

# An Energy-Efficient Mobile Recommender System

Yong Ge<sup>1</sup>, Hui Xiong<sup>1</sup>, Alexander Tuzhilin<sup>2</sup>, Keli Xiao<sup>1</sup>, Marco Gruteser<sup>3</sup>

<sup>1</sup> Rutgers Business School, Rutgers University  
hxiong@rutgers.edu, {yongge, keli}@pegasus.rutgers.edu

<sup>2</sup> Leonard N. Stern School of Business, NYU, atuzhili@stern.nyu.edu

<sup>3</sup> Elect & Comp Engineering, Rutgers University, gruteser@winlab.rutgers.edu

## ABSTRACT

The increasing availability of large-scale location traces creates unprecedented opportunities to change the paradigm for knowledge discovery in transportation systems. A particularly promising area is to extract energy-efficient transportation patterns (green knowledge), which can be used as guidance for reducing inefficiencies in energy consumption of transportation sectors. However, extracting green knowledge from location traces is not a trivial task. Conventional data analysis tools are usually not customized for handling the massive quantity, complex, dynamic, and distributed nature of location traces. To that end, in this paper, we provide a focused study of extracting energy-efficient transportation patterns from location traces. Specifically, we have the initial focus on a sequence of mobile recommendations. As a case study, we develop a mobile recommender system which has the ability in recommending a sequence of pick-up points for taxi drivers or a sequence of potential parking positions. The goal of this mobile recommendation system is to maximize the probability of business success. Along this line, we provide a Potential Travel Distance (PTD) function for evaluating each candidate sequence. This PTD function possesses a monotone property which can be used to effectively prune the search space. Based on this PTD function, we develop two algorithms, *LCP* and *SkyRoute*, for finding the recommended routes. Finally, experimental results show that the proposed system can provide effective mobile sequential recommendation and the knowledge extracted from location traces can be used for coaching drivers and leading to the efficient use of energy.

## 1. INTRODUCTION

Advances in sensor, wireless communication, and information infrastructures such as GPS, WiFi and RFID have enabled us to collect large amounts of location traces (trajectory data) of individuals or objects. Such a large number of trajectories provide us unprecedented opportunity to automatically discover useful knowledge, which in turn deliver

intelligence for real-time decision making in various fields, such as mobile recommendations. Indeed, a mobile recommender system promises to provide mobile users access to personalized recommendations anytime, anywhere. To this end, an important task is to understand the unique features that distinguish pervasive personalized recommendation systems from classic recommender systems.

Recommender systems [3] address the information overloaded problem by identifying user interests and providing personalized suggestions. In general, there are three ways to develop recommender systems. The first one is content-based [15]. It suggests items which are similar to those a given user has liked in the past. The second way is based on collaborative filtering. In other words, recommendations are made according to the tastes of other users that are similar to the target user. Finally, a third way is to combine the above and have a hybrid solution [16]. However, the development of personalized recommender systems in mobile and pervasive environments is much more challenging than developing recommender systems from traditional domains due to the complexity of spatial data and intrinsic spatio-temporal relationships, the unclear roles of context-aware information, and the increasing availability of environment sensing capabilities.

Recommender systems in the mobile environments have been studied before [2, 5, 6, 7, 14, 20, 21]. For instance, the work in [2, 6] targets the development of mobile tourist guides. Also, Heijden et al. have discussed some technological opportunities associated with mobile recommendation systems [21]. In addition, Averjanova et al. have developed a map-based mobile recommender system that can provide users with some personalized recommendations [5]. However, this prior work is mostly based on user ratings and is only exploratory in nature, and the problem of leveraging unique features distinguishing mobile recommender systems remains pretty much open.

In this paper, we exploit the knowledge extracted from location traces and develop a mobile recommender system based on business success metrics instead of predictive performance measures based on user ratings. Indeed, the key idea is to leverage the business knowledge from the historical data of successful taxi drivers for helping other taxi drivers improve their business performance. Along this line, we provide a pilot feasibility study of extracting business-success knowledge from location traces by taxi drivers and exploiting this business information for guiding taxis' driving routes. Specifically, we first extract a group of successful taxi drivers based on their past performances in terms of revenue per en-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

KDD'10, July 25-28, 2010, Washington, DC, US.

Copyright 2010 ACM X-XXXXXX-XX-X/XX/XX ...\$5.00.

ergy use. Then, we can cluster the pick-up points of these taxi drivers for a certain time period. The centroids of these clusters can be used as the recommended pick-up points with a certain probability of success for new taxi drivers in these areas. This problem can be formally defined as a mobile sequential recommendation problem, which recommends sequential pick-up points for a taxi driver to maximize his/her business success. Essentially, a key challenge of this problem is that the computational cost can be dramatically increased as the number of pick-up points increases, since this is a combinatorial problem in nature.

To that end, we provide a Potential Travel Distance (PTD) function for evaluating each candidate route. This PTD function possesses a monotone property which can be used to effectively prune the search space and generate a small set of candidate routes. Indeed, we have developed a route recommendation algorithm, named *LCP*, which exploits the monotone property of the PTD function. In addition, we observe that many candidate routes can be dominated by skyline routes [18], and thus can be pruned by skyline computing. However, traditional skyline computing algorithms are not efficient for querying skyline of all candidate routes because it leads to an expensive network traversal process. Thus, we propose a *SkyRoute* algorithm to compute the skyline for candidate routes. An advantage of searching optimal drive route through skyline computing is that it will save the total online processing time when we try to provide different optimal drive routes defined by different business needs.

Finally, the extensive experiments on real-world location traces of 500 taxi drivers show that both *LCP* and *SkyRoute* algorithms outperform the brute-force method with a significant margin. Also, *SkyRoute* has a much better performance than traditional skyline computing methods [18]. Moreover, we show that, if there is an online demand for different evaluation criteria, *SkyRoute* results in better performances than *LCP*. However, if there is only one evaluation criterion, the performance of *LCP* is the best.

## 2. PROBLEM FORMULATION

In this section, we formulate the problem of mobile sequential recommendation (MSR).

### 2.1 A General Problem Formulation

Consider a scenario that a large number of GPS traces of taxi drivers have been collected for a period of time. In this collection of location traces, we also have the information when a cab is empty or occupied. In this data set, it is possible to first identify a group of taxi drivers who are very successful in business. Then, we can cluster the pick-up points of these taxi drivers for a certain time period. The centroids of these clusters can be used as the recommended pick-up points with a certain probability of success for new taxi drivers in these areas. Then, a mobile sequential recommendation problem can be formulated as follows.

