TRIPPLANNER: Personalized Trip Planning
Leveraging Heterogeneous Crowdsourced Digital Footprints

Chao Chen, Daqing Zhang, Bin Guo, Xiaojuan Ma, Gang Pan, and Zhaohui Wu

Abstract—Planning an itinerary before travelling to a city is one of the most important travel preparation activities. In this paper, we propose a novel framework called TRIPPLANNER, leveraging a combination of Location-based Social Network (i.e. LBSN) and taxi GPS digital footprints to achieve personalized, interactive, and traffic-aware trip planning. First, we construct a dynamic POI network model by extracting relevant information from crowdsourced LBSN and taxi GPS traces. Then, we propose a two-phase approach for personalized trip planning. In the route search phase, TRIPPLANNER works interactively with users to generate candidate routes with specified venues; In the route augmentation phase, TRIPPLANNER applies heuristic algorithms to add user’s preferred venues iteratively to the candidate routes, with the objective of maximizing the route score while satisfying both the venue visiting time and total travel time constraints. To validate the efficiency and effectiveness of the proposed approach, extensive empirical studies were performed on two real-world data sets from the city of San Francisco, which contain more than 391,900 passenger delivery trips generated by 536 taxis in a month, and 110,214 check-ins left by 15,680 Foursquare users in six months.

Index Terms—personalization; trip planning; traffic-aware; crowdsourcing; digital footprints

I. INTRODUCTION

Planning an itinerary is one of the most important and time-consuming travel preparation activities [12], [33]. In order to plan a visit to a popular tourist city, one needs to select a number of preferred Point of Interests (POIs) among hundreds of possible POIs, figure out the order in which they are to be visited, ensure the time it takes to visit each POI and to transit from one POI to the next, and meet one’s time budget. Let us take the following use case as an example:

John is transiting through San Francisco. He rents a car at the SFO airport at 9:00 am and would like to spend several hours for sightseeing, and then leaves for San Jose by train at 15:00 pm from the Caltrain Station. He wants to visit the Golden Gate Bridge, Lombard Street and Fisherman’s Wharf. If time permits, he also wants to squeeze in visits to an art museum and/or lunch at one of the Boudin Bakery locations. In addition, he prefers to have lunch before visiting the Fisherman’s Wharf.

As shown in the above use case, three main factors have to be considered in the design of a trip planning system: 1) the venue\(^1\) constraints, which include the trip starting location (the Airport), the trip ending location (the Caltrain Station), the POIs expected to be covered in the itinerary (e.g. the Golden Gate Bridge), the POI categories which might be added if time permits (e.g. art museum), and the POI visiting order; 2) the time constraints, which include a trip starting and ending time (time budget), the duration of visit at each POI which can be estimated and controlled by users, the transit (driving) time between POIs which may vary depending on the traffic condition of the time of the day, and the operation time of each POI to visit; and 3) user’s preference scores about a specific POI and an itinerary at certain time of the day which are assumed to be computable. The objective of the trip planning system is to interact with users to determine if the user-specified POIs can all be covered within the time budget. If the answer is “no”, the system would iteratively prompt the user to remove one POI at a time until the POIs specified can be fit into one route without compromising the time constraint. If the answer is “yes”, the system would automatically generate an “optimal route” which contains the specified POIs and preferred POI categories, and meets the time constraint according to the predicted driving time of the day.

Apparently, the above problem cannot be solved using the approaches proposed for route search in the previous research [8], [23], [7], [36], as they often assume that the transit time between POIs is constant. In our scenario, the purpose of route search is to find a route that can cover a series of requested POIs specified by users while meeting a time budget. The above issue is also different from route recommendation. Many existing route recommendation systems suggest routes directly based on the similarity between user’s visiting history in other contexts and other people’s trip records in the targeted city [40]. Others identify venues according to a user’s preference and recommend routes based on certain criteria (e.g. with the highest route score) [13], [16], [26], [20]. However, during the actual trip planning, it is common that

---

\(^1\)We use venue and POI interchangeably throughout this paper.
a user may have some additional constraints such as “need to go some specific places,”“go to park before lunch,” “the total travel time should be ≤ 6 hours.” Some prior work asked users to manually select and configure travel routes after recommendation, which was tedious and time consuming [20], [35]. Even worse, there might be no travel routes that can satisfy all the constraints. Another group of route planning work aims to find the optimal (e.g. fastest or shortest) paths in road networks based on the time-varying assumption of each road segment [38]. These studies care only about the edge information in the network, largely ignoring the attributes associated with the nodes (POIs). In other words, they care only about the time on the road, and pay little attention on the opening hours and duration of visit of each POI. Unlike this body of work, we need to consider the characteristics of each POI in the route selection process, e.g. its attractiveness, operation hours, and order of visit. In summary, this paper intends to build a personalized, interactive and traffic-aware trip planning service.

In order to achieve personalization in trip planning, we first need to acquire the information about all POIs in a targeted city and links among them to build a POI network model. So far, different data sources have been exploited, including: 1) websites, Wikipedia, web blogs which contain tourists’ profiles as well as comments that reveal preferences and experiences with POIs [5], [6]; 2) social media sites such as Facebook, Flickr, and LBSN (e.g. Foursquare and Gowalla), which can inform the popularity, functions, operating hours of the POIs as well as individual user’s travel history [8], [21], [26]; and 3) GPS trajectories of people and taxis, which can indicate the stay time in each place and transit time between two places [3], [10], [17], [38], [40]. Apparently, each data source has its strength and weakness in characterizing certain facets of the POI nodes and edges as required by the model. Integrating heterogeneous data sources can provide a more complete picture of the POI network.

In this paper, we develop a novel trip planning framework called TRIPPLANNER. In the front end, TRIPPLANNER allows users to interactively specify their venues of interests with users to generate candidate routes that traverse all specified venues; In the route augmentation phase, the system employs heuristic algorithms to add user-preferred venues (i.e. optional venues if time permits) to the candidate routes iteratively, with the objective of maximizing the route score and satisfying both the venue visiting time and total travel time constraints.

Finally, we validate the efficiency and effectiveness of TRIPPLANNER by extensive evaluations using large-scale real-world data sets. The remaining of this paper is structured as follows. In Section II, we review the related work and show how our work is different from prior research. In Section III, we introduce the framework of our proposed TRIPPLANNER system. After presenting the process of constructing the POI network by leveraging the Foursquare check-in and taxi GPS data sets in Section IV, we elaborate on our two-phase approach in Section V. Evaluation results are reported in Section VI to verify the effectiveness of the proposed approach. Finally, we conclude the paper and chart the future directions in Section VII.

II. RELATED WORK

The related work is organized in two subsections. We first review previous work on extracting information from different data sources to build the POI network model, and then discuss how to recommend a trip to users based on certain assumptions.

A. Construction of POI Network

In trip planning research and applications, people have exploited different data sources to extract node and edge information needed to model a POI network. For example, in [2], [6], [8], [12], [21], [27], [41], many researchers have used geo-tagged photos from photo-sharing sites (e.g. Flickr) to derive the information about POIs, such as location, popularity, characteristics, and proper visiting time and order. In addition, demographics and social relationship of visitors to these POIs can also be extracted. However, it is hard to estimate the dynamic transit time between POIs from geo-tagged photo data. More recently, people began to explore user-generated LBSN digital traces since such data contains rich information that can be used to directly characterize each POI in a tourist city and users’ preferences to each POI [4], [16], [25], [26], [37]. Unfortunately, similar to geo-tagged photo data, LBSN
traces also do not contain dynamic transit time between POIs, especially when driving is considered for travelling in a city. Another popular type of data is GPS trajectory, which can be used to predict the fastest route at certain time of the day in a city [38]. Previous studies have shown that GPS trajectory traces can precisely characterize the transit time between POIs, which is more accurate than Google Maps results [3, 9], [18]; the point-to-point transit time estimated by Google Maps was about 35% off from the actual values on average [3].

