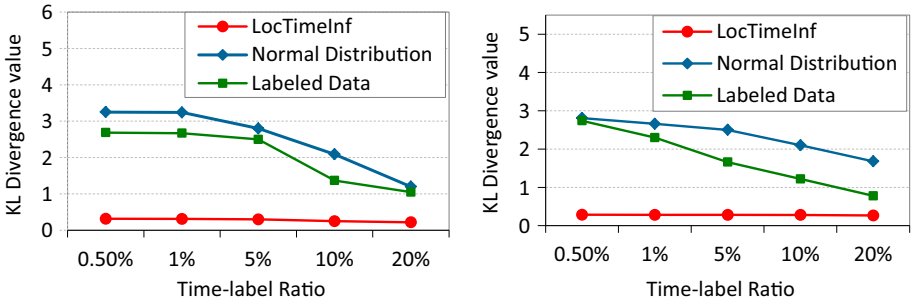

Fig. 2 Average KL divergence values for New York (left) and San Francisco (right) in partially labeled experiment

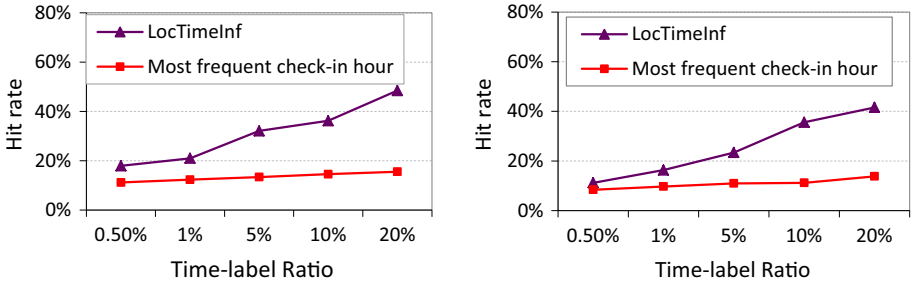

Fig. 3 Hit rates for New York (left) and San Francisco (right), in partially labeled experiment

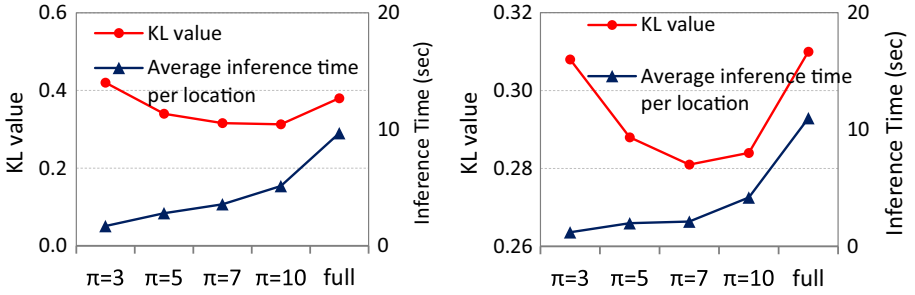

Fig. 4 Average KL divergence values and corresponding inference time for New York (left) and San Francisco (right), by varying the π in partially labeled experiment

The results show that π performs well around 7–10 considering both KL value and inference time. In contrast, we do not suggest small π or complete graph.

Cold-start new locations We evaluate if the proposed auxiliary RCG can solve the cold-start problem. Since new locations are assumed to have no time labels, *normal-distribution* and *labeled-data* methods cannot be applied. We alternatively use spatial kNN to select top-k locations ($k = 3$) which has least geo-distances to new locations, and apply the *normal-distribution* and *labeled-data* methods as the competitors. Figures 5 and 6 show the results. The overall performance is naturally worse than the previous experiment. However, the *LocTimeInf* still can not only outperform the competitors, but also is nearly insensitive to

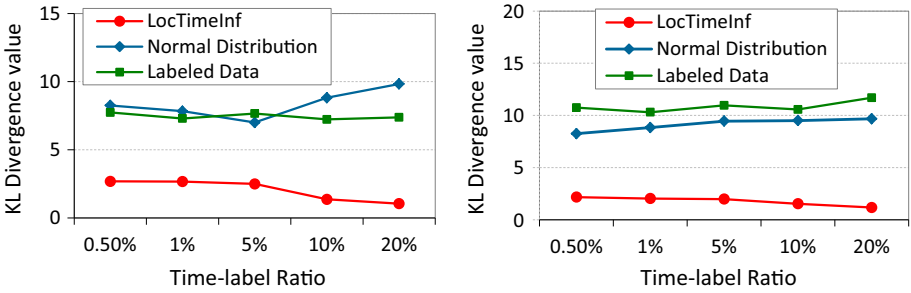

Fig. 5 Average KL divergence values for New York (left) and San Francisco (right), in the cold-start experiment

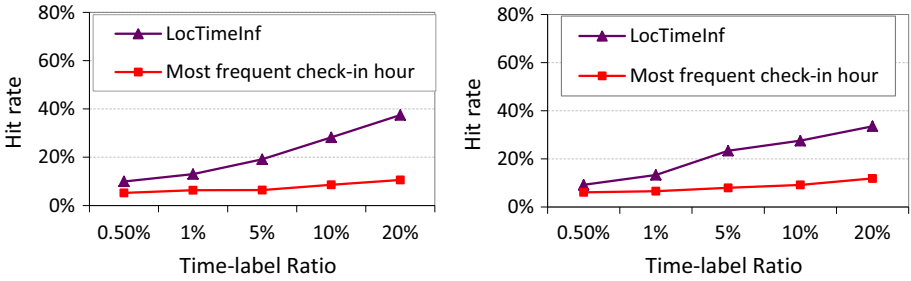

Fig. 6 Hit rates for New York (left) and San Francisco (right), in the cold-start experiment