Assume that a set of  $N$  potential pick-up points,  $\mathcal{C} = \{C_1, C_2, \dots, C_N\}$ , is available. Also, the estimated probability that a pick-up event could happen at each pick-up point is known as  $P(C_i)$ , where  $P(C_i) (i = 1, \dots, N)$  is assumed to be independently distributed. Let  $\mathcal{P} = \{P(C_1), P(C_2), \dots, P(C_N)\}$  denote the probability set. In addition, let  $\vec{\mathcal{R}} = \{\vec{R}_1, \vec{R}_2, \dots, \vec{R}_M\}$  be the set of all the directed sequences (potential driving routes) generated from  $\mathcal{C}$  and

$|\vec{\mathcal{R}}| = M$  is the size of  $\vec{\mathcal{R}}$  - the number of all possible driving routes. Note that the pick-up points in each directed sequence are assumed to be different from each other. Next, let  $L_{\vec{R}_i}$  be the length of route  $\vec{R}_i (1 \leq i \leq M)$ , where  $1 \leq L_{\vec{R}_i} \leq N$ . Finally, for a directed sequence  $\vec{R}_i$ , Let  $\mathcal{P}_{\vec{R}_i}$  be the route probability set which are the probabilities of all pick-up points containing in  $\vec{R}_i$ , where  $\mathcal{P}_{\vec{R}_i}$  is a subset of  $\mathcal{P}$ .

The objective of this MSR problem is to recommend a travel route for a cab driver in a way such that the potential travel distance before having customer is minimized. Let  $\mathcal{F}$  be the function for computing the Potential Travel Distance (PTD) before having a customer. The PTD can be denoted as  $\mathcal{F}(PoCab, \vec{R}, \mathcal{P})$ . In other words, the computation of PTD depends on the current position of a cab (PoCab), a suggested sequential pick-up points ( $\vec{R}$ ), and the corresponding probabilities associated with all recommended pick-up points.

Based on the above definitions and notations, we can formally define the problem as:

#### The MSR Problem

**Given:** A set of potential pick-up points  $\mathcal{C}$  with  $|\mathcal{C}| = N$ , a probability set  $\mathcal{P} = \{P(C_1), P(C_2), \dots, P(C_N)\}$ , a directed sequence set  $\vec{\mathcal{R}}$  with  $|\vec{\mathcal{R}}| = M$  and the current position (*PoCab*) of a cab driver, who needs the service.

**Objective:** Recommending an optimal driving route  $\vec{\mathbb{R}}$  ( $\vec{\mathbb{R}} \in \vec{\mathcal{R}}$ ). The goal is to minimize the PTD:

$$\min_{\vec{R}_i \in \vec{\mathcal{R}}} \mathcal{F}(PoCab, \vec{R}_i, \mathcal{P}_{\vec{R}_i}) \quad (1)$$

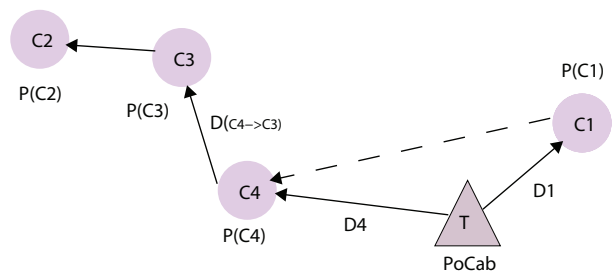


Figure 1: An Illustration Example.

The MSR problem involves the recommendation of a sequence of pick-up points and has combinatorial complexity in nature. However, this problem is practically important and interesting, since it helps to improve the business performances of taxi companies, the efficient use of energy, the productivity of taxi drivers, and the user experiences.

The MSR problem is different from traditional Traveling Salesman Problem (TSP) [4], which finds a shortest path that visits each given location exactly once. The reason is that TSP evaluates a combination of exact  $N$  given locations. In other words, all  $N$  locations have to be involved. In contrast, the proposed MSR problem is to find a subset locations of given  $N$  locations for recommendation. Also, the MSR problem is different from the traditional scheduling problem [8, 17], which selects a set of duties for vehicle drivers. The reason is that all these duties are determined in advance, such as delivering the packages to determined loca-

tions, while the MSR problem consists of uncertain pick-up jobs among several locations. Figure 1 shows an illustration example. In the figure, for a cab T, the closest pick-up point is C1. However, we cannot simply recommend C1 as the first stop in the recommended sequence even if the probability of having a customer at C1 is greater than C4 which is the second closest to T. The reason is that there is still probability that this cab drive cannot find a customer at C1 and then it will cost much more to go to a next pick-up point. Instead, if T goes to C4 first, T might be able to exploit a sequence of pick-up opportunities.

For the MSR problem, there are two major challenges. First, how to find reliable pick-up points from the historical data and how to estimate the successful probability at each pick-up point? Second, there is a computational challenge to search an optimal route.

## 2.2 Analysis of Computational Complexity

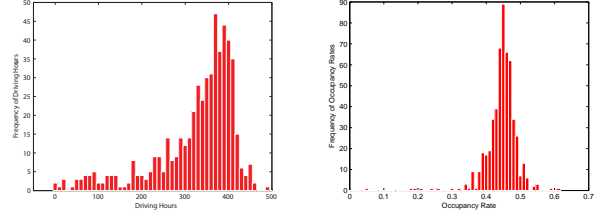
Here, we analyze the computational complexity of the MSR problem. A brute-force method for searching the optimal recommended route has to check all possible sequences in  $\vec{\mathcal{R}}$ . If we assume the cost for computing the function  $\mathcal{F}$  once is 1 ( $Cox(\mathcal{F}) = 1$ ), the complexity of searching a given set  $\mathcal{C}$  with  $N$  pick-up points is as follows.

LEMMA 1. *Given a set of pick-up points  $\mathcal{C}$ , where  $|\mathcal{C}| = N$ ,  $1 \leq L_{\vec{R}_i} \leq N$  and  $Cox(\mathcal{F}) = 1$ , the complexity of searching an optimal directed sequence from  $\vec{\mathcal{R}}$  is  $\mathcal{O}(N!)$*

PROOF. The complexity of searching an optimal sequence is equal to the total number  $M$  of all possible sequences generated from  $\mathcal{C}$ . Since every directed sequence is actually a permutation of pick-up points which form the subset of  $\mathcal{C}$ , we decompose the checking process into two steps: enumeration of non-empty subset  $B$  from  $\mathcal{C}$  and the permutation of pick-up points belonging to the subset  $B$ . For a subset  $B$  with  $i$  different pick-up points, there are totally  $\binom{N}{i}$  different subsets. And the range of integer  $i$  is  $1 \leq i \leq N$ . For each subset  $B$  of  $i$  different element, there are totally  $i!$  different permutations. Thus the total number of all possible directed sequences generated from  $\mathcal{C}$  is  $M = \sum_{i=1}^N \binom{N}{i} \cdot i! < N!(1 + 1 + 1/2) = \frac{5}{2} \cdot N!$ . Thus, we can have  $2 \cdot N! < M < \frac{5}{2} \cdot N!$ . Therefore, the complexity of search optimal directed sequence is  $\mathcal{O}(N!)$ .  $\square$

## 2.3 The MSR Problem with Constraints

As illustrated above, it is computationally prohibited to search for the optimal solution of the general MSR problem. Therefore, from a practical perspective, we consider a simplified version of the MSR problem. Specifically, we put a constraint on the length of a recommended route  $L_{\vec{R}_i}$ . In other words, the length of a recommended route is set to be a constant; that is,  $L_{\vec{R}_i} = \mathcal{L}$ . To simplify the discussion, let  $\vec{R}_i^{\mathcal{L}}$  denote the recommended route with a length of  $\mathcal{L}$ . Based on this constraint, we can simplify the original objective function of the MSR problem as follows.



