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 traveling 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 point-of-interest 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*—Crowdsourcing, digital footprints, personalization, traffic aware, trip planning.**

I. Introduction

PLANNING an itinerary is one of the most important and time-consuming travel preparation activities [11], [32]. 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 San Francisco International Airport at 9:00 A.M. and would like to spend several hours for sightseeing, and then leaves for San Jose by train at 15:00 P.M. from 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 (Caltrain Station), the POIs expected to be covered in the itinerary (e.g., the Golden Gate Bridge), the POI categories that 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 that can be estimated and controlled by users, the transit (driving) time between POIs that 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 that 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" that 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 [7], [8], [22], [35] 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

Manuscript received April 27, 2014; revised July 19, 2014; accepted September 6, 2014. Date of publication November 10, 2014; date of current version May 29, 2015. This work was supported in part by the National High Technology Research and Development Program of China (863) under Grant 2013AA01A605; by the National Key Basic Research Program of China under Grant 2013CB329504; by the New Century Excellent Talents in University Program under Grant NCET-13-0521; and by the National Natural Science Foundation of China under Grant 61332005, Grant 61373119, and Grant 61222209. The Associate Editor for this paper was Q. Zhang. (*Corresponding author: Daqing Zhang*.)

C. Chen is with the Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education, Chongqing University, Chongqing 400044, China, and also with the College of Computer Science, Chongqing University, Chongqing 400044, China (e-mail: cschaochen@cqu.edu.cn).

D. Zhang is with the Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: daqingi2r@yahoo.com).

B. Guo is with the Department of Computer, Northwestern Polytechnical University, Xi'an 710072, China (e-mail: guobin.keio@gmail.com).

X. Ma is with Noah's Ark Lab, Huawei, Shatin, Hong Kong (e-mail: xiaojuan.ma@huawei.com).

G. Pan and Z. Wu are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: gpan@zju.edu.cn; wzh@zju.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TITS.2014.2357835

[1] We use *venue* and *POI* interchangeably throughout this paper.

[39]. Others identify venues according to a user's preference and recommend routes based on certain criteria (e.g., with the highest route score) [12], [15], [19], [25]. However, during the actual trip planning, it is common that a user may have some additional constraints such as "need to go some specific places," "go to park before lunch," and "the total travel time should be $\leq$ 6 h." Some prior work asked users to manually select and configure travel routes after recommendation, which was tedious and time consuming [19], [34]. 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 [37]. 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 that contain tourists' profiles, and comments that reveal preferences and experiences with POIs [5], [6]; 2) social media sites such as Facebook, Flickr, and location-based social networks (LBSNs) (e.g., Foursquare and Gowalla), which can inform the popularity, functions, and operating hours of the POIs and individual user's travel history [8], [20], [25]; 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], [16], [37], [39]. 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 varied constraints. In the back end, it leverages heterogeneous crowdsourced digital footprints for POI network model construction. Through a two-phase query resolution process, TRIPPLANNER could recommend to the user a personalized route with the highest trip score under the total travel time constraint. In summary, there are four main contributions of this paper.

- First, we define a new trip planning problem, which allows users to specify not only the must-visit venues but also optional venue categories if the time permits, given a total travel time budget. We further make more realistic assumption about the transit time between venues that varies with time of the day and day of the week. In other words, the total travel time of the same route may be different.
- Second, we attempt to construct a dynamic POI network model of a city, leveraging *heterogeneous crowdsourced digital footprints* (i.e., Foursquare check-ins and taxi GPS traces) to better utilize the strengths of each data source in characterizing the attributes of the nodes and links of the POI network.
- Third, we propose a *two-phase approach* for *personalized, interactive, and traffic-aware* trip planning. We also propose a new way to score an itinerary, considering both the popularity and individual preference of venues. Specifically, in the *route search phase*, the system works *interactively* 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. Through a case study of planning three trips in San Francisco with different starting times and user preferences, it is shown that TRIPPLANNER can recommend appropriate routes that fully meet users' requirements and yet take into consideration the traffic condition along the chosen routes at the specified time.

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 this paper and chart the future directions in Section VII.

II. RELATED WORK

The related work is organized in two sections. 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], [11], [20], [26], and [40], many researchers have used geotagged photos from photosharing 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 be also extracted. However, it is hard to estimate the dynamic transit time between POIs from geotagged photo data. More recently, people began to explore user-generated LBSN digital traces since such data contain rich information that can be used to *directly* characterize each POI in a tourist city and users' preferences to each POI [4], [15], [24], [25], [36]. Unfortunately, similar to geotagged photo data, LBSN

TABLE I
DIFFERENCES BETWEEN PRIOR WORK AND OUR TRIPPLANNER SYSTEM IN TERMS OF USER PREFERENCES, NODE, EDGE, AND BUDGET CONSTRAINTS

	Paper	User-preferences	Node constraint	Edge constraint	Budget constraint
Route Search	[24]	×	S, E, Specified POIs	Static	×
	[23]	×	S, E, POI categories	Static	×
	[8]	×	S, E, POI categories	Static	Total time
	[19]	×	S, E, POI categories	Static	×
	[30]	×	S, POI categories	Static	×
	[36]	×	POI categories	Static	×
	[14], [22]	×	S, E, POI categories	Traffic-aware	×
Route Recommendation	[16], [21]	✓	S, E	Static	Total time
	[26]	✓	S, E	Static	Total money
Route Planning	[38], [42]	×	S, E	Traffic-aware	×
TRIPPLANNER		✓	S, E, Specified POIs and categories	Traffic-aware	Total time

[1] × denotes the respected factor is NOT considered; ✓ denotes the respected factor is considered;
[2] S is short for the starting place; E is short for the ending place;
[3] Some papers list have some additional node constraints, such as POI visiting order, POI visiting time. For instance, [16] has a visiting time constraint for the POIs. [19], [30] impose a visiting order constraint for the POIs.

