SOR: An Objective Ranking System Based on Mobile Phone Sensing

Xiang Sheng[†], Jian Tang[†], Jing Wang[†], Chenfei Gao[†] and Guoliang Xue[‡]
Department of Electrical Engineering and Computer Science[†]
Syracuse University, Syracuse, NY, 13244.
Email: {xsheng, jtang02, jwang93@syr.edu, cgao03@syr.edu}@syr.edu
Arizona State University, Tempe, AZ, 85287.[‡] Email: xue@asu.edu

Abstract*—Currently, a few online review and recommendation systems (such as Yelp and TripAdvisor) have attracted millions of users and are gaining increasing popularity. They usually rate and rank places and attractions based on subjective ratings provided by users. In this paper, we present design, implementation and evaluation of a mobile phone Sensing based Objective Ranking (SOR) system, which ranks a target place based on data collected via mobile phone sensing. Our system has the following desirable features: 1) it is easy to use; 2) its architecture is so scalable that various embedded and external sensors can be easily integrated into it; 3) an online scheduling algorithm is proposed and used to schedule sensing activities for coverage maximization, which has a constant approximation ratio of $1/2$; 4)a personalizable ranking algorithm is developed and used to rank target places based on various sensor readings and user preferences. We validate and evaluate SOR via both field tests (using real hiking trails and coffee shops in Syracuse, NY as target places) and simulation. The field-testing results show that data collected and processed by SOR can well capture characteristics of target places, and personalizable rankings produced by SOR can well match user preferences. In addition, simulation results well justify effectiveness of the proposed scheduling algorithm.

***Index Terms*—Mobile Phone Sensing, Scheduling, Ranking, Mobile Computing.**

I. Introduction

Mobile phones have evolved as the most important communication and entertainment devices in people's daily life. Many people, however, are not aware that a mobile phone can serve as a powerful sensing device. Most of modern smartphones (such as iPhone5, Google Nexus4, etc.) are also equipped with a rich set of embedded sensors such as camera, GPS, WiFi/3G/4G interfaces, accelerometer, digital compass, gyroscope, microphone and so on. Moreover, external sensors can also be connected to a mobile phone via its Bluetooth interface. For example, as shown in Fig. 1, Sensordrone [27] is a portable and wearable multisensor that can turn a smartphone into an environmental monitor. This small device is equipped with 10 different sensors, including multiple gas sensors, a non-contact thermometer, a humidity sensor, a temperature sensor, a light sensor, a color sensor and a pressure sensor. It can even be connected to more sensors via an expansion connector. Other examples include the well-known Google Glass [11] and Fitbit [8]. These embedded and

This research was supported in part by NSF grants #1218203, #1217611 and #1113398. The information reported here does not reflect the position or the policy of the federal government.

Fig. 1. An external sensor: Sensordrone [27]

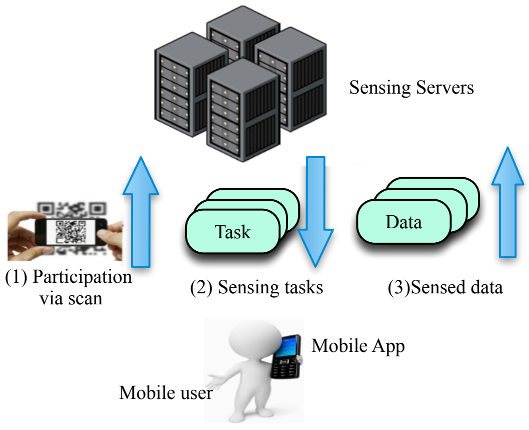


Fig. 2. An overview of SOR

external sensors can enable attractive sensing applications in various domains [16] such as environmental monitoring, social network, healthcare, transportation, safety, etc.

Currently, a few online review and recommendation systems have attracted millions of users and are gaining increasing popularity. For example, Yelp serves as an online urban guide, which provides user reviews, recommendations and rankings of a large variety of local businesses including restaurants, shops, theaters, etc. Another typical example is TripAdvisor, which has become the world's largest website for travelers. It provides user reviews, ratings and ranks for hotels, restaurants, attractions in different places across the whole world. These systems usually rate and rank target places and attractions based on *subjective* ratings and opinions provided by users.

Data collected via mobile phone sensing can be used to evaluate a target place. For example, for a coffee shop, based on readings from microphones, we can know if it is quiet; based on readings from light sensors, we can know if it is bright; based on readings from temperature sensors (on sensordrones), we can know if it is warm. In this paper, we present design, implementation and evaluation of an objective ranking system, which ranks target places based on (objective) data collected via mobile phone sensing. Our objective is not to replace the current ranking/recommendation systems that are based on subjective user ratings but to enhance them with objective data collected via various sensors to provide more comprehensive and objective rankings and recommendations for users.

The proposed mobile phone Sensing based Objective Ranking (SOR) system is illustrated in Fig. 2. In order to use it, the following components must be deployed properly: 1) *Mobile Application:* a mobile application needs to be installed on each participating mobile phone. 2) *Sensing Server:* One or multiple sensing servers need to be deployed to collect sensed data from mobile phones. 3) *2D Barcode:* A 2D barcode needs to be deployed in a target place to trigger a sensing procedure. Next, we describe how the system works:

1) If a mobile user decides to participate, he/she opens the mobile application and scans the 2D barcode in the target place, which will sends a notification (with information about the target place contained in the barcode) to a sensing server and trigger a sensing procedure.
2) A sensing server detects the incoming participation request, calculates a sensing schedule and sends the corresponding sensing tasks to the mobile phone.
3) The mobile application operates sensors to sense according to the provided schedule and sends sensed data back to a sensing server.
4) The sensing server collects and processes sensed data from mobile phones, and stores them into a database.
5) A ranking program ranks the target place based on data collected from mobile phones and user preferences.

It is quite challenging to design such an objective ranking system. First, in order to provide a comprehensive view for target places, the system needs to leverage a large variety of embedded and external sensors to collect various data. Second, a mobile user may participate at any time. The system needs to schedule sensing activities properly to ensure a good coverage over a given scheduling period. It is certainly not desirable to have sensor readings clustered on certain short periods of time. In addition, the system needs to be able to rank a target place based on various sensed data. In the following, the term *"mobile user"* refers to a person who participates in sensing activities and contributes sensed data; while the term *"user"* refers to a person who uses SOR to find out rankings and recommendations. A person can certainly be both user and mobile user.

In our design, we carefully address these challenges. SOR has the following desirable features: 1) it is easy to use because an easy 2D barcode scan triggers a sensing procedure, which is then automatically done without user involvement; 2) its architecture is so scalable that various embedded and external sensors can be easily integrated into it; 3) an online scheduling algorithm is proposed and used to schedule sensing activities for coverage maximization, which has a constant approximation ratio of $1/2$; 4) a personalizable ranking algorithm is developed and used to rank target places based on various sensor readings and user preferences. We summarize our contributions in the following:

1) We design and implement an objective ranking system based on mobile phone sensing, which is the first of its kind.
2) We develop theoretically well-founded and practically efficient scheduling and ranking for the proposed system.
3) We justify effectiveness of the proposed system and algorithms via both field tests (using real hiking trails and coffee shops in Syracuse, NY as target places) and simulation.

Note that the proposed system, ranking algorithm and sensed data can be integrated into existing subjective ranking and recommendation systems to provide a more comprehensive and objective view of target places for users. However, due to space limitation, this paper is only focused on mobile phone sensing and ranking based on objective data collected by mobile phones.

The rest of the paper is organized as follows: We present the software architecture and implementation details of the proposed system in Section II. The proposed scheduling and ranking algorithms are presented in Section III and Section IV respectively. Experimental and simulation results are presented and analyzed in Section V. We discuss related works in Section VI and conclude the paper in Section VII.