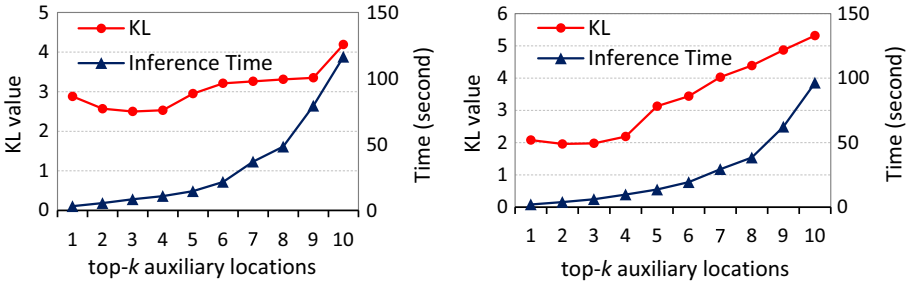

Fig. 7 Average KL values and run time in second for New York (left) and San Francisco (right) in the cold-start experiment

the varied time-label ratio. The performance of two competitors is unstable since nearest locations selected by spatial kNN does not guarantee the exhibitions of similar visiting time distributions.

Recall we discover top-k *auxiliary locations* to solve the cold-start problem. We conduct an experiment to show the effect of k (from 1 to 10) on effectiveness and time efficiency using *LocTimeInf*. The time-label ratio is set as 5%. Figures 7 and 8 show the results. We can find that in general $k = 2 - 4$ can have lower KL values, higher hit rates, and shorter inference time. We think it is because higher k values will not only include noise locations but also increase the computational complexity of inference.

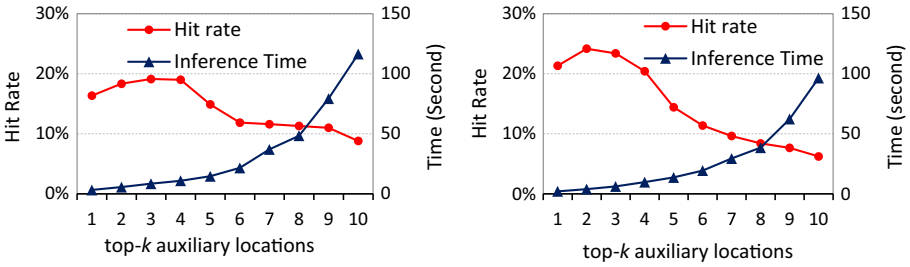

Fig. 8 Hit rates and run time in second for New York (left) and San Francisco (right) in the cold-start experiment

7.2 Evaluation on time-aware pattern mining

We evaluate the time efficiency TTPM using real check-in data. Since no existing work tackles the time-aware transit pattern mining problem. We compare TTPM with a modified Apriori [39] and a modified TAS algorithm [40]. The modified Apriori generates frequent patterns level by level in a breadth-first search. At each level, it combines a frequent k-pattern with another frequent k-pattern to generate a candidate $(k+1)$-pattern. For each candidate $(k+1)$-pattern, we scan the database to count its support and check if it is frequent. Each candidate $(k+1)$-pattern should follow the *maxgap* constraint. The process repeats until no more frequent patterns can be generated. The modified Apriori uses anti-monotone property to prune redundant candidates. On the other hand, TAS is a state-of-art method adopting a prefix projection to efficiently mine the temporally annotated sequential patterns. We modify TAS to not only fit the *maxgap* constraint and the *Route Connected Constraint*.

Figure 9a, b shows the runtime versus *minsup* varied from 1 to 20%, and *maxgap* = 10%. TTPM runs faster than the modified Apriori and TAS. As *minsup* gets smaller, the runtime of TTPM increases slowly, while the other two are more sensitive. We think it is due to that TTPM needs to scan the database only once and remove impossible candidates. Figure 9c, d shows the runtime versus *maxgap* in NY and SF, *maxgap* is varied from 0 to 4, and *minsup* = 10%. As *maxgap* increases, the number of frequent patterns increases, and thus the runtime increases. Nevertheless, TTPM is still the most efficient and scalable. The modified Apriori and TAS need more time due to the drastic increases of candidates generated at each level without checking/pruning strategies, as *maxgap* increases.

7.3 Evaluation on preferred time-aware route planning

We design a *time-aware route cloze test* to validate the quality of the recommended routes. Given real and popular trip routes with time stamp in each location, through randomly removing m consecutive locations in each route, it is possible to test whether a method can successfully predict the missing locations. With the increasing of m, consecutive removal of locations does impose a decent level of difficulty to this cloze test. It is because that as m increases, information that can be used becomes sparse, and mistakes in the earlier positions can lead to follow-up errors in the next positions to be predicted. We re-define Hit Rate as the accuracy measure for this cloze test. Given a total N removals of locations over all routes, and assuming M places out of N is successfully predicted, the hit rate is defined as M/N. Higher hit rate indicates better quality.