traces also do not contain dynamic transit time between POIs, particularly when driving is considered for traveling 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 [37]. Previous studies have shown that GPS trajectory traces can precisely characterize the transit time between POIs, which is more accurate than Google Maps[2] results [3], [9], [17]; 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], [30], [33], [38], [42], 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 [23]. 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 rather than a specific POI location. *Trip planning query* (TPQ) is proposed to address the problem [22]. 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], [18], [29], [35], but most of the studies assumed that the transit time between POIs is constant, with the exception of work like [13], [14], and [21]. 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 that incorporates user preferences, location, and available time to suggest personalized routes [20]. Lu *et al.* present a *personalized trip recommendation* framework that recommended personalized arrangement of visit to venues, given a predefined budget (e.g., time and money) [25]. Hsieh *et al.* propose to utilize users' check-in patterns to recommend popular time-sensitive trips to users [15]. 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.* [37] and Ziebart *et al.* [41] 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, and 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, and 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 composed of three major

[2] http://maps.google.com

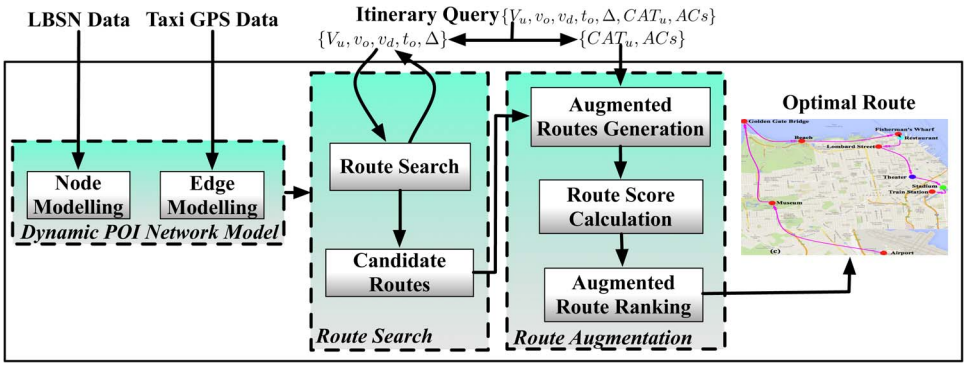

Fig. 1. Framework of our proposed TRIPPLANNER.

parts, namely, a dynamic POI network model, a route search component, and a route augmentation component (see 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, namely, *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 $tt(v_i, v_j)$. The transit time is asymmetric and dynamically changing.

Lemma 1 (Dynamic POI Network has FIFO Property): Given a dynamic network $G = (V, E)$, where the transit time of each edge in G is time dependent. The network is first-input–first-output (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 $(t_1 > t_0)$, 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 Δ; 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 be thus 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 $R = \langle v_1, v_2 \ldots, v_n \rangle$ is *valid* iif

$$aT(v_i) \geq oT(v_i), lT(v_i) \leq cT(v_i) \qquad \forall\, i \in \{1, 2, \ldots, n\}.$$

This implies that the user should visit all venues while they are open. Here, $aT(\cdot), lT(\cdot)$ are the users' arriving and leaving times for the given venue, whereas $oT(\cdot), cT(\cdot)$ refer to the opening and closing times 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 [12] and [19]; otherwise, the route score is defined as 0 (i.e., there exists a case in which a user arrives in 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, i.e., the dynamic POI network model, the route search component, and the route augmentation component. While the dynamic POI network model is prebuilt and maintained offline, the route search and route augmentation components collaboratively answer users' trip queries in real time.

1) 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 traverse 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 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.

Fig. 2. Relevant information of the node provided by Foursquare.

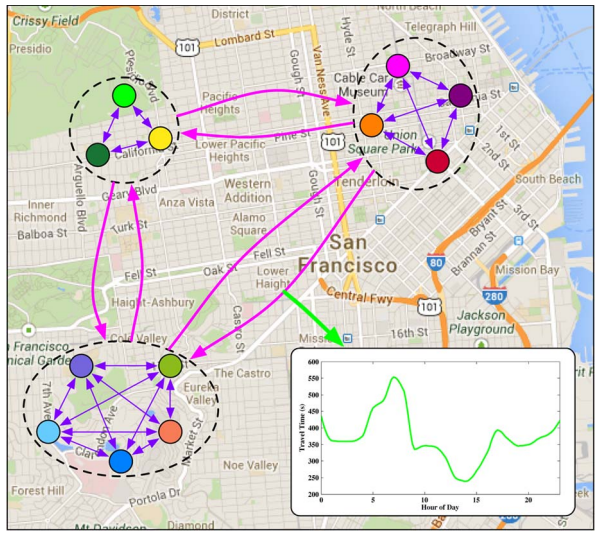

Fig. 3. Illustration of the dynamic network built with Foursquare and taxi GPS data sets.

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 Modeling

A. Node Modeling

Each node in the model corresponds to a POI with five attributes, i.e., *operation time, category that the venue belongs to, popularity, geographical location, and stay time*. For each venue, users provide their expected stay time, whereas we extract information related to the first four attributes from Foursquare data (see 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.[3] 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, i.e., the total number of visitors (tvs) and the total number of check-ins tcs (1). The total visitor number is usually *smaller* than the total check-in number of the same venue since some user checked in repeatedly during a single visit.

$$Pop(v_i) = \frac{2 \times \frac{tvs(v_i)}{c_1} \times \frac{tcs(v_i)}{c_2}}{\frac{tvs(v_i)}{c_1} + \frac{tcs(v_i)}{c_2}} \tag{1}$$

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 [27].

[3]Foursquare categorizes all venues in a three-level hierarchy. More details can be found at http://aboutfoursquare.com/foursquare-categories

Regarding the *geographical location* of a given venue, Foursquare provides the longitude/latitude information together with its address.

