Customized Tour Recommendations in Urban Areas

Aristides Gionis
HIIT and Aalto University
aristides.gionis@aalto.fi

Theodoros Lappas
Stevens Institute of Technology
tlappas@stevens.edu

Konstantinos Pelechrinis
University of Pittsburgh
kpele@pitt.edu

Evimaria Terzi
Boston University
evimaria@cs.bu.edu

ABSTRACT

The ever-increasing urbanization coupled with the unprecedented capacity to collect and process large amounts of data have helped to create the vision of intelligent urban environments. One key aspect of such environments is that they allow people to effectively navigate through their city. While GPS technology and route-planning services have undoubtedly helped towards this direction, there is room for improvement in intelligent urban navigation. This vision can be fostered by the proliferation of location-based social networks, such as `Foursquare` or `Path`, which record the physical presence of users in different venues through *check-ins*. This information can then be used to enhance intelligent urban navigation, by generating customized path recommendations for users.

In this paper, we focus on the problem of recommending customized tours in urban settings. These tours are generated so that they consider (a) the different types of venues that the user wants to visit, as well as the order in which the user wants to visit them, (b) limitations on the time to be spent or distance to be covered, and (c) the merit of visiting the included venues. We capture these requirements in a generic definition that we refer to as the TourRec problem. We then introduce two instances of the TourRec problem, study their complexity, and propose efficient algorithmic solutions. Our experiments on real data collected from `Foursquare` demonstrate the efficacy of our algorithms and the practical utility of the reported recommendations.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.2.8 [**Database Management**]: Database Applications—*Data Mining*

Keywords

urban computing; tour recommendations

WSDM'14, February 24–28, 2014, New York, New York, USA.

http://dx.doi.org/10.1145/2556195.2559893.

1. INTRODUCTION

Based on a recent report from the United Nations, more than 50% of the world's population currently lives in cities. This percentage is projected to increase to 70% by the year 2050 [2]. These facts have motivated the concept of *smart cities* [16], that will provide intelligent services to significantly enhance the quality of urban life. One of the critical services within smart cities is *urban navigation*. While GPS technology has undoubtedly improved people's capability to move easily through a city, intelligent urban navigation goes *beyond* shortest paths. In this paper, we study how we can utilize contextual information to provide customized *tour recommendations* for advanced urban navigation.

In order to accomplish our goal we exploit information provided by a new class of applications: *location-based social networks*. These applications enable users to share their locations through *check-ins* to specific *venues* (e.g., restaurants, bars, parks, museums etc.). This interaction captures the user's physical presence within a city and results in rich datasets that encode real-world activities.

One of the main functionalities currently offered by location-based social networks is the recommendation of *venues* or *location-specific activities*, based on user preferences. For example, `Foursquare`, the largest location-based social network to date, with more than 30 million users, is positioning itself as a location-recommendation engine, used to recommend venues to users based on their own check-in history, as well as that of their friends [1].

A key characteristic of such recommendation systems [8, 25, 33] is that they evaluate each venue *independently*. The result of this evaluation is a personalized probability that a specific user likes the venue, or a general value representing the venue's quality or reputation. The venues with the top-k highest scores are then recommended to the user. The main drawback of such ranking schemes is their inability to evaluate venues in *groups*, and deliver *tour recommendations* that obey certain thematic and spatial constraints.

As an example, consider the scenario shown in Figure 1. A user must travel for business in an unfamiliar part of the megacity in which she lives (e.g., NYC). After her work is completed, she wants to explore this area further. In particular, she is interested in taking a "stroll" to visit an outdoor relaxing spot, (e.g., a nice park — Type A in the figure), before heading to a nice restaurant for lunch (Type B). From there, she wants to go to a popular place for a happy-hour drink (Type C) and then return to her initial point (marked by the bulls-eye in the figure) so she can go back home. Since

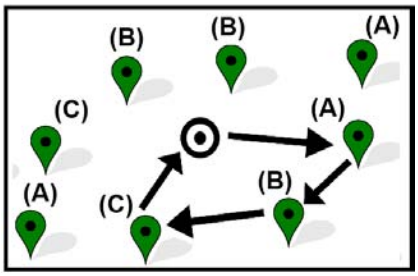


Figure 1: An example of the Tour Recommendation setting.

there are different venues from each type scattered across the map, there are multiple potential tours that respect the given type order (such as the one marked in the figure). The tour-recommendation problem becomes even more complex if we also want to keep the total distance within a given budget, and also maximize the overall merit of the venues included in the tour, as we do in this paper.

Even though a number of tour-recommendation approaches have been recently proposed [14, 23, 27, 34] (for a review see Section 5), those approaches suffer from a number of limitations. These include the dependence on training data in the form of past tours or trajectories, the assumption that all venues belong to the same type and thus service the same user need, and the inability to cater to the user who wants to cover different needs in a specific order during the tour.

In this paper, we develop a framework that addresses these shortcomings by delivering tour recommendations that take into account: (*i*) the different types of venues, as well as the order in which the user wants to visit them; (*ii*) budget constraints expressed in terms of the total distance that the user is willing to traverse (or the total time the user is willing to spend); and (*iii*) the satisfaction that each venue can provide to the user, if included in the tour. In our work, we propose and study the computational and the qualitative effect of two alternative definitions for user satisfaction.

We capture and formalize the above requirements in our definition of the TourRec problem, which asks for a group of venues that adheres to the user constraints and also maximizes the expected user satisfaction. Depending on the way user satisfaction is measured, we define two variants of the TourRec problem: the first variant assumes a (personalized or general) expected satisfaction score for every venue, and asks for the tour with the maximum total satisfaction. The second variant captures the scenario where each venue *covers* certain attractions (e.g., nearby city landmarks, current location-specific events). The goal is then to find a tour that respects the constraints and covers the maximum number of distinct attractions. We study the complexity of both variants and propose efficient algorithmic solutions.

At a high level, the TourRec problem is related to variants of the traveling-salesman problem (e.g., TSP, prize-collecting TSP, orienteering). However, the partitioning of the venues into types, the ordering constraints, as well as the characteristics of our objective functions, necessitate customized algorithms for our problem. In our experiments, we demonstrate the superiority of our methods over competitive baselines via an extensive evaluation on real datasets collected from `Foursquare` for three major cities.

Roadmap: The rest of the paper is organized as follows: we start with the formal definition of our problems in Section 2. The algorithms we design for the different variants of the problems are described in Section 3 and they are experimentally evaluated in Section 4. We review the related work in Section 5 and conclude the paper in Section 6.

2. PROBLEM DEFINITION

Next, we introduce our notation, define the TourRec problem and its variants, and discuss their complexity.

2.1 Preliminaries

Throughout our discussion, we use the term *venue* to refer to locations, landmarks, restaurants, museums etc. We assume that there are n venues $V = \{v_1, ..., v_n\}$, and a function $\text{Dist}(\cdot)$ that measures the physical distance between any two venues. Our assumption is that every venue is associated with a *type*. Indicative types are: museum, gallery, cinema, restaurant, etc. For venue v, we use $T(v)$ to denote its type. We assume that each venue is associated with one type and that there are m *venue types* $\mathcal{T} = \{T_1, \ldots, T_m\}$.