Building on existing work, we leverage taxi GPS trajectory and LBSN trace data to construct a POI network model. Such approach allows us to better characterize both the POI nodes and the edges in the network, making it possible to address a more realistic trip planning problem and design a better trip planning system.

B. Trip Planning

There has been quite some work on trip planning [1], [31], [34], [39], [43], which can be roughly classified into three categories. The first category is route search, which aims to answer a user’s route queries over a given POI network. Traveling Salesman Problem (TSP) is a classical problem in route search [24]. Given a set of specified POIs in a graph and their pairwise distances, the goal of TSP is to find the shortest route that visits each POI exactly once and returns to the original location. However, situations may be much more complicated in the real world. Destination of a trip may be different from the starting point. Furthermore, users may simply have in mind a type of POIs of interests rather than a specific POI location. Trip Planning Query (TPQ) is proposed to address the problem [23]. The goal of TPQ is to find the shortest path between two given locations that covers all of the user-specified node categories. Some research has looked into variations of TSP and TPQ problems with additional constraints [8], [19], [30], [36], but most of the studies assumed that the transit time between POIs is constant, with the exception of work like [14], [15], [22]. Different from prior research, we allow both POIs and POI categories (i.e. types) to be specified in the route query. We also assume that the transit time between POIs is affected by time-dependent traffic condition.

The second category is route recommendation, which suggests POIs or routes to users based on their preferences. It usually assumes that users will not provide POIs or POI categories of interests explicitly. For instance, Kurashima et al. develop a probabilistic model which incorporates user preferences, location and available time to suggest personalized routes [21]. Lu et al. present a Personalized Trip Recommendation (PTR) framework which recommended personalized arrangement of visit to venues, given a predefined budget (e.g. time, money) [26]. Hsieh et al. propose to utilize users’ check-in patterns to recommend popular, time-sensitive trips to users [16]. Different from these studies, we already have information of POIs and/or POI categories specified in the route query. On top of this, we compute a venue score for each POI based on its popularity and users’ preferences.

The third category is route planning with the goal of selecting optimal time-dependent routes. For instance, Yuan et. al. [38] and Ziebart et al. [42] propose to mine the historical taxi GPS traces for the optimal driving path between two chosen POIs, assuming that the transit time is affected by traffic condition. Unlike this category of research, we also consider the priority of each POI, preferred order of visit, as well as the visiting time constraint of each POI in the route optimization process.

A comparison between our work and existing research is further provided in Table I. In summary, our work differs from the previous work in the data sources used, the problem defined, the assumptions given, as well as in the methods developed.

III. TRIPPLANNER SYSTEM

Here, we first introduce several key terminologies. Then, we formally define the research problem of personalized trip
planning. Finally, we give a detailed description of the framework of TripPLANNER system, which is comprised of three major parts: a dynamic POI network model, a route search component, and a route augmentation component (Fig. 1).

A. Key Terminologies

Dynamic POI Network Model: the model can be represented as a directed complete graph \( G = (V, E) \). Each node in \( V \) denotes a venue (i.e. POI), which has five attributes: category, operation time, popularity, geographical location, and stay time (i.e. the duration of visit). Each directed edge \((v_i, v_j)\) in \( E \) represents a link from node \( v_i \) to \( v_j \), which carries the transit time between the two venues, denoted as \( t_t(v_i, v_j) \). The transit time is asymmetric and dynamically changing.

Lemma 1 (Dynamic POI network has First-Input-First-Output Property). Given a dynamic network \( G = (V, E) \), where the transit time of each edge in \( G \) is time-dependent. The network is FIFO since for any arc \((i, j)\) in \( E \), given user \( A \) leaves node \( v_i \) at time \( t_0 \), and user \( B \) leaves node \( v_i \) at time \( t_1 \) \((t_1 > t_0)\), then user \( B \) cannot arrive at node \( v_j \) before user \( A \).

Proof. Proof can be found in Appendix A.

Itinerary Query: an itinerary query \( IQ \) consists of four parts: 1) a user-specified venue list \( V_u \), that the user intends to cover; 2) starting place \( v_o \), and starting time \( t_o \), ending place \( v_d \), and a travel time budget \( \Delta \); 3) a set of user-preferred venue categories \( CAT_u \) (optional venues to visit if time permits); and 4) additional constraints \( ACs \), such as constraints on the time and the order of venues are to be visited. For instance, a user may want to have lunch at noon and visit museums after that. In summary, the query \( IQ \) can thus be represented as \( \{V_u, v_o, v_d, t_o, \Delta, CAT_u, ACs\} \). It should be noted that users may not impose \( ACs \) when planning their visit, and thus the corresponding field is empty.

Valid Route: a route \( R = \{v_1, v_2, \cdots, v_n\} \) is valid if
\[
aT(v_i) \geq oT(v_i), IT(v_i) \leq cT(v_i) \quad \forall i \in \{1, 2, \cdots, n\}
\]
This implies that the user should visit all venues while they are open. Here \( aT(\cdot), IT(\cdot) \) are the users’ arriving and leaving time for the given venue, while \( oT(\cdot), cT(\cdot) \) refer to the opening time and closing time of the given venue respectively.

Route Score: route score is defined as the sum of scores of all venues along the route if it is valid, similar as in [13], [20]; otherwise, the route score is defined as 0 (i.e. there exists case in which a user arrives at at least one venue along the route before it opens or after it closes).

Time Margin: it is defined as the difference between the total travel time of the route and the user’s time budget.

B. Problem Statement

Personalized Trip Planning Problem. Given a dynamic POI network \( G \) in a targeted city and a user’s itinerary query \( IQ \), our objective is to find the optimal valid route with the maximum route score.

C. Framework

As shown in Fig. 1, the proposed framework contains three components: the dynamic POI network model, the route search component, and the route augmentation component. While the dynamic POI network model is pre-built and maintained offline, the route search and route augmentation components collaboratively answer users’ trip queries in real-time.

1) The Dynamic POI Network Model. The key problem of POI network model construction is to separately extract attributes of POI nodes from the Foursquare data set and information of the edges from the taxi GPS data set.

2) Route Search. Given user-specified venues to visit, the starting time, and the time budget, the route search component returns routes that traverses all the intended venues from the starting location to the destination. In particular, the returned routes with a time margin greater than a user-determined threshold become candidate input to the route augmentation component. However, users might list too many venues to cover within the time constraint, or the planned visiting time does not agree with the operating hours of certain venues. If the TripPLANNER system detects any of those cases, it will interact with the user to manually modify the venue list.

3) Route Augmentation. This component aims to augment the candidate generated from the route search module with user-preferred venues inferred from the intended venue categories in the query, maximizing the route score under the given travel time budget. It first pulls together all of the venues that belong to user-preferred venue categories as candidate

![Fig. 1. The framework of our proposed TripPLANNER.](image)
venues. Then for each candidate route, it tries to insert venues in the pool into it to generate an augmented route, without breaking any constraint. In the end, TRIPPLANNER presents the augmented routes to the user, in an order sorted according to their scores in the Augmented Route Ranking module.

In the following two sections, we elaborate on the offline construction of the dynamic POI network, and the online route planning process respectively.

IV. DYNAMIC POI NETWORK MODELLING

A. Node Modelling

Each node in the model corresponds to a POI with five attributes: operation time, category that the venue belongs to, popularity, geographical location, and stay time. For each venue, users provide their expected stay time, while we extract information related to the first four attributes from Foursquare data (Fig. 2).

Operation time of a venue may vary according to the day of the week and even time of the year.

A venue can be associated with two or more category labels with different granularities. Take the Nick’s Crispy Tacos venue shown in Fig. 2 as an example. It has three category labels, among which “Food” is a Level 1 label, “Breakfast Spot” is a Level 2 label, and “Multiplex” is a Level 3 label.