II. Design and Implementation of SOR

In this section, we discuss design and implementation details of SOR, which consists of two major components: Mobile Frontend (i.e., the mobile application running on each participating smartphone) and Sensing Server (backend).

A. Mobile Frontend

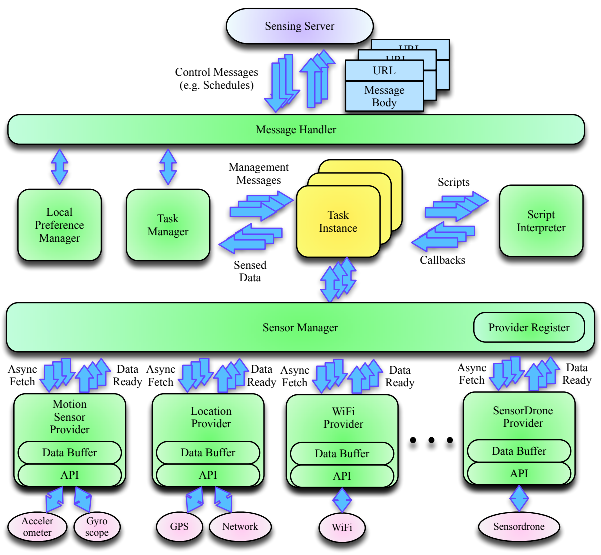


Fig. 3. The architecture of mobile frontend

We implemented the mobile frontend on the Android system. It consists of the following components: *Message Handler, Local Preference Manager, Sensing Task Manager, Script*

Interpreter, Sensor Manager, Providers, which are illustrated in Fig. 3.

The *Message Handler* serves as an interface for communications between the mobile frontend and a sensing server. In SOR, HTTP is used as the communication protocol. All SOR-specific information is encoded as binary data and stored in the message body of an HTTP message. In this way, we can minimize traffic load and enhance security (since the third party system does not know how to decode it). The Message Handler is responsible for encoding/decoding the message body. Every time when a sensing server needs a mobile phone to undertake a sensing task, it will include all necessary information (e.g., when to sense, how to sense, etc) in an HTTP message, which will then be sent to the Message Handler on the mobile frontend. How to sense, i.e., what data to acquire, is described using the Lua [18] scripting language. Note that Lua is a powerful, fast, lightweight, and embeddable scripting language. It combines simple procedural syntax with powerful data description constructs based on associative arrays and extensible semantics, which make it quite fit for sensing task description. Sample Lua scripts (with comments) are presented in Fig. 4. Note that those functions such as `get_light_readings()` and `get_location()` in the scripts are functions defined by us and will be mapped to the callback functions (for asynchronous data fetching).

```
—[[ If Sensordrone is connected, get 5 light readings, with an interval of 2s]]—

if sensordrone_connected() then
    get_light_readings(5, 2)
end

—[[ If GPS query is permitted and the location data could be obtained,
    then, get 2 GPS location readings; otherwise, get 2 coarse locations. ]]—

if fine_location_permitted and gps_data_available() then
      get_location(2, 'fine')
else
      get_location(2, 'coarse')
end
```

Fig. 4. Lua scripts

The other functionalities of the Message Handler include: 1) it dispatches an incoming message to the intended receiver (such as Task Manager, Preference Manager, etc). 2) It encodes data obtained from sensors in a message and sends it to a sensing server. 2) It can communicate with a Google server. This is useful when a sensing server loses track of a particular mobile phone, it can ask the mobile device to ping it via a Google Cloud Messaging server. 3) It can prevent a mobile phone from going to sleep during communications with a server, which is implemented by using the Android system API `powerManager.newWakeupLock()`.

Each incoming task will be served by a task instance, which will be hosted by a thread. The *Task Manager* keeps track of all these task instances. A task instance sends the corresponding Lua scripts to the *Script Interpreter* for translation. The interpreter can interpret both Lua's own functions and the functions we defined for data acquisition. The script interpreter tells the task instance which Java function to call to obtain data from sensors since the Android system cannot recognize Lua scripts. Note that security can be enforced here by only allowing a white list of unharmful functions to be called. A task instance is a self-contained component, which maintains its own status (e.g, running, waiting for data, etc), call proper API functions to acquire data from sensors, and manages data collected from sensors. SOR is a multi-task system, where concurrency is well supported. At one time, there could be multiple task instances running in SOR, which can acquire data from one or multiple sensors simultaneously.

The architecture of the mobile frontend is scalable because various sensors can be easily integrated into it, which is achieved by *Sensor Manager* and *Providers*. If we want to make SOR support a new sensor (embedded or external), we only need to create a *Provider* for that sensor. A Provider is basically a software component which actually operates embedded and external sensors using APIs provided by the Android system and third party respectively to collect data. Note that each Provider maintains a data buffer which buffers data collected from its sensor and can even share them with multiple different tasks. In this way, energy consumed for sensing can be reduced. Note that data acquisition from a sensor is done in an asynchronous manner such that an operation will not block or be blocked by others. When a new sensor is integrated into SOR, the corresponding Provider needs to be registered with the *Sensor Manager* via the *Provider Register*, which keeps a list of currently supported sensors and the corresponding data acquisition functions we defined (such as `get_light_readings()` and `get_location()`). When a task instance requests data by calling such a data acquisition function, the *Sensor Manager* directs the call to the corresponding Provider to actually acquire data from sensors. Moreover, the manager can cancel data acquisition if timeout. Currently, SOR can support all sensors available on a Google Nexus4 smartphone and all sensors available on a Sensordrone.

SOR also allows a user to specify how sensors on his/her phone can be used to participate in sensing activities. For example, a user may not want to expose his/her exact locations to our system, then he/she can disallow the phone to return locations provided by GPS. These preferences are maintained by the *Local Preference Manager* as shown in the figure.

B. Sensing Server

A sensing server consists of *User Info Manager, Application Manager, Participation Manager, Task Scheduler, Data Processor, Personalizable Ranker and Database*, which are illustrated in Fig. 5.

The *Message Handler* in the sensing server is quite similar to its counterpart in the mobile frontend. It communicates with the mobile frontend using HTTP and dispatches incoming message to different components. Note that if it detects that the received message includes sensed data, it will directly store the binary message body into the database, which will be processed later by the *Data Processor*.

The *User Info Manager* maintains user information, including userID, name, token (used to uniquely identify a mobile device), etc. Here, an *application* is defined as a

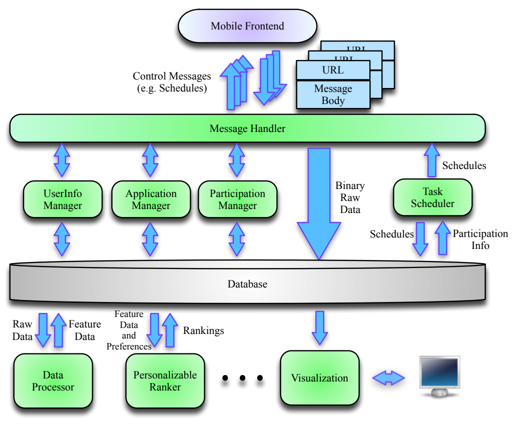


Fig. 5. The architecture of sensing server

procedure of acquiring data from sensors for a target place, which may include multiple sensing tasks. The *Application Manager* manages all necessary information related to each application, including its AppID, its creator (which could be the owner/manager/operator of the corresponding target place), and the Lua scripts defining the corresponding data acquisition procedure (See Fig. 4 for an example).

