Utility-driven Data Acquisition in Participatory Sensing

Mehdi Riahi, Thanasis G. Papaioannou, Immanuel Trummer and Karl Aberer
School of Computer and Communication Sciences
École Polytechnique Fédérale de Lausanne (EPFL)
1015 Lausanne, Switzerland
Email: firstname.lastname@epfl.ch

ABSTRACT

Participatory sensing (PS) is becoming a popular data acquisition means for interesting emerging applications. However, as data queries from these applications increase, the sustainability of this platform for multiple concurrent applications is at stake. In this paper[1], we consider the problem of efficient data acquisition in PS when queries of different types come from different applications. We effectively deal with the issues related to resource constraints, user privacy, data reliability, and uncontrolled mobility. We formulate the problem as multi-query optimization and propose efficient heuristics for its effective solution for the various query types and mixes that enable sustainable sensing. Based on simulations with real and artificial data traces, we found that our heuristic algorithms outperform baseline approaches in a multitude of settings considered.

Categories and Subject Descriptors

C.3 [**Special-purpose and Application-based Systems**]: Participatory Sensing; H.3.3 [**Information Storage and Retrieval**]: Sensor Data Acquisition

General Terms

ALGORITHMS, EXPERIMENTATION

Keywords

sustainability, sensor data sharing, query mix

1. INTRODUCTION

Participatory sensing (PS) is becoming a popular paradigm for collecting and sharing data about phenomena of social interest, such as air quality, well-being, traffic, etc. Even though some people might altruistically participate in such data collection systems, we believe that enough incentives must be provided to people to encourage more participation. The burden that participation imposes on the participants, e.g. battery and network consumption and privacy leakage, should be compensated to guarantee long-term *sustainability* of the system. Moreover, in a popular PS environment, there can be many users/applications that are interested in the data being collected and pose different types of queries, instant or continuous ones. At the same time, some of the users may participate in the sensor data collection. Such PS system can be envisioned by introducing some sort of incentives, e.g. payments from the querying user, to the users from whom the data for the query is collected. It is critical for the sustainability of the system to provide to the users as much utility as possible. In this context, utility is defined as the difference between the value of the query results to the users and the price they pay for obtaining the results.

There exists a large body of work in the area of sensor data acquisition, which either have a single application-specific objective, e.g. achieving complete coverage of the sensing field [16], or assume certain structures for the utility functions, e.g. submodularity as in [1, 15, 10, 9]. Similarly, there is a large body of work in the context of multi-query optimization in sensor networks and in stream processing systems, e.g., [18, 11, 17]. However, the existing approaches cannot be directly applied to the context of PS for the following reasons: 1) because of the uncontrolled mobility of the participants, the query processor needs to deal with data unavailability; and 2) there is a lack of sophisticated utility considerations in the existing work.

The original contributions of this paper are the following:

1. We propose a data acquisition framework in the context of PS that takes into account the factors pertinent to this context and efficiently shares sensor data among queries of different types, so as to enable sustainability. Queries for sensor data come from multiple different applications or users that can have any arbitrary utility considerations.
2. We formulate the optimal data acquisition problem as a multi-query optimization with the objective of maximizing the total utility (or *social welfare*) and propose efficient heuristic solutions for various query types and query mixes.
3. Important query categories, including one-shot and continuous queries, in the context of PS are considered and efficient data acquisition algorithms are proposed for each query type as well as the combination of different query types.
4. We verify the effectiveness of our approach through extensive simulations on real and synthetic data traces.

[1]Partially supported by the EU project OpenIoT (ICT 287305).

The remainder of the paper is organized as follows. In Section 2, we introduce our context and formally define the problem. We present heuristic algorithms for sensor scheduling in Section 3 and evaluate those algorithms experimentally in Section 4. We review the related work in Section 5 and finally we conclude our paper in Section 6.

2. THE CONTEXT

In a PS system several participants carrying heterogeneous sensing devices move in a certain region. The sensing devices communicate with a server, which is called the *aggregator*. Sensing devices take a measurement only when they are selected by the aggregator to do so. Participants ask for a payment for each measurement they provide. Each sensor has a specific sensing range. Each measurement includes a sensor-specific inherent inaccuracy. In this paper, we use the term sensor to refer to the actual sensor on the sensing device, the sensing device, or even the combination of the participant and the sensing device she carries.

End users (or applications) submit queries to the aggregator. The aggregator periodically collects the queries and tries to optimally answer them. Our optimization objective is to maximize the overall utility (or *social welfare*), since this objective matches our requirement for sustainable operation of the system, as opposed to data value maximization or cost minimization. Alternatively, an egalitarian approach could be followed, where the number of users with positive utility is maximized. Utility maximization can be achieved by selecting appropriate sensors for providing measurements, considering the value of the measurements to the queries, the cost of obtaining such measurements, and exploiting possible common data requirements among queries. In a PS context with diverse set of end users who have different criteria for evaluating the quality of query results, the aggregator relies on the end users to provide a valuation function, $v_q(.)$, with each query q. This function returns the value, in real or virtual currency, of a set of measurements that can provide the answer to the query based on the quality of the measurements. Users have a limited budget to spend for obtaining query answers.

Queries issued by end users can fall into two major categories, namely *one-shot queries* and *continuous queries*. One-shot queries are executed only once, while continuous queries are continuously evaluated. Major one-shot queries in the PS context are *point queries*, *spatial aggregate queries over a region*, and *queries over trajectories*. Continuous queries can be split into two sub-categories of *monitoring queries* and *event detection queries*. *Single-sensor* queries only need one sensor reading while *multi-sensor* queries need multiple sensor readings. Figure 1 shows these categories and the query types that we handle explicitly in this paper. Each query category is explained in more details later in this section. Table 1 summarizes our notation.

2.1 Problem Formulation

We assume, without loss of generality, that the system runs for a period of T, e.g., from 6 a.m. to 9 p.m. in a day. This period is discretized into several time slots of fixed length, e.g., 5 minutes. All the sensors communicate with a unique aggregator and if necessary, at the beginning of each time slot announce their location and price of providing a measurement at that location.

The objective is to acquire data for the queries from the

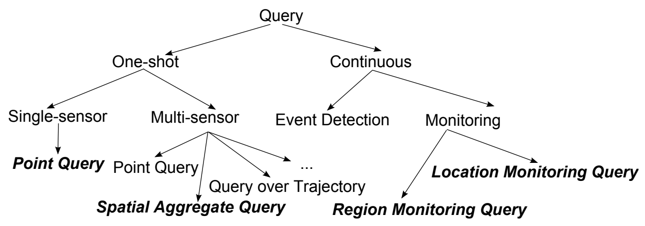


Figure 1: Query categories in the PS context. The query types in boldface are explicitly handled in this paper.

available sensors in order to maximize the utility over T. Formally, we let $\mathcal{Q}$ denote the set of all queries issued from time 1 to T, $\mathcal{S}^t$ denote the set of available sensors at time slot t, and $K : \mathcal{Q} \to \times_{t=1}^{T} 2^{\mathcal{S}^t}$ define an allocation scheme that assigns sensors to each query. $Y(K,t)$ is a function that returns the set of sensors that are assigned to all queries at time t. We denote by $c_s(K,t)$, the cost of sensor s at time t given the allocation K. Let $\mathcal{K}$ denote the set of all possible allocation schemes. The goal is to find allocation $K^* \in \mathcal{K}$ that maximizes the *social welfare*:

$$K^* = \underset{K \in \mathcal{K}}{\arg\max} \big(\sum_{q \in \mathcal{Q}} v_q(K(q)) - \sum_{t=1}^{T} \sum_{s \in Y(K,t)} c_s(K,t) \big). \quad (1)$$

For solving the above problem we need to know in advance all the queries that will be issued over T, and the location and cost of all the sensors at each time slot. However, in a PS system, users must be able to submit new queries whenever they desire and it is not realistic to ask the users to pose all their queries in the beginning of T. Due to the uncontrolled mobility of the sensors, their exact locations at a specific time slot cannot be determined a priori. Moreover, the cost of a sensor might vary from one time slot to another based on the preferences of the sensor owner. Due to the lack of access to all the required information to solve the long-term optimization problem 1, we resort to a *myopic* approach, in which we try to maximize the utility at the current time slot without considering the future state of the system. This approach would be further motivated in a "hotspot" monitoring setting: Consider a hotspot area, e.g. the downtown, of a city where users carrying smart phones continuously enter and exit, and roam around while they are inside it. In this case, the mix of available sensors in the hotspot area dynamically changes and short-term optimization towards monitoring sustainability becomes more important.