To compute the popularity of a given venue, we use two indicators: the total number of visitors (tv) and the total number of check-ins (tcs) (Eq. 1). The total visitor number is usually smaller than the total check-in number of the same venue, since some users checked in repeatedly during a single visit.

\[
\text{Pop}(v_i) = \frac{2 \times \text{tv}(v_i)}{c_1} + \frac{\text{tcs}(v_i)}{c_2}
\]

where \(c_1\) is the maximum visitor number of all venues in the targeted city, and likewise, \(c_2\) is the maximum check-in number of all venues. Note that the most visited venue may be different from the one with the most check-in records. The venue score is fused by the harmonic mean as we want both values to be relatively higher [28].

B. Edge Modelling

Here, we briefly introduce how to infer the dynamic edge values from taxi GPS traces. The technical details are given in Appendix B.

To estimate the dynamic transit time by driving from one node to another (i.e. the value of an edge), we need to consider the time-variant nature of traffic between venues. In this work, we leverage a real world dataset — taxi GPS traces. Such data has two unique features: 1) spatial coverage: a certain number of city taxis can fully cover the whole road network; 2) time coverage: taxis usually operate all day long, which is in line with tourists’ visiting time. These two features of the taxi GPS data enable us to estimate the transit time between any two nodes within any time period.

To simplify the transit time calculation between nodes in the POI network, we first cluster co-located nodes among which walking is the best way to get around. The within-cluster transit time is computed using the average walking speed, while the between-cluster transit time is estimated based on the driving speed at the specific timeslot. Fig. 3 illustrates a simple dynamic POI network. The small circles in different colors refer to the nodes (POIs). Near-by nodes are grouped into clusters (ellipses in the dashed line). Directed edges inside each cluster carry the walking time information between nodes which is independent of the time of the day; while directed edges across clusters carry the transit time information in between which is time-variant. For instance, during rush hours in the morning, the transit time from the upper right cluster to the bottom cluster is more than twice of the least travel time of the day (refer to the green curve in the bottom right of Fig. 3 for a whole-day view of dynamic transit time).
V. The Two-Phase Approach

We take a two-phase approach, i.e. route search and route augmentation, to perform trip planning. Route search retrieves candidate routes traversing all user-specified venues within the time budget. Route augmentation further enriches the candidate routes with user-preferred venues as long as time permits, and recommends to users the optimal routes with the highest scores. The set of user-preferred venues is a subset of venues in the targeted city, which are obtained based on the user-preferred venue categories (CAT) in the itinerary query (IQ) (refer to Appendix C for details).

A. Phase I: Route Search

The route search component works interactively with the user. Given a user’s starting and ending places, specified venue list, and a travel time budget, it first checks and removes any venue that cannot be visited on the intended date. The module then returns all possible routes between the given origin and destination that cover the valid venues in the list, via a simple enumeration. Users may input a long venue list which cannot be covered within the given time budget. In this case, users are asked to shorten the venue list iteratively to ensure a proper time margin. In this process, the system would suggest users to remove venue(s) with a longer distance from the starting and ending places.

Users may also have venue visiting order constraint on the specified venue list. In this case, we use the specified venue sequence as the route search inputs. Afterwards, we need to compute the total travel time for for all remaining routes, considering the arrival time and depart time of the user at each venue. Specifically, the total travel time is computed as the sum of transit time and stay time spent along all the venues in the route, given the traffic-dependent transit time between any two venues and stay time at each venue. The time margins are simply obtained by substituting the time budget, which is same for all remaining routes.

Consequently, routes with time margins bigger than the user-specified threshold and meeting venue order constraint on the specified venue list would be selected as candidate routes. Thus, the computation complexity of this phase is $O(m!)$, where $m$ is the number of user-specified venues after venue list shortening. Note that it may be less than the number of the initial user-input venues.

Moreover, some candidate routes can be further pruned in this phase because they cannot lead to any valid route after the route augmentation phase. Specifically, a route which contains later-arrival venue(s) can be pruned in advance. Here, “later-arrival” means arriving at a venue after its closing time; “earlier-arrival”, on the contrary, refers to arriving at a venue before it opens. The rationale behind is: inserting a new venue before a “later-arrival” venue will further push back the arrival time at this venue; while the “later-arrival” venue would still be late when inserting a venue after it. In a word, the output of the route search phase are all candidate routes that have enough time margins, meet venue visiting order constraint, and do not contain any “later-arrival” venues.

B. Phase II: Route Augmentation

The route augmentation component tries to insert optional user-preferred venues into the candidate routes returned from the previous phase. The route augmentation problem is NP hard and suffers from combination explosion. On one hand, some candidate routes may allow the user to visit only one more optional user-preferred venue, while some of them allow to visit more than one, depending on their time margins and the locations of the user-preferred venues. On the other hand, there are many possible orders to visit the optional user-preferred venues along each candidate route, and different visiting orders would result in different total travel time. The aim for optimization is to maximize the route score without exceeding the time budget. It is very challenging as it tries to satisfy two competing requirements: (1) the route should contain as many user-preferred venues as possible; (2) the route should meet the travel time budget and the venue visiting time constraints. We have to consider the following two factors when selecting new valid venues to optimize the route score.

Arrival Time Delay by Adding New Venues. Apparently, inserting new venues into a given route would increase its total visiting time, adding additional transit time and stay time. The arrival time to some of the existing venues may be delayed. Furthermore, the transit time needed between existing venues might also be different due to the time shift. Taking the diagram in Fig. 4 as an example, after inserting venue $v_5$ in the route, the arrival time to $v_4$, $v_5$, $v_6$, $v_7$ would be delayed, and the transit time between $v_4$ to $v_7$ might also change as the traffic conditions might be different later in the day.

Total Route Score Increased by Adding New Venues. Generally, adding more user-preferred venues would increase the score of a route, but may violate the given constraints if not done properly. We designed a method for route augmentation, which consists of two steps: venue inserting and score maximization. The former aims to find a suitable position in the candidate route to insert a selected venue, while the latter is responsible for maximizing the score of the updated route.
1) The Venue Inserting Algorithm: There are two principles that we should follow when inserting a new venue: the augmented route should be valid and we should minimize the extra cost in time. For a candidate route with \( n \) venues and a new venue \( v_c \) to insert, if the candidate route does not contain any “earlier-arrival” venue, we need to check \( n - 1 \) positions to determine the final augmented route. It should be noted that, for the case that users have order constraint on the user-preferred venues, the positions in which a new venue can be inserted is limited (< \( n - 1 \)). Only the positions that will not violate the venue order constraint are considered. However, if the candidate route does contain “earlier-arrival” venues, we only need to check \( k - 1 \) (< \( n - 1 \)) positions, where \( k \) is the position of the first “earlier-arrival” venue in the candidate route according to Theorem 1. Again, the possible positions can be further narrowed down for the case that users have venue visiting order constraint.

**Theorem 1.** For a candidate route which contains “earlier-arrival” venues, inserting a candidate user-preferred venue behind the first “earlier-arrival” venue could not lead to a valid route.

The pseudo-code of the venue inserting algorithm is shown in Algorithm 1. For clarity, we assume that users do not have venue visiting order constraint on user-preferred venues in the scope of this paper. We first check whether the candidate route contains any “earlier-arrival” venue (Line 1). If it does, the possible positions where the new venue can be inserted are in \([2, k]\); otherwise, the range is \([2, n]\) (Lines 2-5). Note that the “wait” for a venue to open is not considered in this paper, as the total travel time is a hard constraint in our case. The core function of Algorithm 1 is the `augRoute` function shown in Algorithm 2. In this function, the candidate venue is inserted into the given route at each possible position (Lines 3-8). Note that not every position where the candidate venue is inserted can lead to a valid route (Lines 5-7). If no augmented routes are valid or the total travel time cost of all the generated augmented routes exceeds the time budget, the function returns the original input route (Lines 9-11); otherwise, it returns the augmented route with the minimum total travel time (Lines 12-13).