The *Participation Manager* keeps track of a list of sensing tasks and their information, including participating userID, the corresponding token, the corresponding application, the location of the target place, the sensing budget and its status (such as running, waiting for sensing schedule, finished, error, etc). Note that the sensing budget is an integer number specifying how many times the corresponding mobile user can perform data acquisition specified by the corresponding Lua scripts. Initially, it is set to the maximum number of times the mobile user is willing to acquire data from its sensors and it is updated at runtime. Every time when a mobile user scans a 2D barcode, the Participation Manager will first verify whether the user is actually in the target place by acquiring its location and comparing it against the location stored in the Application Manager, and then create a task for it if the user is considered as a truthful user. Moreover, a mobile user's status in the Participation Manager will be changed to "finished" if according to his/her location, he/she leaves the target place.

For each application, the *Sensing Scheduler* applies an online algorithm to calculate a sensing schedule (that specifies when to sense for each participating user) for a scheduling period based on runtime participation information (such as current participating users, their sensing budgets, etc) provided by the Participation Manager (via the database). The duration of a scheduling period can be specified by the creator of the application according to his/her needs. The Sensing Scheduler will also distribute the calculated schedules along with the corresponding Lua scripts to participating mobile phones, and store them into the database. We will describe the proposed scheduling algorithm in greater details in Section III.

In SOR, we chose the PostgreSQL [23] for storing data, which is an open-source Object Relational Database Management System (ORDBMS) with an emphasis on extensibility and standards compliance. The *Data Processor* periodically checks if there are any binary sensed data in the database, and if any, it decodes the data and stores useful information into corresponding tables in the database. Moreover, it also processes raw data to generate more meaningful data for various sensing features (temperature, humidity, roughness of road surface, etc), which will then be stored into the database to serve as input for the *Personalizable Ranker*. The processed data are called *feature data*, which are usually statistics (average, variance, etc) of raw data. The *Personalizable Ranker* leverages a personalizable ranking algorithm to rank target places according to feature data and user preferences. Both data processing and personalizable ranking will be discussed in Section IV. We also implemented a simple *Visualization* module, which can generate figures for feature data in the database such that users can view them easily.

III. Scheduling Algorithm

In this section, we will describe our sensing coverage model, and then present the online scheduling algorithm used in the SOR system.

In our sensing coverage model, we use a set $\mathbf{T}$ of N time instants to divide the time domain within a sensing scheduling period $[t^S, t^E]$ into small time intervals with equal durations. The measurements are scheduled to be taken only at these time instants. Of course, the larger the N, the more accurate the measurement, however, the higher the sensing cost (such as energy consumption). If a sensing feature is measured at time t_i, then we say time instant t_j is *covered* with a probability of $p(t_i, t_j)$, which means that if a measurement is taken at t_i, the reading at t_j stays the same with a probability of $p(t_i, t_j)$. The closer t_j is to t_i, the higher the probability becomes. So a bell-shaped Guassian distribution $\mathcal{N}(\mu, \sigma)$ is used to model these probabilities. Different variance σ can be used to model different sensing features. A large σ is used for those sensing features whose readings do not change drastically over time (such as temperature, humidity, etc), while a small σ is used for those whose readings may change quickly (such as acceleration, orientation, etc). Note that our algorithm is general enough such that other distribution models can also be applied here.

A sensing schedule can be given as a set $\mathbf{\Phi}$ of time instants within a given scheduling period, which specifies when to sense for the corresponding sensors on mobile phones. The probability that time instant t_j is covered by a given sensing schedule $\mathbf{\Phi}$ is:

$$p(t_j, \mathbf{\Phi}) = 1 - \prod_{t_i \in \mathbf{\Phi}} (1 - p(t_i, t_j)). \tag{1}$$

Note that the coverage model discussed here is in the time domain, whereas, most other coverage models are in the spatial domain.

Suppose that we are given a set $\mathbf{T}$ of equally spaced instants within a scheduling period $[t^S, t^E]$ as well as the duration a mobile user k participating in sensing activities $[t_k^S, t_k^E]$, then $\mathbf{T}_k \subseteq \mathbf{T}$ is a subset of time instants in $\mathbf{T}$ that falls in $[t_k^S, t_k^E]$. A sensing schedule of user k, $\mathbf{\Phi}_k$, is a subset of time instants in $\mathbf{T}_k$. In addition, every mobile user k has a sensing budget

N_k^B, which is the number of times he/she is willing to sense during a scheduling period. We are interested in solving the following problem:

$$\max_{\{\mathbf{\Phi}_1,\cdots,\mathbf{\Phi}_K\}} \sum_{t_j \in \mathbf{T}} \sum_{k=1}^{K} p(t_j, \mathbf{\Phi}_k) \tag{2}$$

Subject to:

$$|\mathbf{\Phi}_k| \leq N_k^B, k \in \{1, \cdots, K\}. \tag{3}$$

The scheduling problem is to maximize the total sensing coverage probability by selecting a sensing schedule $\mathbf{\Phi}_k$ for each participating mobile user k, with a cardinality no more than the given budget N_k^B. The goal here is to spread measurements across the whole sensing period and in the meanwhile, ensure fairness by preventing certain mobile users from being abused.

We construct a collection of subsets of the ground set $\mathbf{T}$, $\mathbf{\Lambda} = \{\mathbf{\Psi} : \mathbf{\Psi} \subseteq \mathbf{T}, |\mathbf{\Psi} \bigcap \mathbf{T}_k| \leq N_k^B, k \in \{1, 2, \cdots, K\}\}$. Next, we show that $(\mathbf{T}, \mathbf{\Lambda})$ is a matroid.

Definition 1 (Matroid [10]): A pair (Q, Z) consisting of a ground set Q and a collection Z of subsets of Q is a matroid if:

1) $\emptyset \in Z$;
2) If $X \in Z$ and $Y \subset X$, then $Y \in Z$;
3) for all $X, Y \in Z$, if $|X| > |Y|$ then there exists some $x \in X \setminus Y$ such that $Y \bigcup \{x\} \in Z$.

Theorem 1: $(\mathbf{T}, \mathbf{\Lambda})$ is a matroid.

Proof: It is easy to see that $\emptyset \in \mathbf{\Lambda}$. Suppose that $\mathbf{\Psi}_1 \in \mathbf{\Lambda}$. According to the definition of $\mathbf{\Lambda}$, $\mathbf{\Psi}_1$ satisfies the constraint $|\mathbf{\Psi}_1 \bigcap \mathbf{T}_k| \leq N_k^B, k \in \{1, \cdots, K\}$. And if $\mathbf{\Psi}_2 \subset \mathbf{\Psi}_1$ then we have $|\mathbf{\Psi}_2 \bigcap \mathbf{T}_k| \leq |\mathbf{\Psi}_1 \bigcap \mathbf{T}_k| \leq N_k^B, k \in \{1, \cdots, K\}$. So $\mathbf{\Psi}_2 \in \mathbf{\Lambda}$.