Let Q denote the set of all queries available at the current time slot t, which can include one-shot queries issued for time t and continuous queries that started before or at t. Let S be the set of available sensors at t and c_s denote the reported cost of each sensor s. Let $M : Q \to 2^S$ define an allocation scheme that assigns sensors to each query. $Y(M)$ is a function that returns the set of sensors assigned to queries. Let $\mathcal{M}$ denote the set of all possible allocation schemes. The goal is to find allocation $M^* \in \mathcal{M}$ that maximizes the total utility in the current time slot:

$$M^* = \underset{M \in \mathcal{M}}{\arg\max} \big(\sum_{q \in Q} v_q(M(q)) - \sum_{s \in Y(M)} c_s \big). \quad (2)$$

After finding the best allocation scheme, the cost of each selected sensor s is shared among queries that are answered using the measurement from s. We denote by $\pi_{s,q}$ the amount that query q pays for using data from sensor s.

We must ensure that for each selected sensor s, the total payment from the queries using that sensor is equal to c_s. Moreover, for each query q, which is answered using sensors S_q, its utility must be positive, i.e., $v_q(S_q) - \sum_{s \in S_q} \pi_{q,s} > 0$.

2.2 One-shot Queries

We can distinguish between the queries that only need data from one sensor and queries that ask for several sensor readings. More specifically, spatial aggregate queries and queries over trajectories always require several sensor readings, whereas there exist some point queries that ask for only one sensor reading and some point queries that ask for more than one sensor reading. The former type of point queries is referred to as *single-sensor point queries* and the latter is referred to as *multiple-sensor point queries*. The reason for this distinction is that single-sensor queries can be treated more efficiently due to their special characteristics.

2.2.1 Point Queries

A user who is interested in knowing the value of a phenomenon at a certain location, submits a point query at that location to the system. The queries are required to come with a quality valuation function to valuate the quality of the sensor readings. Generally, the value of a sensor reading for an application is a function of the quality of that sensor reading and the quality of the sensor readings obtained so far. The number of samples required for finding the value of a phenomenon depends on the phenomenon itself and the trustworthiness of the sensors. For example, it might be necessary to take redundant measurements to assess the trustworthiness of a particular sensor that can be used for providing the measurements. For instance, a single-sensor point query q might have the following valuation function:

$$v_q(s) = \begin{cases} B_q \theta_{q,s} & \theta^q_{min} \leq \theta_{q,s} \leq 1, \\ 0 & \theta_{q,s} < \theta^q_{min}, \end{cases} \tag{3}$$

where $0 \leq \theta_{q,s} \leq 1$ is the quality of the sensor reading, θ^q_{min} is the minimum acceptable quality by the query, and B_q is the query budget. This implies that the user is willing to pay B_q for a sensor reading with the highest possible quality.

The quality of a sensor reading depends on the distance of the sensor from the queried location (more accurately, it depends on the correlation between the phenomenon value at the queried location and the location of the sensor,) the inherent sensing inaccuracy, and the trustworthiness of the sensor. We assume that this dependency is given by a user-defined function $\theta_q(s, l_q)$, where l_q is the queried location. The following is an example of such a function:

$$\theta_q(s, l_q) = \begin{cases} (1 - \gamma_s)(1 - \frac{|l_s - l_q|}{d_{max}})\tau_s & \text{if } |l_s - l_q| \leq d_{max} \\ 0 & \text{otherwise,} \end{cases} \tag{4}$$

where γ_s is the inaccuracy of s measured in percentage of the value range of the sensor, $0 \leq \tau_s \leq 1$ is the trustworthiness of s, l_s is the current location of s, and d_{max} is the maximum distance in which the sensors can be considered to provide data. Hereafter, we assume the same function for all queries and we only use θ_s when l_q is implied by the context.

In the case of multiple-sensor point queries, the querying application is requested to provide a more general valuation function $v_q(S)$, that takes as input a set of sensors and determines their value to the query.

2.2.2 Spatial Aggregate Queries

When issuing spatial aggregate queries, applications are interested in an aggregate value of the measurements (e.g. average, min, and max) over a region. Users assign a budget B_q to each query q and spend it based on their valuation of the quality of the result. The quality of an aggregate query answer depends on the qualities of the sensor readings used for providing the answer as well as the coverage of these readings. The application provides, along with the query q, a function $v_q(S_q)$ that evaluates the quality of the result. S_q denotes the set of selected sensors for answering query q. The following is an example of such a function:

$$v_q(S_q) = B_q \mathcal{G}_q(S_q) \frac{\sum_{s \in S_q} \theta_s}{|S_q|}, \tag{5}$$

where $\mathcal{G}_q$ is a function that calculates the coverage of the selected sensors. A simple coverage function can calculate the fraction of the area covered by the sensors, while a more general function might also take into account the dispersion or the importance of the locations that are covered by the selected sensors.

2.2.3 Queries over Trajectories

When a user issues a query over a trajectory, she would like to know the (aggregate) value of a phenomenon over that trajectory. For instance, a user might be interested in knowing the current maximum value of CO_2 in the way from her house to her work. This type of query can be treated as a special case of spatial aggregate query in which instead of providing a region of interest, a trajectory is specified.

2.3 Continuous Queries

Continuous queries are queries that are continuously executed for a certain time period or until they are removed by the users. In general, two categories of continuous queries can be distinguished: 1) *monitoring queries* that ask for continuously monitoring a phenomenon at a certain location or area, and 2) *event detection queries* that ask for monitoring a location or region for detecting the occurrence of an event. In the following example queries, Q1 and Q2 are monitoring queries and Q3 and Q4 are event detection queries.

Q1: `Monitor` CO_2 `level at location` l `in the period` $[t_1, t_2]$.
Q2: `Monitor` CO_2 `level in region` r `in the period` $[t_1, t_2]$.
Q3: `Notify me when` $CO_2 > x$ `with` $confidence > \alpha$ `at location` l `in the period` $[t_1, t_2]$.
Q4: `Notify me when` $avg(CO_2) > x$ `with` $confidence > \alpha$ `in region` R `in the period` $[t_1, t_2]$.

For queries similar to Q1, which are referred to as *location monitoring queries*, applications are requested to provide the desired sampling times $\mathcal{T}$, as well as a valuation function $v_q(\mathcal{T}')$, which returns the value of sampled times $\mathcal{T}'$. Since the locations of sensors, rather unpredictably, change over time, satisfying all the desired sampling times cannot be guaranteed. On the other hand, it is likely that a sensor moves close to a queried location at time $t' \notin \mathcal{T}$. Taking a measurement at these time instances, especially when the sensor can be shared with other queries, can increase the utility of the query at hand. In Section 3.3 we propose an approach to answering location monitoring queries with the objective of increasing the utility of the queries.

In the case of queries similar to Q2, which we refer to them as *region monitoring queries*, we rely on the querying applications to provide their desired sampling points (i.e., sampling locations and times), as well as a valuation function $v_q(.)$, which calculates the value of the measurements (taken at any sampling points). As for location monitoring queries, it might not be possible to satisfy all the desired sampling points. Also, there are opportunities to use other sampling points considering the sharing possibilities with other queries. In Section 3.3 we introduce an approach for answering region monitoring queries considering the opportunistic nature of PS.

In this paper, we don't specifically deal with event detection queries. However, we believe that data acquisition for this type of continuous queries is very similar to data acquisition for monitoring queries. The main difference is that redundant sampling might be needed to ensure the confidence requested by the queries.

2.3.1 *Example Valuation Function for Region Monitoring Queries*

One common approach for finding the valuation of a set of sensors for an application is to use the notion of *expected reduction in variance* [9, 2]. In this approach the phenomenon is modeled as a Gaussian process. Let $\mathcal{V}$ be the set of locations at which a measurement can be performed, i.e., there exists at least one sensor at each of these locations. The state of the phenomenon can be modeled using a set of random variables $\mathcal{X}_{\mathcal{V}}$. Assume, for the moment, that the goal is to select a subset $\mathcal{A} \subseteq \mathcal{V}$ of the locations to maximize the sensing quality $F(\mathcal{A})$ while the budget constraint is satisfied. The value of the phenomenon at the unobserved locations are then predicted based on the process model given the observed locations. The expected reduction in variance at the unobserved locations can be used to measure the quality of sensing if the set $\mathcal{A}$ of locations are selected to take measurements from. This quantity is given by:

$$F(\mathcal{A}) = Var(\mathcal{X}_{\mathcal{V}}) - \int P(\mathbf{x}_{\mathcal{A}}) Var(\mathcal{X}_{\mathcal{V}} | \mathcal{X}_{\mathcal{A}} = \mathbf{x}_{\mathcal{A}}) d\mathbf{x}_{\mathcal{A}}, \quad (6)$$

where $\mathbf{x}_{\mathcal{A}}$ is the measurements observed at locations $\mathcal{A}$. The following valuation function can be used for region monitoring queries:

$$v_q(S) = B_q \cdot F(S) \cdot \frac{\sum_{s \in S} \theta_s}{|S|}, \quad (7)$$

where S is the set of sensors (and their locations). Notice that in the above modeling, the assumption is that the phenomenon is a spatial process. In order to expand the approach for spatio-temporal phenomena, one needs to add a time dimension to the random variables.