We assume a function $f(v)$ that returns a custom value for a given venue v. The value represents the expected satisfaction of the user, if the user visits the particular venue. We assume that higher values of $f(v)$ correspond to higher satisfaction levels. In principle, different users get different satisfaction levels from different venues and therefore the values of $f(v)$ are personalized. Although obtaining accurate estimates of such personalized values is an important problem that is beyond the scope of this paper, when such an estimation function is available (e.g., using the algorithms proposed by Bao et al. [6]) it can be incorporated as it is in our framework. In our implementation we estimate expected user satisfaction using intuitive proxies, such as the total number of check-ins or the average review score.

Our goal is to find a *tour* $\mathcal{R}$ that maximizes the satisfaction of the user and obeys a set of user-specified constrains. A tour is a *sequence* of venues. For example, $\mathcal{R}$ may be equal to $\mathcal{R} = \langle v_1, v_3, v_5 \rangle$, which means that the user is recommended to visit venue v_1, then venue v_3 and, finally, venue v_5. We use the notation $\mathcal{R}(i)$ to refer to the i-th venue in the tour. In addition to the sequence of venues, the tour includes a starting point s and ending point t, which are specified by the user and can be arbitrary locations on the map. We refer to the number of venues in the tour (excluding the source and the destination) as the *size* of the tour.

We consider the following constraints:

Ordering constraints: We assume that the user provides a *type order* π, which expresses precedence constraints in the order in which the user wants to visit venues of different types. The order π is a total order over the types in $\mathcal{T}$ and we say that $T_i \prec_\pi T_j$ if the user expresses a preference of visiting venues of type T_i *before* venues of type T_j. Although in Section 3.3 we discuss how these total order constraints can be relaxed, for now we assume that a tour $\mathcal{R}$ is *consistent with type order* π if for every $1 \leq i \leq |\mathcal{R}| - 1$ we have that if $v = \mathcal{R}(i)$ and $v' = \mathcal{R}(i+1)$, then $T(v) \prec_\pi T(v')$.

The order constraint is motivated by the observation that daily human activities are correlated with the time of the day. For example, people typically have coffee in the morning and visit restaurants in the afternoon and early evening

(lunch and dinner times). Also they cannot visit a museum in the evening since it will be most probably closed. This intuition has been repeatedly studied and verified in relevant literature [26, 15, 22, 17, 3], where it is found that tourists have a very specific opinion on when is the best time within the day for each type of activity or attraction.

In Section 3.3, we show how to extend our framework to address *partial orders on types*, *super types* and *type skips*, in which there is flexibility in the visiting order of venues.

Budget constraints: The budget constraints enable users to specify the total distance D they are willing to cover. In Section 3.3, we discuss how temporal or other types of constraints can also be incorporated. For now, we consider the total distance $\text{ShortestPath}(\mathcal{R})$ of a tour $\mathcal{R}$ that starts at s, traverses the venues in $\mathcal{R}$ in order, and ends in t.

Multiplicity bounds: The multiplicity bounds allow users to specify the number of venues of a particular type that they want to visit. This number is expressed as an interval. That is, the users provide a lower and an upper bound, L_i and U_i, on the number of venues of type T_i they are willing to visit. The set of lower and upper bounds over all venue types is denoted by $B = \{(L_i, U_i)\}_{i=1}^{m}$.

A tour $\mathcal{R}$ is *consistent with multiplicity bounds* B if for every type T_i the number of venues of type T_i in $\mathcal{R}$ are more than or equal to L_i and less than or equal to U_i.

User satisfaction: We use a generic *satisfaction function* $F(\mathcal{R})$ to measure the expected satisfaction of the user with respect to a candidate tour $\mathcal{R}$. We define two variants of this satisfaction function: the *additive* and *coverage* functions, denoted by $\mathcal{A}\text{dd}$ and $\mathcal{C}\text{ov}$, respectively. Each function evaluates $\mathcal{R}$ in a different way as we discuss below.

The additive satisfaction function $\mathcal{A}\text{dd}$*:* $\mathcal{A}\text{dd}$ assumes that there is a benefit $b(v)$ associated with every venue v.[1] This benefit may be generic or a personalized probability that a particular user will like the venue. Given such venue-specific benefits, the additive satisfaction function evaluates the overall satisfaction of a tour $\mathcal{R}$ as $\mathcal{A}\text{dd}(\mathcal{R}) = \sum_{v \in \mathcal{R}} b(v)$.

The coverage satisfaction function $\mathcal{C}\text{ov}$*:* This function is appropriate when each venue is associated with a set $S(v)$ that includes attractions that located in the vicinity of the venue (or even activities that the user can perform within or near the venue). Then, the coverage of a path $\mathcal{R}$ is defined as $\mathcal{C}\text{ov}(\mathcal{R}) = |\cup_{v \in \mathcal{R}} S(v)|$, and represents the total number of **distinct** attractions or activities that the user will have the chance to visit or perform during the tour.

Conceptually, $\mathcal{A}\text{dd}(\mathcal{R})$ is appropriate when the user wants to maximize the quality or the personalized relevance of the venues in the tour. On the other hand, $\mathcal{C}\text{ov}(\mathcal{R})$, is appropriate for more *diverse* routes that offer a wide range of associated attractions and activities.

2.2 The TourRec problem

Given the above notation, our goal is to solve the following generic problem.

Problem 1 (TourRec). *Given a type order π, a distance budget D, multiplicity bounds B, and starting and ending locations s and t, find a a tour $\mathcal{R}$ such that:*

(i) it is consistent with the order π and type bounds B;

(ii) it starts at location s and ends at location t;

(iii) the total covered distance is less than D, i.e., $\text{ShortestPath}(\mathcal{R}) \leq D$*;*

(iv) it maximizes the user satisfaction $F(\mathcal{R})$.

The different variants of the F function give rise to different variants of the TourRec problem. For example, when the total benefit is measured by $\mathcal{A}\text{dd}$, then we call the underlying problem the AdditiveTour problem; when the total benefit is measured by $\mathcal{C}\text{ov}$, then we refer to the resulting problem as the CoveringTour problem.

We have the following NP-hardness results for the complexity of these two problems.

Lemma 1. *The* AdditiveTour *problem is* **NP***-hard.*

For the proof of the above lemma it is sufficient to observe that AdditiveTour contains the Traveling Salesman Problem (TSP) [19]: let us consider the decision version of AdditiveTour, defined as follows: given V, $\mathcal{T}$, B, s, t, D and a satisfaction bound M, is there a tour $\mathcal{R}$ that satisfies all the constraints, has total distance cost at most D and $\mathcal{A}\text{dd}(\mathcal{R}) \geq M$? We can show that TSP can be reduced to this decision version of AdditiveTour. Indeed, consider a TSP instance on a graph $G = (V, E)$ with distance bound D. We then define an instance of the decision AdditiveTour problem as follows. We consider a graph with the same edge distances and a single type of venues $\mathcal{T} = \{T\}$. We also set $|T| = |V| = n$, $L_T = U_T = n$, $s = t$, $b(v) = 1$ for all venues $v \in V$, $M = n$, and distance bound D (the same as in the TSP instance). It is easy to see that the TSP instance has a solution if and only if the transformed instance of the decision AdditiveTour problem has a solution.

Similarly, for CoveringTour we have the following.