(a) Driving Hours

(b) Occupancy Rates

Figure 2: Some Statistics of the Cab Data.

### The MSR Problem with a Length Constraint

**Objective:** Recommending an optimal sequence  $\vec{R}^{\mathcal{L}} (\vec{R}^{\mathcal{L}} \in \vec{\mathcal{R}})$ . The goal is to minimize the PTD:

$$\min_{\vec{R}_i^{\mathcal{L}} \in \vec{\mathcal{R}}} \mathcal{F}(PoCab, \vec{R}_i^{\mathcal{L}}, \mathcal{P}_{\vec{R}_i^{\mathcal{L}}})$$

The computational complexity of this simplified MSR problem is analyzed as follows.

LEMMA 2. *Given  $|\mathcal{C}| = N, L_{\vec{R}_i} = \mathcal{L}$  and  $Cox(\mathcal{F}) = 1$ , the computational complexity of searching an optimal directed sequence with a length of  $\mathcal{L}$  from  $\vec{\mathcal{R}}$  is  $\mathcal{O}(N^{\mathcal{L}})$*

PROOF. Since the length of the recommended route has been fixed, the computational complexity can actually be obtained through modifying equation in PROOF of Lemma 1 as  $M = \binom{N}{\mathcal{L}} \cdot \mathcal{L}!$ , where  $M$  is the number of all the sequences with a length as  $\mathcal{L}$ .  $M$  can be transformed as  $N(N-1) \cdots (N-\mathcal{L}+1)$ . Thus, the computational complexity of this problem is  $\mathcal{O}(N^{\mathcal{L}})$ .  $\square$

The above shows that the computational cost of this simplified MSR problem will dramatically increase as the number of pick-up points  $N$  increases. In this paper, we focus on studying the MSR problem with a length constraint.

## 3. RECOMMENDING POINT GENERATION

In this section, we show how to generate the recommending points and compute the probability of pick-up events at each recommending point from location traces of cab drivers.

### 3.1 High-Performance Drivers

In real world, there are always high-performance experienced cab drivers, who typically have sufficient driving hours and higher customer occupancy rates - the percentage of driving time with customers. For example, Figure 2 (a) and (b) show the distributions of driving hours and occupancy rates of more than 500 drivers in San Francisco over a period of about 30 days. In the figure, we can clearly see that the drivers have different performances in terms of occupancy rates. Based on this observation, we will first extract a group of high-performance drivers with sufficient driving hours and high occupancy rates. The past pick-up records of these selected drivers will be used for the generation of potential pick-up points for recommendation.

### 3.2 Clustering Based on Driving Distance

After carefully observing historical pick-up points of high-performance drivers, we notice that there are relative more

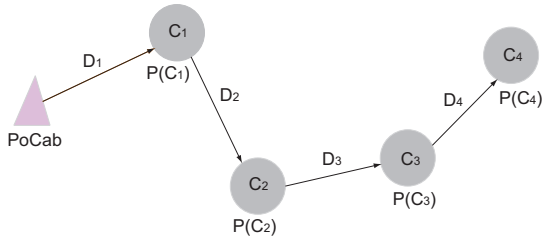


Figure 3: A Recommended Driving Route.

pick-up events in some places than others. In other words, there are the cluster effect of historical pick-up points. Therefore, we propose to cluster historical pick-up points of high-performance drivers into  $N$  clusters. The centroids of these clusters will be used for recommending pick-up points. For this clustering algorithm, we use driving distance rather than Euclidean distance as the distance measure. In this study, we perform clustering based on driving distance during different time periods in order to have recommending pick-up pointers for different time periods. Another benefit of clustering historical pick-up points is to dramatically reduce the computational cost of the MRS problem.

### 3.3 Probability Calculation

For each recommended pick-up point (the centroid of historical pick-up cluster), the probability of a pick-up event can be computed based on historical pick-up data. The idea is to measure how frequent pick-up events can happen when cabs travel across each pick-up cluster. Specifically, we first obtain the spatial coverage of each cluster. Then, let  $\#_T$  denote the number of cabs which have no customer before passing a cluster. For these  $\#_T$  empty cabs, the number of pick-up events  $\#_P$  is counted in this cluster. Finally, the probability of pick-up event for each cluster (each recommended pick-up point) can be estimated as follows.

$$P(C_i)_{1 \leq i \leq N} = \frac{\#_P}{\#_T}, \quad (2)$$

where  $\#_P$  and  $\#_T$  are recorded for each historical pick-up cluster at different time periods.

## 4. SEQUENTIAL RECOMMENDATION

In this section, we design mobile sequential algorithms for searching the optimal route for recommendation.

### 4.1 The Potential Travel Distance Function

First, we introduce the Potential Travel Distance (PTD) function, which will be exploited for algorithm design. To simplify the discussion, we illustrate the PTD function via an example. Specifically, Figure 3 shows a recommended driving route  $PoCab \rightarrow C_1 \rightarrow C_2 \rightarrow C_3 \rightarrow C_4$  for the cab  $PoCab$ , where the length of suggested driving route  $\mathcal{L} = 4$ .

When a cab driver follows this route  $\vec{R}^{\mathcal{L}}$ , he/she may pick up customers at each pick-up point with a probability  $P(C_i)$ . For example, a pick-up event may happen at  $C_1$  with the probability  $P(C_1)$ , or at  $C_2$  with the probability  $\overline{P(C_1)}P(C_2)$ , where  $\overline{P(C_i)} = 1 - P(C_i)$  is the probability that a pick-up event does not happen at  $C_i$ . Therefore, the travel distance before a pick-up event is discretely distributed. In addition, it is possible that there is no pick-up

event happening after going through the suggested route. This probability is  $\overline{P(C_1)} \cdot \overline{P(C_2)} \cdot \overline{P(C_3)} \cdot \overline{P(C_4)}$ . In this paper, since we only consider the driving routes with a fixed length, the travel distance beyond the last pick-up point is set to be  $D_\infty$  equally for all suggested driving routes. Formally, we represent the distribution of the travel distance before next pick-up event with two vectors:  $\mathcal{D}_{\vec{R}^{\mathcal{L}}} = \langle D_1, (D_1 + D_2), (D_1 + D_2 + D_3), (D_1 + D_2 + D_3 + D_4), D_\infty \rangle$  and  $\mathcal{P}_{\vec{R}^{\mathcal{L}}} = \langle P_1, \overline{P(C_1)} \cdot P(C_2), \overline{P(C_1)} \cdot \overline{P(C_2)} \cdot P(C_3), \overline{P(C_1)} \cdot \overline{P(C_2)} \cdot \overline{P(C_3)} \cdot P(C_4), \overline{P(C_1)} \cdot \overline{P(C_2)} \cdot \overline{P(C_3)} \cdot \overline{P(C_4)} \rangle$ . Finally, the Potential Travel Distance (PTD) function  $\mathcal{F}$  is defined as the mean of this distribution as follows.

$$\mathcal{F} = \mathcal{D}_{\vec{R}^{\mathcal{L}}} \cdot \mathcal{P}_{\vec{R}^{\mathcal{L}}} \quad (3)$$

where  $\cdot$  is the dot product of two vectors.