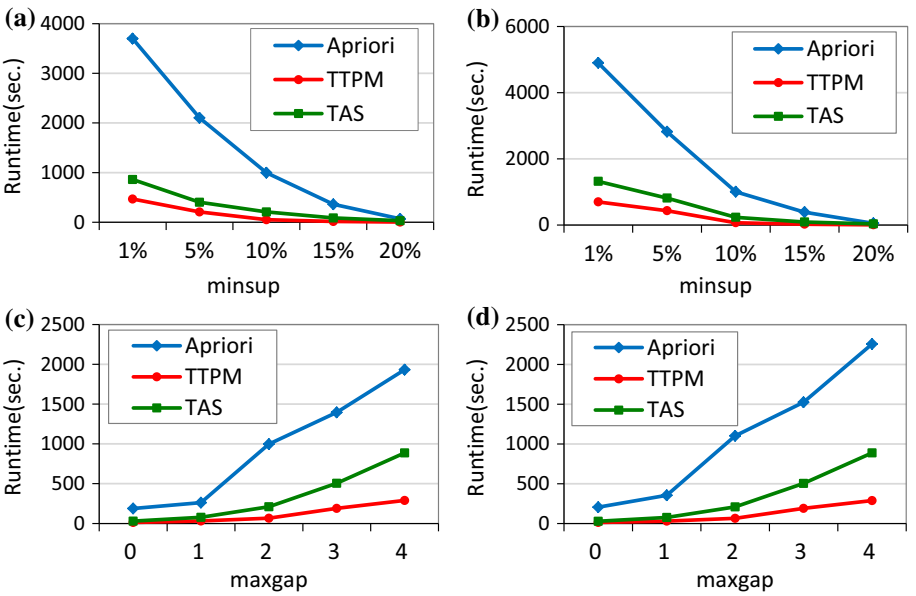

Fig. 9 Runtime versus *minsup* in **a** NYC and **b** SF, and versus *maxgap* in **c** NYC and **d** SF, using Gowalla data

Note that the realistic routes is not necessary to possess the highest RVG scores. But if we generate the unrealistic routes with highest RVG scores and treat them as the ground truth, it would be less meaningful since they cannot reflect the real-world users' location visiting trajectories. Therefore, we use the routes generated by users in Gowalla as the ground truth, and evaluate whether our method can capture users' real routes.

Evaluation settings To examine whether the inferred time labels can help ***PTRP*** achieve higher quality, we do following settings so that the effect of *LocTimeInf* can boost the performance. This enhanced method is denoted by ***PTRP+LocTimeInf***. First, for each location l_i that appears in n_i routes, we randomly divide the participated n_i routes into time-labeled set (90%) and time-unlabeled set (10%). Second, we perform *LocTimeInf* to infer the visiting time labels of location instances in time-unlabeled set. We execute such step up to 10 times to make sure that all the routes have ever served as the time-unlabeled ones. In other words, the time labels in all the locations should be inferred and/or corrected by *LocTimeInf*. Third, for each instance, by comparing the difference between the inferred time label t_i and original time label t_o, we can identify top-$p$percent check-ins with highest differences from the inferred ones. Such top-p percent check-ins are considered as the noise or abnormal ones and are removed in the evaluation datasets. By applying the above procedure, for each location we obtain a new *VTD* which is supposed to lead to more accurate *route visiting goodness* and have better performance on the location cloze test. Due to the page limit, we solely choose $p = 10\%$. The enhanced method (i.e., ***PTRP+LocTimeInf***) will be compared with the pure *PTRP* without *LocTimeInf* (denoted by ***PTRP***), and the corresponding version without using the user clusters to mine the TTPs and construct the routable map, i.e., ***TRP+LocTimeInf*** and ***TRP*** [41]

More competitive methods We compare the proposed method with a strong method, *Guidance Search* [9], which consists of a novel heuristic satisfaction function to guide the search

toward the destination location, and a backward checking mechanism to ensure the effectiveness of the constructed route. We also design the following baseline competitors, which search in the check-in database in a greedy manner to find the route that not only satisfies query requirement, but also maximizes a certain objective function $f(r)$.

- *Distance-based approach* This method chooses the closest location to the current spot as the next to move to. Therefore, it measures the quality of a route by using the objective function $f_{\text{dist}}(r) = \sqrt[n]{\prod_{i=1}^{n}(1/D(l_i, l_{i-1}))}$, where $D(l_i, l_{i-1})$ is the geographical distance between locations.
- *Popularity-based approach* This method chooses the most popular spot of a given time to be the next spot. It rates the route using the goodness function $f_{\text{pop}}(r) = \sqrt[n]{\prod_{i=1}^{n}(N(l_i)/N_{\max})}$.
- *Forward heuristic approach* The forward heuristic chooses a location l_i that possesses the largest bi-gram probability with the previous location $P(l_i|l_{i-1})$ as the next location. Its goodness function is $f_{\text{forw}}(r) = \sqrt[n]{P(l_1)\,P(l_2|l_1)P(l_3|l_2)\cdots P(l_n|l_{n-1})}$.
- *Backward heuristic approach* The backward heuristic chooses l_i that possesses the largest bi-gram probability with the next location $P(l_i|l_{i+1})$ as the next location. The goodness function is designed as $f_{\text{backw}}(r) = \sqrt[n]{P(l_1|l_2)P(l_2|l_3)\cdots P(l_{n-1}|l_n)}$.
- *Frequent sequential patterns* To prove that the mined time-aware transit patterns are more effective than the conventional sequential patterns for planning time-aware routes. We perform the sequential pattern mining method [28] and select the patterns with highest RVG scores to predict the removed locations.