2.4 Costs

Sensor owners participate in the system as long as the resource consumption on their devices as well as their location privacy loss are compensated. In this regard, each sensor asks for a certain price in return for providing a measurement to the aggregator. Therefore, the cost of obtaining a measurement from sensor s which is located at l_s, consists of two components as demonstrated in the following equation:

$$c_s(\mathcal{E}_s, H_s, l_s) = c_s^e(\mathcal{E}_s) + c_s^p(p_s(H_s, l_s)), \quad (8)$$

where $\mathcal{E}_s$ is the remaining energy, and H_s is the history of revealed locations of s. c_s^e is a function that gives the energy cost of taking a measurement and transmitting it to the aggregator, and c_s^p is a function that calculates the cost of the sensor's privacy loss due to revealing its location. The privacy loss is computed by the function p_s. We do not impose any restrictions on the form of these two functions.

Symbol	Semantic
$\theta_{q,s}$	quality of readings from sensor s for query q (in $[0,1]$)
τ_s	trustworthiness of sensor s (in $[0,1]$)
γ_s	inaccuracy of sensor s (in $[0,1]$)
B_q	budget for query q
v_q	utility function for query q
$\mathcal{E}_s$	remaining energy for sensor s
H_s	history of revealed locations for sensor s
l_s	location of sensor s
l_q	location queried by query q
$c_s/c_s^e/c_s^p$	total/energy/privacy cost for sensor s
T/t	total considered time period/specific time slot
$\mathcal{Q}/Q$	all queries in T/ in the current time slot
$\mathcal{K}/K$	set of all possible allocation schemes/specific allocation scheme
$\mathcal{M}/M$	set of all possible allocation schemes/specific allocation scheme in one time slot
$\mathcal{T}/\mathcal{T}'$	set of desired sampling times/set of sampled times for a location monitoring query
$\pi_{q,s}$	the payment of query q to sensor s

Table 1: Summary of introduced symbols

3. OUR DATA ACQUISITION APPROACH

In this section we describe our approach to the problem of utility-driven data acquisition for a mixture of queries of different types. We first introduce data acquisition for each query type. Data acquisition for the query mix, which is based on the data acquisition algorithms for individual query types, is explained in the end of this section.

3.1 Single-Sensor Point Queries

We present two algorithms for answering single-sensor point queries. The first one finds the optimal solution but does not scale to large problem instances. The second one is an efficient heuristic approximation.

3.1.1 *Optimal Scheduling*

When there exist only point queries in the current time slot, we can express the optimized sensor allocation problem as a Binary Integer Linear Program (BILP). Assume n sensors are available and L locations are queried. For each queried location l, by m_l queries, we define a binary variable $Y_i^l \in \{0,1\}$ for each $i = 1, \ldots, n$, which states whether sensor i is assigned to location l or not. For each sensor i, let $X_i \in \{0,1\}$ denote whether sensor i is assigned to any location or not. We denote by c_i the cost of sensor i. The following BILP solves the problem of optimally assigning sensors to answer single-sensor point queries:

$$\begin{aligned} &\max \sum_{l=1}^{L} \sum_{i=1}^{n} v_l'(s_i) Y_i^l - \sum_{i=1}^{n} c_i X_i, \\ &\text{s.t. } Y_i^l \leq X_i \quad \forall i, l, \quad \text{and} \quad \sum_{i=1}^{n} Y_i^l \leq 1 \quad \forall l. \end{aligned} \quad (9)$$

In the above formula $v'_l(s_i)$ is defined as:

$$v'_l(s_i) = \begin{cases} v_l(s_i) & \text{if } v_l(s_i) > 0 \\ -1 & \text{otherwise,} \end{cases} \tag{10}$$

where $v_l(s_i) = \sum_{q \in Q_l} v_q(s_i)$ in which Q_l is the set of queries at location l.

We split the sensor cost among queries proportionally to the value it yields to each query (*proportionate cost allocation.*) In other words, if $Y_s^l = 1$, then the user who has issued the query $q^* \in Q_l$ has to pay according to the following:

$$\pi_{q^*,s} = \frac{v_{q^*}(s) \cdot c_s}{\sum_{a=1}^{L} v'_a(s) Y_s^a}. \tag{11}$$

This cost sharing scheme ensures that each query receives a positive net benefit because a sensor is selected only if the total valuations it yields is greater than its cost. It follows that $\pi_{q,s} < v_q(s)$ for each q that is answered by sensor s.

3.1.2 Heuristic Scheduling

Instances of the optimization problem (9) can be solved optimally by an ILP solver as long as the input size is not very large. When the input size is very large, we can resort to an approximation algorithm. We define the utility function $u : 2^S \to \mathcal{R}$ as the following:

$$u(S') = \sum_{l \in L} \max_{s \in S'} v_l(s) - \sum_{s \in S'} c_s, \tag{12}$$

where $S' \subseteq S$. Then the optimal sensor allocation problem reduces to finding $S^* \subseteq S$ such that $S^* = \arg\max_{S' \subseteq S} u(S')$. Once the optimal set of sensors is determined, each sensor is assigned to a query location for which it yields the best valuation compared to other sensors. It can be shown that $u(.)$ is a (non-monotone) submodular function.

A $\frac{1}{3}$-approximation algorithm for non-monotone submodular functions, referred to as *Local Search* algorithm, is proposed in [3], which works as follows. It starts by adding the sensor which maximizes the utility function to the set of selected sensors W. Then it iteratively adds to W those sensors that increase the utility more than a certain threshold. In the next step, it removes from W the sensors that have become obsolete and then goes to the previous step. These steps are repeated until no obsolete sensors are found. If $u(W) \geq u(S \backslash W)$, then the set W is returned, otherwise $S \backslash W$ is returned as the set of selected sensors. This algorithm requires at most $O(n^3 \log n)$ calls to the utility function, where n is the number of available sensors. It is worthwhile to mention that a randomized local search algorithm is also proposed in [3], which achieves a $\frac{2}{5}$-approximation of the optimal solution. However, in our experiments we only use the Local Search algorithm.

3.2 Multiple-Sensor One-shot Queries

There exist queries which ask for measurements from more than one sensor. These queries include, but not limited to, *spatial aggregate queries*, *queries over trajectories*, and *multiple-sensor point queries.* At each time slot several of these queries can arrive to the aggregator. Many of them require data about the same phenomenon over different (potentially overlapping) regions. In order to maximize the overall utility, the aggregator must exploit as much as possible the common data requirements among these queries and select the best set of sensors that provide the required data. The problem of finding the optimal set of sensors is a combinatorial problem, since we have to enumerate all possible sensor assignments to queries and select the one that maximizes the overall net benefit. Therefore, we propose a greedy approach, presented in Algorithm 1, that iteratively selects sensors that maximize the partial overall utility.

The objective is to maximize the following utility function:

$$u(S') = \sum_{q \in Q} v_q(S') - \sum_{s \in S'} c_s, \tag{13}$$

where S' is the set of selected sensors, and Q is the set of queries. When all v_q's are submodular, it can be shown that $u(.)$ is also a submodular (non-monotone) function. While the algorithms proposed in [3] have proven performance guarantees for submodular functions, it is shown that the greedy algorithm can perform arbitrarily badly compared to the optimal solution. However, because the valuation functions are taken as black boxes, we use Algorithm 1 unless we have knowledge about submodularity of the valuation functions. In this case, we can adapt the aforementioned algorithms for non-monotone submodular function. The reason behind using Algorithm 1 instead of always utilizing the adapted approximate algorithms in [3] is that when the utility functions are not submodular, experimentally the former performs better in terms of total utility and it's also faster. It is worth mentioning that, for example, function (5) is not submodular, even though it is known that the coverage function is submodular. Involving sensor quality in evaluation of a set of sensors destroys the submodularity of the function.