From the definition of the PTD function, we know that the evaluation of a suggested drive route is only determined by the probability of each pick-up point and the travel distance along the suggested route, except the common  $D_\infty$ . These two types of information associated with each drive route  $\vec{R}_i^{\mathcal{L}}$  can be represented with one  $2\mathcal{L}$ -dimensional vector  $\mathcal{DP} = \langle DP_1, \dots, DP_l, \dots, DP_{2\mathcal{L}} \rangle$ . Let us consider the example in Figure 3, where  $\mathcal{L} = 4$ . The 8-dimensional vector  $\mathcal{DP}$  for this specific driving route is  $\mathcal{DP} = \langle D_1, \overline{P(C_1)}, D_2, \overline{P(C_2)}, D_3, \overline{P(C_3)}, D_4, \overline{P(C_4)} \rangle$ .

However, to find the optimal suggested route, if we use a brute-force method, we need to compute the PTD for all directed sequences with a length  $\mathcal{L}$ . This involves a lot of computation. Indeed, many suggested routes can be removed without computing the PTD function, because all pick-up points along these routes are far away from the target cab. Along this line, we identify a monotone property of the Function  $\mathcal{F}$  as follows.

**LEMMA 3. The Monotone Property of the PTD Function  $\mathcal{F}$ .** *The PTD Function  $\mathcal{F}(\mathcal{DP})$  is strictly monotonically increasing with each attribute of vector  $\mathcal{DP}$ , which is a  $2\mathcal{L}$ -dimensional vector.*

**PROOF.** A proof sketch is as follows. By the definition of the function  $\mathcal{F}$  in Equation 3, we can first derive the polynomial form of  $\mathcal{F}$ . From the polynomial form of  $\mathcal{F}$ , we can observe that the degree of each variable is one. Also,  $D_\infty$  is assumed to be one big enough constant. To prove the monotonicity of  $\mathcal{F}$ , it is equally to prove that the coefficient of each variable is positive. This is easy to show. The proof details are omitted due to the space limit.  $\square$

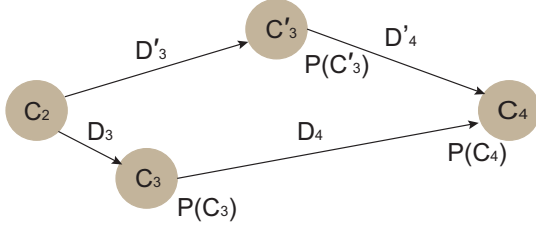
### 4.2 The $\mathcal{LCP}$ Algorithm

In this subsection, we introduce the  $\mathcal{LCP}$  algorithm for finding an optimal driving route. In  $\mathcal{LCP}$ , we exploit the monotone property of the PTD function and two other pruning strategies, *Route Dominance* and *Constrained Subroute Dominance*, for pruning the search space.

**DEFINITION 1. Route Dominance.** *A recommended driving route  $\vec{R}^{\mathcal{L}}$ , associated with the vector  $\mathcal{DP}$ , dominates another route  $\vec{\widetilde{R}}^{\mathcal{L}}$ , associated with the vector  $\widetilde{\mathcal{DP}}$ , iff  $\exists 1 \leq l \leq 2\mathcal{L}, DP_l < \widetilde{DP}_l$  and  $\forall 1 \leq l \leq 2\mathcal{L}, DP_l \leq \widetilde{DP}_l$ . This can be denoted as  $\vec{R}^{\mathcal{L}} \Vdash \vec{\widetilde{R}}^{\mathcal{L}}$ .*

By this definition, if a candidate route  $A$  is dominated by a candidate route  $B$ ,  $A$  cannot be an optimal route. Next, we provide a definition of constraint sub-route dominance.

**DEFINITION 2. Constrained Sub-route Dominance.** Consider that two sub-routes  $\vec{R}_{sub}$  and  $\vec{R}'_{sub}$  with an equal length (the number of pick-up points) and the same source and destination points. If the associated vector of  $\vec{R}_{sub}$  dominates the associated vector of  $\vec{R}'_{sub}$ , then  $\vec{R}_{sub}$  dominates  $\vec{R}'_{sub}$ , i.e.  $\vec{R}_{sub} \Vdash \vec{R}'_{sub}$ .



**Figure 4: Illustration: the Sub-route Dominance.**

For example, as shown in Figure 4,  $\vec{R}_{sub}$  is  $C_2 \rightarrow C_3 \rightarrow C_4$  and  $\vec{R}'_{sub}$  is  $C_2 \rightarrow C'_3 \rightarrow C_4$ . The associated vectors of  $\vec{R}_{sub}$  and  $\vec{R}'_{sub}$  are  $\mathcal{DP}_{sub} = \langle D_3, \overline{P(C_3)}, D_4, \overline{P(C_4)} \rangle$  and  $\mathcal{DP}'_{sub} = \langle D'_3, \overline{P(C'_3)}, D'_4, \overline{P(C_4)} \rangle$  respectively. Then the dominance of  $\vec{R}_{sub}$  over  $\vec{R}'_{sub}$  is determined by the dominance of these two vectors. Here, we have the constraints that two routes have the same length as well as the same source and destination. The constrained sub-route dominance enables us to prune the search space in advance. This is shown in the following proposition.

**PROPOSITION 1. LCP Pruning.** For two sub-routes  $A$  and  $B$  with a length  $\mathcal{L}$ , which includes only pick-up points, if sub-route  $A$  is dominated by sub-route  $B$  under Definition 2, the candidate routes with a length  $\mathcal{L}$  which contain sub-route  $A$  will be dominated and can be pruned in advance.

Let us study the example in Figure 4. If  $\mathcal{L} = 3$  and  $\vec{R}_{sub}$  ( $C_2 \rightarrow C_3 \rightarrow C_4$ ) dominates  $\vec{R}'_{sub}$  ( $C_2 \rightarrow C'_3 \rightarrow C_4$ ), the candidate  $PoCab \rightarrow C_2 \rightarrow C_3 \rightarrow C_4$  dominates the candidate  $PoCab \rightarrow C_2 \rightarrow C'_3 \rightarrow C_4$  by Definition 1. Thus we can prune the candidate contains  $\vec{R}'_{sub}$  in advance before online recommendation. Specifically, the *LCP* algorithm will enumerate all the  $\mathcal{L}$ -length sub-routes, which include only pick-up points, and prune the dominated sub-routes by Definition 2 offline. This pruning process could be done offline before the position of a taxi driver is known. As a result, *LCP* pruning will save a lot of computational cost since it reduces the search space effectively.

