iDiary: From GPS Signals to a Text-Searchable Diary

Dan Feldman, Andrew Sugaya, Cynthia Sung, Daniela Rus
Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA, USA 02139
{dannyf, asugaya, crsung, rus}@csail.mit.edu

ABSTRACT

This paper describes a system that takes as input GPS data streams generated by users' phones and creates a searchable database of locations and activities. The system is called iDiary and turns large GPS signals collected from smartphones into textual descriptions of the trajectories. The system features a user interface similar to Google Search that allows users to type text queries on their activities (e.g., "Where did I buy books?") and receive textual answers based on their GPS signals.

iDiary uses novel algorithms for semantic compression (known as coresets) and trajectory clustering of massive GPS signals in parallel to compute the critical locations of a user. Using an external database, we then map these locations to textual descriptions and activities so that we can apply text mining techniques on the resulting data (e.g. LSA or transportation mode recognition).

We provide experimental results for both the system and algorithms and compare them to existing commercial and academic state-of-the-art. This is the first GPS system that enables text-searchable activities from GPS data.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, Experimentation

Keywords

GPS, text query, semantic compression, activity recognition, mobile device

1. INTRODUCTION

Mobile devices such as smart phones are playing an increasingly important role in our lives and can be the source of very large and useful data about the users carrying them. Our goal is to develop systems and algorithms that take signals generated from sensors such as GPS streams and convert them into text-searchable systems. For an individual user, such a system would automatically create a textual diary that captures the spatio-temporal activities of the user. For example, the user could pose the query "What are the restaurants I visited during SenSys 2013?" or "What are the roads I drove on today?" For groups of users, such a system captures their spatio-temporal interactions, for example "Where did I meet with X during SenSys 2013?", or "When did Y and X meet for the first time?"

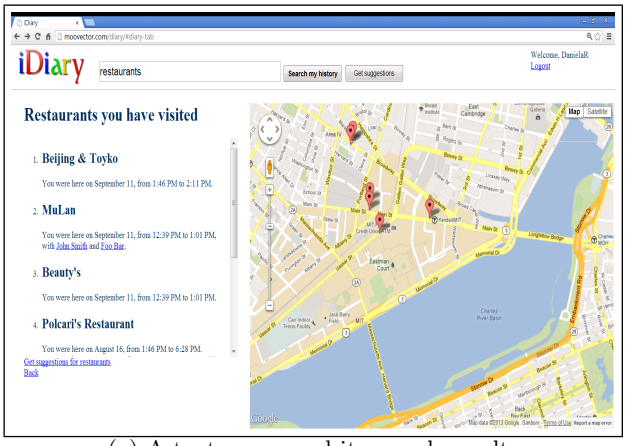
Translating massive amounts of raw GPS data points into useful human-understandable information is not currently possible because (a) compared to fields such as machine learning and text mining, very little is known about handling these kinds of data sets, both in theory and practice, (b) the user interface of existing GPS applications, such as "Google Latitude," targets map-navigation over text search, and (c) the data is huge (e.g., a single smart phone can generate on the order of 1GByte of data per day).

We have developed a system called "iDiary" that collects, stores, and processes GPS data signals generated by mobile phones and converts this data into spatio-temporal textual descriptions of trajectories and locations of the users. The locations are further correlated to businesses and activities associated with those businesses using text analysis on Yelp reviews.

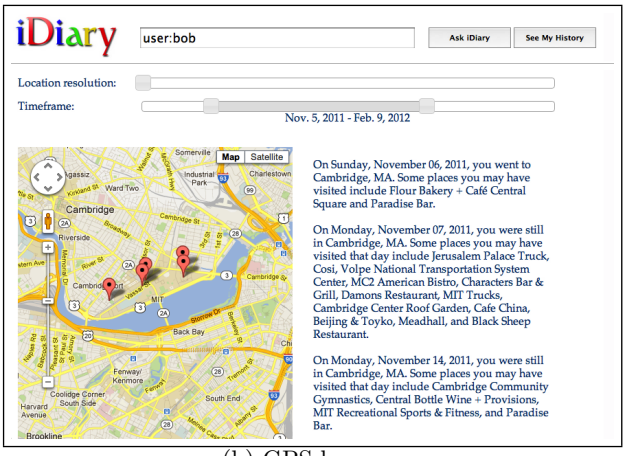
In a small pilot study we demonstrate the end-to-end capabilities of this system. Users carry a smart phone enabled with an app that collects their GPS data stream. The GPS signal is analyzed to identify a set of critical data points in it, thus greatly reducing the amount of data stored by the system (for example, for a user's traversal of a particular street, the relevant points are the location and time when the user entered the street, the location and time when the user exited the street, and the locations and times when the user entered buildings on the street). This small number of critical points are then converted to text using reverse geocoding. Users can then input text queries into an interface similar to "Google Search" and receive summaries of their recorded activities or recommendations for alternate activities based on the data gleaned from the rest of the network. iDiary allows users to search and manage their data without having to deal with maps or databases, and it provides one of the first systems to enable text-searchable activities from GPS data.