Lemma 2. *The* CoveringTour *problem is* **NP***-hard.*

The proof of Lemma 2 relies on the fact that the CoveringTour problem when there is a single type of venues $\mathcal{T} = \{T\}$ such that $|T| = |V| = n$, $B = \{(L, U)\}$ with $L = U = k$, and $s = t$ then our problem is identical to the $k-$Maximum Coverage problem [19]. In fact, we can show that the CoveringTour problem is NP-hard even when $L_i = U_i = 1$ for every type i and $D = \infty$. In this case, CoveringTour is identical to the Maximum Coverage with Group BudgetConstraints problem [10].

3. ALGORITHMS

We begin this section by describing our algorithms for AdditiveTour. Then, we demonstrate how these can be adapted for solving CoveringTour.

3.1 Algorithms for AdditiveTour

First, we consider the version of the AdditiveTour problem where we specify a total order π over the venue types and we want to have *exactly* one venue from every type in our tour. For this case, we have the following result:

Lemma 3. *When π is a total order, and $L_i = U_i = 1$ for all types $T_i \in \mathcal{T}$, then the* AdditiveTour *problem can be solved optimally in time $\mathcal{O}(n^2 D)$.*

Proof. This version of the AdditiveTour problem can be solved via dynamic programming. More specifically, we

[1] $b(v)$ is essentially a specific instantiation of the $f(v)$ function mentioned earlier.

define the table $B(v, d)$ to denote the maximum value of the $\mathcal{A}$dd function one can achieve with a walk consistent with π, which starts at s, ends at v, and covers a total distance which is at most d. Clearly, for n venues and maximum distance D, the size of the table B is $(n + 2) \times D$.

First, we sort the venue types according to the order π — the ordering of the venues within each type does not matter and can be arbitrary. Then, for every $v \in T_k$ we can evaluate $B(v, d)$ by following recursion

$$B(v, d) = \max_{v' \in T_{k-1}} \left\{ B(v', d - \text{Dist}(v, v')) + b(v) \right\}. \quad (1)$$

The time required for evaluating Equation (1) is $\mathcal{O}(n^2 D)$. Thus, the dynamic-programming algorithm is pseudo polynomial. For the special case where all the venues are on a graph with unit edge weights, the time required for the above recursion is polynomial in n. □

We refer to the dynamic-programming algorithm that implements recursion (1) as the `Rel-DP` algorithm.

In addition to the exact pseudo-polynomial `Rel-DP` algorithm, we can also have a fully polynomial-time approximation scheme (FPTAS) for the problem. This scheme firsts scales down the instance of the input problem so that all distances are polynomial in n. It then runs the `Rel-DP` algorithm on the scaled-down instance and returns the solution. In more detail, given a parameter ϵ, we define $K = \frac{\epsilon D}{n}$. We then scale down every distance by $\text{Dist}'(v, v') = \lfloor \frac{\text{Dist}(v,v')}{K} \rfloor$, and we run the `Rel-DP` algorithm with distances Dist′ and distance bound $D' = \lfloor \frac{D}{K} \rfloor$. The resulting dynamic-programming table has dimensions $(n+2) \times \lfloor \frac{n}{\epsilon} \rfloor$ and thus the running time of the algorithm is $\mathcal{O}(n^2 \lfloor \frac{n}{\epsilon} \rfloor)$. It is also easy to see that the returned solution is a $(1 - \epsilon)$-approximation to the optimal solution.[2] Summarizing, we have:

LEMMA 4. *When π is a total order and $L_i = U_i = 1$ for all types $T_i \in \mathcal{T}$, for any $\epsilon > 0$ the* AdditiveTour *problem be approximated within factor $(1 - \epsilon)$ in time $\mathcal{O}(n^2 \lfloor \frac{n}{\epsilon} \rfloor)$.*

Efficiency consideration: The $O(n^2 D)$ running times of the `Rel-DP` algorithm is a worst-case estimates. In practice, the running time of depends on the distribution of venues into types as well as the binning of D, i.e., the size of the step we adopt for distances when we construct the dynamic-programming tables. Our experiments show that appropriate binning can lead to very efficient algorithms.

3.2 Algorithms for CoveringTour

Again, we consider the version of the CoveringTour problem in which the user wants to visit *exactly one* venue from each type. We can solve this version of CoveringTour by modifying Recursion (1) as follows:

$$C(v, d) = \max_{v' \in T_{k-1}} \{ C(v', d - \text{Dist}(v', v)) + \mathcal{C}\text{ov}(v \mid \mathcal{R}_{v', d - \text{Dist}(v', v)}) \}, \quad (2)$$

where $C(v, d)$ keeps the best coverage of a path that ends in v and has total shortest path d, and $\mathcal{R}_{v',d}$ denotes the actual path that ends in v' and is within the distance budget d.

We refer to the dynamic-programming algorithm that implements Recursion (2) as the `Cover-DP` algorithm. The time complexity of evaluating Recursion (2) is the same as the time complexity for evaluating Recursion (1), i.e., $O(n^2 D)$. Note that the speedups discussed in the previous paragraph can be also applied here.

[2]Keep in mind that we are dealing with a maximization problem.

3.3 Extensions

In this section we present different extensions to our framework. We extend the applicability of our work and address intuitive scenarios that can emerge in urban settings.

Relaxing the total order constraint: Although the above discussion assumed a total order on the venue types, our algorithms can be easily modified for arbitrary partial type orders. In this section we provide three different ways to relax the total-order constraint: *partial orders of types*, *super types*, and *type skips*.

Partial orders. The first way to relax the total-order constraint is to allow the venue types to satisfy any partial order. In this scenario, our goal is to find a tour which visits venues in a way that respects a user-submitted partial order. we assume that a user provides a partial order, and the aim is to find a tour for which the ordering of the venues in the tour satisfies the partial order. A simple solution to this problem is to consider all *linear extensions* of the given partial order. Recall that a linear extension of a partial order is a total order that is consistent with the partial order. Therefore, we can enumerate all those total orders, solve the recommendation problem in each total order, and pick the best solution. Even though in the general case the number of linear extensions can be exponentially large, in our setting, we expect that a user specifies a small number of venue types (less than 5), so given a partial order it is possible to try all consistent total orders. Our results in Section 4.2 indicate that each total order instance can be computed very fast. Furthermore, the total number of distinct venue types in our dataset is 9, hence, the number of consistent total orders for any query is relatively small as well. In addition, each consistent total order can be evaluated independently, allowing for a naturally parallelizable implementation.

In practice, we should not expect from users to be familiar with the concept of partial orders in order to be able to specify their constraints. However, it is possible to design a system in which users specify their constraints as pair-wise precedence constraints via a simple user interface, and the constraints are represented internally as a partial order.

Super-types. An alternative family of ordering constraints, simpler than partial orders, are *bucket orders*, or as we call them in our setting, *super-types*. In this case, users can specify that they could visit *any* type of venues among a given subset. For example, users can specify that they are interested in visiting a museum OR a gallery OR a landmark and then go for lunch to a restaurant OR to a cafe. Those type groupings are called super-types. Our algorithms can then be applied directly to satisfy any such order. The idea is to simply consider super-types as types and apply the algorithms we developed in the previous sections.

The concept of super-types, coupled with allowing to specify multiple venues from each type, which we will discuss shortly, increases the flexibility of the user by a great deal. For example, a user may specify a super-type consisting of 4 types and asking for a recommendation that contains at most 3 venues from this super-type, and the same individual type can be repeated.