### 4.3 The *SkyRoute* Algorithm

In this subsection, we show how to leverage the idea of skyline computing for identifying representative skyline routes among all the candidate routes. Here, we first formally define skyline routes.

**DEFINITION 3. Skyline Route.** A recommended driving route  $\vec{R}^{\mathcal{L}}$  is a skyline route iff  $\forall \vec{R}_i^{\mathcal{L}} \in \vec{\mathcal{R}}, \vec{R}_i^{\mathcal{L}}$  cannot dominate  $\vec{R}^{\mathcal{L}}$  by Definition 1. This is denoted as  $\vec{R}_i^{\mathcal{L}} \not\vdash \vec{R}^{\mathcal{L}}$ .

The skyline route query retrieves all the skyline routes with a length of  $\mathcal{L}$ . Formally, we use  $\vec{\mathcal{R}}_{Skyline}$  to represent the set of all the skyline routes.

**LEMMA 4. Joint Principle of Skyline Routes and the PTD Function  $\mathcal{F}$ .** The optimal driving route determined by the PTD function  $\mathcal{F}$  should be a skyline route. This is denoted as  $\vec{R}^{\mathcal{L}} \in \vec{\mathcal{R}}_{Skyline}$

**PROOF.** (Proof Sketch.) This lemma can be proved by contradiction. Assume that  $\vec{R}_1^{\mathcal{L}}$  is an optimal driving route and is not a skyline route. By Definition 3,  $\vec{R}_1^{\mathcal{L}}$  must be dominated by some driving route denoted as  $(\vec{R}_i^{\mathcal{L}})$ , which is a skyline route. By Definition 1, each attribute of the vector associating with  $\vec{R}_1^{\mathcal{L}}$  should be not smaller than the corresponding attribute of the vector associating with  $\vec{R}_i^{\mathcal{L}}$ . Also, there must be one attribute, for which the value of vector associating with  $\vec{R}_1^{\mathcal{L}}$  is bigger than that of vector associating with associating with  $\vec{R}_i^{\mathcal{L}}$ . Then, by Lemma 3, the function  $\mathcal{F}$  value of the vector associating with  $\vec{R}_1^{\mathcal{L}}$  should be less than that of the vector associating with  $\vec{R}_i^{\mathcal{L}}$ . Therefore,  $\vec{R}_1^{\mathcal{L}}$  should not be the optimal drive route.  $\square$

With the joint principle of skyline routes and the PTD function  $\mathcal{F}$  in Lemma 4, it is possible to first find skyline routes and then search for the optimal driving route from the set of skyline routes. This way can eliminate lots of candidates without computing the PTD function  $\mathcal{F}$ . Next, we show how to compute skyline routes.

Indeed, skyline computing, which retrieves non-dominated data points, has been extensively studied in the database literature [9, 11, 13, 18]. However, most of these algorithms cannot be directly used to find skyline routes in the MSR problem, because vectors associated with suggested routes are generated through an expensive cluster network traversal process. In Particular, the performances of traditional skyline computing algorithms degrade significantly when the network size increases or the length of suggested driving route is increased. Also, there are a large memory requirement for storing these vectors during the traditional skyline computing process. Moreover, for real-world applications, the position of empty cab is dynamic. Therefore, the recommended driving routes are dynamic in a real-time fashion. This means that we cannot have the indices for the multi-dimensional data points(vector  $\mathcal{DP}$ ) in advance, which is desired for many traditional skyline computing algorithms [19]. To this end, we design a *SkyRoute* algorithm for computing skyline routes by exploiting the unique properties of skyline routes for the purpose of efficient computation.

The basic idea of the *SkyRoute* algorithm is to prune some candidate routes, which are comprised of the dominated sub-routes and cannot be skyline routes, at a very early stage. This idea is based on the observation that any recommended driving routes are composed of sub-routes and different routes can cover the same sub-routes. The search space will be significantly reduced, since lots of candidate routes containing the dominated sub-routes will be discarded from further consideration as skyline routes. In the following, we first introduce two propositions for candidate routes pruning based on dominated sub-routes.

**PROPOSITION 2. Backward Pruning.** *If a sub-route  $R_1$  from PoCab to an intermediate pick-up point  $C_i$  is dominated by another sub-route  $R_2$  from PoCab to  $C_i$  under the sub-route dominance By Definition 2, then all the candidate routes  $\vec{R}_{\ni R_1}^{\mathcal{L}}$ , which have  $R_1$  as a precedent sub-route will be dominated by the candidate routes  $\vec{R}_{\ni R_2}^{\mathcal{L}}$ . The only difference between  $\vec{R}_{\ni R_1}^{\mathcal{L}}$  and  $\vec{R}_{\ni R_2}^{\mathcal{L}}$  is from PoCab to  $C_i$ . Therefore, those candidate routes  $\vec{R}_{\ni R_1}^{\mathcal{L}}$  can be pruned in advance.*

**PROPOSITION 3. Forward Pruning.** *If a sub-route  $R_1$  from one pick-up point  $C_i$  to another pick-up point  $C_j$  is dominated by another sub-route  $R_2$  from  $C_i$  to  $C_j$  under the sub-route dominance by Definition 2, then all the candidate routes  $\vec{R}_{\ni R_1}^{\mathcal{L}}$ , which contain  $R_1$  as sub-route will be dominated by the candidate routes  $\vec{R}_{\ni R_2}^{\mathcal{L}}$ . The only difference between  $\vec{R}_{\ni R_1}^{\mathcal{L}}$  and  $\vec{R}_{\ni R_2}^{\mathcal{L}}$  is from  $C_i$  to  $C_j$ , Therefore, those candidate routes  $\vec{R}_{\ni R_1}^{\mathcal{L}}$  can be pruned in advance.*

With the proposition of Backward Pruning, it is possible to decide some dominated sub-routes and discard some candidate routes which contain these dominated sub-routes. Also, the benefit of the proposition of Forwarding Pruning is the ability to prune some dominated sub-routes as well as some candidate routes offline, since both probabilities and distances between pick-up points can be obtained before any online recommendation of driving routes. Note that only sub-routes with a length less than  $\mathcal{L}$  need to be considered in the above discussion.

Figure 5 shows the pseudo-code of the *SkyRoute* algorithm. As can be seen, during offline processing, *SkyRoute* checks the dominance of sub-routes with a length  $\mathcal{L}$  by Definition 2 and prunes the ones dominated by others. This process is also applied in the *LCP* algorithm. In addition, *SkyRoute* can also prune sub-routes with different lengths with Forward Pruning in proposition 3. During online processing, results of offline processing are used as candidate routes. From line 2 to line 5, *SkyRoute* iteratively checks the sub-routes with *PoCab* as the source node and prunes the candidate routes containing dominated sub-routes with Backward Pruning in proposition 2. Then, in line 6, the candidate set is obtained after all the pruning process. Finally, a skyline query [18] is conducted on this candidate set to find skyline routes. Note that the online search time of the optimal driving route should include the time of online process of *SkyRoute* and the search time on the set of skyline routes.