We prove that condition 2) is also satisfied by contradiction. Suppose that $\mathbf{\Psi}_1 \in \mathbf{\Lambda}, \mathbf{\Psi}_2 \in \mathbf{\Lambda}$, and $|\mathbf{\Psi}_1| > |\mathbf{\Psi}_2|$, but there does not even exist any element x such that $x \in \mathbf{\Psi}_1 \setminus \mathbf{\Psi}_2$ and $\mathbf{\Psi}_2 \bigcup \{x\} \in \mathbf{\Lambda}$. If this statement is true, then $\mathbf{\Psi}_2 \bigcup \{x_k\} > N_k^B, \forall x_k \in \{\mathbf{\Psi}_1 \setminus \mathbf{\Psi}_2\} \bigcap \mathbf{T}_k, k \in \{1, \cdots, K\}$. This means for any k, if $\{\mathbf{\Psi}_1 \setminus \mathbf{\Psi}_2\} \bigcap \mathbf{T}_k \neq \emptyset$, then $|\mathbf{\Psi}_2 \bigcap \mathbf{T}_k| = N_k^B$. So $|\mathbf{\Psi}_1 \bigcap \mathbf{T}_k| \leq |\mathbf{\Psi}_2 \bigcap \mathbf{T}_k|, \forall \{\mathbf{\Psi}_1 \setminus \mathbf{\Psi}_2\} \bigcap \mathbf{T}_k \neq \emptyset, k \in \{1, \cdots, K\}$. And since $\{\mathbf{\Psi}_1 \setminus \mathbf{\Psi}_2\} \subset \mathbf{\Psi}_1$, and all the elements in $\mathbf{\Psi}_1 \bigcap \mathbf{\Psi}_2$ are shared by both $\mathbf{\Psi}_1$ and $\mathbf{\Psi}_2$, $|\mathbf{\Psi}_1| \leq |\mathbf{\Psi}_2|$, which is in contradiction to our assumption. This completes our proof. ■

The scheduling problem can be re-formulated as:

$$\max_{\mathbf{\Psi} \in \mathbf{\Lambda}} \sum_{t_j \in \mathbf{T}} p(t_j, \mathbf{\Psi}). \tag{4}$$

This scheduling problem falls in a class of problems of maximizing a submodular set function over a matroid [10] because its objective function $f(\mathbf{\Psi}) = \sum_{t_j \in \mathbf{T}} p(t_j, \mathbf{\Psi})$ has been shown to be a non-negative, monotone and submodular function in [31] and we show that $(\mathbf{T}, \mathbf{\Lambda})$ is a matroid in Theorem 1. We present a simple greedy algorithm to solve it in the following.

The basic idea of the proposed algorithm is to keep adding into the solution the time instant that can result in the maximum incremental coverage until no mobile users can be scheduled to sense more without violating their budget constraints. The running time of this algorithm is $O(|\mathbf{T}|^2 \cdot g(|\mathbf{\Lambda}|))$, where $g(|\mathbf{\Lambda}|)$ is the running time for testing whether $\mathbf{\Psi}_{h-1} \bigcup \{s\} \in \mathbf{\Lambda}$ or not. In our algorithm, this can be quickly done in constant time by maintaining a counter for each mobile user and checking if its value exceeds the given budget. So the overall time complexity is $O(|\mathbf{T}|^2) = O(N^2)$. Hence the proposed algorithm is time efficient.

Algorithm 1 The sensing scheduling algorithm

```
Step 1  $\mathbf{\Psi}_0 := \emptyset$; $l := 1$;
Step 2  while $\exists x \in \mathbf{T} \setminus \mathbf{\Psi}_{l-1}$ s.t. $\mathbf{\Psi}_{l-1} \bigcup \{x\} \in \mathbf{\Lambda}$
            $x^* := \text{argmax}_{x' \in \mathbf{T} \setminus \mathbf{\Psi}_{l-1}} f(\mathbf{\Psi}_{l-1} \bigcup \{x'\})$
                    $-f(\mathbf{\Psi}_{l-1})$;
            $\mathbf{\Psi}_l := \mathbf{\Psi}_{l-1} \bigcup \{x^*\}$;
            $l := l + 1$;
        endwhile
Step 3  return $\mathbf{\Psi}_h$;
```

In addition, according to Theorem 1, the scheduling problem is to maximize a non-decreasing submodular set function over a matroid. It has been shown in [10], that a simple greedy algorithm (similar to that shown above) gives a $\frac{1}{2}$-approximation for a class of such problems. Therefore, Algorithm 1 is a $\frac{1}{2}$-approximation algorithm for the scheduling problem (4).

IV. Data Processing and Personalizable Ranking

In this section, we discuss how raw data collected from mobile frontend are processed and fed to the ranking algorithm as input to calculate ranks for a target place.

A. Data Processing

In SOR, for a target place, data collected by sensors of certain type in a given scheduling period are stored as a set of 3-tuples $(t, \Delta t, \mathbf{d})$. t is the timestamp, whereas, Δt is a short period of time (typically several seconds). Note that SOR takes multiple (instead of one) readings within $[t, t + \Delta t]$ to ensure high sensing quality. The number of readings to be taken during this period can be specified in the Lua scripts. $\mathbf{d}$ is the corresponding set of readings.

Ranking is conducted based on values of a set of *humanly understandable features*, such as temperature, WiFi signal strength, roughness of road surface. For a target place, raw data need to be processed to calculate a value for each feature, which will then be used by the ranking algorithm as input. Note that the methods for calculating these values from raw data may vary with features. For example, for temperature, we take an average over all temperature sensors' readings; however, for roughness of road surface, we take an average of standard deviations of accelerometers' readings within Δt. Readings from different types of sensors may be combined to generate the value for a feature too. SOR calculates these statistics (*feature data*) and stores them into the database. When they are needed for ranking, they are read from the database into a matrix $\mathbf{H} =< h_{ij} >, i \in \{1, \cdots, N\}, j \in \{1, \cdots, M\}$, where N and M are the numbers of target places and features respectively. For simplicity, we focus on places belonging to a certain category (such as coffee shop, hiking trail, etc) here. SOR can certainly deal with multiple categories by using multiple such matrices.

However, data in $\mathbf{H}$ cannot be directly used for ranking since the purpose of ranking is to recommend suitable places for individual users. If ranking is done on the absolute temperature, then a very hot (or cold) place may be ranked one of top places, which is certainly not preferred by most people. Hence, our personalizable ranking algorithm will further process these values based on user preferences, which will be discussed next.

B. Personalizable Ranking Algorithm

In this section, we present a personalizable ranking algorithm based on user preferences. Our algorithm uses the same sensed data as input for all users but can produce different rankings for different users based on their preferences. The input of the algorithm includes: 1) $\mathbf{H} =< h_{ij} >, i \in \{1, \cdots, N\}, j \in \{1, \cdots, M\}$ (read from the database); 2) $\mathbf{U} =< u_j >, j \in \{1, \cdots, M\}$, where u_j is the value preferred by the user on feature j; 3) $\mathbf{W} =< w_j >, j \in \{1, \cdots, M\}$, where w_j is the weight given by the user on feature j to express his/her emphasis; We outline the proposed algorithm in the following and then explain every step in details.

Algorithm 2 Personalizable Ranking Algorithm

```
Step 1  Process $\mathbf{H} =< h_{ij} >$ further and store results
        to a new matrix $\mathbf{\Gamma} =< \gamma_{ij} >$ according to user
        preferences by $\gamma_{ij} := |h_{ij} - u_j|, i \in \{1, \cdots, N\}, j \in \{1, \cdots, M\}$;

Step 2  Sort the target places in $\mathbf{\Gamma} =< \gamma_{ij} >$ on the column
        by column basis to produce an individual ranking
        $\mathbf{R}_j$ for every feature $j$;

Step 3  Aggregate individual rankings to output a final rank-
        ing based on $\mathbf{W} =< w_j >, j \in \{1, \cdots, M\}$ using a
        min-cost flow based algorithm (described below).
```

In the first step, the algorithm calculates the distances between numbers in $\mathbf{H}$ and the values preferred by a user and then store them into another $N \times M$ matrix $\mathbf{\Gamma} =< \gamma_{ij} >$. For example, suppose that the temperature (suppose it is the jth feature) in target place i is h_{ij}, then $\gamma_{ij} := |h_{ij} - u_j|$, where u_j is the temperature preferred by the user. If the user does not input a desirable temperature, the system provides a default value, e.g. 73°F, based on common sense. Moreover, for some features (such WiFi signal strength), if it is always the larger (the smaller) the better, then a very large (small) default value is always used as the preferred value.