**Algorithm 1 Venue Inserting Algorithm**

**Input:** A candidate route \( R = \langle v_1, v_2, \ldots, v_n \rangle \);
A candidate venue \( v_c \);
A user-specified total travel time budget \( \Delta \);

**Output:** An augmented route \( \text{aug}R \)

1: if \( R \) has “earlier-arrival” venues then
2:  \( k = \text{pos}(R) / \text{pos}(R) \) gets the index of the first “earlier-arrival” venue in \( R \)
3:  \text{aug}R = \text{augRoute}(R, v_c, [2, k], \Delta) 
4: else 
5:  \text{aug}R = \text{augRoute}(R, v_c, [2, n], \Delta) 
6: end if

The algorithms above illustrate how to insert one venue to a candidate route. If there are multiple venues to add, this process will iterate through the list, again following the proposed principles. In the rest of the paper, we use the expression \( \text{aug}R + \{v_{c1}, v_{c2}, \ldots, v_{cn}\} \) to denote the operation of inserting the venue list \( \{v_{c1}, v_{c2}, \ldots, v_{cn}\} \) to the candidate route \( R \) sequentially. Note that for the same set of candidate venues, different inserting orders may result in different augmented routes (e.g. \( R + \{v_{c1}, v_{c2}\} \neq R + \{v_{c2}, v_{c1}\} \)).

2) Route Score Maximization Algorithms: We first present mathematical formulation of our route score maximization algorithms, then introduce how to compute the route score according to the user’s preferences. In the end, we propose three heuristic algorithms to maximize the route score.

**Mathematical Formulation.** For a given user \( u_i \), a set of candidate venues \( \{v_{ci}\}_{i=1}^{N} \), and a candidate route \( R \), the route score maximization problem is:

\[
\max \quad \mathcal{R}(u_i, R + \{x_{i}v_{ci}\}_{i=1}^{N})
\]

Subject to:

\[
\begin{align}
x_i & \in \{0, 1\} \\
x_{i}v_{ci}.\text{cat} & \cup x_{i}v_{ci}.\text{cat} \cup \cdots \cup x_{n}v_{cN}.\text{cat} \subseteq \text{CAT}_u
\end{align}
\]

\[
\mathcal{T}(R + \{x_{i}v_{ci}\}_{i=1}^{N}) \leq \Delta
\]

where Eq. 2 refers to the objective function (i.e. the route score) for maximization. It is subjected to three constraints, as shown in Eqs. 3-5. Eq. 4 defines the constraint for the augmented venue selection, i.e. only the user-preferred venues can be selected for route augmentation, but not necessarily covering all venue categories, due to the total travel time constraint. Eq. 5 emphasizes that the total time cost of the newly augmented route should be within the predefined travel time budget \( \Delta \).

**Route Score Calculation.** The route score calculation algorithm is the core of the route augmentation component, which estimates the attractiveness of a candidate route to a particular user. The route score is defined as the sum of all its venue scores, and thus the venue scoring method is vital.

**Venue Scoring.** On one hand, the score of a venue is determined by its popularity (\( \text{Pop} \), as shown in Eq. 1), which is objective (denoted as \( \text{VS}_\text{obj} \)); On the other hand, the venue score is also related to individual user’s personal interests revealed in his/her check-in history, which is subjective. For instance, the scores of “Art & Museum” venues should be higher for a user, if he/she visits venues in this category more often than the others as shown in the Foursquare check-in records. The normalized check-in preference value (\( \text{VS}_\text{sub} \)) of the venue \( v_j \) for user \( u_i \) is calculated by Eq. 6. For simplicity, only the level-1 category labels (i.e. the nine category labels defined by Foursquare) are used in the scope of this paper.

\[
\text{VS}_\text{sub}(u_i, v_j) = \frac{\text{tcs}(u_j, \{v_{j,\text{cat}}\})}{\text{tcs}(u_j)}
\]
where \( tcs(u_j) \) represents the total number of check-ins that the user \( u_j \) conducted in Foursquare, while \( tcs(u_j, \{v_{i,c} \}) \) stands for the total number of check-ins at venues belonging to the same category \( v_i \).

Finally, the venue score can be computed according to Eq. 7, considering both the venue popularity and the user preferences.

\[
\begin{align*}
\mathcal{VS}(u_j, v_i) &= \mathcal{VS}_{obj}(v_i) + \mathcal{VS}_{sub}(u_j, v_i) \\
&= \frac{1}{\Delta} \sum_{v_{i,c} \in V_{u_j}(i)} tcs(v_{i,c}) \cdot x(v_{i,c}) + \frac{1}{\Delta} \sum_{v_{i,c} \in V_{u_j}(i)} tcs(v_{i,c}) \cdot x(v_{i,c})
\end{align*}
\]

Three Heuristic Algorithms. As discussed previously, there are two important steps in route augmentation: selecting new venues, and inserting them into the candidate routes sequentially. It is not trivial since we have to make a trade-off between the individual venue scores and the total number of venues that can be added. For example, inserting a far away venue with a very high venue score might forbid adding more new venues, since it has already used up the time budget. In contrast, inserting a close-by venue with an average venue score first would allow including more new venues. It is difficult to say which strategy would achieve a higher route score in the end. Hence, we propose three heuristic algorithms for maximizing the route score in the route augmentation phase. Note that added venues are all user-preferred venues.

**Algorithm 3 Travel Time Minimizer Algorithm**

**Input:** A candidate route \( R = \{v_{1}, v_{2}, \ldots, v_{n}\} \); A set of new venues \( V_{c} \) (i.e. user-preferred venues); A user-specified total travel time budget \( \Delta \).

**Output:** An augmented route \( R \) of the candidate route.

1: \( augR = \emptyset \)
2: for \( i := 1 \) to \( |V_c| \) do
3: \( v_{ci} = V_c(i) \)
4: \( augR = \{R + v_{ci}\} \cup augR \)
5: //Call the venue inserting algorithm
6: \( TC(i) = +\infty \)
7: //the total travel time cost of the route is set +\infty if the selected venue can not be inserted
8: \( TC(i) = TC(R + v_{ci}) \)
9: end if
10: end for
11: if \( arg\min(TC) \neq +\infty \) then
12: \( k = \arg\min(TC) \)
13: \( R = augR(k) \)
14: \( V_c = V_c - v_{ck} \)
15: end if
16: Repeat Lines 1–14
17: Until \( R \) keeps unchanged

**Travel Time Minimizer:** The basic idea of this algorithm is to insert as many new venues as possible, given the fact that the route score would be higher as the number of venues increases in general. Thus at each venue inserting iteration, our proposed heuristic is that the venue closest to the candidate route (measured by the additional travel time) would be selected first for insertion, regardless of its venue score. We illustrate the core part of the travel time minimizer algorithm in Algorithm 3.

For each candidate route returned by the route search phase, in each iteration, we need to examine all new venues in order to select one for the newly augmented route (Lines 2–10). This is quite time-consuming especially when the size of the venue list is big. We use the venue inserting algorithm as shown in Algorithm 1 for each new venue (Lines 3–4). If the newly augmented route is not valid, its total travel time would be set to +\infty; otherwise, it would be updated to that of the newly augmented route (Lines 5–9). At the end of each iteration, the route in the \( augR \) set with the minimum total travel time cost will be selected as the input (Lines 11–15) for the next round of venue inserting, again via the same heuristic (Line 16). The algorithm would terminate when no new route can be generated (Line 17). Note that the inserted venue needs to be removed from the new venue list before the next iteration (Line 14). Therefore, the computation complexity in each iteration for this algorithm has an upper bound of \( O(n - 1)^N \), where \( n \) is the number of existing venues in the original candidate route, and \( N \) is the total number of user-preferred venues.

**Venue Score Maximizer:** The basic idea of this algorithm is to prioritize high-scored venues. Thus in each iteration, the venue with the highest venue score that can lead to a valid route would be inserted first, no matter how far away it is from the candidate route. Algorithm 4 illustrates the core part of the proposed venue score maximizer.