#### 4.4 Obtaining the Optimal Driving Route

For both *LCP* and *SkyRoute* algorithms, after all the pruning process, we will have a set of final candidate routes for a given taxi driver. To obtain the optimal driving route, we can simply compute the PTD function  $\mathcal{F}$  for all the remaining candidate routes with a length  $\mathcal{L}$ . Then, the route with the minimal PTD value is the optimal driving route for this given taxi driver.

#### 4.5 The Recommendation Process

Even though we can find the optimal drive route for a given cab with its current position, it is still a challenging problem about how to make the recommendation for

#### ALGORITHM *SkyRoute*( $\mathcal{C}, \mathcal{P}, \text{Dist}, \mathcal{L}, \text{PoCab}$ )

Input:

- $\mathcal{C}$ : set of cluster nodes with central positions
- $\mathcal{P}$ : probability set for all cluster nodes
- Dist*: pairwise drive distance matrix of cluster nodes
- $\mathcal{L}$ : the length of suggested drive route
- PoCab*: the position of one empty cab

Output:

- $\vec{R}_{\text{Skyline}}$ : list of skyline drive routes.

#### Online Processing

1. Enumerate all candidate routes by connecting *PoCab* with each sub-route of  $\mathcal{R}_{\text{sub}}^{\mathcal{L}}$  obtained in step 10 during Offline Processing
2. **for**  $i = 2 : \mathcal{L} - 1$
3.     Decide dominated sub-routes with  $i$ th intermediate cluster and prune the corresponding candidates by using proposition 2
4.     Update the candidate set by filtering the pruned candidates in step 3
5. **end for**
6. Select the remained candidate routes with length of  $\mathcal{L}$  from the loop above
7. Final typical skyline query to get  $\vec{R}_{\text{Skyline}}$  from those candidate routes in step 6

#### Offline Processing(*LCP*)

8. Enumerate all sub-routes with length of  $\mathcal{L}$  from  $\mathcal{C}$
9. Prune and maintain dominated Constrained Sub-routes with length of  $\mathcal{L}$  using proposition 3
10. Maintain the remained non-dominated sub-routes with length of  $\mathcal{L}$ , denoted as  $\mathcal{R}_{\text{sub}}^{\mathcal{L}}$

Figure 5: The *SkyRoute* Algorithm

many cabs in the same area. In this section, we address this problem and introduce a strategy for the recommendation process in the real world.

A simple way is to suggest all these empty cabs to follow the same optimal drive route, however there is naturally an overload problem, which will degrade the performance of the recommender system. To this end, we employ load balancing techniques [10] to distribute the empty cabs to follow multiple optimal drive routes. The problem of load balancing has been widely used in distributed systems for the purpose of optimizing a given objective through finding allocations of multiple jobs to different computers. For example, the load balancing mechanism distributes requests among web servers in order to minimize the execution time. For the proposed mobile recommendation system, we can treat multiple empty cabs as jobs and multiple optimal drive routes as computers. Then, we can deal with this overload problem by exploiting existing load balancing algorithms. Specifically, in this study, we apply the *circulating mechanism* for the recommender systems by exploiting a Round Robin algorithm [22], which is a static load balancing method.

Under the *circulating mechanism*, to make recommendation for multiple empty cabs, a round robin scheduler alternates the recommendation among multiple optimal drive routes in a circular manner. As shown in Figure 6, we could search  $k$  optimal drive routes and recommend the NO.1 route to the first coming empty cab. Then, for the second empty cab, the NO. 2 drive route will be recommended. Assume there are more than  $k$  empty cabs, recommendations are repeated from NO. 1 route again after the  $k$ th empty cab. In practice, to achieve this, one central dispatch (processor) is needed to maintain the empty cabs and as-

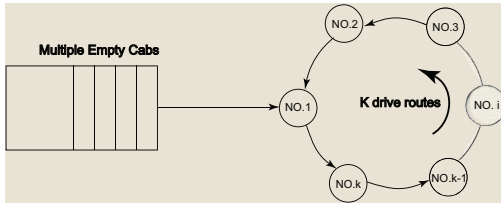


Figure 6: Illustration of the *Circulating Mechanism*.

signments among the top-k driving routes. Note that the load balancing techniques are not the focus of this paper.

## 5. EXPERIMENTAL RESULTS

In this section, we evaluate the performances of the proposed two algorithms: *LCP* and *SkyRoute*.

### 5.1 The Experimental Setup

**Real-world Data.** In the experiments, we have used real-world cab mobility traces, which are provided by the Exploratorium - the museum of science, art and human perception through the cabspotting project [1]. This data set contains GPS location traces of approximately 500 taxis collected around 30 days in the San Francisco Bay Area. For each recorded point, there are four attributes: latitude, longitude, fare identifier and time stamp. In the experiments, we select the successful cab drivers and generate the cluster information as follows. Specifically, we select cab drivers with total driving hours over 230 and occupancy rates greater than 0.5. In total, we obtain 20 cab drivers and their location traces. Based on this selected data, we generate potential pick-up points and the pick-up probability associated with each pick-up point for different time periods. In the experiments, we focus on two time periods: *2PM-3PM* and *6PM-7PM*. For these two time periods, we obtain 636 and 400 historical pick-up points respectively. After calculating the pairwise driving distance of pick-up points with the Google Map API, we use Cluto [12] for clustering. All default parameters are used in the clustering process except for “-clmethod=direct”. Please note that, since the driving distance measured by the Google Map API depends on the driving direction, we use the average to estimate the distance between each pair of pick-up points. Finally, we group the historical pick-up points into 10 clusters. The traveling distances between clusters are measured between centroids of clusters with the Google Map API.

**Synthetic data.** To enhance validation, we also generate synthetic data for the experiments. Specifically, we randomly generate potential pick-up points within a specified area and generate the pick-up probability associated with each pick-up point by a standard uniform distribution. In total, we have 3 synthetic data sets with 10, 15 and 20 pick-up points respectively. For this synthetic data, we use the Euclidean distance instead of the driving distance to measure the traveling distance between pick-up points. Also, for both real-world and synthetic data, we randomly generate the positions of the target cab for recommendation.

**Experimental Environment.** The algorithms were implemented in Matlab2008a. All the experiments were conducted on a Windows 7 with Intel Core2 Quad Q8300 and 6.00GB RAM. The search time for the optimal driving route and the skyline computing time are two main performance metrics. All the reported results are the average of 10 runs.

### 5.2 An Illustration of Optimal Driving Routes

Here, we show some optimal driving routes determined by the PTD function  $\mathcal{F}$  on real-world data.