In the second step, for all target places belonging to a category (such as coffee shop or hiking trail), the algorithm produces a ranking $\mathbf{R}_j$ (i.e a sorted list) on each feature j by sorting all the target places in the ascending order of the corresponding feature values on the column by column basis. We call such rankings *individual rankings* in the following.

In the third step, the algorithm *aggregates* individual rankings produced (based on a single feature) in the second step to generate the final ranking. In order to do it, we need a metric measuring distance between two rankings. In SOR, the *Kemeney distance* [14], [15] is chosen for this purpose since it has been shown to have good spam resistance [7] and has been widely used for ranking in various domains such as webpage ranking, consensus and etc. Suppose that an index function $\pi(i, \mathbf{R})$ returns the index of item i (target place i in our case) in ranking $\mathbf{R}$.

Definition 2 (Kemeney Distance [14], [15]): The Kemeney distance between two rankings $\mathbf{R}_1$ and $\mathbf{R}_2$,

$$d_K(\mathbf{R}_1, \mathbf{R}_2) = \sum_{i=1}^{N} \sum_{i'=1}^{N} \mathbf{1}(\text{sgn}((\pi(i, \mathbf{R}_1) - \pi(i', \mathbf{R}_1)) * (\pi(i, \mathbf{R}_2) - \pi(i', \mathbf{R}_2))) < 0), \tag{5}$$

where $\mathbf{1}(\cdot)$ is the indicator function and sgn$(\cdot)$ is the sign function.

Intuitively, the Kemeney distance counts the number of pairwise violations between two rankings. For example, given two rankings of three items $\{A, B, C\}$:

$$\begin{aligned} \mathbf{R}_1 &: A, B, C \\ \mathbf{R}_2 &: B, C, A \end{aligned} \tag{6}$$

then the Kemeney distance between them is $d_K(\mathbf{R}_1, \mathbf{R}_2) = 2$, since there are two pairwise violations, (A, B) and (A, C).

We consider personalizable ranking, which allows users to emphasize (or de-emphasize) certain features by assigning weights to them. Let $\mathbf{\Omega}$ be a collection of M individual rankings on all features $\mathbf{\Omega} = \{\mathbf{R}_j : j \in \{1, \cdots, M\}\}$. We come up with a new metric, called *weighted K-ranking distance*, to evaluate the quality of a ranking based on user preferences.

The weighted K-ranking distance from a ranking $\mathbf{R}$ to a collection of individual rankings $\mathbf{\Omega}$ is:

$$\kappa_K(\mathbf{R}, \mathbf{\Omega}) = \sum_{j=1}^{M} w_j * d_K(\mathbf{R}, \mathbf{R}_j), \tag{7}$$

where w_j is the weight assigned to feature j by the user. The ranking problem is to find a ranking $\mathbf{R}^*$ such that its weighted ranking distance to $\mathbf{\Omega}$ is minimized among all rankings, i.e.,

$$\mathbf{R}^* = \underset{\mathbf{R}}{\operatorname{argmin}}\, \kappa_K(\mathbf{R}, \mathbf{\Omega}). \tag{8}$$

Unfortunately, as showed by [7], computing such an optimal (aggregated) ranking $\mathbf{R}^*$ is NP-hard, even for the unweighted case with $|\mathbf{\Omega}| = 4$. So we need to have an effective heuristic algorithm.

Spearman's footrule distance [6], $d_f(\cdot)$, has been widely used to approximate the Kemeney distance:

$$d_f(\mathbf{R}_1, \mathbf{R}_2) = \sum_{i=1}^{N} |\pi(i, \mathbf{R}_1) - \pi(i, \mathbf{R}_2)|, \tag{9}$$

where $\mathbf{R}_1$ and $\mathbf{R}_2$ are two rankings as discussed above. The footrule distance has the following property [6]:

$$d_K(\mathbf{R}_1, \mathbf{R}_2) \le d_f(\mathbf{R}_1, \mathbf{R}_2) \le 2 d_K(\mathbf{R}_1, \mathbf{R}_2). \tag{10}$$

Similarly, we define the *weighted f-ranking distance* as:

$$\kappa_f(\mathbf{R}, \mathbf{\Omega}) = \sum_{j=1}^{M} w_j * d_f(\mathbf{R}, \mathbf{R}_j). \tag{11}$$

Instead of solving the original ranking problem defined above, we can solve a footrule distance based ranking problem:

$$\mathbf{R}^* = \underset{\mathbf{R}}{\operatorname{argmin}}\, \kappa_f(\mathbf{R}, \mathbf{\Omega}). \tag{12}$$

It has been shown in [7] that the *unweighted* of the footrule distance based ranking problem could be transferred to a minimum cost perfect matching problem, which can be solved efficiently in polynomial time.

Next, we show that our *weighted* version can be efficiently solved by constructing an auxiliary flow graph and using a min-cost flow based algorithm. First, we construct a flow graph to assist computation $\mathbf{G}(\mathbf{V} \bigcup \mathbf{V}' \bigcup \{s, z\}, \mathbf{E})$. In this graph, each vertex $v_i \in \mathbf{V}$ corresponds to a target place i and each vertex $v_{i'} \in \mathbf{V}'$ corresponds to a rank. There is a directed edge $e \in \mathbf{E}$ from each $v_i \in \mathbf{V}$ to every $v_{i'} \in \mathbf{V}'$, whose cost is set to $\sum_{\mathbf{R}_j \in \mathbf{\Omega}} w_j * |\pi(i, \mathbf{R}_j) - i'|$ and capacity is set to 1. Note that for a target place i, the cost here basically gives the sum of distances to all individual rankings (suppose that its final rank is i'). Moreover, to complete a flow graph, we introduce a virtual source s, which has a directed edge to each $v_i \in \mathbf{V}$ with a cost of 0 and a capacity of 1. And, there is a virtual sink z, which has a directed edge coming from each $v_{i'} \in V'$ with a cost of 0 and a capacity of 1 too.

The importance of the flow graph lies in the fact that a min-cost $s - z$ flow with an amount of N on the graph gives a ranking that minimizes the weighted f-ranking distance. It is known that the min-cost flow in such a flow graph (whose link capacities are all 1) can be efficiently found by a linear programming based algorithm [1], which is guaranteed to generate an integer flow since the corresponding co-efficient matrix is totally unimodular. Moreover, it can be easily shown that the optimal solution to our footrule distance based ranking problem is a $\frac{1}{2}$-approximate solution to the original (Kemeney distance based) problem due to the property (10) described above.

V. Validation and Performance Evaluation

We validated and evaluated SOR via both field tests and simulation. Specifically, we field-tested two kinds of places, hiking trails and coffee shops, in or around the city of Syracuse; and we evaluated the performance of the proposed scheduling algorithm via simulation.

A. Field Tests for Hiking Trails

In the first sets of field tests, we collected data from three hiking trails in or around Syracuse, namely, the *Green Lake Trail* [12] (in the Green Lake State Park), the *Long Trail* and the *Cliff Trail*(both of them are in the Clark Reservation [3]) The field tests were conducted during 11:00AM-2:00PM Nov. 17, 2013. In each test, there were 7 participating mobile phones, which are all Google's Nexus4 smartphones.