Type skips. Another way to relax the total order is to allow types to be skipped. So far we have assumed that exactly one venue from every type must be included in the recommended tour (i.e., $L_i = U_i = 1$). However, our methods can be generalized to find tours that are allowed to skip some types. For the AdditiveTour problem, this is captured by the following lemma.

Lemma 5. *Consider the* AdditiveTour *problem where* $L_i, U_i \in \{0, 1\}$ *(with* $L_i \leq U_i$*) for all types* $T_i \in \mathcal{T}$*. In this case, the* AdditiveTour *problem can be solved optimally in time* $\mathcal{O}(n^3 D)$.

Proof. We are using the same benefit table B of dimension $(n+2) \times D$ and the same ordering of venues as in the previous proof. However, Recursion (1) is modified to the following recursive evaluation:

$$B(v,d) = \max_{\substack{k'=1\ldots k-1 \\ v' \in T_{k'}}} \left\{ B(v', d - \text{Dist}(v,v')) + \mathcal{A}\text{dd}(v) \right\}. \quad (3)$$

The main difference between Recursion (1) and (3) is that in the latter when forming the tour that ends in venue $v \in T_k$, all tours ending in venues of all types that precede type T_k are examined (as opposed to only the tours that end in venues of type T_{k-1}). □

The complexity of recursion (3) is $O(n^3 D)$. A similar modification can be used to allow types to be skipped in the CoveringTour problem.

Multiple venues per type: Lemmas 3 and 5 assume that *exactly* and *at most* one venue from each venue type will be added in the final tour, respectively. For the AdditiveTour problem, the variant where multiple venues of the same type are required (i.e., $U_k > 1$) can be handled by extending the dynamic programming recursion of Equation (1). This can be done as follows: consider again the dynamic programming table $B(v,d)$ that stores the maximum benefit solution ending at v of type T_k and having total cost at most d. Then for every $v \in T_k$ we can evaluate $B(v,d)$ as follows:

$$B(v,d) = \max_{\substack{v' \in T_{k-1} \\ d' \leq d}} \left\{ B(v', d - d') + \text{Path}(v', v, d - d') \right\}. \quad (4)$$

In the above equation, $\text{Path}(v', v, x)$ is the benefit achieved by the best path that starts at v' ends at v, has cost at most x and visits the required a number of places in T_k. Computing $\text{Path}(v', v, x)$ is in fact the orienteering problem [30] and therefore known algorithms for this problem [11] can be applied as a subroutine to the above recursion.

For the CoveringTour problem, we can allow for multiple venues per type by creating copies of all the venues of category T_i and treating every copy as a different category. Observe that, for this problem, picking the same venue will not improve the coverage function, while it will increase the cost of the path. Therefore, the algorithm will always prefer to pick new venues.

Time budget and venue delays: In practice, users may have an upper bound on the time that they are willing to allocate to their tour, instead of on the distance that they are willing to cover. In this case, we can transform distance into time by multiplying with the appropriate velocity (e.g., walking speed). The pairwise distance between venues can then be measured in time units without affecting our algorithms. In addition, the time-based formulation allows us to consider the expected duration of a visit to a particular venue. For example, a visit to MoMA (Museum of Modern Art) in New York City could last a lot longer than a visit to a small local gallery. Our algorithms can be easily extended to account for this, by adding the temporal cost associated with each venue v to the existing cost of transitioning from v to any other venue v'.

Multiple transportation types: In practice, the user may have different options from moving from one place to another, e.g., walking, taking a taxi, metro etc. Each such option would incur a different time cost. This can be modeled by considering different types of edges connecting every pair of two venues. Then the dynamic-programming recursions would simply have to iterate over all possible types of edges. This would increase the computational complexity of all the above algorithms by a factor of L — where L is the maximum number of transportation options that can be observed between two nodes.

Table 1: Statistics for the **NYC**, **SF** and **London** datasets.

	NYC	**SF**	**London**
# Check-ins	322 504	91 887	47 916
# Venues	12 728	3 903	2 740

4. EXPERIMENTS

Our experiments with real data from `Foursquare` demonstrate the practical utility of our formulations and the corresponding algorithms.

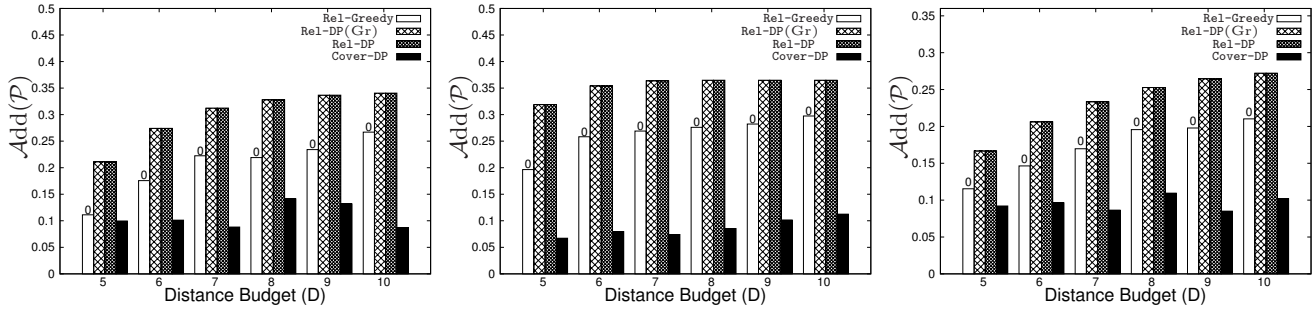
4.1 Experimental setup

The `Foursquare` datasets: We obtain an initial set of venues using the publicly available dataset of Cheng et al. [13], which includes geo-tagged, user-generated content that was pushed to Twitter's public feed from Sept 2010 to Jan 2011. From this dataset, we select tweets related to `Foursquare` check-ins into popular venues (with at least 10 check-ins). Our final dataset consists of 6 699 516 check-ins.

The classification of venues to types in `Foursquare` is hierarchical. In our experiments, we use the upper level of this classification and thus each one of our venues can belong to one of the following 9 types: *Arts & Entertainment*, *College & University*, *Food*, *Professional & Other Places*, *Nightlife Spots*, *Great Outdoors*, *Shop & Service*, *Travel & Transport* and *Residence*. The original dataset did not include the category information for the venues, so we did the mapping of venues to types by crawling `Foursquare.com`.

From this dataset, we generate three smaller datasets, which correspond to venues in three large cities, New York City (**NYC**), San Francisco (**SF**) and London (**London**). Each one of these datasets contains all the check-ins that took place within a 10-mile radius from the corresponding city center. Table 1 shows the statistics of these datasets.

In all three datasets, we consider the venues that belong to the category *Great Outdoors* to be the attractions/activities that one wants to cover in the CoveringTour problem. Hence, we use the rest 8 categories as the (ordinary) venue types for both the AdditiveTour and CoveringTour problems. For our experiments, we compute a universal relevance score $b(v)$ for every venue v. This is computed as the fraction of check-ins for venue v over the total number of check-ins in



(a) **NYC** (b) **SF** (c) **London**

Figure 2: The value of $\mathcal{A}$dd($\mathcal{R}$) for tours $\mathcal{R}$ output by the different algorithms for distance bounds $D \in \{5, 6, 7, 8, 9, 10\}$ (in miles) for all three datasets.

venues of type $T(v)$. Similarly, for venue v, we compute the set of nearby activities, denoted by $S(v)$ as the set of venues of type *Greater Outdoors* that are within 1 mile from v.