The capabilities of the iDiary system are enabled by novel algorithms for (1) analyzing and compressing massive stream-

SenSys'13, November 11–15, 2013, Rome, Italy.



(a) A text query and its search results



(b) GPS-browser

Figure 1: Snapshots from our system on real GPS data. The application has two modes: (a) The user can type a query such as "restaurant" on the screen and get a map with textual descriptions of visited restaurants. (b) The user can view a summary of his activities and can use the time-frame or resolution slider to change the number of output points in the summary.

ing GPS signals and (2) for extracting activity information from existing text repositories such as Yelp reviews. This paper describes the iDiary system, the enabling algorithms, and the experimental results from a pilot study with data generated using smart phones carried by several users over a period of over one year. The main innovation of this system is mapping GPS data into a text searchable database in a computationally effective way.

1.1 Our Contributions

1. iDiary system and iPhone application that allow users to collect their GPS points (traces) through their smartphone and search over the data using text queries in free form ("Where did I buy books"). Snapshots from the application's user interface are shown in Fig. 1 and 2.

2. New algorithms for analyzing and clustering the signals and converting them into a semantic database of the underlying patterns.

3. Novel ways of combining text mining techniques with GPS data to allow text search on GPS signals.

4. Experimental results for our system and algorithms.

This paper is organized as follows: Section 2 describes previous work on activity recognition and GPS data analysis. Section 3 gives an overview of the iDiary system. Section 4 describes the algorithms used in more detail. Section 5 contains the results of experiments on our system. We conclude with final thoughts in Section 6.

2. RELATED WORK

Mobile information retrieval analytics was identified at the Second Strategic Workshop in Information Retrieval in Lorne [5] as one of the six most interesting directions for future research. While applications exist to track users' mobile data and some effort has been made to analyze this information for certain types of activities (e.g. [15]), no system yet exists that combines these two research efforts to allow users to browse and query their recent activities.

Systems for Analyzing Location Data. The idea of collecting location data using the mobile phone is not new. For example, [24] outlines a framework for collecting and storing mobile sensor data for analysis. PEIR [22] uses this framework to collect users' GPS data from their mobile phones and determine their common transportation modes and environmental impact. Applications for tracking user activities and locations have also appeared in the commercial realm. Foursquare allows users to check-in to locations [1] and produces a searchable timeline [2]; however, data collection in this system is not automated. Google Latitude [25] can track where a user has been automatically and shows the user's activity on a high-level scale (i.e., work, home, out), as well as a list of previously visited locations. However, the application lacks low-level activity recognition (e.g., "drinking coffee") and searchable history. To our knowledge, no previous system provides text search capabilities over raw GPS data.

Textual Description of Activity. Recent work in sensor data analysis has demonstrated the possibility of recognizing human activities from raw data streams. For example, [6] used accelerometer data collected from different parts of the body to identify different activities typically performed around the household. The work most similar to our system is [19], which addressed the challenge of determining a user's activity based on location information alone. However, this approach required activities to be labeled manually. In [23, 26], GPS traces were automatically labeled with activities based on the character of nearby businesses, but the labels themselves were chosen by humans. To our knowledge, no methods exist that allow fully automated labeling of human activities.

Semantic Compression. The work above generally assume small data sets or store GPS data only long enough