Algorithm 1: Greedy Sensor Selection

```
Data: Set $Q$ of queries, $S$ of available sensors, and
      quality valuation function $v_q$ of each query $q$.
Result: $S \backslash \tilde{S}$ is the set of selected sensors.
1  $\tilde{S} \leftarrow S$
2  $\forall q \in Q, S_q \leftarrow \emptyset$
3  while $\tilde{S} \neq \emptyset$ do
4      $\forall s \in \tilde{S}, Q_s \leftarrow \emptyset$
5      foreach $s \in \tilde{S}$ and $q \in Q_{l_s}$ do
6          $\delta v_{q,s} \leftarrow v_q(S_q \cup \{s\}) - v_q(S_q)$
7          if $\delta v_{q,s} > 0$ then $Q_s \leftarrow Q_s \cup \{q\}$
8      $a \leftarrow \arg\max_{s \in \tilde{S}} \sum_{q \in Q_s} \delta v_{q,s} - c_s$
9      if $\sum_{q \in Q_a} \delta v_{q,a} - c_a > 0$ then
10         $\forall q \in Q_a; S_q \leftarrow S_q \cup a;\ \pi_{q,a} \leftarrow \frac{\delta v_{q,a} \cdot c_a}{\sum_{q \in Q_a} \delta v_{q,a}}$
11         $\tilde{S} \leftarrow \tilde{S} \backslash a$
12     else Leave the while loop
```

THEOREM 1. *Let $S' = S \backslash \tilde{S}$ denote the set of selected sensors after Algorithm 1 terminates. Let $S_q = S_q^{(m)} = \{s_1, s_2, ..., s_m\}$ be the set of selected sensors for query q, where m is the number of these sensors. We have the following properties:*

1. $\sum_{s \in S_q} \delta v_{q,s} = v_q(S_q), \ \ \forall q \in Q.$
2. *If $S' \neq \emptyset$, then $\sum_{q \in Q} v_q(S_q) > \sum_{s \in W} c_s$, that is, the total utility is positive.*
3. *$v_q(S_q) > \sum_{s \in S_q} \pi_{q,s}, \ \ \forall q \in Q$, that is, the individual utility is not negative.*
4. *The algorithm requires $O(|Q||S|^2)$ calls to the valuation functions.*

PROOF. The first property is proved using the definition of $\delta_{q,s}$, the partial utility of a sensor s for a query q:

$$\sum_{s \in S_q} \delta v_{q,s} = \sum_{i=1}^{m} \delta v_{q,s_i} = \sum_{i=1}^{m} v_q(S_q^{(i-1)} \cup \{s_i\}) - v_q(S_q^{(i-1)})$$
$$= \sum_{i=1}^{m} v_q(S_q^{(i)}) - v_q(S_q^{(i-1)}) = v_q(S_q^{(m)}) - v_q(S_q^{(0)})$$
$$= v_q(S_q^{(m)}) = v_q(S_q).$$

The second property can be easily proved by using property 1 and the fact that the algorithm ensures $\sum_{q \in Q} \delta v_{q,s} - c_s > 0$ for each selected sensor s. The proof of the third property is straightforward in the same way as for property 2 and by using the definition of proportionate cost allocation. The algorithm goes through the sensors in $\tilde{S}$ in every iteration (at most $|S|$ iterations) and this continues until $\tilde{S}$ becomes empty. In each iteration all queries are considered. Therefore, the time complexity of Algorithm 1 is $O(|Q||S|^2)$. $\square$

3.3 Continuous Queries

We propose Algorithm 2 for providing the required data for a set of location monitoring queries. Each query q continuously needs the value of a phenomenon at location $q.l$ in the time period $q.t_1$ to $q.t_2$. The desired sampling times of query q is denoted by $q.\mathcal{T}$. The main objective of the algorithm is to obtain a measurement for each $t \in q.\mathcal{T}$. However, because of the uncertainty in succeeding to satisfy all the desired sampling times, we also follow an opportunistic approach to obtain measurements at all $t' \notin q.\mathcal{T}$.

Function CreatePointQuery(t, q)

```
Data: $t$ is the current time and $q$ is the query
Result: A point query for query $q$ at time $t$
1 if $t = q.t_1$ then
2     $q.\mathcal{T}' \leftarrow \emptyset$; $q.\widehat{C} \leftarrow 0$
3     $q.lst \leftarrow -\infty$; $q.nst \leftarrow q.\mathcal{T}.first$
4 $\Delta v_t \leftarrow v_q(q.\mathcal{T}' \cup \{t\}) - v_q(q.\mathcal{T}')$
5 if $t \in q.\mathcal{T}$ OR $q.nst = \infty$ OR $q.lst < q.\mathcal{T}'.last$ then
  $\Delta v \leftarrow \Delta v_t$
6 else $\Delta v \leftarrow min\{\alpha(v_q(q.\mathcal{T}') - q.\widehat{C}), \Delta v_t\}$
7 return A point query $q_l$ with the valuation function
  with the maximum value of $\Delta v$.
```

Procedure ApplyResults(t, q, π)

```
Data: $t$ is the current time, $q$ is the query, and $\pi$ is the
      amount that $q$ must pay.
1 if $\pi \geq 0$ then
2     $q.\mathcal{T}' \leftarrow q.\mathcal{T}' \cup \{t\}$
3     $q.\widehat{C} \leftarrow q.\widehat{C} + \pi$
4     if $t = q.nst$ then $q.lst \leftarrow t$; $q.nst \leftarrow q.\mathcal{T}.next(t)$
5 else if $t \in \mathcal{T}$ then $q.lst \leftarrow t$; $q.nst \leftarrow q.\mathcal{T}.next(t)$
```

At each time slot t, for each available location monitoring query, function *CreatePointQuery* is called to create a point query at the queried location. After execution of the created point queries, procedure *ApplyResults* is invoked to apply the results for each query. Consider one location monitoring query q. If $t \in q.\mathcal{T}$, or if sampling at the last sampling time has been failed, or t is greater than the final requested sampling time, a point query is created. The maximum value for the valuation function of the point query is denoted by Δv, which is the valuation of taking a sample at time t. When none of these conditions hold, the current extra budget is calculated and a fraction, denoted by parameter α, of this extra budget is used for a point query. The reason behind using only a fraction of the extra budget is to be able to keep some extra budget for uncertain future samples. A natural way for specifying α is to start with a small value and increase it (or possibly decrease it) as we learn the difference between the utility obtained compared to the expected utility and how much utility is expected to achieve in future. In this algorithm, $\mathcal{T}.first$ returns the first sampling time in $\mathcal{T}$, and $\mathcal{T}.next(t)$ returns the first sampling time which is greater than t. Note that although omitted in the algorithm, v_q considers the quality of the collected sensor readings or the expected quality of a sensor reading before the actual sensor selection at the current time.

Algorithm 2: Sensor Selection for Location Monitoring Queries at time t

```
Data: Set $Q$ of location monitoring queries, and
      quality valuation function $v_q$ of each query $q$.
1 $Q_p \leftarrow \emptyset$
2 foreach $q \in Q$ do
3     $Q_p \leftarrow Q_p \cup$ CreatePointQuery$(t, q)$;
4 Select sensors for point queries in $Q_p$ and for each point
  query calculate its payment $\pi_{q,t}$. If the point query is
  not satisfied, set $\pi_{q,t} \leftarrow -\infty$.
5 foreach $q \in Q$ do
6     ApplyResults$(t, q, \pi_{q,t})$
```

Algorithm 3 is used for answering a set of region monitoring queries. The algorithm uses two main functions: *CreatePointQueries* and *ApplyResults*. The first is called for generating the required point queries, and the second is called for applying the results after execution of point queries. Consider a single region monitoring query q with region r_q. At each time t, a query-specific function f_q is consulted for obtaining the desired sampling locations based on the current locations and costs of sensors in r_q and the remaining budget. For each sampling location, a point query is created with the valuation function equal to the valuation improvement by the sensor at that location. The generated point queries, Q_t, are then executed along with all other point queries, e.g., using one of the algorithms introduced in Section 3.1.

After execution of point queries we can make use of the sensors that are selected for other queries if they fall into r_q. The maximum total cost contribution from query q for these sensors is $\alpha(C_t - \widehat{C}_t)$, where C_t is the expected cost to be spent, and $\widehat{C}_t$ is the actual cost spent in time t. The control parameter α is used for determining how much extra budget to keep for the next time slots. The actual cost contribution depends on the sensors' costs and their valuation improvement for the query.