Baseline heuristics: We compare our algorithms with a `Greedy` baseline adopted from the operations research literature [29]. This `Greedy` algorithm is iterative: at step t, it picks a venue from the t-th type T_t in the order π. More specifically, among all venues v_t in T_t it identifies the four venues with the highest value of the ratio of the objective function divided by the distance Dist(v_t, w_{t-1}), where w_{t-1} is the venue picked at the $t-1$ step. Then one of these four venues is finally chosen at random with probability proportional to the corresponding value of the ratio. The algorithm proceeds until it reaches the destination or it runs out of distance budget. If the algorithm terminates before reaching the destination, it reports a *failure*. In all our experiments, we repeat the above process for 100 times and we report the solution with the highest value of the objective function. We refer to the `Greedy` algorithm that uses the $\mathcal{A}$dd (resp. $\mathcal{C}$ov) objective as the `Rel-Greedy` (resp. `Cov-Greedy`) algorithm.

4.2 Quality evaluation

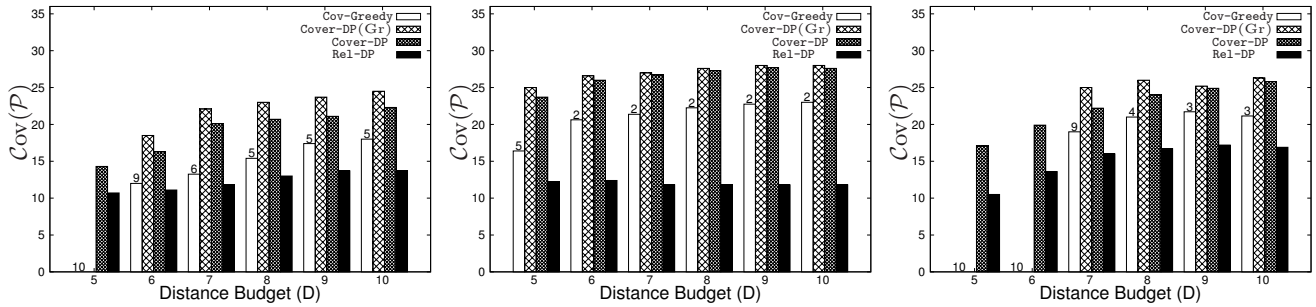
Using the above set up we evaluate the quality of the various algorithms for solving the AdditiveTour and the CoveringTour problems respectively. When evaluating the quality of our algorithms for AdditiveTour, we measure the $\mathcal{A}$dd of the proposed tours, while when we evaluate the algorithms for the CoveringTour problem, we measure the $\mathcal{C}$ov or the output tours. In all cases, higher values of the objective mean better-quality results.

In all the experiments we present we have $L_i = U_i = 1$ for all types. We evaluate our algorithms with regard to the distance budget (D) that a user is willing to walk and the number of venue types (k) of the requested tour. In all cases the trends are similar: the `Rel-DP` and the `Cover-DP` algorithms are clearly the best algorithms for AdditiveTour and CoveringTour respectively. In addition to giving lower-quality solutions, the `Greedy` heuristics often fail to find a solution, even if one exists.

Varying the distance budget D: In this experiment, we study the effect of the distance bound D on the results reported by each algorithm. We generate 10 random type orders. Each type order includes 6 (distinct) types chosen randomly from the set of available ordinary types (i.e., $k = 6$). We evaluate our algorithms for each value of $D \in \{5, 6, 7, 8, 9, 10\}$ (in miles) over all generated orders.

Figures 2 and 3 show the average value achieved by the algorithms for the $\mathcal{A}$dd and $\mathcal{C}$ov objectives, respectively. Observe that all values are averages over the 10 different type orders. Although the dynamic programming algorithms always report a solution if one exists, the greedy approaches may fail to find a solution. In such cases the reported average is taken over the type orders for which the algorithm successfully reported a tour. The number on top of the bar corresponding to a greedy algorithm represents the number of instances (out of 10) for which the algorithm reported a failure. For a fair comparison, we also report the average value of $\mathcal{A}$dd achieved by the dynamic programming algorithms over the instances for which the greedy baselines actually found a solution (the `Rel-DP`(Gr) and `Cover-DP`(Gr) bars for the $\mathcal{A}$dd and $\mathcal{C}$ov objectives, respectively). The plots also include the $\mathcal{A}$dd value achieved by `Cover-DP` and the $\mathcal{C}$ov value achieved by `Rel-DP`. Our goal is to verify that a customized approach is required for each objective function.

As can be seen in Figure 2, `Rel-DP` finds a solutions for all instances and for all values of D. The same is also true for `Rel-Greedy` (and hence, `Rel-DP` and `Rel-DP`(Gr) match). However, with respect to the objective function ($\mathcal{A}$dd), `Rel-DP` consistently outperforms `Rel-Greedy`. Another observation is that the optimal benefit (reported by `Rel-DP`) peeks after the distance bound has been sufficiently relaxed, and stops increasing after that point (which differs among cities). The results for the $\mathcal{C}$ov objective are shown in Figure 3. While the general trends are very similar to the ones we made for $\mathcal{A}$dd, the figure motivates additional observations. First, we observe that `Cov-Greedy` fails to report a solution for a significant number of order types. For example, for the **NYC** dataset, the `Cov-Greedy` failed to find a solution for 5 out of 10 instances for $D = 10$. In fact, `Cov-Greedy` failed to report *any* solutions (e.g., for the $D \in \{5, 6\}$ for **London** or for $D = 5$ for Pnyc). Due to this failures, the `Cover-DP` bar differs from the `Cover-DP`(Gr) bar — recall that the latter is the performance of `Cover-DP` over the instances for which `Cov-Greedy` was able to deliver a solution. Nevertheless, in all cases, both `Cover-DP` and `Cover-DP`(Gr) outperform `Cov-Greedy`. This demonstrates the inability of `Cov-Greedy` to leverage the overlap among the sets of attractions covered by the various venues.



(a) **NYC** (b) **SF** (c) **London**

Figure 3: The value of $\mathcal{C}\text{ov}(\mathcal{R})$ for tours $\mathcal{R}$ found by the different algorithms for distance bounds $D \in \{5, 6, 7, 8, 9, 10\}$ (in miles) for all three datasets.

Finally, we observe that `Cover-DP` and `Rel-DP` failed to achieve high values for the objectives functions that they were not customized for ($\mathcal{A}$dd and $\mathcal{C}$ov respectively). In fact, they were even outperformed by the greedy baselines in those tasks. This verifies our intuition that both problems are non-trivial, and require customized algorithms.