**Algorithm 4 Venue Score Maximizer Algorithm**

**Input:** A candidate route \( R = \{v_{1}, v_{2}, \ldots, v_{n}\} \); A set of new venues \( V_{c} \) (i.e. user-preferred venues); A user-specified total travel time budget \( \Delta \).

**Output:** An augmented route \( R \).

1: \( V_c \leftarrow \operatorname{sort}(V_c) \)
2: \( i = 1; v_{ci} = V_c(i) \)
3: \( augR = R + v_{ci} \)
4: while \( augR = R + v_{ci} \) do
5: \( if \ R + v_{ci} \in \Delta \) then
6: \( i = i + 1; v_{ci} = V_c(i) \)
7: \( augR = R + v_{ci} \)
8: else
9: \( \text{Break} \)
10: end if
11: end while
12: \( R = augR \)
13: \( V_c = V_c - v_{ck} \)
14: Repeat Lines 1~14
15: Until \( R \) keeps unchanged

In each iteration, we first sort the new venues in the descending order of venue scores defined in Eq. 7 (Line 1). This sorting operation can save computation time as we only need to check whether the high-ranked venues can yield a valid augmented route. If yes, there is no need to examine the rest of the venue list as in Algorithm 3. In the best case, the first venue (with the highest venue score) meets the requirement (Lines 2~3); in the worst case, all new venues will be checked (Lines 5~10). At the end of each iteration, the route with the highest route score will become the candidate route for the next iteration (Lines 12~14). The termination condition is the same as that of the travel time minimizer (Line 15). Again, the inserted venue would be excluded from further operations (Line 13). Therefore, the computation complexity in each iteration for this algorithm varies from \( O(N \log N) \) (i.e. the best case) to \( O(N \log N + (n - 1)^N) \) (i.e. the worst case). Note that \( O(N \log N) \) is the complexity of sorting operation.

The above two algorithms are used as baseline methods. The first heuristic algorithm only considers the number of the venues added, while the second one emphasizes merely on the scores of the inserted venues. As a result, the routes of the first algorithm would be generally longer (i.e. containing more venues), compared to the second algorithm. It is because the second heuristic algorithm, given the same time budget con-
strait, favors having one venue with a high venue score over two nearby average venues, even though the latter case might lead to a higher route score. To overcome the limitations of these two baseline methods, we propose our gravity maximizer.

Gravity Maximizer. Inspired by Newton’s law of universal gravitation which is capable of modelling human mobility patterns (the travel behaviors to places, travel patterns, etc.) [7], [32], we introduce a gravity model that uses the venue scores and the venue distances to the candidate route collectively for route augmentation. In our gravity model, the spherical distance between the candidate route and the new venue is analogy with the distance defined in Newton’s gravity model, where the location of the candidate route is obtained by averaging the locations of all venues that it contains. Likewise, the average venue score of the candidate route and the score of new venue corresponds to the mass. Finally, the gravity can be computed using Eq. 8.

$$G(v_{ci}, R) = \frac{\sum_{i=1}^{n} VS(u_j, v_i)}{\text{dist}(v_{ci}, R)^{\lambda}}$$

In the proposed gravity maximizer, the new venues are sorted in the descending order of their gravity values computed via Eq. 8, instead of the venue scores. The rest of procedure is exactly the same as that of venue score maximizer. Thus the two methods are similar in the computation complexity, with an extra cost of the venue’s gravity computation in the gravity maximizer.

In fact, the ranking based on gravity values would be degraded to that of venue scores if we set $\lambda = 0$, as gravity values would be determined by venue scores only. In other words, the gravity maximizer and venue score maximizer algorithms would reach the same result when $\lambda = 0$. On the contrary, as can be inferred from Eq 8, if we set $\lambda$ to be extremely high (e.g. $\lambda > 5$), the gravity values would be dominantly influenced by the distance to the candidate route, introducing a bias towards the “closest” venue (i.e. with the smallest distance to the candidate route). This agrees with the basic idea of the travel time minimizer algorithm. Furthermore, with a large negative $\lambda$ (e.g. $\lambda < -5$), “distant” venues would be ranked higher, which should be avoided. We will investigate how different $\lambda$ values affect the algorithm’s performance in Section VI-B.

3) Augmented Route Ranking: The algorithms discussed in Section V aim to optimally augment the set of candidate routes returned from the Phase I (i.e. route search). Augmented Route Ranking operation then picks out the augmented route with the highest route score to answer the user’s itinerary query (IQ). Note that if multiple “optimally” augmented routes possess the same route score as they may contain the same venues but in different order, the route with a smaller “total travel time” would be ranked higher.

VI. SYSTEM EVALUATION

Here, we present the evaluation results that aim to: (1) validate the efficiency and effectiveness of the trip planning algorithms and (2) demonstrate the usefulness and personalization capability of the trip planning system. We first describe the experiment setup, results of the parameter sensitivity study, as well as evaluation on algorithm efficiency and effectiveness, and then discuss several issues which need to be addressed further.

A. Experiment Setup

Data Preparation. We used Foursquare check-in data of San Francisco from April 2010 to October 2010, and the taxi GPS traces of the same city from the CabSpotting project (http://cabspotting.org/) to construct the POI network of San Francisco. The Foursquare data contains 110,214 check-ins generated by 15,680 users. The taxi GPS data contains 391,938 passenger-delivery trips generated by 536 taxis in June 2008. We did not include data from the vacant taxis since they might not drive at a normal speed when searching for passengers. Although we could not find two data sets from the same period for evaluation, the process of our proposed framework is data-independent, and the results can be easily updated once we are able to provide these heterogeneous data from the same period. The procedure of the POI network construction has been discussed in Section IV, and more details can be found in Appendix B.

Evaluation Environment. All the evaluations in the paper are run in Matlab on an Intel Xeon W3500 PC with 12-GB RAM and running Windows 7 operation system.

B. Parameter Sensitivity Study

As discussed in Section V, we have only one internal parameter $\lambda$ in the proposed gravity maximizer algorithm (Eq. 8), and no internal parameter in the other two baselines. We are thus interested in how it affects the optimal route score. We do not set $\lambda$ to extreme values as discussed; instead, we vary $\lambda$ in the range of [-3,3] with an interval of 0.1. The optimal scores under different $\lambda$ values, in comparison with the two baseline algorithms are shown in Fig. 5(a). As the figure suggests, the optimal route score generated by the travel time minimizer algorithm is always the lowest since it does not take the individual score of candidate venues into consideration. As expected, the optimal route score computed by the gravity maximizer algorithm and the venue score maximizer algorithm are the same when $\lambda$ is around 0. We also find that the gravity maximizer algorithm yields higher optimal route score than the venue score maximizer algorithm when $\lambda$ is within the range of [0.5, 2.3].

We also show the change in computation time of the gravity maximizer algorithm under different $\lambda$ values in Fig. 5(b). More specifically, the computation time fluctuates with the increase of $\lambda$. However, the maximum time cost is no longer than 1.45 seconds, which is acceptable. Considering the trade-off between route score and computation time, we choose $\lambda = 1.5$ for the rest of the evaluations.

C. Efficiency Evaluation

The efficiency of the three algorithms depends on several parameters, such as the total number of venues ($N$) in the targeted city, the number of user-preferred venue categories ($k$), the number of user-specified venues ($m$), and user-defined
algorithms are less time consuming compared to the travel time minimizer algorithm, which is consistent with the complexity analysis. Furthermore, the computation time of the travel time minimizer algorithm is almost proportional to $N$. This is logical as travel time minimizer needs to examine the additional travel time introduced by each venue in the candidate list. On the contrary, the computation time of the venue score maximizer and gravity maximizer algorithms only goes up slightly as the number of venues increases. Moreover, these two algorithms took less than one second to generate the result. The gravity maximizer algorithm generally took a slightly longer time than the venue score maximizer because of the additional gravity value calculation for each user-preferred venue. In this experiment, we fix $k = 3$ and $\Delta = 10$ hours.