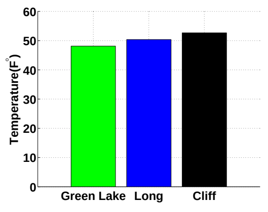
For hiking trails, we collected data of 5 sensing features that hikers usually care about most (listed below). We used the following methods to process sensed data to produce values for each feature: 1) temperature: it is an average of all temperature sensor readings; 2) humidity: it is an average of all humidity sensor readings; 3) roughness of road surface: it is an average of the standard deviations of all accelerometer's readings within Δt (a short sampling period described in Section IV); 4) curvature: it is calculated based on GPS locations using the method presented in [17]. 5) altitude change: it is the standard deviation of averages of all altitude sensor readings within Δt. The feature data are presented in Figure 6.

In order to justify effectiveness of personalizable ranking in SOR, we came up with three virtual hikers, namely, *Alice, Bob* and *Chris*, whose preferences are described using hiker profiles shown in Fig. 7. Note that a user can express his/her preferences by setting preferred feature values and weights. The weight can be set to an integer in $\{0, 1, 2, 3, 4, 5\}$ with '0' meaning that he/she doesn't care and '5' indicating he/she really cares. For example, Alice is assumed to be an experienced hiker who prefers difficult trails. So she sets all the preferred values for the roughness, curvature and altitude change to MAX (a relatively large integer pre-configured in SOR), and sets all their weights to 5. We then present the rankings of the three target hiking trails computed by SOR via mobile phone sensing for three hikers in Table I.

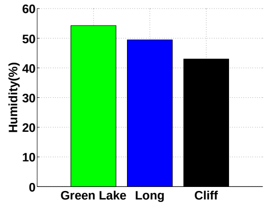
TABLE I
Rankings of Hiking Trails Computed by SOR

User	No. 1	No. 2	No. 3
Alice	Cliff Trail	Long Trail	Green Lake Trail
Bob	Long Trail	Cliff Trail	Green Lake Trail
Chris	Green Lake Trail	Long Trail	Cliff Trail

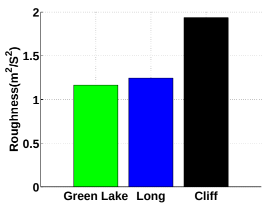
To validate these ranking results, we established the *ground truths* using pictures taken during field tests and real user comments collected from Internet via Google (mainly from *www.cnyhiking.com, www.outdoorexperiencereview.com www.hikespeak.com, nysparks.com, etc*), which are shown in Fig. 8 and Fig. 9 respectively. Note that in the table, we also summarize user comments as key opinions for quick references. From these ground truths, we can see that the Cliff Trail is rocky so it is indeed a difficult trail. The other two trails are flat and fairly easy, especially the Green Lake trail (according to a real user comment "...This trail is almost entirely flat"). In addition, the Green Lake Trail is around a lake (see its picture) so it is supposed to be humid and a little cooler. According to rankings produced for Alice (an experienced hiker who prefers difficult trails), Cliff Trail is ranked No. 1, followed by the Long Rail (which is a little more difficult than the Green Lake Trail). Similarly, for Bob (a beginner who likes dry and even trails), the Long Trail is recommended as the top choice, followed by the Cliff trail, which is difficult but drier than the Green Lake Trail. Since Bob cares more about humidity than difficulty (according to the corresponding weights), so Cliff Trail is ranked higher than Green Lake Trail. For Chris (a beginner who likes jogging near a lake/sea/river), the Green Lake Trails



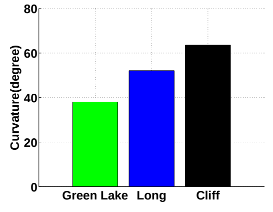
(a) Temperature



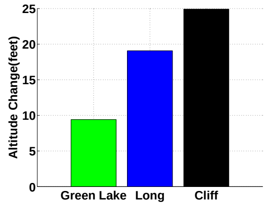
(b) Humidity



(c) Roughness of road surface

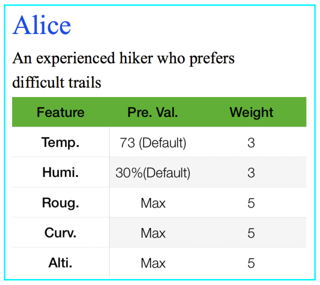


(d) Curvature



(e) Altitude change

Fig. 6. Feature data for hiking trails

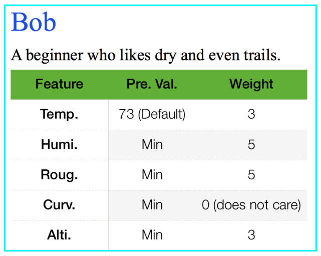


Alice

An experienced hiker who prefers difficult trails

Feature	Pre. Val.	Weight
Temp.	73 (Default)	3
Humi.	30%(Default)	3
Roug.	Max	5
Curv.	Max	5
Alti.	Max	5

(a) Alice's profile

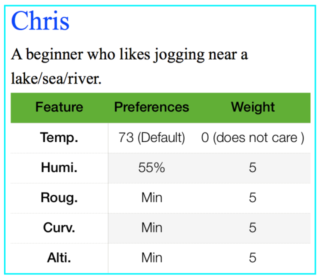


Bob

A beginner who likes dry and even trails.

Feature	Pre. Val.	Weight
Temp.	73 (Default)	3
Humi.	Min	5
Roug.	Min	5
Curv.	Min	0 (does not care)
Alti.	Min	3

(b) Bob's profile



Chris

A beginner who likes jogging near a lake/sea/river.

Feature	Preferences	Weight
Temp.	73 (Default)	0 (does not care)
Humi.	55%	5
Roug.	Min	5
Curv.	Min	5
Alti.	Min	5

(c) Chris's profile

Fig. 7. Hiker profiles

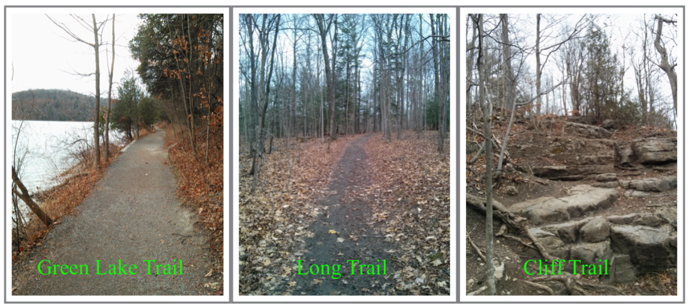


Fig. 8. Ground truth 1: pictures of the target hiking trails

	Real Users' Comments	Key Opinions
Green Lake	"…trails around the lake to run…" "…this is a great running trail…" "…trails… are wide and mostly flat …quite suitable for beginner…" "…The trail is almost entirely level…"	very flat, easy.
Long	"...wood land and meadow…" "... which runs through a meadow""...parts are easy…"	flat, easy.
Cliff	"…rocky outcrops…""...There are a couple of steep billy climbs…""This trail is extremely difficult,..."	rocky, difficult.

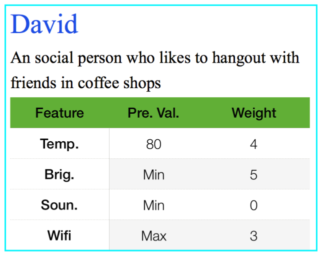
Fig. 9. Ground truth 2: real user comments for the target hiking trails

is certainly recommended as the first choice. We can conclude that data collected and processed by SOR can well capture characteristics of target places, and personalizable rankings produced by SOR can well match user preferences.

B. Field Tests for Coffee Shops

In the second sets of field tests, we collected data from three coffee shops in Syracuse, namely, the *Tim Hortons* (985 East Brighton Avenue Syracuse, NY 13205), the *Barns & Noble (B&N) Cafe* (3454 E. Erie Blvd, Syracuse, NY, 13214) and a *Starbucks* (177 Marshall St, Syracuse, NY 13210). The field tests were conducted during 11:00AM-2:00PM Nov. 15, 2013. In each test, there were 12 participating mobile phones, which are all Google's Nexus4 smartphones.