In Figure 7, we plot the potential pick-up points within the time period *6PM-7PM* and the assumed position of the target cab for recommendation. During this time period, the optimal drive routes evaluated by the PTD function are  $PoCab \rightarrow C1 \rightarrow C3 \rightarrow C2$ ,  $PoCab \rightarrow C1 \rightarrow C3 \rightarrow C2 \rightarrow C7$  and  $PoCab \rightarrow C4 \rightarrow C1 \rightarrow C3 \rightarrow C2 \rightarrow C7$  for  $\mathcal{L} = 3$ ,  $\mathcal{L} = 4$  and  $\mathcal{L} = 5$  respectively. Please note that we have documented more results and illustrations in a Web site listed in Appendix.



Figure 7: Illustration: Optimal Driving Routes.

Table 1: Some Acronyms.

BFS:	Brute-Force Search .
<i>LCPS</i> :	Search with <i>LCP</i>
SR(BNL):	Search via Skyline Computing algorithm <i>SkyRoute</i> + <i>BNL</i> .
SR(D&C):	Searching via Skyline Computing Algorithm <i>SkyRoute</i> + <i>D&amp;C</i> .

### 5.3 An Overall Comparison

In this subsection, we show an overall comparison of computational performances of several algorithms.

First, in *SkyRoute*, after the pruning process proposed in this paper, we apply some traditional skyline computing methods to find the skylines from the remained candidate set. Here, we employ two skyline computing methods, *BNL* and *D&C* [18]. In this experiment, all acronyms of evaluated algorithms are given in Table 1. Note that, for BFS, we only compute the PTD value for all candidate routes one by one and find the maximum value as well as the optimal driving route. Also, most information, such as the locations of potential pick-up points and the pick-up probability, can be known in advance. The online computations are the distance from the target cab to pick-up points and PTD function.

Figure 8 shows the online search time of optimal driving routes evaluated by the PTD function for different values of  $\mathcal{L}$  on both synthetic data and real-world data. The search

time shown here includes all the time for online processing. As can be seen, *LCPS* outperforms BFS and SR(D&C)S with a significant margin for all different lengths of the optimal drive route on both synthetic and real data. The reason why searching via skyline computing takes longer time than *LCPS* or BFS is that skyline computing is partially online processing and takes a lot of time. Although we only show the results of the time period 6PM – 7PM, a similar trend has also been observed in other time periods. Due to the space limit, we omit these results here. However, we have documented more results in a Web site listed in Appendix.

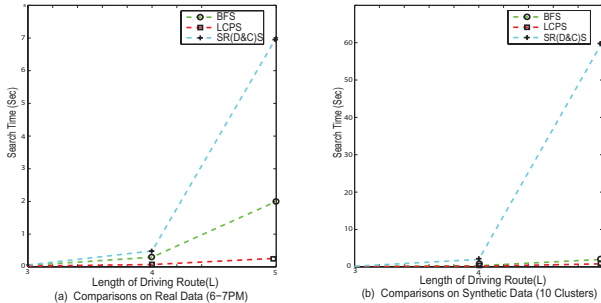


Figure 8: A Comparison of Search Time.

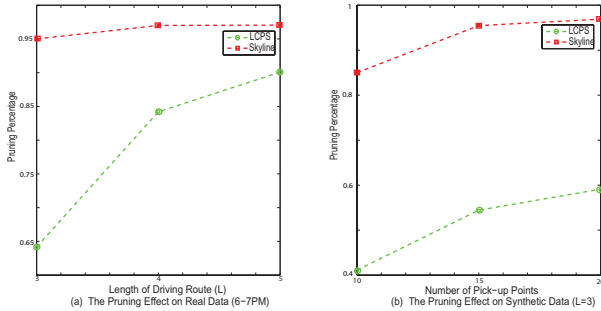


Figure 9: The Pruning Effect

In terms of the pruning effect, both *LCP* and *SkyRoute* can prune the search space significantly as shown in Figure 9, where we show the pruning ratios of *LCP* and *Skyroute*. Note that the pruning ratio is the number of pruned candidates divided by the original number of all the candidates.

In addition, for *LCPS*, the pruning process can be done in advance. This saves a lot of time for online search. In particular, Table 2 shows a comparison of online search time between BFS and *LCPS* across different numbers of pick-up points and different lengths of driving routes on both synthetic and real-world data. As can be seen, *LCPS* always outperforms BFS with a significant margin.

Table 2: A Comparison of Search Time (Second) between BFS and *LCPS*

10 Synthetic Pick-up Clusters			
	$\mathcal{L} = 3$	$\mathcal{L} = 4$	$\mathcal{L} = 5$
BFS	0.051643	0.300211	2.000949
<i>LCPS</i>	<b>0.043750</b>	<b>0.165401</b>	<b>0.803290</b>
15 Synthetic Pick-up Clusters			
BFS	0.142254	1.925054	23.517042
<i>LCPS</i>	<b>0.095364</b>	<b>0.611193</b>	<b>4.322053</b>
Real Data (2-3PM)			
BFS	0.045933	0.297187	1.991507
<i>LCPS</i>	<b>0.036736</b>	<b>0.141536</b>	<b>0.622932</b>

Finally, Figure 10 shows the online search time of optimal driving routes ( $\mathcal{L} = 3$ ) across different numbers of pick-up

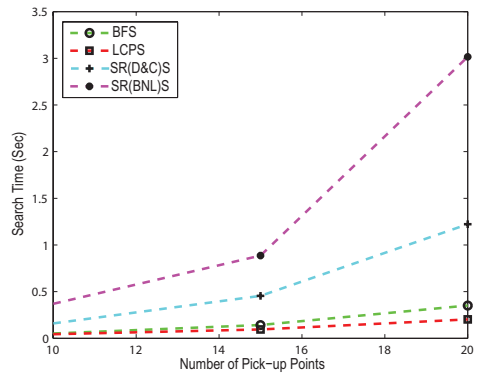


Figure 10: A Comparison of Search Time ( $\mathcal{L} = 3$ ) on the Synthetic Data set.

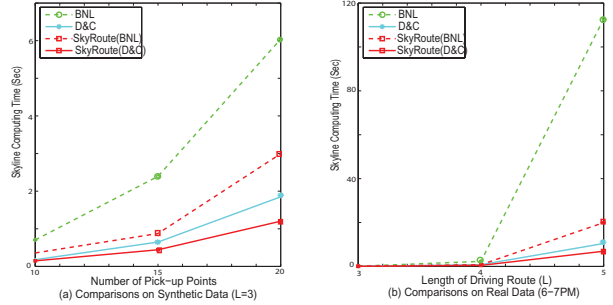


Figure 11: A Comparison of Skyline Computing

points on synthetic data. In the figure, a similar trend of performances can be observed as in Figure 8.

## 5.4 A Comparison of Skyline Computing

In this subsection, we evaluate the performances of different skyline computing algorithms.