Sensor data sharing is possible when the query regions overlap. This potential data sharing can be incorporated in Algorithm 3 by providing the input set $SC_{r,t}$ to the function f_q as a set containing weighted costs of sensors. For example, when some sensors are already selected for other queries, a weight of zero can be assigned to their costs in $SC_{r,t}$. Also,

```
Function CreatePointQueries($t, q, S_{r,t}, SC_{r,t}$)
  Data: $t$ is the current time, $q$ is the query, $S_{r,t}$ is the
        set of sensors in region $q.r$ at time $t$, and $SC_{r,t}$
        is their corresponding locations and costs
  Result: A set of point queries, the expected budget for
          the queries, and the set of sensors which are
          supposed to answer the point queries
1 if $t = q.t_1$ then
2     $q.S \leftarrow \emptyset$, $q.\widehat{C} \leftarrow 0$
3 $C_t \leftarrow 0,\ Q_t \leftarrow \emptyset$
4 $S_t \leftarrow f_q(S_{r,t}, SC_{r,t}, q.B - q.\widehat{C})$
5 foreach $s \in S_t$ do
6     Create a point query $q_s$ with the valuation function
      $v_{pq} = v_q(S_t) - v_q(S_t \backslash \{s\})$.
7     $Q_t \leftarrow Q_t \cup q_s;\ C_t \leftarrow C_t + c_s$
8 return $\{Q_t,\ C_t,\ S_t\}$
```

a heuristic approach for increasing the selection chance of a sensor which can be shared by k region monitoring queries, is to reduce its cost by a factor of $w(k)$, where w is a function that returns a real value between 0 and 1.

```
Procedure ApplyResults($q, Q_t, C_t, S_t, \pi, A_{r,t}$)
  Data: $q, Q_t, C_t, S_t$ are as for CreatePointQueries, $\pi$ is
        the amount that $q$ pays for the satisfied point
        queries, and $A_{r,t}$ is the set of sensors in region
        $q.r$ selected for other queries
1 foreach $q_s \in Q_t$, if $q_s$ is not satisfied do
2     $S_t \leftarrow S_t \backslash \{s\}$
3 $\widehat{C}_t \leftarrow \pi$
4 Contribute to the costs of sensors in $A_{r,t} \backslash S_t$ by the
  maximum amount of $\alpha(C_t - \widehat{C}_t)$ and update $\widehat{C}_t$
  accordingly.
5 $q.S \leftarrow q.S \cup (S_t \cup A_{r,t})$; $q.\widehat{C} \leftarrow q.\widehat{C} + \widehat{C}_t$
```

Because of the query budget constraints, a mechanism is needed to decide which sensors to take measurements from and when. In the context of sensor networks, this problem is referred to as *sensor selection problem.* To be able to support a wide range of applications, the queries are requested to provide a method for specifying the desired set of sampling points at each time slot (f_q in Algorithm 3.) In PS with uncontrolled mobility, applications are faced with an obstacle for finding out all their desired sampling points in advance: only at the current time we know which sensors are located in the queried region. As a workaround, instead of finding upfront all the desired sampling points, at each time slot we can select the best sampling locations based on the available sensors in the queried region.

We propose Algorithm 4 as an example approach for finding the set of best sensors to query for the current time t_c. The sensors in the queried region along with their costs and locations are provided as input to the algorithm. We assume that the current location of sensors will not change in the future. Even with this simplifying assumption, the problem is NP-complete. The proposed solution is hence a greedy approach. Notice that even though the algorithm selects (sensor) locations for different time instances, we are only interested in the locations for t_c. The multiplication of the sensing quality improvement by the fraction of the remaining time over the duration of the query is an attempt to increase the chance of selecting sensors for the current time.

```
Algorithm 3: Sensor Selection for Region Monitoring
Queries at Time $t$
  Data: Set $Q$ of region monitoring queries, and quality
        valuation function $v_q$ of each query $q$.
1 $Q_p \leftarrow \emptyset$
2 forall the $q \in Q$ do
3     Compute $S_{r,t}$ and $SC_{r,t}$
4     $X[q] \leftarrow$ CreatePointQueries($t, q, S_{r,t}, SC_{r,t}$)
5     $Q_p \leftarrow Q_p \cup X[q].Q_t$
6 Select sensors for point queries in $Q_p$.
7 foreach $q \in Q$ do
8     $\pi \leftarrow$ the payment of $q$ for the satisfied point queries
      in $X[q].Q_t$
9     $A_{r,t} \leftarrow$ selected sensors in region $q.r$ at time t for
      other queries
10    ApplyResults($q, X[q].Q_t, X[q].C_t, X[q].S_t, \pi_q, A_{r,t}$)
```

```
Algorithm 4: Sampling point selection for a region
monitoring query at time $t_c$.
  Data: Set $S$ of available sensors in queried region $r_q$ of
        query $q$, the budget $B$, and function $F$ that
        quantifies the value of a set of sensors.
  Result: $S_{t_c}$ is the set of locations to query at current
          time $t_c$.
1 $C \leftarrow 0$
2 $S_t \leftarrow \emptyset$ for all $t = t_c, \ldots, q.t_2$
3 while $C < B$ do
4     foreach $s \in S$ do
5         foreach $t = t_c$ to $q.t_2$ do
6             if $s \notin S_t$ then
7                 $\delta_{s,t} \leftarrow (F(S_t \cup \{s\}) - F(S_t))\,\theta_s \frac{q.t_2 - t}{q.t_2 - q.t_1}$
8     $(s^*, t^*) \leftarrow \arg\max_{s,t} \delta_{s,t}$
9     $S_{t^*} \leftarrow S_{t^*} \cup \{s^*\}$
10    $C \leftarrow C + c_{s^*}$
```

3.4 Query Mix

When the aggregator receives queries of different types, it has the possibility of sharing the sensors among them and hence increasing the total utility. Indeed, since individually finding an optimal set of sensors for multiple point or aggregate queries is NP-Complete, finding the optimal set of sensors for the combination of queries is also NP-Complete. We therefore propose Algorithm 5 for selecting sensors considering the commonalities between the queries at hand.

This algorithm consists of four stages. In the first stage, the required point queries are generated for available location monitoring and region monitoring queries. For doing so, the functions *CreatePointQuery* used in Algorithm 2 and *CreatePointQueries* used in Algorithm 3 are called. In the second step, all the queries are jointly provided to Algorithm 1 as the input. This greedy algorithm selects the sensors with the objective of increasing the total utility and computes the amount that each query will be charged for using the data from the assigned sensors. In the next stage, the results of the point queries generated for continuous queries are applied using the procedures *ApplyResults* used in Algorithms 2 and 3. The cost contribution from region monitoring queries for the extra sensors that they can use necessitates the adjustment of the payments for other queries sharing the same sensors. In the last stage, selected sensors are asked to send their measurements, which are then sent to the higher level query processor. Finally, the users are charged the amount that is calculated in the previous stage and each selected sensor is paid its announced price.

Algorithm 5: Data Acquisition for Query Mix

```
Data: Set $Q_{agg}, Q_p, Q_{lm}, Q_{rm}$ of aggregate, point,
      location monitoring, and region monitoring
      queries, set $S$ of available sensors, and quality
      valuation function $v_q$ of each query $q$.
    ▷ Point query creation for continuous queries
1 [Function CreatePointQuery ] Create required point
  queries for location monitoring queries $Q_{lm}$. Let $Q_p^{lm}$
  denote the generated point queries.
2 [Function CreatePointQueries ] Create required point
  queries for region monitoring queries $Q_{rm}$. Let $Q_p^{rm}$
  denote the generated point queries.
    ▷ Sensor selection
3 [Algorithm 1] Input all the queries
  $Q_{agg} \cup Q_p \cup Q_p^{lm} \cup Q_p^{rm}$ to Algorithm 1 for sensor
  selection.
4 [Algorithm 2 and 3] Run Algorithms 2 and 3 for
  applying the results of the corresponding point queries.
    ▷ Payment adjustment
5 Adjust the payments to be asked from the queries based
  on the potential cost contribution resulting from Step 4.
    ▷ Data acquisition and accounting
6 Ask the selected sensors to provide their measurements.
7 Provide the data to the query processor. Charge the
  users whose queries have been satisfied and pay the cost
  of selected sensors.
```

4. EXPERIMENTAL EVALUATION

In order to prove the effectiveness of our utility-driven data acquisition framework, we have conducted a thorough simulation study using real and synthetic mobility datasets. In the following we first introduce these datasets and then for each query type presents the experiments and their results.

4.1 Setup

We consider a simulation period of 50 time slots in all the experiments. At each time slot new queries are generated and then executed jointly with the existing continuous queries, if any. The inaccuracy of each sensor is chosen randomly from the interval $[0, 0.2]$. We refer to the maximum number of readings that a sensor can provide as the *lifetime* of the sensor. When the number of measurement taken by a sensor reaches its lifetime, it cannot be used anymore in the subsequent time slots. Unless otherwise stated, the lifetime is equal to the simulation period. We use two simple energy cost models: A *fixed cost model* defined by $c_s^e(\mathcal{E}_s) = C_s$, and a *linear cost model* defined by $c_s^e(\mathcal{E}_s) = C_s(1 + \beta(1 - \mathcal{E}_s))$, where C_s is a fix price, and β is the cost increment factor.