2) Varying $k$: We show the computation time of the three algorithms under different $k$ in Fig. 6(b). In general, the computation time increases with $k$. This is because a larger $k$ often leads to a bigger number of user-preferred venues for augmentation. Again, the computation time of the travel time minimizer algorithm is much longer than that of the other two algorithms under the same setting, for the same reason as when $N$ varies. For the venue score maximizer and gravity maximizer algorithms, their computation time increases more significantly as $k$ becomes bigger, compared to that under different $N$. This is indeed caused by the increase of the number of user-preferred venues. As $N$ increases, both the number of user-preferred and non-user-preferred venues would increase. However, all non-user-preferred venues can be excluded from the route augmentation process and thus have no impact on the computation time. In contrast, any change in $k$ would be completely and directly reflected on the change in the number of user-preferred venues. In this experiment, we fix $N = 300$ and $\Delta = 8.5$ hours.

3) Varying $\Delta$: Fig. 6(c) shows the change in computation time of the three algorithms under given total travel time budget $\Delta$. Similar to the previous two cases, the travel time minimizer algorithm needs more time as $\Delta$ increases, much more than the other two algorithms of which the computation time was similar and no more than one second. In general, more user-preferred venues are allowed to be added which results in more venue inserting iterations in the route augmentation process, especially for the travel time minimizer algorithm since its objective is to minimize the introduced travel time at each iteration. In this experiment, we fix $N = 300$ and $k = 3$. 

travel time budget ($\Delta$). The first two variables determine the number of user-preferred venues (i.e. candidate new venues). The number of user-specified venues and travel time budget have an impact on the number of candidate routes produced in Phase I (i.e. the route search phase), as well as on the number of user-preferred venues that can be inserted in Phase II. Particularly, at most $m$ candidate routes can be produced. The number of user-specified venues ($m$) is common for all three algorithms, affecting the computation time in both the route search phase and the route augmentation phase. For simplicity, we fix $m = 5$ in all the evaluations. In the following experiments, we mainly study how the choice of $N$, $k$ and $\Delta$ affects the computation time of the three algorithms, varying only one parameter at a time.

It should be noted that all the candidate routes are augmented in parallel. In other words, the total computation time in the route augmentation phase is equal to the maximum computation time among all candidate routes. The efficiency is measured by the total time cost in both phases.

1) Varying $N$: The relationship between the computation time of the three algorithms and the total number of venues in the city ($N$) is shown in Fig. 6(a). Results suggest that the proposed venue score maximizer and gravity maximizer
D. Effectiveness Evaluation

Similar to the study of efficiency, we assessed the effectiveness of route augmentation algorithms under the same settings. The optimal route scores returned by the three algorithms with varying \( N, k \) and \( \Delta \) are shown in Fig. 7(a), Fig. 7(b), and Fig. 7(c) respectively. In Fig. 7(a), the experiment setting is \( m = 5, k = 3, \Delta = 10 \) hours; in Fig. 7(b), the setting is \( N = 300, m = 5, \Delta = 8.5 \) hours; and in Fig. 7(c), the setting is \( N = 300, m = 5, k = 3 \). In all three cases, the proposed gravity maximizer algorithm consistently outperformed the other two baseline methods in terms of optimizing the route score. Fig. 7(a) shows that the optimal route score of the travel time minimizer algorithm decreases gradually as \( N \) increases, as opposed to the gravity maximizer and venue score maximizer algorithms. This is because the inherent characteristic of the travel time minimizer algorithm biases towards venues that are closer but probably with a smaller score as \( N \) increases. Results also suggest that, compared to the venue score maximizer algorithm, the gravity maximizer algorithm is more likely to find the global optimal route score. In Fig. 7(b) and Fig. 7(c), all three algorithms achieved higher optimal route score with bigger \( k \) and \( \Delta \). However, such increase dramatically slowed down when \( k > 5 \), probably due to the time budget constraint we impose.

E. Case Study

We further tested the personalization capability of the TRIPPLANNER system in the case that two users with different personal interests submit the same query \((IQ_1)\) to the system. To be more specific, according to their check-in history, one of the users \((u_1)\) preferred Great Outdoors and Restaurants venues, while the other user favored more of the Arts & Entertainments and Restaurants venues. To demonstrate the traffic-aware capability of our TRIPPLANNER, we designed a second case in which \( u_1 \) modified the query and set a different trip starting time \((IQ_2)\). To verify that the route recommended by TRIPPLANNER is optimized, we introduced a third case in which the recommended route in response to \( IQ_2 \) by \( u_1 \) was compared to an average route. Queries in all three cases share the following information: i) The users start and end the trip both at the the Caltrain Station; ii) User-specified venues include Museum, Golden Gate Bridge, Beach, Lombard Street and Fisherman’s Wharf; iii) the total travel time budget \( \Delta \) is set to 11 hours; iv) the optional user-preferred categories are \{Restaurants, Arts & Entertainments, Great Outdoors\}; and v) the dining time is set to [11:00 am, 12:59 pm] for lunch and [17:30 pm, 20:00 pm] for dinner. Table II lists the information of the three cases we designed, including the corresponding user, starting time, and results of the recommended route.

**Case I: Personalization Capability.** This case intends to demonstrate the personalization capability of TRIPPLANNER with two different users. As shown in Fig. 8(a) and 8(b), given the same time budget, both users can accommodate four more preferred venues in their trips additional to the must-visit venues (i.e. \( R_1 \) and \( R_2 \)). Further investigation showed that, even though not explicitly requested, TRIPPLANNER recommended restaurants to both users around lunch and dinner time since they are food lovers (as shown in Fig. 8(e)).

<table>
<thead>
<tr>
<th>Case</th>
<th>Users</th>
<th>Starting Time</th>
<th>Recommended Route</th>
<th>Route Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>( u_1 )</td>
<td>10:00 am</td>
<td>( R_1 )</td>
<td>14.4176</td>
</tr>
<tr>
<td></td>
<td>( u_2 )</td>
<td>10:00 am</td>
<td>( R_2 )</td>
<td>14.6883</td>
</tr>
<tr>
<td>II</td>
<td>( u_1 )</td>
<td>08:30 am</td>
<td>( R_3 )</td>
<td>13.6602</td>
</tr>
<tr>
<td></td>
<td>( u_2 )</td>
<td>08:30 am</td>
<td>( R_4 )</td>
<td>12.9087</td>
</tr>
</tbody>
</table>

**Case II: Traffic-aware Capability.** This case looked into the traffic-aware capability of TRIPPLANNER. We compared two queries \((IQ_1 \text{ and } IQ_2)\) of the same user \( u_1 \) that only differ in the starting time (Table II). The recommended routes \((R_1 \text{ and } R_3)\) are shown in Fig. 8(a) and 8(c) respectively. Only three preferred venues can be added in \( R_3 \), as it starts around the morning rush hours and thus needs more transit time compared to the other two routes \((R_1 \text{ and } R_2)\), resulting in a smaller route score of 13.6602. Similar to \( R_1 \) and \( R_2 \), proper lunch and dinner are planned for the user. In addition, the user is suggested to visit the far-away Golden Gate Bridge first since most of venues such as museums are not yet opened early in the morning.

**Case III: Route Score Optimization Capability.** In this case, we are interested in the difference between the optimal...
Louvre lovers might spend the whole day in the particular POI is not trivial. It depends on the user’s interest actually estimating the stay time for each individual user at a stay time at a venue can be obtained in advance. However, score based on a user’s check-in history to characterize the individual. In this study, we intentionally add a subjective characterize the relative importance of different POIs for each user’s preferred stay time at different venues [12].

In the following, we discuss some issues of TripPLANNER, which need to be addressed in future work.