Varying the Tour size k: In this experiment, we evaluate the effect of the size of the tour k (i.e., the number of venue types in the tour) on the performance of the algorithms. We set $D = 6$ miles, and generate 10 random permutations of k (distinct) types, for $k \in \{2, 3, 4, 5, 6, 7, 8\}$, emulating the process that we followed in the previous experiment. The results for the **NYC** dataset for both the $\mathcal{A}$dd and $\mathcal{C}$ov objectives are shown in Figure 4.[3]

The `Rel-DP` and `Cover-DP` algorithms outperform the greedy baselines for the $\mathcal{A}$dd and $\mathcal{C}$ov objectives, respectively. This demonstrates that our algorithms are able to include additional venues and improve their respective objectives, while respecting the distance bound. On the other hand, the values of the greedy approaches may only increase trivially or even decrease, as k becomes larger and more types are added to the tour. In fact, for the $\mathcal{C}$ov objective, `Cov-Greedy` often fails to find a solution, especially for larger values of k.

Binning of distances: As we have already discussed in Section 3, the running time of both the `Rel-DP` and `Cover-DP` depends on the granularity of the bins that we use to break the distance budget D. We expect that finer granularity gives better results in terms of the objective functions, yet is computationally more expensive. Tables 2 and 3 show the values of the objective functions $\mathcal{A}\text{dd}(\mathcal{R})$ and $\mathcal{C}\text{ov}(\mathcal{R})$ as well as the corresponding running times of the `Rel-DP` and the `Cover-DP` algorithms for bins of sizes $b = 0.1$ and $b = 0.01$ miles. For the results reported here we set $D = 10$. From the two tables we observe that bins of size 0.1 of miles achieve the same values of the objectives while they result in 10 times faster algorithms. Observe that the running time of `Cover-DP` is larger than the running time of `Rel-DP`. The reason for that is that the former needs to evaluate a more complex objective — which in every step requires the evaluation of the intersection of two sets: the set of already-covered activities and the activities introduced by any newly-considered venue.

[3]The results for the other two datasets were very similar and omitted due to space constraints as well.

Table 2: Value of $\mathcal{A}\text{dd}(\mathcal{R})$ and running times achieved by `Rel-DP` for bin sizes $b = 0.1$ or 0.01 miles and $D = 10$ miles.

	$\mathcal{A}\text{dd}(\mathcal{R})$		Running time (ms)	
	$b = 0.01$	$b = 0.1$	$b = 0.01$	$b = 0.1$
London	0.27	0.27	976.9	103.7
SF	0.36	0.36	1176.7	125.4
NYC	0.34	0.34	1072.5	139.3

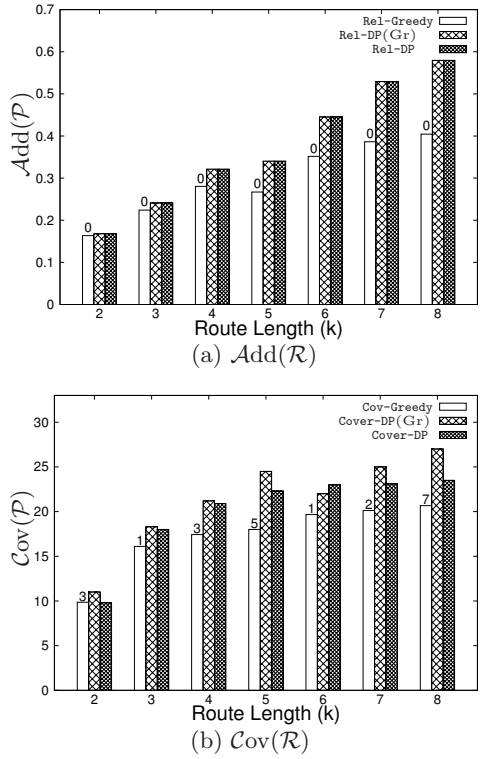
Table 3: Value of $\mathcal{C}\text{ov}(\mathcal{R})$ and running times achieved by `Rel-DP` for bin sizes $b = 0.1$ or 0.01 miles and $D = 10$ miles.

	$\mathcal{C}\text{ov}(\mathcal{R})$		Running time (ms)	
	$b = 0.01$	$b = 0.1$	$b = 0.01$	$b = 0.1$
London	24.7	24.7	4587.6	384.8
SF	26.5	26.5	6596.1	547.6
NYC	21.3	21.3	4612.8	445.5

4.3 Anecdotal evaluation

Here, we use visualization in order to provide some further insight on the tours output by the different algorithms as solutions to the AdditiveTour and CoveringTour problems. Due to lack of space we show here the results for a request of a tour in London — results in other cities convey similar intuition. The query we consider requires a tour of size $k = 4$ (i.e., four venue types) and the input type order is π=$\{1, 3, 6, 5\}$, where types *1,3,6,5* correspond to the `Foursquare` types "*Arts & Entertainment*", "*Food*", "*Shop & Services*" and "*Nightlife Spot*" respectively. For the results we report here we assume that both the starting and the ending point of the tour is London's city center.

Solving the AdditiveTour problem for London with a distance bound D= 6 miles, gives the tours presented on the maps at Figure 5; `Rel-DP` reports the solution depicted on the left, while `Rel-Greedy` reports the solution on the right. The point labeled "0" is the starting and ending point of the tour, while the red point-marks correspond to the venues recommended and are labeled with their type number. We also report the total benefit obtained from the two algorithms (top left corner of each map), and the benefit of each individual venue. While both tours begin with the



(a) $\mathcal{A}$dd($\mathcal{R}$)

(b) $\mathcal{C}$ov($\mathcal{R}$)

Figure 4: The $\mathcal{A}$dd($\mathcal{R}$) and $\mathcal{C}$ov($\mathcal{R}$) for tours $\mathcal{R}$ output by the different algorithms for the **NYC** dataset and varying the tour size $k \in \{2, 3, 4, 5, 6, 7, 8\}$.

same venue (the Tate Museum of Modern Arts),[4] they proceed to include different venues, thus leading to different $\mathcal{A}$dd values. In fact, the value achieved by `Rel-DP` is almost two times greater than that achieved by `Rel-Greedy`.

Solving the CoveringTour problem for London with the same parameters gives the tours presented on the maps at Figure 6. `Cover-DP` reports the solution depicted on the left, while `Cov-Greedy` reports the solution on the right. We now present with a dotted circle centered at each venue, the area that the latter *covers* with respect to locales of type "Greater Outdoors", which as aforementioned are the attractions for our CoveringTour problem. Furthermore, the actual attractions covered are shown on the map as blue mark-points. As one can see, the tours that are provided by the two algorithms are completely different in this case. For instance, `Cover-DP` recommends Alice to start her tour from the British Museum, while `Cov-Greedy` recommends the Tate Museum of Modern Arts. However, again `Cover-DP` clearly outperforms `Cov-Greedy`, since the number of "blue nodes" covered by the former is 20, as compared to the 14 attractions covered by the `Cov-Greedy`.

5. RELATED WORK

The proliferation of location-based systems has given rise to literature on spatial recommendations, which is clearly relevant to our work. Related to our study are also combinatorial problems that deal with finding tours on graphs. We review this work below.

Spatial recommendations: The rapid proliferation of location-based social networks has motivated a volume of work that focuses on methods for personalized location (or venue) recommendations [8, 25, 33] to social-network users. These methods, deploy collaborative-filtering techniques [33], geometric embeddings [8] or they even incorporate features present in the users' social network [25] in order to associate every venue with a score, which encodes the probability of a user liking (or visiting) a particular venue recommendation. Contrary to our own work, all these approaches evaluate each venue independently of the rest. Thus, venues are not considered as groups or as stops in a tour and therefore thematic redundancy or spatiotemporal constraints cannot be taken into consideration. This is the key difference between our work and all these described above.