For coffee shops, we collected data of 4 sensing features that customers usually care about most: 1) temperature (temperature sensor on the Sensordrone), 2) brightness (light sensor on the Sensordrone), 3) WiFi signal strength (WiFi interface), and 4) background noise level (microphone). For all these four features, we took averages of all corresponding sensor readings. The feature data are presented in Figure 10.

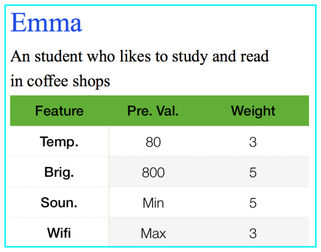


David

An social person who likes to hangout with friends in coffee shops

Feature	Pre. Val.	Weight
Temp.	80	4
Brig.	Min	5
Soun.	Min	0
Wifi	Max	3

(a) David's profile



Emma

An student who likes to study and read in coffee shops

Feature	Pre. Val.	Weight
Temp.	80	3
Brig.	800	5
Soun.	Min	5
Wifi	Max	3

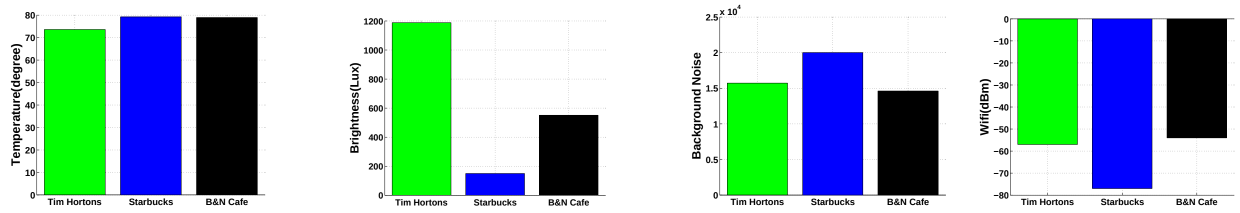
(b) Emma's profile

Fig. 11. Customer profiles

Similarly, we came up with two virtual customers, namely, *David* and *Emma*, whose preferences are described using customer profiles (with preferred values and weights) shown in Fig. 11. We then present the rankings of the three target coffee shops computed by SOR for both customers in Table II.

Again, we compare the ranking results with *ground truths*, which are pictures taken during field tests and real user comments collected from Internet via Google (mainly from Yelp and Foursquare), shown in Fig. 12 and Fig. 13 respectively.

From the ground truths, we can see that the Starbucks is crowded, noisy and dark. While the other two coffee shops



(a) Temperature (b) Brightness (c) Background noise (d) Wifi

Fig. 10. Feature data for coffee shops

TABLE II
RANKINGS OF COFFEE SHOPS COMPUTED BY SOR

User	No. 1	No. 2	No. 3
David	Starbucks	B&N Cafe	Tim Hortons
Emma	B&N Cafe	Tim Hortons	Starbucks

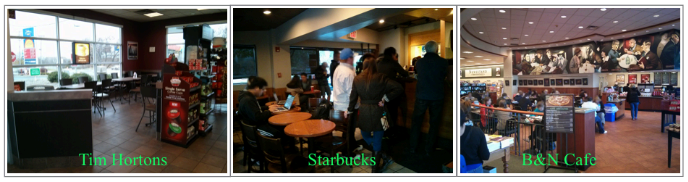


Fig. 12. Ground truth 1: pictures of the target coffee shops

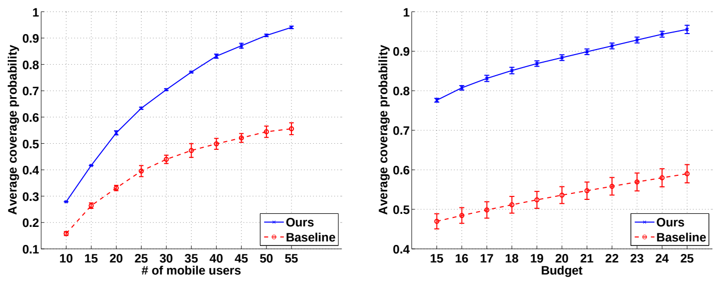
	Real Users' Comments	Key opinions
Tim Hortons	"…it is not as noisy as I expected. ..not many people…" "I felt a little bit cold when I was in the store…" "…large windows…It feels very bright…"	quiet, cold, bright.
Starbucks	"…it was a very loud theater…" "…This location is dark and kind of gloomy…" "… Always crowded, never a place to sit, long lines…" " … way too small… considering the noise level and proximity of the tables to each other…"	crowded, loud, dark.
B&N Cafe	"The in-store Starbucks has plenty of tables and is fairly quiet." "…Only place to study…" "…It is very quiet… It is the best public place for studying or reading in Syr…"	quiet, good for study.

Fig. 13. Ground truth 2: real user comments for the target coffee shops

are quiet and bright. The Tim Hortons is a little colder than the B&N Cafe, however, very bright due to a big window (See Fig. 12). David is a social person who likes to hang out with friends in coffee shops so he prefers a not-so-bright and warm place but does not really care about noise. According to the ranking produced for him, the Starbucks is ranked No. 1, followed by the B&N Cafe (since it is not as bright as the Tim Hortons). For Emma (a student who likes to read and study in relatively warm coffee shops), the B&N is recommended as the top choice, followed by the Tim Hortons which is a little colder than B&N Cafe. In the coffee shop case, we can make the same conclusion as the hiking trail case.

C. Simulation for the Sensing Scheduling Algorithm

We evaluated performance of the proposed sensing scheduling algorithm in large cases via simulation. In the simulation, the duration of sensing scheduling period was set to 3 hours, which is divided by 1080 time instants. The arrival (leaving) times of mobile users were randomly generated, following a uniform distribution between 0 (the corresponding arrival time) and 10800s. We used a bell-shaped Guassian distribution (with $\mu = 0$ and $\sigma = 10$s) to model coverage, as discussed in Section III. A simple scheduling algorithm served as the baseline: a mobile phone starts to sense every 10s since its arrival for N_k^B times, where N_k^B is the corresponding budget. The average coverage probability was used as performance metric, which is the sum of coverage probabilities (objective function) divided by the total number of time instants in the scheduling period (i.e., 1080). In the first simulation scenario, we changed the number of mobile users from 10 to 50 with a step size of 5 and the budgets of all mobile users were fixed to 17. In the second scenario, we changed the budget from 15 to 25 with a step size of 1 and the numbers of mobile users were fixed to 40. We presented the results in Fig. 14. Note that every number in the figure is an average over 10 runs.



(a) Varying # of mobile users (b) Varying budget

Fig. 14. Performance of the sensing scheduling algorithm

From the figure, we can see that on average, our scheduling algorithm outperforms the baseline algorithm by 65% in terms of average coverage probability. From Fig. 14(a), we can see that when 55 users participate in sensing, our algorithm leads to almost 100% coverage. In order to achieve an average coverage probability of 80%, our scheduling algorithm need no more than 40 users (with a budget of 17) while the baseline algorithm can only reach an average coverage probability of 50% with 40 users. Similar observations can be made from Fig. 14(b). No matter which method is used, the average coverage probability always increases with the number of mobile users and budget as expected. In addition, we observe that the variance of the coverage probability given by our scheduling algorithm is always less than that given by the baseline algorithm, which means our algorithm is more stable and is suitable for various situations.