**Venue Stay Time.** In the current study, we assume that the stay time at a venue can be obtained in advance. However, actually estimating the stay time for each individual user at a particular POI is not trivial as well as his/her time budget. For instance, the museum lovers might spend the whole day in the Louvre, while some people only spend 2 hours to visit the most famous artworks, especially when the trip schedule is tight. In the future, we plan to explore other data sources and techniques to estimate each user’s preferred stay time at different venues [12].

**Route Score.** There is no objective way to quantitatively characterize the relative importance of different POIs for each individual. In this study, we intentionally add a subjective score based on a user’s check-in history to characterize the attractiveness of a POI to him/her, in addition to its popularity. Thus, the system would have problem if the user did not have check-in logs. To deal with this case, we can ask the user to input his check-in preference manually. Although the proposed scoring method that leverages the existing literature seems to work well, further research is needed to identify more effective ways to automatically assign attractiveness scores to different POIs and arrange the visiting order accordingly.

**Issues Regarding Data Sources.** In this study, we leverage two crowdsourced digital footprints to construct the dynamic POI model. In real life, nodes are also dynamic. Some of them may attract more and more visitors as time goes by, while others may become less popular. Also their operation time may change. To capture such change, we need to use a more recent data and update the nodes rather regularly. For example, for a venue, we can count the number of check-ins and the number of visitors during the past four months to renew its popularity score. We can also update its operation time information from other data sources, such as OpenStreetMap and official websites of the venues.

We propose the use of actual taxi GPS trace data to estimate the time-dependent transit time among venues (i.e. edge modelling). And thus, to demonstrate the validity and advantages of this approach, we choose driving as the travel mode. If we incorporate other data sources, such as subway or bus schedule timetable, our framework (i.e. two-phase approach) can estimate the transit time among venues by different travel modes more realistically, and can work adaptively to suggest the optimal route.
VII. CONCLUSION AND FUTURE WORK

In this study, we have developed a novel framework called TripPLANNER for personalized, interactive and traffic-aware trip planning. It leverages two heterogeneous data sources and considers factors including the varying transit time between POIs, user preferences, and the total travel time budget. First, we constructed the dynamic POI network model by extracting relevant information from crowdsourced Foursquare and taxi GPS traces. Then we proposed a two-phase approach for personalized trip planning with a comprehensive route scoring method and a novel route search-augmentation-ranking process. Using two real-world data sets which contain more than 391,900 passenger-delivery trips and 110,200 check-ins in the city of San Francisco, we compared our proposed route augmentation method with two baseline algorithms, and showed that our method is more efficient and effective than the baseline approaches. We further conducted a case study to validate the capability of our framework in recommending adaptive and optimal itineraries.

In the future, we plan to broaden and deepen this work in several directions. First, we intend to extend the scenarios to multi-day itinerary planning. Second, we would like to deploy our system on mobile devices, enabling a series of pervasive smart travel and transportation planning services. Third, we plan to test our system with real users in actual practices, collecting feedback on how to improve the service further.

ACKNOWLEDGEMENTS

The authors want to thank the anonymous reviewers and the editors for their helpful comments and suggestions. The authors also want to thank Dingqi Yang for the preparation of the Foursquare data set, and other colleagues for insightful discussions. This research was partly supported by the National High Technology Research and Development Program of China (863) under Grant No. 2013AAA01A605, the National Key Basic Research Program of China under Grant No. 2013CB329504, New Century Excellent Talents in University Program under Grant No. NCET-13-0521, and the National Natural Science Foundation of China under Grant No. 61332005, 61373119, 61222209. D. Zhang is the corresponding author of this paper.

REFERENCES

Fig. 9. User A and user B travel from \( v_i \) to \( v_{i+1} \) through the same path (a) and different paths (b).


APPENDIX A

PROOF OF THE FIFO PROPERTY

Proof. For two users (A and B), if user A starts from \( v_i \) earlier than user B (\( sT_A < sT_B \)), then user A would arrive at \( v_{i+1} \) at least the same time as user B (\( aT_A \leq aT_B \)).

Since the taxi GPS traces can provide more information about potential routes, such as the length, estimated travel time, and popularity [9], we assume that we could recommend the user with the fastest driving routes at the given time of the day for any given pair of origin and destination (i.e. OD).

Consider the case in Fig. 9(a): both users (user A and B) drive from \( v_i \) to \( v_{i+1} \) through the same path (i.e. Path 1). User A would be ahead of user B since user A departs from \( v_i \) earlier than user B. Assuming that the traffic conditions in the former segment of Path 1 might be better at the time when user B departs (i.e. \( sT_B \)), user B would take less time to complete this segment. Therefore, it is likely that user A and user B meet at a certain point (e.g. the point with star marker) in the route before reaching the destination. If so, user A and user B would arrive at \( v_{i+1} \) at the same time; otherwise, user A would arrive at \( v_{i+1} \) earlier than user B.

Consider the case in Fig. 9(b): user A drives from \( v_i \) to \( v_{i+1} \) through Path 1 at time \( sT_A \), while user B drivers from \( v_i \) to \( v_{i+1} \) through Path 2 at time \( sT_B \). For comparison, we assume that a user C drives from \( v_i \) to \( v_{i+1} \) through Path 2 at time \( sT_C \). Since the fastest driving route between \( v_i \) and \( v_{i+1} \) when departing at the time \( sT_A \) is Path 1, we conclude that user A would arrive at \( v_{i+1} \) earlier than user B. Similar to the case in Fig. 9(a), as user C departs from \( v_i \) earlier than user B through the same path (i.e. Path 2), user C would arrive at \( v_{i+1} \) no later than user B. Therefore, user A would arrive at \( v_{i+1} \) earlier than user B.

\[\square\]

APPENDIX B

EDGE MODELLING

In this appendix, we first introduce the representation and indexing of the taxi GPS trajectory, then estimate the transit time between any two given venues (edge values), depending on the user’s departure time.

A. Taxi Trajectory Representation and Indexing

Fig. 10 illustrates a taxi delivery trajectory. Each small circle point refers to a GPS sampling point; each triangle point refers to a venue in the targeted city. Note that the bigger circles (e.g., \( C_i \)) are the clusters of adjacent venues, constructed using the popular mean-shift algorithm [11], [21].

During a taxi ride, the driver may go through several venue clusters, and how long it needs to transit between any two passing-by clusters can be inferred. Thus, we represent the taxi trajectory as a sequence of venue clusters. For example, the taxi trajectory in Fig. 10 can be represented as:

\[\langle(\bar{T}_1, C_1), (\bar{T}_2, C_2), (\bar{T}_k, C_k)\rangle\]

where \( \bar{T}_i \) is the average value of the sampling time between the taxi’s first entry of and first exist from the venue cluster \( C_i \). Consequently, from this trajectory, we can deduce that the transit time needed from \( C_i \) to \( C_j \) when leaving \( C_i \) at time \( \bar{T}_i \) is \( \bar{T}_j - \bar{T}_i \), from \( C_i \) to \( C_k \) at time \( \bar{T}_k \) is \( \bar{T}_k - \bar{T}_i \), and from \( C_j \) to \( C_k \) departing from \( C_j \) at time \( \bar{T}_j \) is \( \bar{T}_k - \bar{T}_j \). Thus, this trajectory can be further represented as three quadruples pairs:

1. \(\langle[\bar{T}_1], C_i, C_j, \bar{T}_j - \bar{T}_i\rangle\)
2. \(\langle[\bar{T}_1], C_i, C_k, \bar{T}_k - \bar{T}_i\rangle\)
3. \(\langle[\bar{T}_j], C_j, C_k, \bar{T}_k - \bar{T}_j\rangle\)

Operation \( [\cdot] \) is to get the corresponding time slot of the day for a given point of time. In particular, we divide a day into five time slots in the scope of this paper, as shown in Table III. Note that second and fifth time slots are the rush hours, and more transit time is needed usually. In total, there are \( \frac{n(n-1)}{2} \)
number of quadruples pairs for a given taxi trajectory, where \( n \) is the number of venue clusters that the taxi bypasses.