Effects of β We first report the hit rate by varying β from 0 to 1 (i.e., the weight controlling the importance between visiting time and transition time), as shown in Fig. 10a, b. We set *minsup* = 10% and *maxgap* = 2. Our goal is to investigate whether β will affect the hit rate. We set the number of missing locations = 3. The results show that the balance weighting (e.g., $\beta \sim 0.5$) provides a positive influence to the performance. When β is around 0.5, *PTRP+LocTimeInf* can consistently produce the best results. The hit rates of some baselines are lower than 3%. Moreover, when β is from 0.25 to 0.75, both *PTRP+LocTimeInf* and *TRP+LocTimeInf* significantly outperform *Guidance Search* (up to 12 and 5%, respectively). It is not surprising because the mined TTP can not only provide representative route segments to guide the construction of routes but also capture the visiting and transit behaviors. The result that *PTRP+LocTimeInf* and *PTRP* significantly outperforms *TRP+LocTimeInf* and *TRP* (up to 10%) further exhibits the power of user clustering for preferred route search. We can also find that *PTRP+LocTimeInf* method can improve the performance of *PTRP* slightly. The results implicitly verify that *LocTimeInf* is can generate correct time labels and is useful to boost the quality.

Effects of number of guessing instances per route We vary the number of missing instances per route and report the hit rates. We set $\beta = 0.5$ (i.e., visiting time and transit time are equally important), *minsup* = 10% and *maxgap* = 2. The results are shown in Fig. 10c, d. The hit rate of each method decreases when the number of missing instance increases. Nevertheless, the proposed methods still significantly outperforms the *Guidance Search* as well as the other baselines. The results that *PTRP+LocTimeInf* significantly outperforms other methods again guarantees the capability of the proposed *LocTimeInf* and doing user clustering to collectively model user visiting preferences.

Effects of various settings in user clustering We evaluate how different information used in *GST-Clus* user clustering affects the performance. Recall that three factors, geographical

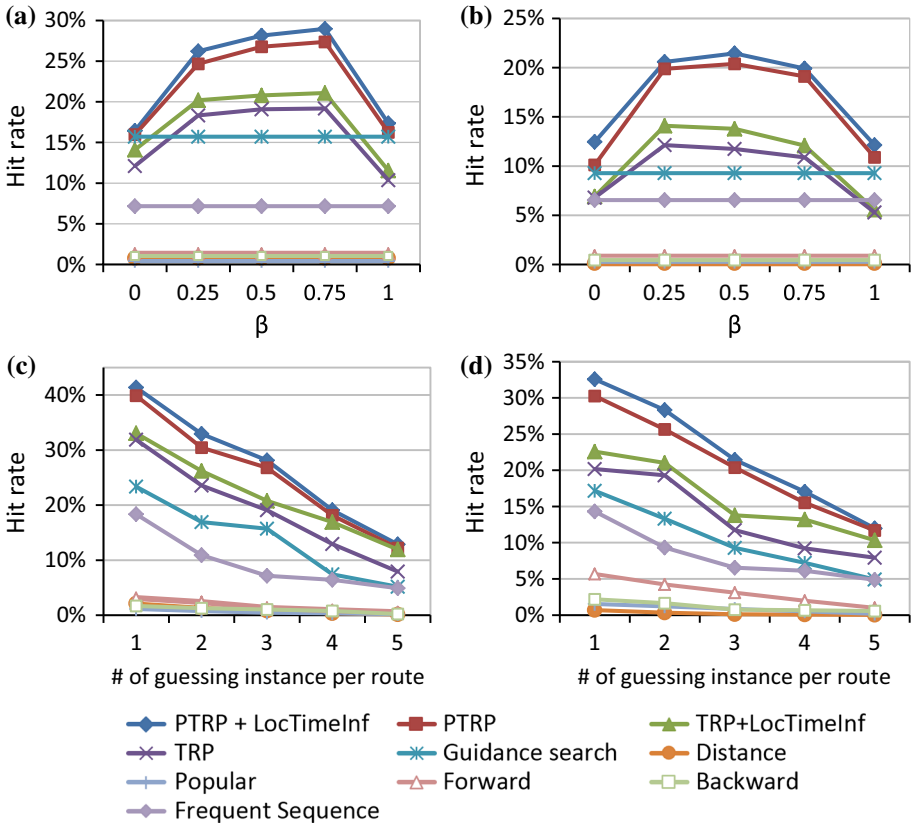

Fig. 10 Hit rates by varying the β value (**a**, **b**), and the number of guessing instance per route (**c**, **d**) in NYC (left) and SF (right)