We assume the aggregator is a trusted entity and therefore, the sensors always report their true locations to the aggregator. However, the other consumers of the data are not trusted. The privacy computation model employed in the simulations works as follows: each sensor keeps a history of the times when it has reported a measurement to the aggregator. The size of the history is called the *privacy window* and is denoted by w. The privacy loss is the weighted average of the time distances between the times when a data is reported and the current time t:

$$p_s(H_s, l_s) = \frac{w + \sum_{t' \in H_s} (w - (t - t'))}{\frac{w(w+1)}{2}}. \tag{14}$$

Function (14) puts more weight on the recent data reporting times. Therefore, by applying this function, the sensor device tries to avoid reporting measurements in consecutive time instances, hence hiding its trajectory. We consider 5 different privacy sensitivity levels (PSL) for the sensor devices, namely *Zero, Low, Moderate, High,* and *Very High*, which are, respectively, mapped to values $0, 0.25, 0.5, 0.75, 1$. The privacy cost function is defined as:

$$c_s^p(p_s(H_s, l_s)) = PSL_s * p_s(H_s, l_s) * C_s. \tag{15}$$

In all the experiments we set $C_s = 10$ and unless stated otherwise, we use the fixed cost model for energy and we set the privacy sensitivity level to *Zero*.

A trust value in the interval $[0, 1]$ is assigned to each sensor. A trust value of zero indicates that the sensor readings cannot be trusted at all, while the trust value of one implies that the sensor readings are fully trusted. Even though the trustworthiness of the sensors can change over the course of time, for simplicity, we assume that this parameter remains unchanged over the whole simulation period. Since the trust or reputation assessment of sensors is not the focus of this work, we assume that there is a trust assessment mechanism in place which assigns trustworthiness values to the sensors upon initialization. In the simulations, unless specified otherwise, the sensors are assumed to be fully trusted.

4.2 Datasets

We use two mobility datasets: RWM generated based on the *random waypoint model* [6], and RNC which is a real mobility dataset from Nokia campaign (http://opensense.epfl.ch). In RWM each sensor moves from its current location with a speed randomly selected between zero and a sensor-specific maximum speed. The direction of the movement is either up, down, left, or right, and is randomly selected. The movements are limited to a region of 80×80 grids. Upon initialization the maximum speed of each sensor is set randomly to 4 or 5, which are spread randomly in the region. Only a central subregion of 50×50 is considered by the aggregator as the working region (or the "hotspot"). That is, only the queries and sensors that are bounded by this subregion are considered, but sensors can enter and leave this subregion. The default number of sensors for the experiments using RWM is 200.

RNC is derived from a data collection campaign in Lausanne, Switzerland consisting of location information of 180 participants. The whole region of movement is griditized into grids of length 100 meters. Only a region of 237×300 grids is considered and the working subregion is set to be a subregion of size 100×100. Because of the high sparsity of this mobility data, we have shifted the movements times to have more users in the same day. We also added some dummy users with the mobility patterns of the existing users but with randomly selected starting location and time of the movement from the real trajectories. This resulted in having in total 635 sensors in the whole region and on average 120 sensors in the working subregion in each time slot.

In the simulations involving region monitoring queries, we use Intel Lab dataset (http://db.csail.mit.edu/labdata). The simulations are performed over a 20×15 region. The reason for using this data set is that in the experiments for region monitoring query we need to have real sensor readings in addition to mobility data. Since the sensors in the Intel Lab deployment are stationary, we assign the sensor readings to the grids in which they are located. Then we use a random waypoint model for generating mobility data for 30 imaginary sensors. The sensor reading which is assigned to a grid is reported as the data for the imaginary sensor that is located in that grid.

4.3 Single-Sensor Point Queries

We have implemented a baseline algorithm which in each time slot takes queries one by one and for each query selects the sensor with maximum utility. A sensor that is selected to answer a query at a certain location is also assigned to all other queries at that location. The cost of the selected sensors is set to zero for the remaining queries. This algorithm resembles execution on query arrival and data buffering for the duration of a time slot.

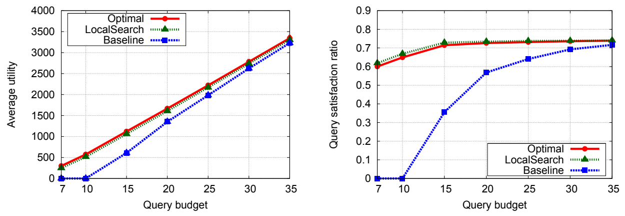
In each time slot 300 users submit point queries each with the location randomly picked over the working region. The valuation function (3) with $\theta^q_{min} = 0.2$ is used for all point queries. For finding the quality of each sensor reading, function (4) is used with $d_{max} = 5$ for the experiments on RWM and with $d_{max} = 10$ for the experiments on RNC.

Figure 2(a) shows the average utility achieved by different algorithms per time slot w.r.t. the query budget when RWM is used. It can be seen that the Local Search algorithm finds solutions close to the optimal ones. In this experiment, the query budget is the same for all the queries. Figure 2(b) shows the fraction of point queries that are answered (satisfaction ratio) by different algorithms. Since the baseline algorithm does not efficiently benefit from sensor sharing among queries, it cannot answer any queries when the query budget is small (i.e., 7, 10). On the contrary, the optimal and Local Search algorithms can always answer more than 60% of the queries. When the query budget is big enough, the average utility and the satisfaction ratio achieved by the algorithms become very close since the queries can afford the cost of any sensor. As the budget increases, the satisfaction ratio converges to around 73%. This shows that regardless of the amount of budget, about 27% of the queries can never be answered because of the lack of sensors with acceptable quality in their vicinities. We recall that our goal is to maximize utility, not to maximize the satisfaction ratio nor the quality of results. This means that the optimal algorithm might not always achieve the best satisfaction ratio compared to the heuristic algorithms. In other words, achieving higher utility sometimes requires refusing answering queries for which a lower total utility can be achieved.

Figures 3(a) and 3(b) show the results when RNC is used. Similar patterns as for the experiment with RWM are observed. However, the average utilities and satisfaction ratios are smaller than their counterparts in Figures 2(a) and 2(b). Besides the difference in the mobility patterns, the reason is that the simulation area in the experiments with RNC is larger, hence the sensors are more sparsely distributed. Hereafter, we only present the results on RNC dataset.

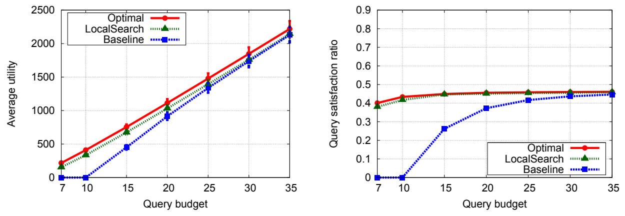
In practice we cannot assume that all the queries have the same budget. Therefore, in the next experiment we chose the query budget uniformly at random in *budget mean* ±10. Figures 4(a) and 4(b) show that the results are very similar to when the fixed budget scheme is used. Therefore, in order to highlight more easily the efficiency of the algorithms, in all the next experiments we use the fixed query budget scheme.

Figures 5(a) and 4(b) illustrate that as the number of queries increases, the possibility of sharing sensors among more queries increases, which results in more utility and slightly higher satisfaction ratio. In the next experiment, we randomly pick the privacy sensitivity level of each sensor and we set the sensors use the linear energy cost function with β randomly chosen in [0, 4]. The results are depicted in Figures 6(a) and 6(b) for lifetime 50 and in Figures 6(c) and 6(d) for lifetime 25. The figures demonstrate that in general the utility and satisfaction ratio drop when the participants become privacy sensitive and use non-constant energy cost (compare to Figures 3(a) and 3(b).) The difference in the utilities when the lifetime is 50 and when it is 25 is very small, which implies that only a few sensors are worn out during the simulation. Due to their mobility, sensors might enter and leave the working region at any time, which prevents sensors to be exhaustively used.



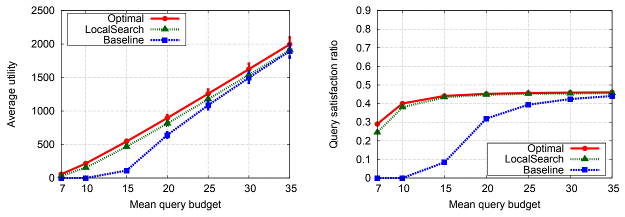
(a) (b)

Figure 2: Single-sensor point queries, RWM dataset, a) average utility per time slot, b) satisfaction ratio.