VI. Related Work

Comprehensive reviews for mobile phone sensing systems and applications can be founded in [16] and [26]. In the following, we briefly introduce a few representative systems developed for transportation, social networking, healthcare and environment monitoring.

Nericell [20] is a mobile phone sensing based road condition and traffic monitoring system, which uses various sensors on a mobile phone to detect potholes, bumps, braking and honking. Another system is VTrack [29], which, however, tracks the traffic delays and congestions. In [5], the authors proposed a system aimed at early detection and alert of dangerous vehicle maneuvers. As its name suggests, Micro-blogs [9] is a mobile phone sensing system developed for social networks. CenceMe, developed by Miluzzo *et al.* [19], represents the first system that combines the inference of the presence of individuals using sensor-enabled mobile phones with sharing of this information through social networks. A healthcare system, UbiFit Garden [2], infers people's activities throughout everyday life via mobile phone sensing. In [4], Cui *et al.* studied the environment impact of air temperature on thermal comfort, motivation, performance and their relationship.

PEIR (Personal Environmental Impact Report) [21] is a mobile phone sensing application that uses location data sampled from phones everyday to calculate personalized estimates of environmental impact and exposure. Another example is the ear-phone [24] system for noise monitoring.

Sensing scheduling and coordination have been studied in the context of mobile phone sensing recently. A protocol, Aquiba [28], was proposed to exploit opportunistic collaboration of pedestrians for mobile phone sensing. In [30], several mechanisms were introduced for automated mapping of urban areas that provide a virtual sensor abstraction to applications. Spatial and temporal coverage metrics were also presented for measuring the quality of acquired data. In a recent work [25], Sheng *et al.* presented optimal algorithms and practical heuristic algorithms for energy-efficient collaborative sensing scheduling problems.

To the best of our knowledge, we are the first to build a system to leverage mobile phone sensing for ranking and recommendation.

VII. Conclusions

In this paper, we presented design, implementation and evaluation of SOR. SOR is easy to use and its architecture is so scalable that various sensors can be easily integrated into it. We presented an online scheduling algorithm for coverage maximization, which has a constant approximation ratio of $1/2$; moreover, we presented a personalizable ranking algorithm, which ranks target places based on various sensor readings and user preferences. Both of them have been used in SOR for scheduling and ranking respectively. We validated and evaluated SOR via both field tests (using real hiking trails and coffee shops in Syracuse as target places) and simulation. Field-testing results showed that data collected and processed by SOR can well capture characteristics of target places, and personalizable rankings produced by SOR can well match user preferences. In addition, simulation results showed that the proposed scheduling algorithm outperforms a baseline algorithm by 65% in terms of average coverage probability.

References

[1] R. K. Ahuja, T. L. Magnanti and J. B. Orlin, Network flows: theory, algorithms, and applications, Prentice Hall, 1993.

[2] S. Consolvo *et al.* , Activity sensing in the wild: a field trial of ubifit garden, *ACM SIGCHI'2008*, pp. 1797–1806.

[3] Clark reservation, http://nysparks.com/parks/126/details.aspx

[4] W. Cui, G. Cao, J. Park,and Q. Ouyang and Y. Zhu, Influence of indoor air temperature on human thermal comfort, motivation and performance *Building and Environment*, Vol. 68, 2013 pp. 1144–122.

[5] J. Dai, J. Teng, X. Bai, Z. Shen and D. Xuan, Mobile phone based drunk driving detection, *IEEE PervasiveHealth'10*.

[6] P. Diaconis and R. L. Graham, Spearman's footrule as a measure of disarray, *Journal of the Royal Statistical Society*, Series B, pp. 262–268, 1977.

[7] C. Dwork, R. Kumar, M. Naor and D. Sivakumar, Rank aggregation methods for the web, *WWW'2001*, pp. 613–622.

[8] Fitbit, *http://www.fitbit.com*.

[9] S. Gaonkar *et al.* , Micro-blog: sharing and querying content through mobile phones and social participation, *ACM MobiSys'2008*, pp. 174–186.

[10] L. Gargano and M. Hammar, A note on submodular set cover on matroids, *Discrete Mathematics*, Vol. 309, No. 18, 2009, pp. 5739–5744.

[11] Google Glass, *http://www.google.com/glass/start/*.

[12] Green lake state park, http://nysparks.com/parks/172.

[13] E. Koukoumidis, L. S. Peh and M. R. Martonosi, SignalGuru: leveraging mobile phones for collaborative traffic signal schedule advisory, *ACM MobiSys'2011*, pp. 127–140.

[14] J. Kemeny, Mathematics without numbers, *Daedalus'88*, pp. 571-91.

[15] J. Kemeny and S. Lawrence, Mathematical models in the social sciences. *Boston:Ginn*, 1960.

[16] N. D. Lane *et al.* , A survey of mobile phone sensing, *IEEE Communications Magazine*, Vol. 48, No. 9, 2010, pp. 140–150.

[17] L. Li, X. Zhao and G. Xue, Unobservable re-authentication for smartphones, *NDSS'2013*, pp. 24–27.

[18] Lua, *http://www.lua.org/*.

[19] E. Miluzzo *et al.* , Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application, *ACM SenSys'2008*, pp. 337–350.

[20] P. Mohan, V. N. Padmanabhan and R. Ramjee, Nericell: rich monitoring of road and traffic conditions using mobile smartphones, *ACM SenSys'2008*, pp. 323–336.

[21] M. Mun *et al.* , PEIR, the personal environmental impact report, as a platform for participatory sensing systems research, *ACM Mobisys'2009*, pp 55–68.

[22] E. Ngai and J. Xiong, Adaptive collaborative sensing using mobile phones and stationary sensors, *IEEE/IFIP DSN-W'2008*, pp. 280–285.

[23] PostgreSQL, *http://www.postgresql.org/*.

[24] R. K. Rana *et al.* , Ear-phone: an end-to-end participatory urban noise mapping system, *ACM IPSN'2010*, pp. 105–116.

[25] X. Sheng, J. Tang and W. Zhang, Energy-efficient collaborative sensing with mobile phones, *IEEE Infocom'2012*, pp. 1916–1924.

[26] X. Sheng, J. Tang, X. Xiao and Guoliang Xue, Sensing as a Service: Challenges, Solutions and Future Directions, *IEEE Sensors Journal*, Vol. 13, No. 10, pp. 3733–3741

[27] Sensordrone, *http://sensorcon.com/sensordrone/*.

[28] N. Thepvilojanapong, S. I. Konomi, Y. Tobe, Y. Ohta, M. Iwai and K. Sezaki, Opportunistic collaboration in participatory sensing environments, *ACM MobiArch'2010*, pp. 39–44.

[29] A. Thiagarajan *et al.* , VTrack: accurate, energy-aware road traffic delay estimation using mobile phones, *ACM Sensys'2009*, pp. 85–98.

[30] H. Weinschrott, F. Durr and K. Rothermel, StreamShaper: coordination algorithms for participatory mobile urban sensing, *IEEE MASS'2010*, pp. 195–204.

[31] D. Yang, X. Fang and G. Xue, ESPN: Efficient server placement in probabilistic networks with budget constraint, *IEEE Infocom'2011*, pp. 1269–1277.

[32] R. Zhang, J. Shi, Y. Zhang, and C. Zhang, Verifiable privacy -preserving aggregation in people-centric urban sensing systems, *IEEE Journal of Selected Areas on Communications*, Vol. 30, No.9, 2013, pp. 268–278.

[33] X. Zhu, Q. Li and G. Chen, APT: Accurate Outdoor Pedestrian Tracking with Smartphones, *IEEE Infocom'13*, pp. 2508–2516.