(G), social (S), and temporal (T), collectively determine the distance between users. By changing the corresponding weights, i.e., ω_g, ω_s, and ω_t, we aim to understand which kind of information (and their combination) can lead to a better result. This experiment is conducted by also varying four important parameters: β, the number of guessing instance per route (#GI), and the DBSCAN parameters ϵ and $minPts$. The results are shown in Table 2, in which the tags *G, S, T, GS, GT, ST, and GST* means that (ω_g, ω_s, ω_t) are (1,0,0), (0,1,0), (0,0,1), (0.5,0.5,0), (0.5,0,0.5), (0,0.5,0.5), and (0.33,0.33,0.33), respectively. It is apparent that *GST* leads to the best performance. For using single factor, G and T are competitive while S is worst. Such result reveals that relatively friends may not be correlated to a user's visiting preference too much, comparing to their historical visited locations and time. In addition, the experimental results suggest that we can derive better performance when *minPts* is between 250 and 750 and *epsilon* is between 0.1 and 0.5. Such parameters can be learned from the validation data.

Effects of *minsup* and *maxgap* We investigate the effects of *minsup* and *maxgap*, and the results are shown in Fig. 11. It is obvious that the hit rates will increase when *minsup* decreases due to the drastic increasing of TTP patterns to be measured. We also report the time efficiency (in second). The runtime of PR-Search grows exponentially when the number of

Table 2 Hit rates by varying β, the number of guessing instance per route (#GI), and the DBSCAN parameters ϵ and $minPts$, under different combinations of geographical, social, and temporal parts

	NY							SF						
	G	S	T	GS	GT	ST	GST	G	S	T	GS	GT	ST	GST
β														
0.00	0.14	0.10	0.14	0.14	0.16	0.14	0.16	0.10	0.08	0.11	0.11	0.11	0.11	0.12
0.25	0.24	0.20	0.24	0.23	0.25	0.24	0.26	0.18	0.15	0.19	0.19	0.20	0.19	0.20
0.50	0.27	0.23	0.27	0.27	0.28	0.28	0.28	0.19	0.13	0.20	0.20	0.20	0.20	0.22
0.75	0.27	0.23	0.26	0.27	0.28	0.27	0.29	0.16	0.13	0.19	0.17	0.19	0.19	0.19
1.00	0.16	0.13	0.16	0.16	0.17	0.17	0.18	0.11	0.07	0.11	0.11	0.11	0.11	0.13
#GI														
1	0.40	0.36	0.40	0.40	0.41	0.41	0.41	0.30	0.27	0.31	0.31	0.32	0.31	0.32
2	0.32	0.27	0.32	0.32	0.33	0.33	0.33	0.26	0.21	0.27	0.27	0.27	0.27	0.28
3	0.27	0.22	0.27	0.27	0.28	0.28	0.28	0.19	0.12	0.20	0.20	0.20	0.20	0.21
4	0.18	0.15	0.18	0.18	0.19	0.19	0.21	0.14	0.12	0.14	0.16	0.16	0.16	0.18
5	0.11	0.07	0.12	0.11	0.13	0.13	0.13	0.10	0.08	0.10	0.10	0.11	0.10	0.11
ϵ														
0.10	0.26	0.21	0.26	0.26	0.27	0.27	0.30	0.17	0.14	0.17	0.17	0.18	0.17	0.18
0.25	0.28	0.25	0.28	0.28	0.29	0.29	0.29	0.20	0.15	0.20	0.20	0.21	0.20	0.21
0.50	0.27	0.23	0.27	0.27	0.28	0.28	0.29	0.19	0.17	0.20	0.20	0.20	0.20	0.21
0.75	0.24	0.22	0.24	0.24	0.25	0.24	0.27	0.14	0.11	0.14	0.14	0.14	0.14	0.16
1.00	0.17	0.11	0.18	0.18	0.19	0.20	0.20	0.11	0.10	0.11	0.12	0.12	0.12	0.13
$minPts$														
100	0.20	0.15	0.20	0.20	0.21	0.21	0.22	0.17	0.16	0.18	0.18	0.18	0.18	0.19
250	0.26	0.19	0.26	0.25	0.26	0.26	0.29	0.20	0.16	0.20	0.20	0.21	0.20	0.23
500	0.27	0.21	0.27	0.27	0.28	0.28	0.29	0.19	0.15	0.20	0.20	0.20	0.20	0.21
750	0.26	0.22	0.26	0.25	0.26	0.26	0.26	0.20	0.17	0.20	0.20	0.21	0.20	0.22
1000	0.24	0.19	0.24	0.24	0.25	0.24	0.25	0.16	0.14	0.16	0.16	0.16	0.16	0.16

patterns increases (i.e., lower *minsup* values). We suggest that users can choose proper *minsup* according to their scenarios of usages. For example, lower *minsup* can help a tourist agency introduce different tourists for users. In contrast, a backpacker may prefer higher *minsup* values to derive the instant time-aware routes. On the other hand, as for *maxgap*, different from *minsup*, when *maxgap* increases, the hit rates grow slowly while the PR-Search time grows sharply. Therefore, we suggest not choosing a higher *maxgap* value. Setting *maxgap* as 1 or 2 will lead to a good balance between hit rate and PR-Search time.