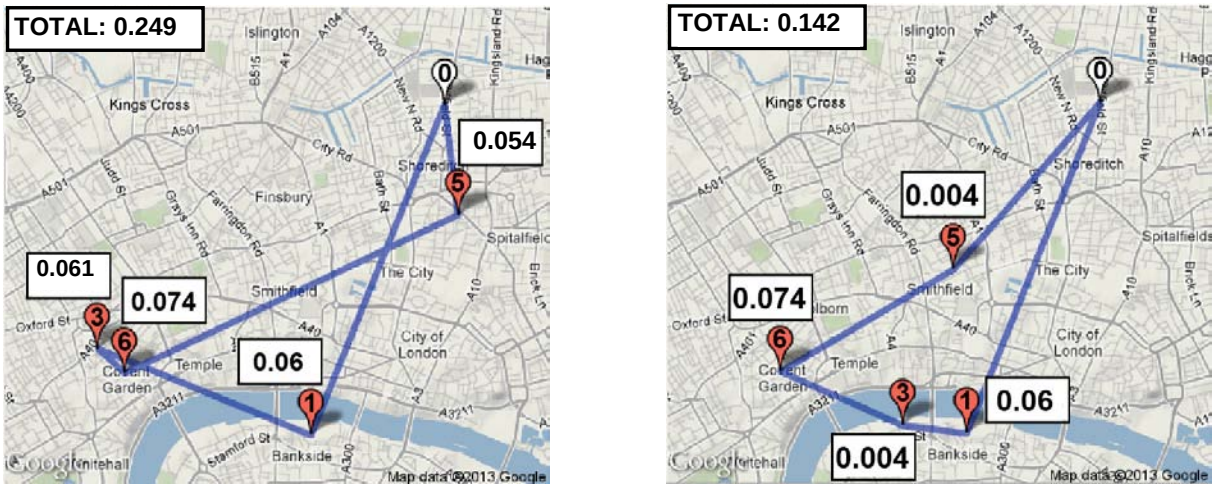
Motivated by the limitation of the above methods, recent work has focused on recommending *tours* of locations. For instance, De Choudhury et al. [14] focus on segmenting streams of spatiotemporally tagged photos into paths, and then assembling these paths into itineraries. Similar works are those by Kurashima et al. [24] and Yoon et al. [34], who utilize spatiotemporal traces (e.g., photo streams, GPS trajectories) to recommend spatiotemporally affordable tours In addition to being applicable only in the presence of training sequences of spatiotemporally tagged photographs (or other similar traces), these approaches cannot handle multiple types of venues that cater to different user needs. This limitation is also shared by the relevant works on interactive tour recommendation [27, 18, 23], which iteratively personalize or improve a tour based on user feedback. Clearly, by not allowing for venues of multiple types, all of these approaches can also not support the user's requirement to satisfy her needs in a particular order.

The support for multiple types of venues is considered by Ardissono et al. [5]. However, the proposed system expects the user to *manually* select a venue from each desired type. A tour traversing the selected venues is then proposed. Finally, in a recent demo paper, Xie et al. [32] describe a single-type system for tour-recommendations, and briefly describe how this could be extended to handle multiple types. The proposed method is not comparable to ours, since it does not allow the user to specify the order in which the user wants to visit the different types. In addition, even though no experimental evaluation is provided, the authors suggest that multiple venues can be combined to form a spatiotemporally efficient tour via a simple greedy approach. However, our experiments on real data demonstrate the inadequacies of the greedy approach in practical scenarios.

Further, approaches that focus on reconstructing and recommending routes based on existing location trajectories (e.g., [31, 12]) are complementary to our work; the output of our methods can be used as input to such approaches.

Tours in operations research: At a high level, there is a connection between our problems and the orienteering problem (OP) [30] studied in operations-research and theoretical computer science. In OP the input consists of a set of locations with an associated benefit for each location, and a cost for the traversal of an edge between two locations. The goal is to visit a set of locations such that the benefit is maximized, while the total distance covered is within a bound D_{max}. The first difference between our problem and

[4]We defer from mentioning commercial venues for the rest of the types.



(a) `Cover-DP` (b) `Cov-Greedy`

Figure 5: Anecdotal results for the **London** dataset when tours are obtained as solutions to the AdditiveTour problem; distance budget $D = 6$ miles, $\pi = \{$*Arts & Entertainment (1), Food (3), Shop & Services (6), Nightlife Spot (5)*$\}$.

the OP is that our locations (which we call venues) are associated with types, and only a certain number of each type can be included in the tour. In addition, the user-specified ordering constraints over these types are also absent from OP. Thus, the OP-specific heuristics [11, 21, 28] are not directly applicable to our problem.

The connection between our problems and the Traveling Salesman Problem (TSP) [4] and its prize-collecting versions [7, 9] is equally abstract. The TSP problem considers a set of cities and distances between them and asks for the shortest possible tour that visits each city exactly ones and returns to the original city. The prize-collecting version of the same problem assumes that there is a benefit associated with visiting every city and the goal is to maximize the total benefit by visiting cities, divided by the total distance covered by the tour. The consideration of multiple venue types, the ordering constraints and the freedom to not visit all the venues (but only some from every type) make our problems distinct from all the above formulations. Moreover, the CoveringTour version of our problem is very distinct from any prize-collecting problem that has appeared in relevant literature, since the benefit induced by every venue does not depend solely on the venue itself, but also on the rest of the venues included in the tour. In short, the existing algorithms for the different TSP versions are not applicable for solving the problems that we study in our work.

Travel package recommendations: Tangential to the spatial recommendation problems, is that of travel package recommendations, where the goal is to recommend vacation packages (i.e., destination, hotels and organized activities). Although these studies (e.g., [20]) also fall into the general realm of - group - recommendation systems, they are distinct from our work. First, their focus is on creating combinatorial packages that satisfy different constraints, rather than on the concept of creating ordered routes which is essential to our work. We consider such systems to be complementary to ours: once a user is in a given urban area, possibly picked by one of the package-recommendation approaches discussed above, our work provides a system that is able to provide a customized tour based on her preferences.