Although the exact value of the *stay time* at a given venue cannot be precisely derived from the check-in data, it could be approximated using the average stay time of tourists. Note that users might estimate the stay time when planning the trip and adjust it during the actual visit.

B. Edge Modeling

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 paper, we leverage a real-world data set—taxi GPS traces. Such data set has two unique features: 1) *spatial coverage*: a certain number of city taxis can fully cover the whole road network; and 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 colocated nodes among which walking is the best way to get around. The within-cluster transit time is computed using the average walking speed, whereas the between-cluster transit time is estimated based on the driving speed at the specific time slot. Fig. 3 illustrates a simple dynamic POI network. The small circles in different colors refer to the nodes (POIs). Nearby 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, whereas 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. 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_u 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 (OD) that cover the valid venues in the list via a simple enumeration. Users may input a long venue list that 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. Afterward, we need to compute the total travel time for all remaining routes, considering the arrival and departure times of the user at each venue. Specifically, the total travel time is computed as the sum of transit and stay times 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 the 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 $\mathcal{O}(m!)$, 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 that 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, whereas the "later-arrival" venue would be still late when inserting a venue after it*. In other words, the outputs 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.

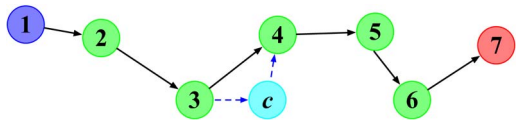

Fig. 4. Illustrative example of inserting a venue into a candidate route.

Remark: As this paper targets the travel route planning issue for tourists visiting a city for a day trip by renting a car, we thus assume that the number of user-specified venues is not big [28]. Therefore, the route search problem is just finding all the routes that meet the visiting order and time constraints, and the time spent on route search in a city scale is relatively short.

For the case that users specify no venue, the route search phase will generate one candidate route from the starting place to the ending place with no other POIs in between. The route augmentation phase (discussed in the following section) would insert possible POIs as time permits to generate a near-optimal route with maximal score.

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*, whereas 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 times. 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; and 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 be also different due to the time shift. Taking the diagram in Fig. 4 as an example, after inserting venue v_c 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 and 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, i.e., *venue inserting* and *score*

maximization. The former aims to find a suitable position in the candidate route to insert a selected venue, whereas the latter is responsible for maximizing the score of the updated route.

1) 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 that contains "earlier-arrival" venues, inserting a candidate user-preferred venue behind the first "earlier-arrival" venue could not lead to a valid route.

The pseudocode 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 Δ;
Output: An augmented route $augR$
1: **if** R has "earlier-arrival" venues **then**
2: $k = pos(R)//pos(R)$ gets the index of the first "earlier-arrival" venue in R
3: $augR = augRoute(R, v_c, [2, k], \Delta)$
4: **else**
5: $augR = augRoute(R, v_c, [2, n], \Delta)$
6: **end if**

Algorithm 2 Venue Inserting Function

1: **Function** $augR = augRoute(R, v_c, [a, b], \Delta)$
2: $newR = \emptyset$
3: **for** $k := a$ **to** b **do**
4: $tmpR = \langle v_1, v_2, \ldots, v_c, \ldots, v_n \rangle$ //the index of v_c in $tmpR$ is k, and the venue orders in R kept unchanged in $tmpR$
5: **if** $tmpR$ is *valid* **then**
6: $newR = newR \cup tmpR$
7: **end if**
8: **end for**
9: **if** $newR$ is empty or $\min[\mathcal{TC}(newR)] > \Delta$ **then**
10: $augR = R$
11: **else**
12: $augR = \arg\min \mathcal{TC}(newR)$ //select the newly augmented route with the minimal total travel time cost
13: **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 this paper, we use the expression $R + \{v_{c1}, v_{c2}, \ldots, v_{cn}\}$ 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 and 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

$$\max \mathcal{RS}\left(u_i, R + \{x_i v_{ci}\}_{i=1}^{N}\right). \tag{2}$$

Subject to:

$$x_i \in \{0, 1\} \tag{3}$$

$$x_1 v_{c1}.cat \cup x_2 v_{c2}.cat \cup \ldots \cup x_n v_{cN}.cat \subseteq CAT_u \tag{4}$$

$$\mathcal{TC}\left(R + \{x_i v_{ci}\}_{i=1}^{N}\right) \leq \Delta \tag{5}$$

where (2) refers to the objective function (i.e., the route score) for maximization. It is subjected to three constraints, as shown in (3)–(5). Equation (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. Equation (5) emphasizes that the total time cost of the newly augmented route should be within the predefined travel time budget Δ.

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 [Pop, as shown in (1)], which is objective (denoted as $\mathcal{VS}_{\text{obj}}$). 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 $\mathcal{VS}_{\text{sub}}$ of the venue v_i for user u_j is calculated by (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.

$$\mathcal{VS}_{\text{sub}}(u_j, v_i) = \frac{tcs\,(u_j, \{v_i.cat\})}{tcs(u_j)} \tag{6}$$

where $tcs(u_j)$ represents the total number of check-ins that the user u_j conducted in Foursquare, whereas $tcs(u_j, \{v_i.cat\})$ 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 (7), considering both the venue popularity and the user preferences.

$$\mathcal{VS}(u_j, v_i) = \mathcal{VS}_{\text{obj}}(v_i) + \mathcal{VS}_{\text{sub}}(u_j, v_i). \tag{7}$$

Three Heuristic Algorithms: As previously discussed, there are two important steps in route augmentation, i.e., *selecting* new venues and *inserting* them into the candidate routes sequentially. It is not trivial since we have to make a tradeoff between the individual venue scores and the total number of venues that can be added. For example, inserting a faraway 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.

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.