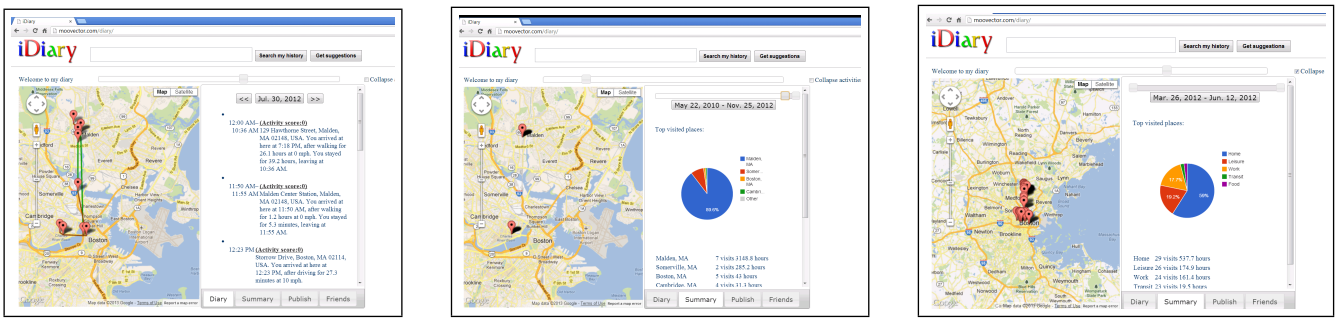


Figure 2: (left) An auto-generated text that describes the user trajectories as a diary. The user can change the relevant time interval, and details resolution of the map using the sliders. (middle) Text summary of places that were visited on a given time interval. The pie-chart above the list represents the fraction of time each place was visited. (right) The summary of visited businesses can be replaced by summary of activities (Food, Leisure, etc), using the category that is assigned to each business by Yelp.

to extract activity information. The growing availability of mobile devices and the large amount of data produced would overwhelm any system that attempts to store the data for future re-analysis. Thus real-life implementation of any such system requires some sort of semantic compression of the data. Gonzalez et al. [17] demonstrated that human motions follow spatial and temporal patterns, indicating that GPS data can be compressed by recording the small map of a user's frequently traversed locations. This idea has led to an explosion of methods (see [16, 18, 7] and references therein) aimed at extracting this map from trajectories, but few can accommodate the large amounts of data that are expected for the iDiary system.

Coresets are frequently used to compress data and allow more efficient processing while providing guarantees on the optimality of the output. Coresets for approximating trajectories by line segments have been proposed [13, 4] under a variety of assumptions. The coreset we use for iDiary is based on [14] and uses the fact that GPS signals, unlike general sets of points, are usually sampled at constant frequency. Hence, a set of points that are lying on a segment can be represented exactly by its end-points. Our coreset contains a large set of GPS points (actually, larger than the input), but since most of them lie on segments, it can be stored using constant space.

Text Search. We use Latent Semantic Analysis (LSA) to produce the "semantic space" for our text search (see Fig. 7). When the database is small enough, a straightforward application of SVD to the data suffices. For larger and on-line corpus (e.g., Wikipedia and blogs), we use coresets for LSA. Such coresets were suggested in [11, 12] for computing low-dimensional semantic space that is spanned by few input points. Since the rows of document-term matrix are sparse, this semantic space takes much less space than the regular SVD factorization that is usually used for LSA. Our coreset construction is inspired by their construction and also uses weighted random sampling.

3. FROM GPS POINTS TO TEXT-SEARCH INTERFACE

A user Bob goes from home to work and back everyday on weekdays, and on weekends he visits the park near his

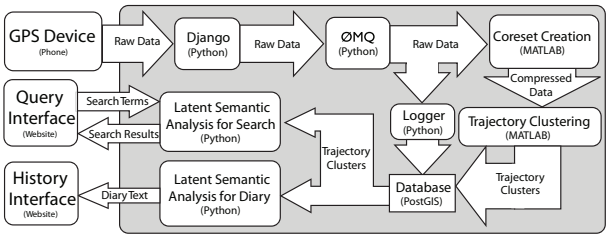


Figure 3: The code architecture of the system and the data flow associated with it. A rounded rectangle represents a section of code, and the arrows show the input and outputs for each such section. The language used to write a section of code is in parentheses.

home. Sometimes on the way to work, Bob uses the train and sometimes his car. Occasionally he must drop his child off at the kindergarten. On the way to work, Bob sometimes stops at the gas station. He sometimes goes to a location he never visits again, but in general his movements occur among these five main locations: home, work, park, kindergarten, and gas station.

Fig. 4 shows Bob's *semantic map*, a graph whose edges correspond to the trajectories that Bob takes. A node appears when trajectories converge or diverge (e.g, Home and Junction). These can be considered as Bob's decision states. Each edge represents several routes (roads, streets) that are approximated in the figure as linear segments between the nodes. For example, a trajectory from the node Home to node Work might go through routes labeled $a, b, \ldots, i$ in Fig 4. Each trajectory is usually used more than once over time, at different speed (car, foot, bicycle, etc). The car/man icon near an edge of the graph in the figure refers to the common mode transportation that is used on the corresponding trajectory.