6. CONCLUSIONS

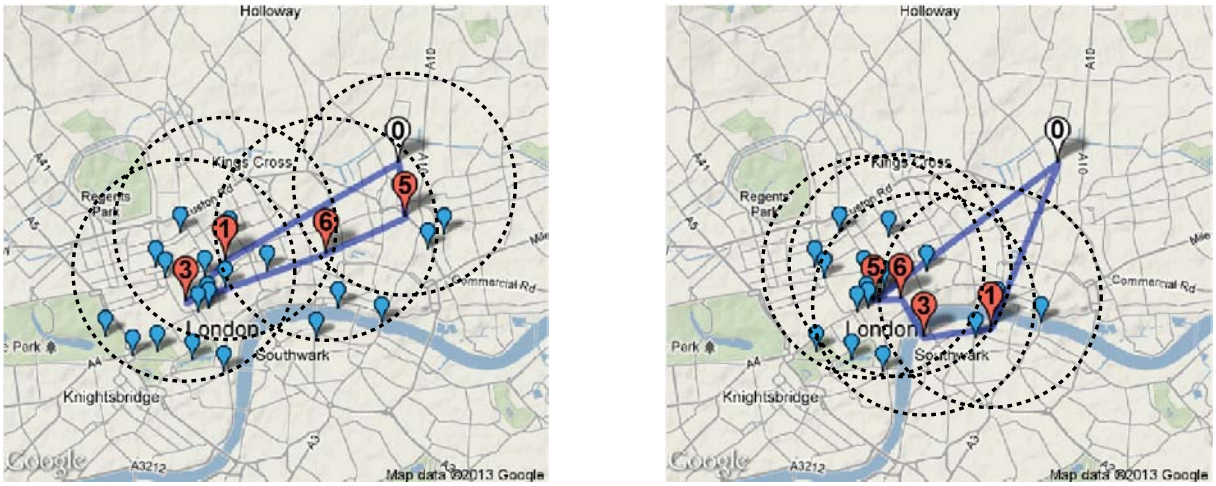
We have presented two alternative instantiations of a framework for generating customized tour recommendations as a paradigm of an intelligent urban navigation service. The first one, AdditiveTour, assigns a benefit to each location and retrieves the sequence of venues with the maximum total benefit, constrained by a given distance budget and a preferred visitation order for the types of venues in the tour. The second formulation, CoveringTour, assigns to each venue a set of associated attractions *covered* by it (i.e., are within range). The goal is to maximize the total number of attractions covered, under the same set of constraints. We study the complexity of these problems and provide efficient algorithmic solutions. Furthermore, we evaluate our methods on a large dataset from `Foursquare`, the largest location-based social network to date. The results demonstrate the practical utility of the proposed formulations and the efficacy of the proposed algorithms.

Acknowledgments

This research was partially funded by NSF grants III 1218437 and CAREER 1253393 and gifts from Microsoft and Google.

7. REFERENCES

[1] Foursquare as a recommendation engine: https://foursquare.com/about/.
[2] United nations-world urbanization prospects: The 2011 revision – highlights: http://esa.un.org/unup.
[3] A. M. W. Alan A. Lew, C. Michael Hall. *A Companion to Tourism*. Wiley, 2004.
[4] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. *The Traveling Salesman Problem: A Computational Study*. 2006.
[5] L. Ardissono, A. Goy, G. Petrone, M. Segnan, and P. Torasso. Intrigue: Personalized recommendation of tourist attractions for desktop and hand held devices. *Applied Artificial Intelligence*, 17(8-9), 2003.



(a) `Cover-DP` (b) `Cov-Greedy`

Figure 6: Anecdotal results for the **London** dataset when tours are obtained as solutions to the CoveringTour problem; distance budget $D = 6$ miles, $\pi = \{$*Arts & Entertainment (1), Food (3), Shop & Services (6), Nightlife Spot (5)*$\}$.

[6] J. Bao, Y. Zheng, and M. Mokbel. Location-based and preference-aware recommendation using sparse geo-social networking data. In *ACM SIGSPATIAL GIS*, 2012.

[7] R. Bar-Yehuda, G. Even, and S. Shahar. On approximating a geometric prize-collecting traveling salesman problem with time windows. *Journal of Algorithms*, 55(1), 2005.

[8] B. Berjani and T. Strufe. A recommendation system for spots in location-based online social networks. In *SNS*, 2011.

[9] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. P. Williamson. A note on the prize collecting traveling salesman problem. *Math. Program.*, 59, 1993.

[10] C. Chekuri and A. Kumar. Maximum coverage problem with group budget constraints and applications. In *APPROX-RANDOM*, 2004.

[11] C. Chekuri and M. Pál. A recursive greedy algorithm for walks in directed graphs. *IEEE FOCS*, 2005.

[12] Z. Chen, H. Chen, X. Zhou, Y. Zheng, and X. Xie. Searching trajectories by locations: An efficiency study. In *ACM SIGMOD*, 2010.

[13] Z. Cheng, J. Caverlee, K. Lee, and D. Sui. Exploring millions of footprints in location sharing services. In *ICWSM*, 2011.

[14] M. De Choudhury, M. Feldman, S. Amer-Yahia, N. Golbandi, R. Lempel, and C. Yu. Automatic construction of travel itineraries using social breadcrumbs. In *HT*, 2010.

[15] B. Dellaert, A. Borgers, and H. Timmermans. A day in the city: Using conjoint choice experiments to model urban tourists' choice of activity packages. *Tourism Management*, 16(5):347 – 353, 1995.

[16] P. Doherty. Smart cities: How to build sustainable and resilient environments in an increasingly urbanized world. In *McGraw Hill Financial Global Institute White Paper*, 2013.

[17] D.Scott. Exploring time patterns in people's use of a metropolitan park district. In *Leisure Sciences*, 1997.

[18] S. Dunstall, M. Horn, P. Kilby, M. Krishnamoorthy, B. Owens, D. Sier, and S. Thièbaux. An automated itinerary planning system for holiday travel. In *Journal of IT and Tourism*, 2003.

[19] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[20] Y. Ge, Q. Liu, H. Xiong, A. Tuzhilin, and J. Chen. Cost-aware travel tour recommendation. In *KDD*, 2011.

[21] B. Golden, L. Levy, and R. Vohra. The orienteering problem. In *Naval Research Logistics*, 1987.

[22] N. Grinberg, M. Naaman, B. Shaw, and G. Lotan. Extracting diurnal patterns of real world activity from social media. In *AAAI ICWSM*, 2013.

[23] K. Kim, H. Kim, and J. Ryu. Triptip: a trip planning service with tag-based recommendation. In *CHI (Extended abstract)*, 2009.

[24] T. Kurashima, T. Iwata, G. Irie, and K. Fujimura. Travel route recommendation using geotags in photo sharing sites. In *CIKM*, 2010.

[25] A. Noulas, S. Scellato, N. Lathia, and C. Mascolo. A random walk around the city: New venue recommendation in location-based social networks. In *SocialCom*, 2012.

[26] D. G. Pearce. Tourist time-budget. *Annals of Tourism Research*, 15(1):106 – 121, 1988.

[27] S. Roy, G. Das, S. Amer-Yahia, and C. Yu. Interactive itinerary planning. In *ICDE*, 2011.

[28] W. Souffriau, P. Vansteenwegen, J. Vertommen, G. V. Berge, and D. V. Oudheusden. A personalized tourist trip design algorithm for mobile tourist guides. In *Applied Artificial Intelligence*, 2008.

[29] T. Tsiligirides. Heuristic methods applied to orienteering. In *The Journal of Operational Research Society*, 1984.

[30] P. Vansteenwegen, W. Souffriau, and D. V. Oudheusden. The orienteering problem: A survey. In *European Journal of Operational Research*, 2011.

[31] L.-Y. Wei, Y. Zheng, and W.-C. Peng. Constructing popular routes from uncertain trajectories. In *ACM KDD*, 2012.

[32] M. Xie, L. Lakshmanan, and P. Wood. Comprec-trip: a composite recommendation system for travel planning. In *ICDE (demo)*, 2011.

[33] M. Ye, Y. Peifeng, and W.-C. Lee. Location recommendation for location-based social networks. In *SIGSPATIAL GIS*, 2010.

[34] H. Yoon, Y. Zheng, X. Xie, and W. Woo. Social itinerary recommendation from user-generated digital trails. In *Personal and Ubiquitus Computing*, 2012.