Algorithm 3 Travel Time Minimizer Algorithm

Input: A candidate route $R = \langle v_1, v_2, \ldots, v_n \rangle$;
A set of new venues V_c (i.e., user-preferred venues);
A user-specified total travel time budget Δ;
Output: An augmented route
1: $augR = \emptyset$
2: **for** $i := 1$ **to** $|V_c|$**do**
3: $v_{ci} = V_c(i)$
4: $augR = \{R + v_{ci}\} \cup augR$
//Call the venue inserting algorithm
5: **if** $R + v_{ci} = R$ **then**
6: $TC(i) = +\infty$
//the total travel time cost of the route is set $+\infty$ if the selected venue cannot be inserted
7: **else**
8: $TC(i) = \mathcal{TC}(R + v_{ci})$
9: **end if**
10: **end for**
11: **if** $any(TC) \neq +\infty$**then**
12: $k = \arg\min_i(TC)$
13: $R = augR(k)$
14: $V_c = V_c - v_{ck}$
15: **end if**
16: **Repeat lines** 1–14
17: **Until** R **keeps unchanged**

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, particularly 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 and 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 $\mathcal{O}(n-1)^N$, 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 $R = \langle v_1, v_2, \ldots, v_n \rangle$;
A set of new venues V_c (i.e., user-preferred venues);
A user-specified total travel time budget Δ;
Output: An augmented route
1: $V_c \leftarrow sort(V_c)$
//Sort new venues according to their scores in descending order defined in (7)
2: $i = 1; v_{ci} = V_c(i)$
3: $augR = R + v_{ci}$

4: **while** $augR = R$ **do**
5: **if** $i < |V_c|$ **then**
6: $i = i + 1; v_{ci} = V_c(i)$
7: $augR = R + v_{ci}$
8: **else**
9: **Break**
10: **end if**
11: **end while**
12: $R = augR$
13: $V_c = V_c - v_{ci}$
14: **Repeat lines** 2–14
15: **Until** R **keeps unchanged**

In each iteration, we first *sort* the new venues in the descending order of venue scores defined in (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 and 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 $\mathcal{O}(N \log |N|)$ (i.e., the best case) to $\mathcal{O}(N \log |N| + (n-1)^N)$ (i.e., the worst case). Note that $\mathcal{O}(N \log |N|)$ 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, whereas 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 with the second algorithm. It is because the second heuristic algorithm, given the same time budget constraint, favors having one venue with a high venue score over two nearby average venues, although 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 modeling human mobility patterns (the travel behaviors to places, travel patterns, etc.) [7], [31], 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 analogous 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 correspond to the *mass*. Finally, the gravity can be computed using

$$G(v_{ci}, R) = \frac{\mathcal{VS}(u_j, v_{ci}) \times \frac{1}{n} \sum_{i=1}^{n} \mathcal{VS}(u_j, v_i)}{dist(v_{ci}, R)^{\lambda}}. \quad (8)$$

In the proposed *gravity maximizer*, the new venues are *sorted* in the descending order of their gravity values computed via (8), instead of the venue scores. The rest of the procedure is exactly the same as that of the *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 the *venue score maximizer* algorithms would reach the same result when $\lambda = 0$. On the contrary, as can be inferred from (8), if we set λ 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 toward 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 λ (e.g., $\lambda < -5$), "distant" venues would be ranked higher, which should be avoided. We will investigate how different λ 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 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 experimental setup, results of the parameter sensitivity study, and evaluation on algorithm efficiency and effectiveness and then discuss several issues that need to be addressed further.

A. Experimental 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 contain 110 214 check-ins generated by 15 680 users. The taxi GPS data contain 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

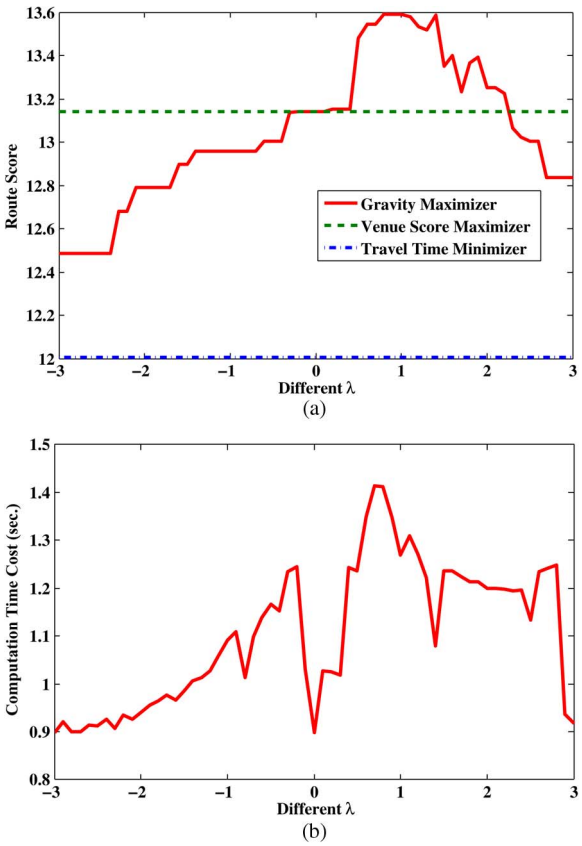

Fig. 5. Results of parameter sensitivity study. (a) Optimal route scores under different λ. (b) Computation time cost under different λ.

been discussed in Section IV, and more details can be found in Appendix B.

Evaluation Environment: All the evaluations in this 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 λ in the proposed *gravity maximizer* algorithm [(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 λ to extreme values as discussed; instead, we vary λ in the range of $[-3, 3]$ with an interval of 0.1. The optimal scores under different λ 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 λ is around 0. We also find that the *gravity maximizer* algorithm yields higher optimal route score than the *venue score maximizer* algorithm when λ is within the range of [0.5 2.3].