Figure 3 shows the code architecture of the system we have built to extract this kind of semantic map and story from a user's GPS signal. The computation flow consists of three main parts: data storage, activity recognition, and user interaction.

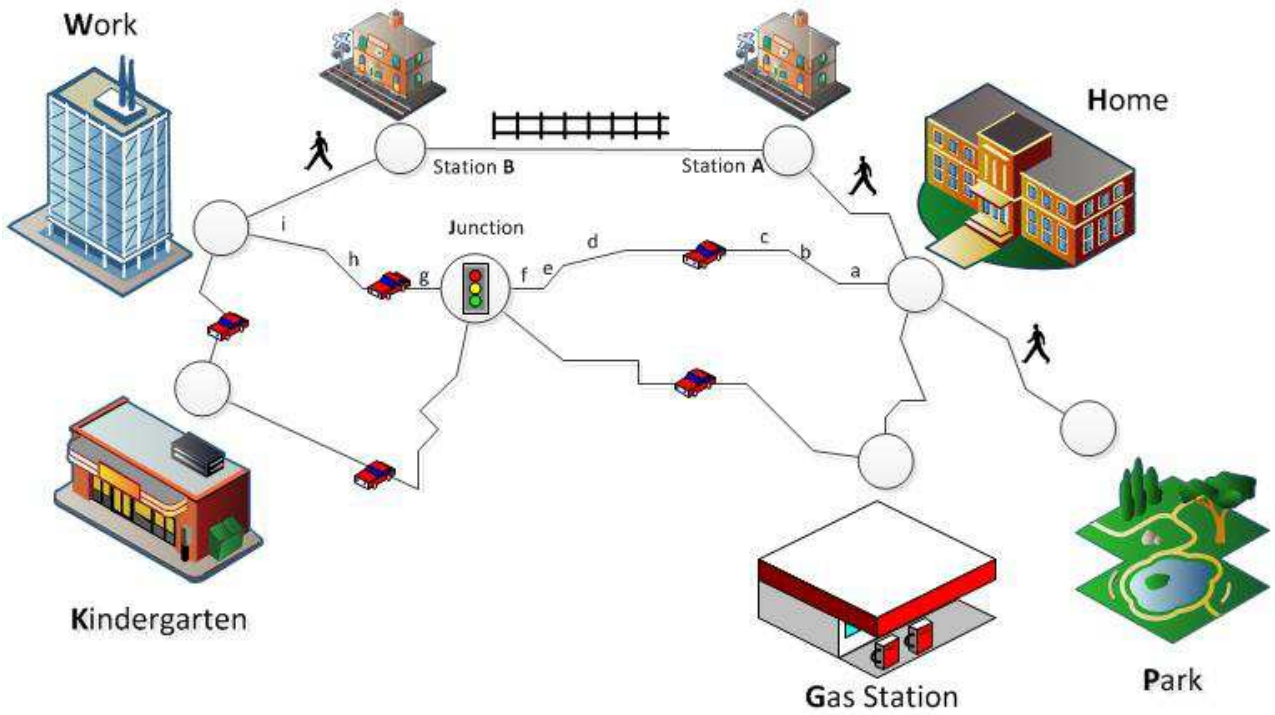


Figure 4: Semantic map of a user's trajectories.

Handling GPS-noise.

The key component of our system is the mapping of GPS locations to points of interest. A number of papers (e.g., [8, 27]) have discussed how correctly assigning a place based on GPS observation is difficult because multiple places are often within the margin of error in the GPS. For example, using GPS points that were received from the modern smartphone of Bob, it might still be hard to tell if he is drinking coffee at Starbucks or shopping in the store nextdoor.

We use Barabási's observation on repeated visits to reduce this problem. Roughly speaking, according to the law of large numbers, the mean of a collection of (GPS) sample points from a single ground-truth position, say, Starbucks, will become closer to the ground-truth over time. Of course, if the variance of the GPS noise is too large, and Bob does not stay in Starbucks long enough, the mean of the sample would still be a bad approximation to Starbucks. However, if Bob visits Starbucks frequently or several times, we can use his repeated visits to refine the esimate of his location. iDiary uses a (k, m)-segment mean algorithm to recognize and cluster the points of these visits over time. It then computes the mean for all these points in the cluster, from all the visits in Starbucks. After either frequent or long enough visits in the same place, this mean should approximate well the actual place. The same idea holds for general segments in this space, as frequently visited roads. Places that Bob visits very rarely and for a very short time might not be approximated well, but in some sense are also less relevant to the map of Bob's common trajectories.

Energy saving.