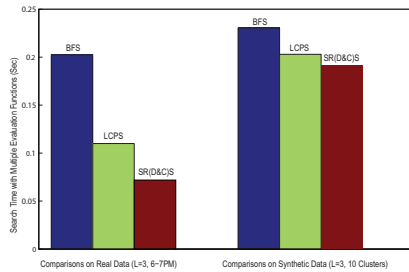
This experiment was conducted across different numbers of pick-up points and different lengths of recommended driving routes on both synthetic and real-world data. As shown in Figure 11, *SkyRoute* with *BNL* or *D&C* can lead to better efficiency compared to traditional skyline computing methods. The above indicates that *SkyRoute* is an effective method for computing Skyline routes.

Furthermore, we have observed that the computation cost of *BNL* or *D&C* varies on different data sets with the same size of candidate routes. The reason is that *BNL* or *D&C* has different computation complexity for the best and worse cases. Therefore, even with the same number of pick-up points and the same length of driving routes, the running time of *SkyRoute(BNL)*, (*SkyRoute(D&C)*) or *SR(D&C)S* is different as shown in Figure 11 and Figure 8

## 5.5 Case: Multiple Evaluation Functions

Here, we show the advantages of searching optimal driving routes through skyline computing. Specifically, we evaluate the following business scenario. When there are business needs for different ways to define optimal driving routes, which can be measured by different evaluation functions.

As can be seen in Figure 8 and Figure 10, the search of an optimal driving route via skyline computing does not outperform *LCPS* or BFS, because it takes the most part of total online processing time for computing skylines. However, for a target cab and fixed potential pick-up points, we only need to compute skylines once. And the search space can be



**Figure 12: A Comparison of Search Time for Multiple Optimal Driving Routes**

pruned drastically as shown in Figure 9. In other words, if the goal is to provide multiple optimal driving routes based on different business needs at the same time. Skyline computing will have an advantage.

To illustrate this benefit of skyline computing, we design 5 different evaluation functions (including PTD) to select 5 corresponding optimal drive routes. Note that all these evaluation functions have the monotonicity Property as stated in lemma 3. Due to the space limitation, we omit the details of these evaluation functions. Then, we search five different optimal driving routes simultaneously with the methods shown in Table 1 on both synthetic data and real-world data. Figure 12 shows the comparisons of computational performances with  $\mathcal{L} = 3$ . As can be seen, SR(D&C)S outperforms LCPS and BFS with a significant margin.

## 6. CONCLUDING REMARKS

In this paper, we developed an energy-efficient mobile recommender system by exploiting the energy-efficient driving patterns extracted from the location traces of Taxi drivers. This system has the ability to recommend a sequence of potential pick-up points for a Taxi driver in a way such that the potential travel distance before having customer is minimized. To develop the system, we first formalized a mobile sequential recommendation problem and provided a Potential Travel Distance (PTD) function for evaluating each candidate sequence. Based on the monotone property of the PTD function, we proposed a mobile recommendation algorithm, named *LCP*. Moreover, we observed that many candidate routes can be dominated by skyline routes, and thus can be pruned by skyline computing. Therefore, we also proposed a SkyRoute algorithm to efficiently compute the skylines for candidate routes. An advantage of searching an optimal drive route through skyline computing is that it can save the overall online processing time when we try to provide different optimal driving routes defined by different business needs.

Finally, experimental results showed that the *LCP* algorithm outperforms the brute-force method and *SkyRoute* with a significant margin when searching only one optimal driving route. Moreover, the results showed that *SkyRoute* leads to better performances than brute-force and *LCP* when there is an online demand for different optimal drive routes defined by different evaluation criteria.

## 7. REFERENCES

- [1] <http://cabspotting.org/>.
- [2] G. Abowd, C. Atkeson, and et al. Cyber-guide: A mobile context-aware tour guide. *Wireless Networks*, 3(5):421–433, 1997.
- [3] G. Adomavicius and A. Tuzhilin. Towards the next

generation of recommender systems: A survey of the state-of-the art and possible extensions. *TKDE*, 2005.

- [4] D. L. Applegate, R. E. Bixby, and et al. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.
- [5] O. Averjanova, F. Ricci, and Q. N. Nguyen. Map-based interaction with a conversational mobile recommender system. In *The 2nd Int'l Conf on Mobile Ubiquitous Computing, Systems, Services and Technologies*, 2008.
- [6] F. Cena, L. Console, and et al. Integrating heterogeneous adaptation techniques to build a flexible and usable mobile tourist guide. *AI Communications*, 19(4):369–384, 2006.
- [7] K. Cheverst, N. Davies, and et al. Developing a context-aware electronic tourist guide: some issues and experiences. In *the SIGCHI Conference on Human Factors in Computing Systems*, pages 17–24, 2000.
- [8] M. Dell'Amico, M. Fischetti, and P. Toth. Heuristic algorithms for the multiple depot vehicle scheduling problem. *Management Science*, 39(1):115–125, 1993.
- [9] D. Papadias, G. Y. Tao, and B. Seeger. Progressive skyline computation in database systems. *ACM TODS*, 30(1):43–82, 2005.
- [10] D. Grosu and A. T. Chronopoulos. Algorithmic mechanism design for load balancing in distributed systems. *IEEE TSMC-B*, 34(1):77–84, 2004.
- [11] J. Chomicki, J. P. Godfrey, and D. Liang. Skyline with presorting. In *ICDE*, pages 717–719, 2003.
- [12] G. Karypis. Cluto: <http://glaros.dtc.umn.edu/gkhome/views/cluto>.
- [13] T. Kian-Lee, E. Pin-Kwang, and B. C. Ooi. Efficient progressive skyline computation. In *Vldb*, 2001.
- [14] B. N. Miller, I. Albert, and et al. MovieLens unplugged: Experiences with a recommender system on four mobile devices. In *international conference on Intelligent user interfaces*, 2003.
- [15] R. J. Mooney and L. Roy. Content-based book recommendation using learning for text categorization. In *Workshop Recom. Sys.: Algo. and Evaluation*, 1999.
- [16] M. Pazzani. A framework for collaborative, content-based, and demographic filtering. *Artificial Intelligence Review*, 1999.
- [17] R. Portugal, H. R. Lourenc4o, and J. P. Paixao. Driver scheduling problem modelling. *Public Transport*, 1(2):103–120, 2009.
- [18] S. Borzsonyi, K. Stocker, and D. Kossmann. The skyline operator. In *ICDE*, pages 421–430, 2001.
- [19] Y. Tian, K. C.K. Lee, and W.-C. Lee. Finding skyline paths in road networks. In *GIS*, pages 444–447, 2009.
- [20] A. Tveit. Peer-to-peer based recommendations for mobile commerce. In *the 1st international workshop on Mobile commerce*, 2001.
- [21] H. van der Heijden, G. Kotsis, and R. Kronsteiner. Mobile recommendation systems for decision making 'on the go'. In *ICMB*, 2005.
- [22] Z. Xu and R. Huang. Performance study of load balancing algorithms in distributed web server systems. In *TR*, CS213 Univ. of California, Riverside.

## APPENDIX

More results have been documented at the Web site: <http://www.pegasus.rutgers.edu/~yongge/kdd2010/mobile.html>