We also show the change in computation time of the *gravity maximizer* algorithm under different λ values in Fig. 5(b). More specifically, the computation time fluctuates with the increase in λ. However, the maximum time cost is no longer than 1.45 s, which is acceptable. Considering the tradeoff 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 travel time budget Δ. 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. In particular, 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 Δ 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* algorithms are less time consuming compared with 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 the *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 1 s 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 $\Delta = 10$ h.

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, as compared with that under different

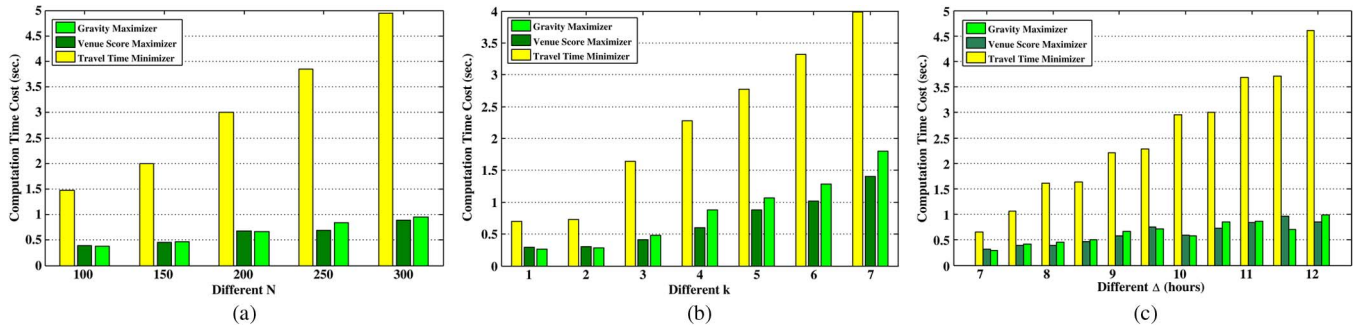

Fig. 6. Results of efficiency evaluation. (a) Computation time cost by varying N. (b) Computation time cost by varying k. (c) Computation time cost by varying Δ.

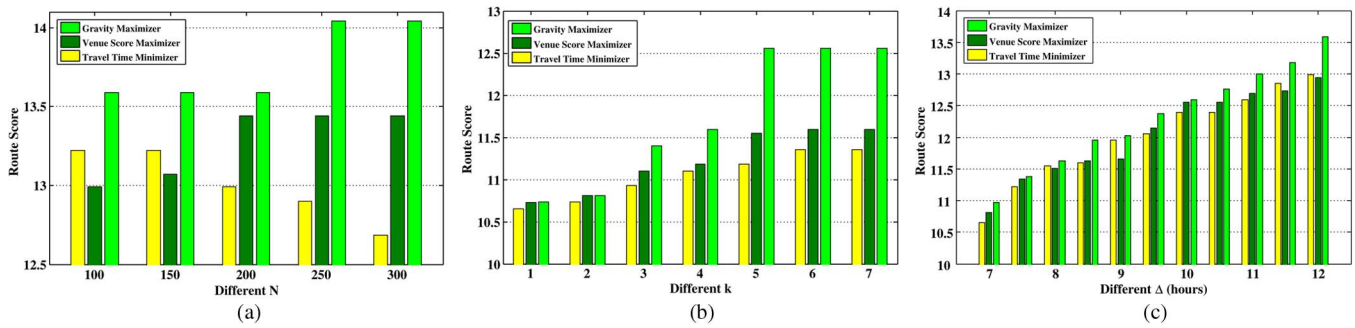

Fig. 7. Results of effectiveness evaluation. (a) Optimal route score by varying N. (b) Optimal route score by varying k. (c) Optimal route score by varying Δ.

N. This is indeed caused by the increase in 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 $\Delta = 8.5$ h.

3) Varying Δ: Fig. 6(c) shows the change in computation time of the three algorithms under given total travel time budget Δ. Similar to the previous two cases, the *travel time minimizer* algorithm needs more time as Δ increases, much more than the other two algorithms of which the computation time was similar and no more than 1 s. In general, more user-preferred venues are allowed to be added, which results in more venue inserting iterations in the route augmentation process, particularly 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 $k = 3$.

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 Δ are shown in Fig. 7(a)–(c), respectively. In Fig. 7(a), the experiment setting is $m = 5$, $k = 3$, $\Delta = 10$ h; in Fig. 7(b), the setting is $N = 300$, $m = 5$, $\Delta = 8.5$ h; and in Fig. 7(c), the setting is $N = 300$, $m = 5$, and $k = 3$. 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 gradually decreases 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 toward venues that are *closer* but probably with a *smaller* score as N increases. Results also suggest that, compared with the *venue score maximizer* algorithm, the *gravity maximizer* algorithm is more likely to find the global optimal route score. In Fig. 7(b) and (c), all three algorithms achieved higher optimal route score with bigger k and Δ. However, such increase dramatically slowed down when $k > 5$, probably due to the time budget constraint we impose.