7.4 User study

We develop an online trip route recommendation system (termed TRP) based on the proposed framework. The snapshot of TRP is shown in Fig. 12. TRP is used to conduct a user study to examine whether the routes recommended by the proposed method are reasonable, useful and acceptable by real users. For each city, we randomly select two paired locations as source–destination to construct the routes for the user study. The route length varies from 4 to 7.

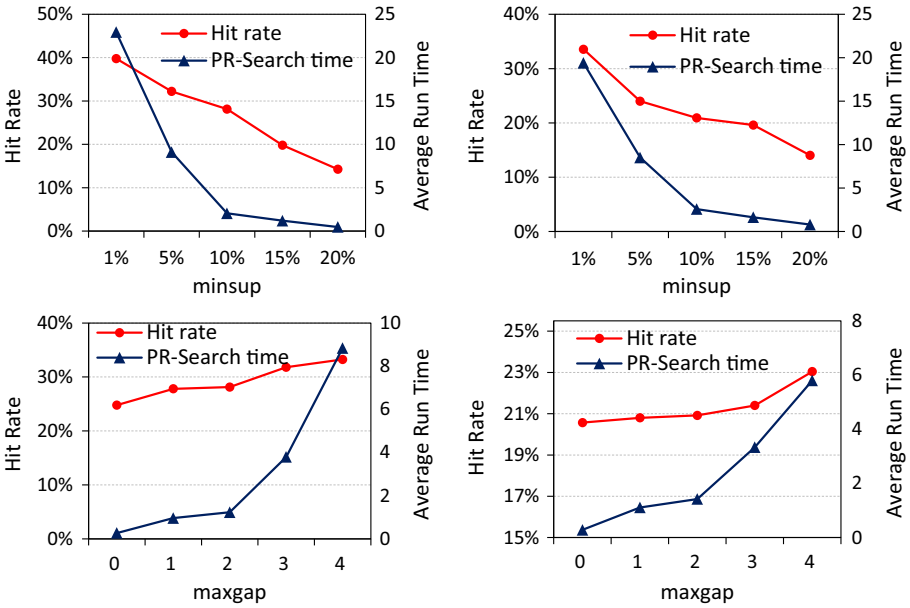

Fig. 11 Hit rates and the corresponding runtime for NY (left) and SF (right), by varying *minsup* and *maxgap*

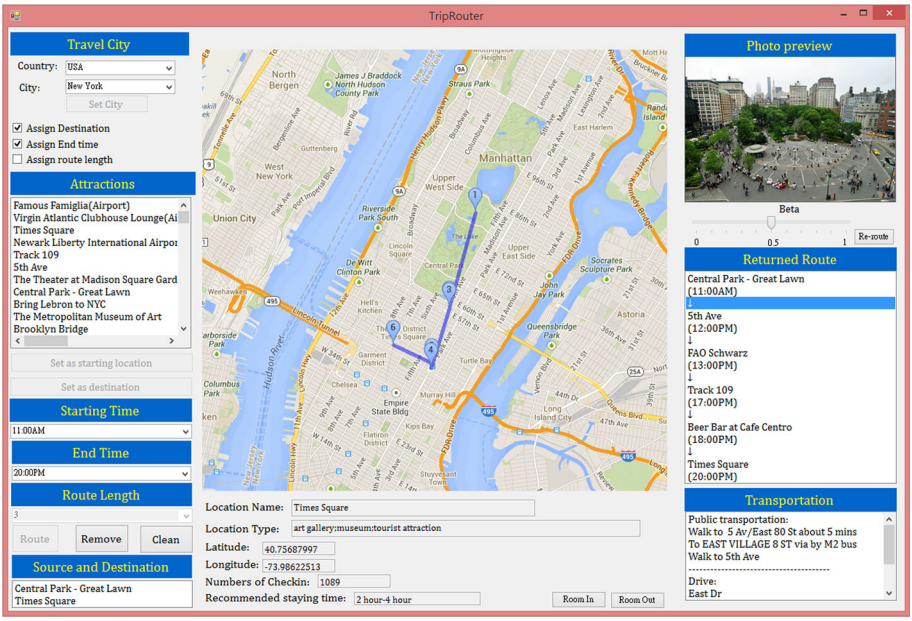

Fig. 12 The system snapshot of TRP

For comparison, we produce four kinds of routes for user study. The first three routes are generated by our TRP with $\beta = 0$, $\beta = 0.5$ and $\beta = 1$ (i.e., the parameter β determines the importance between visiting time and transition time). The fourth is the popularity-based

Table 3 The criteria with questions for our user study.

Criteria	Question
Location popularity (LP)	Do you think these recommended locations are popular ones?
Visiting order (VO)	Is the visiting order of locations in the route acceptable?
Visiting time (VT)	How you feel about the visiting time of locations in the route?
Transition time (TT)	Do you think the transition time between locations is reasonable?
Overall acceptance (OA)	If you are a traveler, do you want to adopt this route?