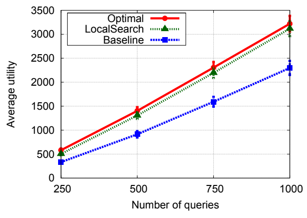
(a) (b)

Figure 3: Single-sensor point queries, a) average utility per time slot, b) satisfaction ratio.

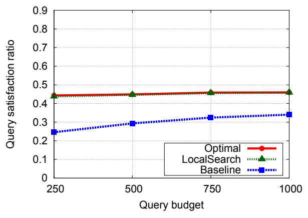


(a) (b)

Figure 4: Uniformly distributed budget, a) average utility per time slot, b) satisfaction ratio of point queries.

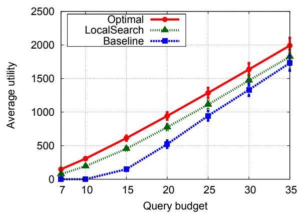


(a)

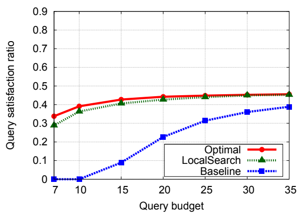


(b)

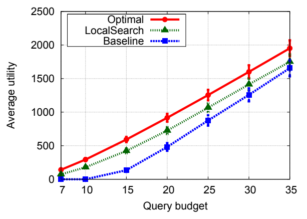
Figure 5: Varying the number of queries, with query budget fixed to 15. a) Average utility per time slot, b) satisfaction ratio of point queries.



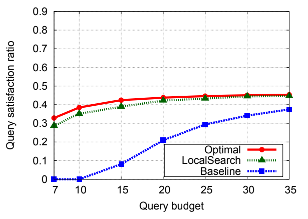
(a)



(b)



(c)



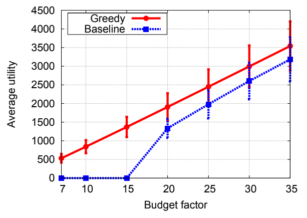
(d)

Figure 6: Random privacy sensitivity level and linear energy cost function, a) average utility per time slot - lifetime 50, b) satisfaction ratio of point queries - lifetime 50, c) average utility per time slot - lifetime 25, d) satisfaction ratio of point queries - lifetime 25.

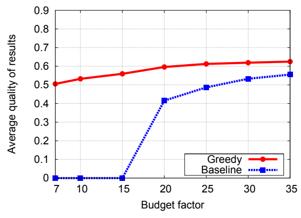
4.4 Spatial Aggregate Queries

Since other types of multiple-sensor one-shot queries introduced in this paper can be treated similarly to the spatial aggregate queries, we only consider this query type. We have implemented a baseline algorithm for answering multiple-sensor one-shot queries which resembles sequential execution of queries with data buffering. It takes the queries one by one and for each query selects the sensors that result in best utility. The cost of the selected sensors is set to zero for the subsequent queries in the time slot. The valuation function (5) is used for all queries. The sensing range of sensors is set to 10 units. In each time slot the number of aggregate queries is selected uniformly at random with the mean of 30 queries. The queried regions are generated randomly in the working region. The query budget for each query q is set to $\frac{\mathcal{A}(r_q)}{1.5r_s}b$, where r_s is the average coverage of the sensors (which is set to d_{max}), and b is the budget factor.

Figure 7(a) shows the average utility per time slot w.r.t. the budget factor. Algorithm 1 not only always significantly outperforms the baseline, but also can answer queries even when the budget is small. Figure 7(b) shows the average quality of results for the answered queries. The average quality of results for a query is the valuation of the set of selected sensors for that query divided by the maximum value of its valuation function.



(a)



(b)

Figure 7: Aggregate queries, a) average utility per time slot, b) average quality of results.

4.5 Location Monitoring Queries

We use the technique proposed in [19] for determining the sampling times for a location monitoring query. This algorithm works on the historical data and selects the sampling times such that the residuals of the model based on the values at the sampling times and the model given all the historical data is minimized. The number of sampling times is assumed to be fixed and is given to the algorithm. This approach assumes that the data values for the current time interval are almost the same as the data values in the same time interval in the past. Even though this is a weak assumption, it shall not affect the effectiveness of our data acquisition approach, which is designed to work with any sampling approach and any valuation function. We use a dataset containing a trace of ozone measurements from a deployments in Zurich, Switzerland (http://www.opensense.ethz.ch). A linear regression model is used to model the data. We use the following valuation function:

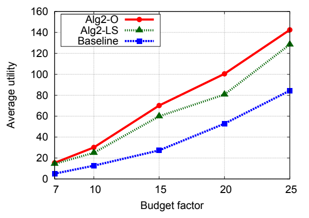
$$v_q(\mathcal{T}', \Theta) = B_q G(\mathcal{T}') \frac{\sum_{\theta \in \Theta} \theta}{|\Theta|}, \tag{16}$$

$$G(\mathcal{T}') = \frac{\sum_{i=1}^{N} r_i^2 | \mathcal{T}}{\sum_{i=1}^{N} r_i^2 | \mathcal{T}'}, \tag{17}$$

where $\mathcal{T}$ is the desired sampling times, $\mathcal{T}'$ and Θ are the set of timestamps and qualities of the samples taken so far, B_q is the query budget, N is the number of historical data items, and $r_i|T$ is the difference between the actual value of the ith data item and the modeled value from the model generated using only data items with timestamps in T.

At each time slot the number of existing queries and new queries is always less than 100. The location for each new query is randomly selected in the working subregion. The duration of each query is randomly chosen from $[5, 20]$ and the number of desired sampling times is set to one third of the query duration. The budget assigned to each query is equal to its duration times the budget factor. The parameter α is set to the constant value 0.5. Figure 8(a) shows the average utility per time slot w.r.t. the budget factor using Algorithm 2 compared to a baseline approach. *Alg2-O* and *Alg2-LS* state that, respectively, the optimal solution and the Local Search algorithm are used for answering point queries. In the baseline approach point queries are generated only at the desired sampling times and then the baseline approach introduced in Section 4.3 is used for answering the point queries. The average quality of results is shown in Figure 8(b). The relatively small values for the average utility and average quality of results stem from the lack of enough sensors close to the queried locations and the weak assumption in the technique used in determining the best sampling times, which assumes similar periodic patterns in the data.

Nevertheless, our approach still outperforms the baseline.


(a)

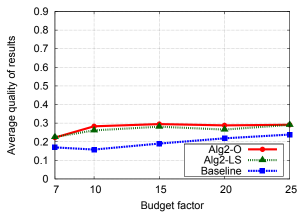
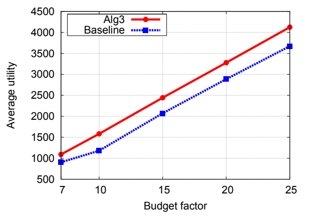

(b)

Figure 8: Location monitoring queries, a) average utility per time slot, b) average quality of results.


(a)

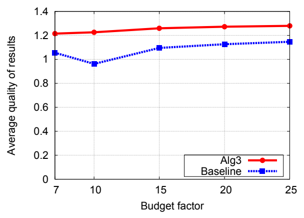

(b)

Figure 9: Region monitoring queries, a) average utility per time slot, b) average quality of results.

4.6 Region Monitoring Queries

In this experiment we assign the valuation function (7) to all region monitoring queries. The parameters of the Gaussian model are learned from a fraction of sensor readings in Intel Lab dataset. Function f_q in Algorithm 3 is implemented based on Algorithm 4. At each time slot one query is created with the query region randomly generated in the simulation area. The duration of the query is randomly chosen in $[5, 20]$. The budget assigned to each query is calculated as $\frac{\mathcal{A}(r_q)}{3\pi r_s^2}b$, where r_s is the average coverage distance of the sensors (2 in this case), and b is the budget factor. The parameter α is set to the constant value 0.5. The following weight function is used to modify the cost of a sensor which falls into the region of k region monitoring queries:

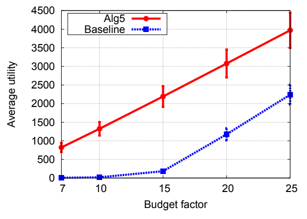
$$w(k) = \begin{cases} 11 - k & k < 10 \\ 0.1 & otherwise. \end{cases} \qquad (18)$$

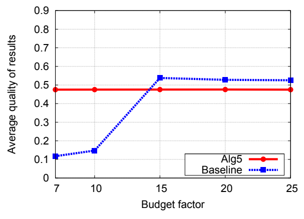
Figure 9(a) shows the average utility per time slot w.r.t. the budget factor using Algorithm 3 compared to a baseline approach. In Algorithm 3 we use the optimal solution for answering point queries. In the baseline approach we do not use cost weighting and we omit sharing sensors that are selected for other queries and are not at the locations requested by the query. In addition, the baseline approach introduced in Section 4.3 is used for answering the point queries. Figure 9(b) shows that, most of the times, the average quality of results is more than 1, which means that the valuation of sensors selected for each query is more than what was requested by the queries. Note that this is possible since $F(\mathcal{A})$ is not bounded by 1.