TABLE II
INFORMATION ABOUT THE THREE DESIGNED CASES

	Users	Starting Time	Recommended Route	Route Score
Case I	u_1	10:00 am	R_1	14.4176
	u_2	10:00 am	R_2	14.6883
Case II	u_1	10:00 am	R_1	14.4176
	u_1	08:30 am	R_3	13.6602
Case III	u_1	08:30 am	R_3	13.6602
	u_1	08:30 am	R_4	12.9087

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, whereas the other user favored the *Arts & Entertainments* and *Restaurants* venues more. To demonstrate the *traffic-aware*

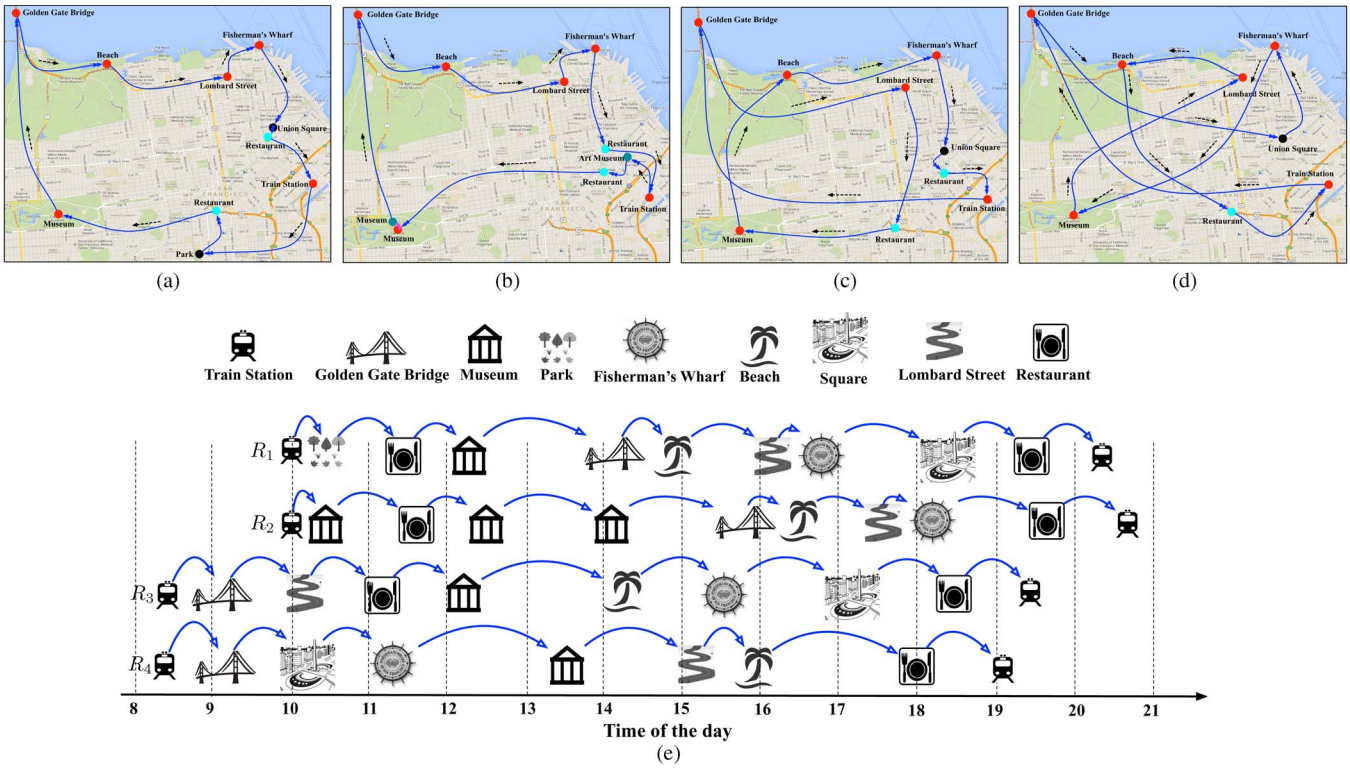

Fig. 8. Results of the case study. (a)–(d) Trip routes on Google map (in the spatial dimension). (Best viewed in the enlarged digital version). (a) R_1. Starting time: 10:00 A.M. (b) R_2. Starting time: 10:00 A.M. (c) R_3. Starting time: 08:30 A.M. (d) R_4. Starting time: 08:30 A.M. (e) Comparison of four routes in the temporal dimension.

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. 1) The users start and end the trip both at *Caltrain Station*. 2) User-specified venues include *Museum, Golden Gate Bridge, Beach, Lombard Street, and Fisherman's Wharf*. 3) The total travel time budget Δ is set to 11 h. 4) The optional user-preferred categories are *{Restaurants, Arts & Entertainments, Great Outdoors}*. 5) The dining time is set to [11:00 A.M., 12:59 P.M.] for lunch and [17:30 P.M., 20:00 P.M.] 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 (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, although 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)]. For user u_1, the other two venues added belong to the *Great Outdoors* category, whereas two more museums from the *Arts & Entertainments* category appeared in the augmented itinerary for u_2. As illustrated in Fig. 8(e), u_1 arrives at *Caltrain Station* a bit earlier than u_2, suggesting that routes R_1 and R_2 have different travel times. These results clearly indicate the ability of TRIPPLANNER to customize both specified and top-ranked preferred venues in the recommended trip, according to users' preferences.

Case II: Traffic-Aware Capability: This case looked into the *traffic-aware* capability of TRIPPLANNER. We compared two queries (IQ_1 and IQ_2) of the same user u_1 that only differ in the starting time (see Table II). The recommended routes (R_1 and R_3) are shown in Fig. 8(a) and (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 with the other two routes (R_1 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 faraway *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 route* versus an *average route*. Fig. 8(d) shows a randomly selected augmented route R_4, which is generated by our proposed *gravity maximizer* algorithm under the same query as Fig. 8(c). The user is also suggested to visit the faraway *Golden Gate Bridge* first in R_4. Although the average route includes all the user-specified venues and meets the total travel time budget constraint, it only accommodates two preferred venues due to the long transit time caused by taking an *inefficient* venue visiting order. As a result, the user only has time to take a quick snack for lunch if taking R_4. Therefore, its route score (12.9087) is much lower than that of the recommended optimal route R_3.

F. Discussion

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. It depends on the user's interest and his/her time budget. For instance, the museum lovers might spend the whole day in the *Louvre*, whereas some people only spend 2 h to visit the most famous artworks, particularly 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 [11].