We can derive hundreds of thousands of quadruples pairs from the taxi GPS trace dataset using this representation. The quadruples pairs with the same first three elements (i.e. time slot, the departure cluster, and the arrival cluster) will have the same Id.

**B. Transit Time Estimation**

Since the best way to move from one venue to anther inside the same cluster is by walking, the transit time between within-cluster venues is simply estimated by the ratio between their spherical distance and the walking speed (i.e. 4.5 km/h). For venues in different clusters (e.g. \( C_i \) and \( C_j \)), the transit time depends on the user’s departure time (e.g. 9:00 AM), and it can be estimated by averaging the fourth element in all quadruples pairs with the index of \( (3, C_i, C_j) \). Considering the difference in driving speed between a traveller and local taxi drivers, and the time spent on parking, we multiply the transit time estimated from taxi trajectories with a constant \( c_1 \) (=1.3) in the scope of this paper, and add a constant parking time \( c_2 \) (=3 minutes).

**APPENDIX C**

**VENUE CATEGORY ENCODING AND RETRIEVING**

In this appendix, we propose a simple venue category encoding mechanism to retrieve user-preferred venues from all city venues efficiently, according to the user-preferred venue categories \( (CAT_u) \) in the itinerary query \( (IQ) \). We first briefly introduce how Foursquare organizes venue category, and then describe our proposed solution.

In Foursquare, venues are organized in a three-level ontology structure (Fig. 11). It has 9 categories on the first level and 412 sub-/sub-subcategories (i.e. level 2 and level 3). Further analysis showed that, in Foursquare, each venue is often marked with more than one category labels distributed across different levels. For example, the average number of labels per venue in San Francisco is 1.421. In addition, when planning for an actual visit, users may describe a venue at different level of details as their knowledge, background, and experiences vary. For instance, when a user specifies the “Food” category, not only venues marked with “Food”, but also those associated with the child category labels, e.g. “New American Restaurant” and “Bakery” should be returned.

Based on these two facts, we propose a simple venue encoding mechanism that assigns a unique number to each venue category label with information about its superordinate integrated. Specifically, we use a 6-digit number to encode a venue category label. For Level 1 categories, only the first two digits are used and the rest digits are set to zero. For example, 05000100 refers to the “Outdoors & Recreations” category label. For a Level 2 category label (sub-category), the first two digits refers to its parent category, the middle two digits encodes its position among all siblings, and the last two digits remain zero. Taking 0501100 as an example, it refers to the “Bridge” label which is a child of “Outdoors & Recreations”. Similarly, for Level 3 category labels (sub-sub-category), all six digits are non-zeros. The former four digits denote to which the refer to its category and sub-category it belongs, respectively.

With this coding system, the user-preferred venues can be retrieved through the following four steps.

**Step 1**: For each user-specified venue category, we encode it into a 6-digit number. The number is denoted as \( Num_i \).

**Step 2**: For each venue in the targeted city, we encode its venue category labels into corresponding 6-digit numbers. Note that one venue may be assigned to more than one label numbers.

**Step 3**: Given the encoded number for the user-specified venue category (i.e. \( Num_i \), the output of Step 1), we retrieve the user-preferred venues from all venues in the targeted city. Specifically,

- if the user-specified venue category has no child labels (i.e. the last two digits of \( Num_i \) are non-zeros), venues in the targeted city with the exactly same encoded number as \( Num_i \) would be marked as user-preferred venues;
- otherwise, venues that contain the encoded numbers

**TABLE III**

<table>
<thead>
<tr>
<th>Time slot</th>
<th>Specific time duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>00:00~05:59</td>
</tr>
<tr>
<td>2</td>
<td>06:00~07:59</td>
</tr>
<tr>
<td>3</td>
<td>08:00~10:59</td>
</tr>
<tr>
<td>4</td>
<td>11:00~16:59</td>
</tr>
<tr>
<td>5</td>
<td>17:00~19:59</td>
</tr>
<tr>
<td>6</td>
<td>20:00~23:59</td>
</tr>
</tbody>
</table>

*Fig. 11. Ontology structure of venue categories.*
$Num_i$ or any of child labels of $Num_i$ should be retrieved. For example, if $Num_i=010100$, venues contain encoded numbers from 010100 to 0101099 are all returned.

**Step 4:** Repeat Step 1–3 until all user-specified venue category labels are checked. Venues retrieved in each round will be unified to form the final output.

---

**Chao Chen** is an associate professor at College of Computer Science, Chongqing University, Chongqing, China. He obtained his Ph.D. degree from Pierre and Marie Curie University and Institut Mines-TELECOM/TELECOM SudParis, France in 2014. He received the B.Sc. and M.Sc. degrees in control science and control engineering from Northwestern Polytechnical University, Xi’an, China, in 2007 and 2010, respectively. Mr. Chen was a co-recipient of the Best Paper Runner-Up Award at MobiQuitous 2011. In 2009, he worked as a Research Assistant with Hong Kong Polytechnic University, Kowloon, Hong Kong. His research interests include pervasive computing, social network analysis, data mining from large-scale taxi data, and big data analytics for smart cities.

**Daqing Zhang** is a professor at Institute of Software, School of Electronics Engineering and Computer Science, Peking University, China. He obtained his Ph.D. from University of Rome “La Sapienza” and the University of L’Aquila, Italy in 1996. His research interests include large-scale data mining, urban computing, context-aware computing, and ambient assistive living. He has published more than 180 refereed journal and conference papers, all his research has been motivated by practical applications in digital cities, mobile social networks and elderly care. Dr. Zhang is the Associate Editor for four journals including ACM Transactions on Intelligent Systems and Technology. He has been a frequent Invited Speaker in various international events on ubiquitous computing. He is the winner of the Ten Years CoMoReA Impact Paper Award at IEEE PerCom 2013, the Best Paper Award at IEEE UIC 2012 and the Best Paper Runner Up Award at MobiQuitous 2011.

**Bin Guo** is a professor from Northwestern Polytechnical University, China. He received his Ph.D. degree in computer science from Keio University, Japan in 2009 and then was a post-doc researcher at Institut Mines-TELECOM/TELECOM SudParis in France. His research interests include ubiquitous computing, mobile crowd sensing, and HCI.

**Xiaojuan Ma** is a researcher at Huawei Noah’s Ark Lab. Before joining Noah’s Ark, she was a postdoctoral researcher in the Human-Computer Interaction Institute at Carnegie Mellon University (CMU), and was awarded Computing Innovation Fellow by Computing Research Association. Dr. Ma received her Ph.D. degree from the Computer Science Department of Princeton University in 2010. She worked as a research fellow in the Department of Information Systems, National University of Singapore before joining CMU. Dr. Ma’s background is in Human-Computer Interaction. She is particularly interested in human computation and crowdsourcing, multimedia-augmented communication support for both human-human and human-robot interactions, design, visual/auditory perception, and (computational) linguistics.

**Gang Pan** received the B.Sc. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1998 and 2004, respectively. He is currently a Professor with the College of Computer Science and Technology, Zhejiang University. He has published more than 90 refereed papers. He visited the University of California, Los Angeles, Los Angeles, during 2007–2008. His research interests include pervasive computing, computer vision, and pattern recognition. Dr. Pan has served as a Program Committee Member for more than ten prestigious international conferences, such as IEEE International Conference on Computer Vision and IEEE Computer Society Conference on Computer Vision and Pattern Recognition.

**Zhaohui Wu** received the B.Sc. and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, China, in 1988 and 1993, respectively. He is currently a professor with the Department of Computer Science, Zhejiang University. His research interests include distributed artificial intelligence, semantic grid, and pervasive computing. Dr. Wu is a Standing Council Member of the China Computer Federation.