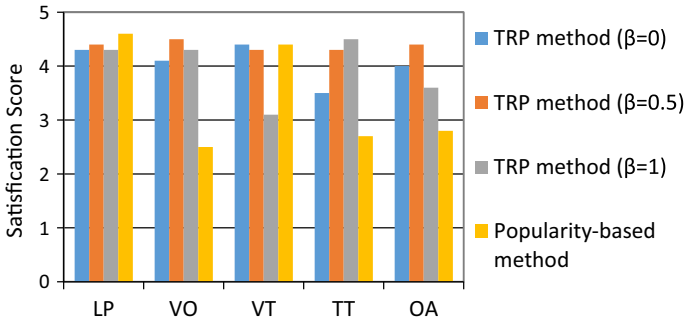

Fig. 13 Results of subjective evaluation for four methods

route recommendation which sequentially selects the most popular neighboring locations with the popular visiting time slots. Note that other traditional route planning methods cannot recommend the visiting time of locations so that we do not compare with them here. We invite 20 backpackers who had ever gone to at least 10 countries to conduct the user study. Please refer to Table 3 for the evaluation criteria. For each user, we ask him/her to give a 0–5 score to each criterion/question for each constructed route, in which "5" refers to the highest satisfaction while "0" is the lowest. We report the average value for each criterion in Fig. 13.

In general, TRP with $\beta = 0.5$ produces scores greater than 4 for all the criteria since the mined TTP patterns can ensure the quality of popularity and visiting order, and the TRP method can boost the effectiveness of visiting time and transit time as well. For the case of $\beta = 0$, it also gets higher scores of LP, VO, and VT. However, it has relatively lower scores of TT due to the lack of addressing the transit time. TRP with $\beta = 1$ gets the highest score of TT since it emphasizes on the transition time. However, its score of VT is lower than those of $\beta = 0$ and $\beta = 0.5$. That says, sometimes $\beta = 1$ would recommend improper visiting time for users. On the other hand, the popularity-based route ensures that the visiting locations associated visiting time stamps are popular, but the scores for the remaining four criteria are significantly lower. The user study gives us a quick view that the β can be set as 0.5 to achieve a better user satisfaction.

8 Conclusion

This paper attempts to tackle a preference-based time-aware route planning problem: given a query with starting location, starting time, and/or the destination, finding a route which is not only representative but also proper in terms of visiting time on locations along the

route. A four-stage model, consisting of inferring the visiting time of locations (*LocTimeInf*), geographical-social-temporal user clustering (GST-Clus), time-aware transit pattern mining (TTPM), and PTRP, are developed to construct the desired routes, with promising experimental results on Gowalla and Foursquare data. Our model is data-driven, which assures diverse results can be learned from different cities and location-based services where visiting patterns may vary with culture and characteristics of the city and the data.

Acknowledgements This work was sponsored by Ministry of Science and Technology (MOST) of Taiwan under Grants 104-2221-E-006-272-MY2, 106-2628-E-006-005-MY3, and 106-2221-E-006-221.

References

1. Lu H-C, Lin C-Y, Tseng VS (2011) Trip-mine: an efficient trip planning approach with travel time constraints. IEEE MDM 1:152–161
2. Wei L-Y, Zheng Y, Peng W-C (2012) Constructing popular routes from uncertain trajectories. In: ACM KDD
3. Yoon H, Zheng Y, Xie X, Woo W (2011) Social itinerary recommendation from user-generated digital trails. Pers Ubiquitous Comput 16:469–484
4. Yuan Q, Cong G, Ma Z, Sun A, Thalmann N M (2013) Time-aware point-of-interest recommendation. In: ACM SIGIR
5. Monreale A, Pinelli F, Trasarti R, Giannotti F (2009) Where next: a location predictor on trajectory pattern mining. In: ACM KDD
6. Sadilek A, Kautz H, Bigham J P (2012) Finding your friends and following them to where you are. In: ACM WSDM
7. Zaki MJ (2011) SPADE: an efficient algorithm for mining frequent sequences. Mach Learn 42:31–60
8. Cho E, Myers SA, Leskovec J (2011) Friendship and mobility: User movement in location-based social networks. In: ACM KDD
9. Hsieh H-P, Li C-T, Lin S-D (2014) Measuring and recommending time-sensitive routes from location-based data. ACM TIST 5:45
10. Chiang M-F, Lin Y-H, Peng W-C, Yu P S (2013) Inferring distant-time location in low-sampling-rate trajectories. In: ACM KDD
11. Cheng A-J, Chen Y-Y, Huang Y-T, Hsu W H, Liao H-Y M (2011) Personalized travel recommendation by mining people attributes from community-contributed photos. In: ACM Multimedia
12. Lu X, Wang C, Yang J-M, Pang Y, Zang L (2010) Photo2trip: generating travel routes from geo-tagged photos for trip planning. In: ACM Multimedia
13. Kurashima T, Iwata T, Irie G, Fujimura K (2010) Travel route recommendation using geotags in photo sharing sites. In: ACM CIKM
14. Sharifzadeh M, Kolahdouzan M, Shahabi C (2008) The optimal sequenced route query. VLDB 17:765–787
15. Cao X, Chen L, Cong G, Xiao X (2012) Keyword-aware optimal route search. VLDB 5:1136–1147
16. Zheng K, Shang S, Yuan J, Yang Y (2013) Towards efficient search for activity trajectories. In: IEEE ICDE
17. Quercia D, Schifanella R, Aiello L M (2014) The shortest path to happiness: recommending beautiful, quiet, and happy routes in the city. In: ACM HyperText
18. Tan M, Wang B, Wu Z, Wang J, Pan G (2016) Weakly supervised metric learning for traffic sign recognition in a LIDAR-equipped vehicle. IEEE Trans Intell Transp Syst 17(5):1415–1427
19. Ye M, Yin P, Lee W-C, Lee D-L (2011) Exploiting geographical influence for collaborative point-of-interest recommendation. In: ACM SIGIR
20. Liu X, Liu Y, Aberer K, Miao C (2013) Personalized point-of-interest recommendation by mining users' preference transition. In: ACM CIKM
21. Liu B, Fu Y, Yao Z, Xiong H (2013) Learning geographical preferences for point-of-interest recommendation. In: ACM KDD
22. Noulas A, Scellato S, Lathia N, Masolo C (2012) Mining user mobility features for next place prediction in location-based services. In: IEEE ICDM
23. Lian D, Zhao C, Xie X, Sun G, Chen E, Rui Y (2014) GeoMF: joint geographical modeling and matrix factorization for point-of-interest recommendation. In: ACM KDD