Route Score: There is no objective way to quantitatively characterize the relative importance of different POIs for each individual. In this paper, 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 a problem if the user did not have check-in logs. To deal with this case, we may 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 paper, 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, whereas others may become less popular. In addition, their operation time may change. To capture such change, we need to use 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 modeling). 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 paper, 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 that 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. 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 multiday 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.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers and the editors for their helpful comments and suggestions. The authors would also like to thank D. Yang for the preparation of the Foursquare data set and other colleagues for insightful discussions.

REFERENCES

[1] A. Aguiar, F. Nunes, M. Silva, P. Silva, and D. Elias, "Leveraging electronic ticketing to provide personalized navigation in a public transport network," *IEEE Trans. Intell. Transp. Syst.*, vol. 13, no. 1, pp. 213–220, Mar. 2012.

[2] Y. Arase, X. Xie, T. Hara, and S. Nishio, "Mining people's trips from large scale geo-tagged photos," in *Proc. MM*, 2010, pp. 133–142.

[3] R. K. Balan, K. X. Nguyen, and L. Jiang, "Real-time trip information service for a large taxi fleet," in *Proc. MobiSys*, 2011, pp. 99–112.

[4] J. Bao, Y. Zheng, and M. F. Mokbel, "Location-based and preference-aware recommendation using sparse geo-social networking data," in *Proc. GIS*, 2012, pp. 199–208.

[5] S. Basu Roy, G. Das, S. Amer-Yahia, and C. Yu, "Interactive itinerary planning," in *Proc. IEEE ICDE*, 2011, pp. 15–26.

[6] I. Brilhante, J. A. Macedo, F. M. Nardini, R. Perego, and C. Renso, "Where shall we go today? Planning touristic tours with TripBuilder," in *Proc. ACM CIKM*, 2013, pp. 757–762.

[7] F. Calabrese, G. Di Lorenzo, L. Liu, and C. Ratti, "Estimating origin–destination flows using mobile phone location data," *IEEE Pervasive Comput.*, vol. 10, no. 4, pp. 36–44, Apr. 2011.

[8] X. Cao, L. Chen, G. Cong, and X. Xiao, "Keyword-aware optimal route search," *Proc. VLDB Endowment*, vol. 5, no. 11, pp. 1136–1147, Jul. 2012.

[9] V. Ceikute and C. Jensen, "Routing service quality—Local driver behavior versus routing services," in *Proc. IEEE MDM*, 2013, pp. 97–106.

[10] C. Chen, D. Zhang, N. Li, and Z.-H. Zhou, "B-Planner: Planning bidirectional night bus routes using large-scale taxi GPS traces," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 4, pp. 1451–1465, Aug. 2014.

[11] M. De Choudhury *et al.*, "Automatic construction of travel itineraries using social breadcrumbs," in *Proc. HT*, 2010, pp. 35–44.

[12] T. Deng, W. Fan, and F. Geerts, "On the complexity of package recommendation problems," *SIAM J. Comput.*, vol. 42, no. 5, pp. 1940–1986, 2013.

[13] I. Hefez, Y. Kanza, and R. Levin, "TARSIUS: A system for traffic-aware route search under conditions of uncertainty," in *Proc. GIS*, 2011, pp. 517–520.

[14] A. V. Hill and W. C. Benton, "Modelling intra-city time-dependent travel speeds for vehicle scheduling problems," *J. Oper. Res. Soc.*, vol. 43, no. 4, pp. 343–351, Apr. 1992.

[15] H.-P. Hsieh, C.-T. Li, and S.-D. Lin, "Exploiting large-scale check-in data to recommend time-sensitive routes," in *Proc. UrbComp*, 2012, pp. 55–62.