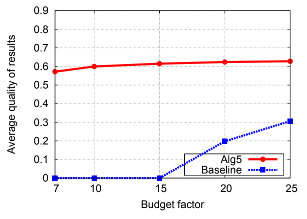
4.7 Query Mix

In addition to Algorithm 5, we have implemented a baseline algorithm for answering a mixture of queries of different types. In this algorithm, first the aggregate queries are executed using the baseline algorithm for aggregate queries. The cost of selected sensors is set to zero for subsequent queries in the current time slot. In the next step, the required point queries are generated for continuous queries and then they are executed along with the point queries issued by end users using the baseline algorithm for answering single-sensor point queries. This baseline resembles sequential execution of queries in one time slot with buffering data for the period of that time slot. The number of point, aggregate, and location monitoring queries is the same as in the experiments for each individual query type. Due to the lack of complete measurement data in RNC, we exclude region monitoring queries in this experiment. Sensor lifetime is set to 25 and a random privacy sensitivity level is assigned to each sensor. The linear energy cost function is used by each sensor with parameter β randomly chosen in [0, 4].

Figure 10(a) shows the average utility per time slot w.r.t. the budget factor. It can be seen that Algorithm 5 significantly outperforms the baseline approach. As Figures 10(b), 10(c), and 10(d) show, the quality of results produced by the baseline approach for each query type is either zero or very small when the budget is small. In contrast, our approach can satisfy many queries even when the budget is small thanks to more efficient sensor sharing.


(a)


(b)


(c)

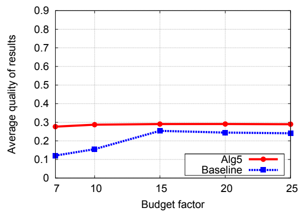

(d)

Figure 10: Mix of point, aggregate and location monitoring queries. a) average utility per time slot for query mix, b) average quality of results for point queries, c) average quality of results for aggregate queries, d) average quality of results for location monitoring queries.

We have also performed experiments with different trust value distributions and we observed that the more trustworthy the sensors are, the more utility they bring to the queries. This is indeed expected because in all utility functions that have been considered, the utility has a direct relation with the trustworthiness of the sensors.

5. RELATED WORK

A utility-based sensor selection framework is proposed in [1] in which the applications can specify the utility of each set of sensors in a wireless sensor network. Submodular and supermodular utility function classes are considered. The goal is to select a sequence of sets to maximize the total utility while not exceeding the available energy. In [15], the problem of sensor selection, where a set of sensors is selected according to the maximum a posteriori or the maximum likelihood rules, is formulated as optimizations of submodular functions over uniform matroids. A heuristic approach based on convex optimization is proposed in [7] for the sensor selection problem with the objective of minimizing the estimation error. In our scenario, the network model is different and the objective is to maximize the net benefit. As we allow multiple applications, which potentially have different valuation functions, we cannot identify up front in which function category our utility function falls.

Simultaneous placement and scheduling of sensors is considered in [10], where an algorithm is proposed to efficiently and simultaneously decide where to place sensors and when to activate them using the submodularity of the utility function. Two distributed sensor scheduling approaches are proposed in [5, 4]. These works are based on the assumption that the utility function is submodular. In this paper we pursue a centralized approach, which is not restricted to submodular utility functions in order to be able to handle applications with diverse requirements.

The work of [9] is perhaps the closest to this paper. We distinguish our work in two main ways: 1) they try to maximize the utility of data collection for the queried locations assuming that the sensors are fully trusted and the budget is fixed. In contrast, we aim for maximizing the utility of several concurrent queries, potentially of different types, assuming that the sensors are not fully trusted; 2) they assume that the phenomenon follows a known distribution and utilize this for the near-optimal sensor selection, whereas we don't have any explicit assumption on the phenomenon and we obtain the utility functions from the applications.

The problem of multi-query processing has been systematically defined in [14] in the context of relational database systems. Lifetime-based and event-based queries are introduces along with normal queries in sensor networks in [12]. Optimization techniques such as reordering of predicates and event query batching has been used to preserve power. Merging multiple user queries into one network query and then extracting user data streams from network data streams is proposed in [13]. Optimizing multiple aggregate queries in sensor networks is studied in [18] with the objective of minimizing the communication cost while taking into account the processing limitations of the sensor nodes. In order to reduce energy consumption in a wireless sensor network, rewriting a new monitoring query based on the existing ones and evaluating it in the base station rather than injecting it into the network is proposed in [11]. In the AdaptiveCQ framework [17], for efficient processing of multiple continuous queries, the intermediate results of queries are shared at a fine level without materializing them on disk. [8] proposes a query planner for distributed stream processing systems which exploits overlaps between queries and sharing partial results with the objective of efficient resource allocation. In our approach, data sharing is implied without using techniques such as query rewriting.

6. CONCLUSION

We proposed a holistic data acquisition framework for participatory sensing (PS) environments, where multiple applications may pose multiple queries of different types. We formulated the problem of optimal multi-query data acquisition with the objective of maximizing the total utility. We proposed heuristic algorithms for maximizing the utility in a myopic way for the most important query types or their mixes in this context. As a particular example, we considered efficient data acquisition for continuous queries in a PS environment with no guarantees on the data availability neither spatially nor temporally. As a future work, we plan to exploit knowledge on the sensor mobility.

7. REFERENCES

[1] F. Bian, D. Kempe, and R. Govindan. Utility based sensor selection. In *Proc. of IPSN*, 2006.

[2] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proc. of VLDB*, 2004.

[3] U. Feige et al. Maximizing non-monotone submodular functions. In *Proc. of FOCS*, 2007.

[4] D. Golovin, M. Faulkner, and A. Krause. Online distributed sensor selection. In *Proc. of IPSN*, 2010.

[5] M. Huber, A. Kuwertz, F. Sawo, and U. Hanebeck. Distributed greedy sensor scheduling for model-based reconstruction of space-time continuous physical phenomena. In *Proc. of FUSION*, 2009.

[6] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Proc. of Mobile Computing*, 1996.

[7] S. Joshi and S. Boyd. Sensor selection via convex optimization. *Trans. Sig. Proc. of*, 57(2), Feb. 2009.

[8] E. Kalyvianaki, W. Wiesemann, Q. H. Vu, D. Kuhn, and P. Pietzuch. Sqpr: Stream query planning with reuse. In *Proc. of ICDE*, 2011.

[9] A. Krause, E. Horvitz, A. Kansal, and F. Zhao. Toward community sensing. In *Proc. of IPSN*, 2008.

[10] A. Krause, R. Rajagopal, A. Gupta, and C. Guestrin. Simultaneous optimization of sensor placements and balanced schedules. *IEEE Trans. Automat. Contr.*, 56(10):2390–2405, 2011.

[11] Y. W. Lee, K. Y. Lee, and M. H. Kim. Energy-efficient multiple query optimization for wireless sensor networks. In *Proc. of SENSORCOMM*, 2009.

[12] S. Madden et al. The design of an acquisitional query processor for sensor networks. In *Proc. of SIGMOD*, 2003.

[13] R. Muller and G. Alonso. Efficient sharing of sensor networks. In *Proc. of MASS*, 2006.

[14] T. K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13(1):23–52, Mar. 1988.

[15] M. Shamaiah, S. Banerjee, and H. Vikalo. Greedy sensor selection: Leveraging submodularity. In *Proc. of CDC*. IEEE, 2010.

[16] K.-P. Shih, Y.-D. Chen, C.-W. Chiang, and B.-J. Liu. A distributed active sensor selection scheme for wireless sensor networks. In *Proc. of ISCC*, 2006.

[17] W. H. Tok and S. Bressan. Efficient and adaptive processing of multiple continuous queries. In *Proc. of EDBT*, 2002.

[18] N. Trigoni, Y. Yao, A. Demers, J. Gehrke, and R. Rajaraman. Multi-query optimization for sensor networks. In *Proc. of DCOSS*, 2005.

[19] Z. Yan, J. Eberle, and K. Aberer. Optimos: Optimal sensing for mobile sensors. In *Proc. of MDM*, 2012.