24. Liu Y, Liu C, Liu B, Qu M, Xiong H (2016) Unified point-of-interest recommendation with temporal interval assessment. In: ACM KDD
25. Yao L, Sheng Q Z, Qin Y, Wang X, Shemshadi A, He Q (2015) Context-aware point-of-interest recommendation using tensor factorization with social regularization. In: ACM SIGIR
26. Zhao S, Zhao T, Yang H, Lyu M R, King I (2016) STELLAR: Spatial-temporal latent ranking for successive point-of-interest recommendation. In: Proceedings of the 13th AAAI conference on artificial intelligence (AAAI-16). AAAI Association, Phoenix, pp 315–321
27. Li X, Cong G, Li X-L, Pham T-A N, Krishnaswamy S (2015) Rank-GeoFM: A Ranking based geographical factorization method for point of interest recommendation. In: ACM SIGIR
28. Agrawal R, Srikant R (1995) Mining sequential patterns. In: IEEE ICDE
29. Ayres J, Gehrke JE, Yiu T, Flannick J (2002) Sequential pattern mining using a bitmap representation. In: ACM SIGMOD
30. Pei J, Han J, Mortazavi-Asl B, Pinto H (2001) PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In: IEEE ICDE
31. Yan X, Han J, Afshar R (2003) CloSpan: Mining closed sequential patterns in large datasets. In: SIAM SDM
32. Wang J, Han J, Li C (2007) Frequent closed sequence mining without candidate maintenance. IEEE TKDE 19:8
33. Becker GS (1965) A theory of the allocation of time. Econ J 75(299):493–517
34. Zhu X, Ghahramani Z, Lafferty J (2003) Semi-supervised learning using Gaussian fields and harmonic functions. In: ICML
35. Shi J, Mamoulis N, Wu D, Cheung D W (2014) Density-based place clustering in geo-social networks. In: ACM SIGMOD
36. Wang D, Pedreschi D, Song C, Giannotti F, Barabasi A-L (2011) Human mobility, social ties, and link prediction. In: ACM KDD
37. Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. ACM KDD 96:226–231
38. Cheng Z, Caverlee J, Lee K, Sui D (2011) Exploring millions of footprints in location sharing services. In: ICWSM
39. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules. VLDB 1215:487–499
40. Giannotti F, Nanni M, Pedreschi D (2006) Efficient mining of temporally annotated sequences. In: SIAM SDM
41. Hsieh H-P, Li C-T (2014) Mining and planning time-aware routes from check-in data. In: ACM CIKM

Cheng-Te Li is an assistant professor at Department of Statistics, National Cheng Kung University (NCKU), Tainan, Taiwan. Before joining NCKU, he was an Assistant Research Fellow at Research Center for Information Technology Innovation (CITI) in Academia Sinica, Taiwan. He received his M.S. and Ph.D. degrees from Graduate Institute of Networking and Multimedia, National Taiwan University, in 2009 and 2013, respectively. His research interests include social and information networks, data mining, and social media analytics. His international recognition includes Exploration Research Award of Pan Wen Yuan Foundation 2016, Facebook Fellowship 2012 Finalist Award, ACM KDD Cup 2012 First Prize, IEEE/ACM ASONAM 2011 Best Paper Award, and Microsoft Research Asia Fellowship 2010.

Hsin-Yu Chen is a M.S. graduate student in the Department of Statistics, National Cheng Kung University. She is under the supervision of Prof. Cheng-Te Li. Her research topics contain computational statistics, data mining, and social network analysis.

Ren-Hao Chen is a graduate student in the Institute of Computer and Communication Engineering, National Cheng Kung University, Taiwan, supervised by Prof. Hsun-Ping Hsieh. His research interests include on big data analytics, spatio-temporal data mining, and artificial intelligence. He is now mainly involving in the projects related to urban dynamics inference, social-based recommender system and route planning.

Hsun-Ping Hsieh is now an Assistant Professor and leads Urban Science and Computing Lab (UCLAB) at Department of Electrical Engineering, National Cheng Kung University, Taiwan. He received his Ph.D. degree in Graduate Institute of Networking and Multimedia at National Taiwan University. Hsun-Ping's international recognition includes ACM KDD Cup 2010 First Prize, Garmin Fellowship 2014 and 2013, and Best Intern Award at Microsoft Research Asia. His research interests include big data mining, urban computing, and AI-based smart city services.