[16] T. Hunter, P. Abbeel, and A. M. Bayen, "The path inference filter: Model-based low-latency map matching of probe vehicle data," *IEEE Trans. Intell. Transp. Syst.*, vol. 15, no. 2, pp. 507–529, Apr. 2014.
[17] T. Hunter, P. Abbeel, R. Herring, and A. Bayen, "Path and travel time inference from GPS probe vehicle data," in *Proc. NIPS*, 2009, pp. 1–8.
[18] Y. Kanza, R. Levin, E. Safra, and Y. Sagiv, "Interactive route search in the presence of order constraints," *Proc. VLDB Endowment*, vol. 3, no. 1/2, pp. 117–128, Sep. 2010.
[19] M. Khabbaz, M. Xie, and L. V. S. Lakshmanan, "Toprecs+: Pushing the envelope on recommender systems," *IEEE Data Eng. Bull.*, vol. 34, no. 2, pp. 61–68, Jun. 2011.
[20] T. Kurashima, T. Iwata, G. Irie, and K. Fujimura, "Travel route recommendation using geotags in photo sharing sites," in *Proc. CIKM*, 2010, pp. 579–588.
[21] R. Levin and Y. Kanza, "TARS: Traffic-aware route search," *GeoInformatica*, vol. 18, no. 3, pp. 1–40, Jul. 2014.
[22] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng, "On trip planning queries in spatial databases," in *Proc. SSTD*, 2005, pp. 273–290.
[23] S. Lin, "Computer solutions of the traveling salesman problem," *Bell Syst. Tech. J.*, vol. 44, no. 10, pp. 2245–2269, Dec. 1965.
[24] B. Liu, Y. Fu, Z. Yao, and H. Xiong, "Learning geographical preferences for point-of-interest recommendation," in *Proc. ACM KDD*, 2013, pp. 1043–1051.
[25] E. H.-C. Lu, C.-Y. Chen, and V. S. Tseng, "Personalized trip recommendation with multiple constraints by mining user check-in behaviors," in *Proc. GIS*, 2012, pp. 209–218.
[26] X. Lu, C. Wang, J.-M. Yang, Y. Pang, and L. Zhang, "Photo2Trip: Generating travel routes from geo-tagged photos for trip planning," in *Proc. MM*, 2010, pp. 143–152.
[27] Y. Lun Chou, *Statistical Analysis With Business and Economic Applications*. New York, NY, USA: Holt, Rinehart and Winston, 1969.
[28] M. Papandrea, M. Zignani, S. Gaito, S. Giordano, and G. Rossi, "How many places do you visit a day?" in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops*, 2013, pp. 218–223.
[29] M. Sharifzadeh, M. Kolahdouzan, and C. Shahabi, "The optimal sequenced route query," *VLDB J.*, vol. 17, no. 4, pp. 765–787, Jul. 2008.
[30] W. Souffriau and P. Vansteenwegen, "Tourist trip planning functionalities: State-of-the-art and future," in *Proc. Current Trends Web Eng.*, 2010, pp. 474–485.
[31] J. Truscott and N. M. Ferguson, "Evaluating the adequacy of gravity models as a description of human mobility for epidemic modelling," *PLoS Comput. Biol.*, vol. 8, no. 10, p. e1002699, Oct. 2012.
[32] P. Vansteenwegen, W. Souffriau, G. V. Berghe, and D. V. Oudheusden, "The city trip planner: An expert system for tourists," *Expert Syst. Appl.*, vol. 38, no. 6, pp. 6540–6546, Jun. 2011.
[33] F.-Y. Wang, "Parallel control and management for intelligent transportation systems: Concepts, architectures, applications," *IEEE Trans. Intell. Transp. Syst.*, vol. 11, no. 3, pp. 630–638, Sep. 2010.
[34] M. Xie, L. V. Lakshmanan, and P. T. Wood, "Breaking out of the box of recommendations: From items to packages," in *Proc. ACM Conf. Recommender Syst.*, 2010, pp. 151–158.
[35] B. Yao, M. Tang, and F. Li, "Multi-approximate-keyword routing in GIS data," in *Proc. GIS*, 2011, pp. 201–210.
[36] M. Ye, P. Yin, W.-C. Lee, and D.-L. Lee, "Exploiting geographical influence for collaborative point-of-interest recommendation," in *Proc. SIGIR*, 2011, pp. 325–334.
[37] J. Yuan, Y. Zheng, X. Xie, and G. Sun, "T-drive: Enhancing driving directions with taxi drivers' intelligence," *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 1, pp. 220–232, Jan. 2013.
[38] J. Zhang *et al.*, "Data-driven intelligent transportation systems: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1624–1639, Dec. 2011.
[39] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma, "Mining interesting locations and travel sequences from GPS trajectories," in *Proc. WWW*, 2009, pp. 791–800.
[40] Y.-T. Zheng, Z.-J. Zha, and T.-S. Chua, "Mining travel patterns from geotagged photos," *ACM Trans. Intell. Syst. Technol.*, vol. 3, no. 3, pp. 56:1–56:18, May 2012.
[41] B. D. Ziebart, A. L. Maas, A. K. Dey, and J. A. Bagnell, "Navigate like a cabbie: Probabilistic reasoning from observed context-aware behavior," in *Proc. UbiComp*, 2008, pp. 322–331.
[42] K. Zografos, K. Androutsopoulos, and V. Spitadakis, "Design and assessment of an online passenger information system for integrated multimodal trip planning," *IEEE Trans. Intell. Transp. Syst.*, vol. 10, no. 2, pp. 311–323, Jun. 2009.

Chao Chen 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, and the Ph.D. degree from Pierre and Marie Curie University, Paris, France, and Institut Mines-Telecom/Telecom SudParis, Evry, France, in 2014.

In 2009, he was a Research Assistant with The Hong Kong Polytechnic University, Kowloon, Hong Kong. He is currently an Associate Professor with the College of Computer Science, Chongqing University, Chongqing, China. He is also with the Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education, Chongqing, China. His research interests include pervasive computing, social network analysis, data mining from large-scale taxi data, and big data analytics for smart cities.

Dr. Chen was a corecipient of the Best Paper Runner-Up Award at Mobiquitous 2011.

Daqing Zhang received the Ph.D. degree from the University of Rome "La Sapienza", Rome, Italy, and the University of L'Aquilam, L'Aquila, Italy, in 1996.

He is currently a Professor with the Institute of Software, School of Electronics Engineering and Computer Science, Peking University, Beijing, China. He has published more than 180 referred journal and conference papers; all his research has been motivated by practical applications in digital cities, mobile social networks, and elderly care. His research interests include large-scale data mining, urban computing, context-aware computing, and ambient assistive living.

Dr. Zhang is the Associate Editor of four journals, including the *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 the IEEE PerCom 2013, the Best Paper Award at the IEEE UIC 2012, and the Best Paper Runner Up Award at Mobiquitous 2011.

Bin Guo received the Ph.D. degree in computer science from Keio University, Tokyo, Japan, in 2009

He was a Postdoctoral Researcher with the Institut Mines-Telecom/Telecom SudParis, Evry, France. He is currently a Professor with Northwestern Polytechnical University, Xi'an, China. His research interests include ubiquitous computing, mobile crowd sensing, and human–computer interaction.

Xiaojuan Ma received the Ph.D. degree in computer science from Princeton University, Princeton, NJ, USA, in 2010.

She is currently a Researcher with Noah's Ark Lab, Huawei, Shatin, Hong Kong. Before joining Noah's Ark Lab, she was a Postdoctoral Researcher with the Human-Computer Interaction Institute, Carnegie Mellon University (CMU), Pittsburgh, PA, USA. She was a Research Fellow with the Department of Information Systems, National University of Singapore, Singapore, before joining CMU. Her 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.

Dr. Ma is a Computing Innovation Fellow of the Computing Research Association.

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 visited the University of California, Los Angeles, CA, USA, in 2007–2008. He has published more than 90 refereed papers. 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 the IEEE International Conference on Computer Vision and the 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 College of Computer Science and Technology, 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.