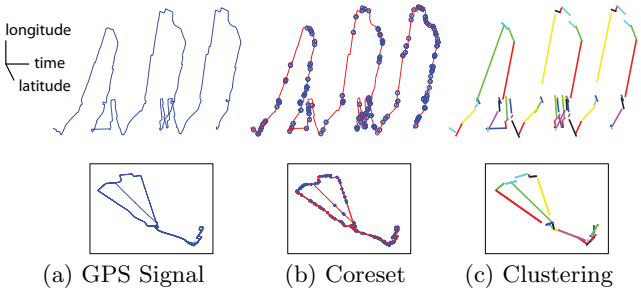
The system can be used for saving energy and battery life, based on two simple observations. First, if the user stays in the same place, the corresponding signal in the $(latitude, longitude, time)$ space will be approximately horizontal. In this case, we can sample the GPS less frequently. In general, smaller sampling rate helps very little to save energy as most of the consumed energy is due to the connection of the device to the satellites. However, when the frequency is low enough, depending on the specification of the device, the GPS sensor can be turned off until the next sample. This would make sense, for example, during night when the user is sleeping and the device standing still.

The second observation is that, based on our clustering algorithms and semantic map, we can *predict* the trajectory of Bob, even if he is not standing still. For example, if he goes to work using approximately the same route every morning, or if there is a single path in the semantic map from the current location to the next vertex (e.g., along the path $a \ldots f$ in Fig. 4), it might be easy to extrapolate the GPS points from historical data and turn off the GPS device.

3.1 GPS Data Storage

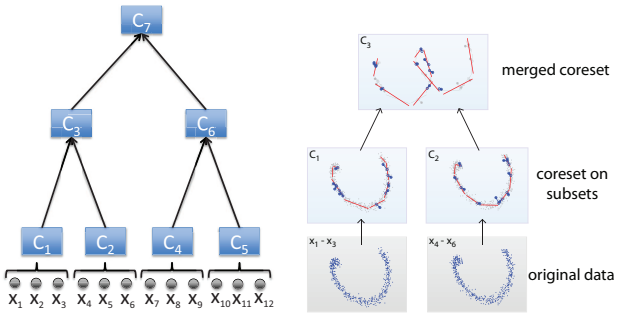
Collecting the signal. After Bob installs our application on his mobile device, the application collects his GPS points (traces/signal) and transmits them to our lab's server. This is done in the background using the location services of the iPhone. Since these services consume large amounts of power, we save battery by switching off the GPS (and consequently, data collection) when the user is stationary for ten minutes. The GPS is switched back on and begins collecting data once the user moves approximately half a kilometer, which is detected using wi-fi and/or cell towers. Bob's GPS data is shown in Fig. 5(a).

Compressing the signal. When the server receives Bob's GPS points, it compresses them for efficient storage and



(a) GPS Signal (b) Coreset (c) Clustering

Figure 5: Top: Images of Bob's trajectory and approximations in 3-D (time, latitude, longitude) space. Bottom: Projections on $\mathbb{R}^2$ (latitude, longitude). (a) Bob's raw GPS signal P, consisting of $n \sim 5000$ GPS points sampled at times $t = 1, \cdots, n$. (b) The coreset of the GPS signal, consisting of the k-segments mean of the signal, together with a few sampled weighted representative from the GPS points. (c) Clustering of the k segments that approximate the 3D-signal in time with m motion patterns, shown in m different colors; the 2-D projection is Bob's semantic map.



(a) Coreset tree (b) Construction of C_3

Figure 6: (a) Tree construction for generating coresets in parallel or from data streams [13]. Black arrows indicate "merge-and-reduce" operations. The intermediate coresets $C_1, \ldots, C_7$ are numbered in the order in which they would be generated in the streaming case. In the parallel case, C_1, C_2, C_4 and C_5 would be constructed in parallel, followed by C_3 and C_6, finally resulting in C_7. (b) Example construction of C_3 on a GPS signal that recorded a trajectory of 6 routes over time. The first (left) half of the input contains the first 3 routes. The second half (right) contains routes 4-6.

to speed up our following algorithms. In order to do this, we observe that the signal contains a lot of redundant information that can be removed. In particular, when people move from place to place (by foot, car, train, etc.), their trajectories can usually be described using simple linear routes. Hence, given an observed GPS signal, we approximate the trajectory by a small sequence of k segments (linear routes). We use this linear simplification of the signal to construct its semantic compression (coreset); see Definition 4.4. Our coreset consists of the k-segment mean with a "smart" non-uniform sample of weighted representative GPS points. Since the size of the coreset is constant, this provides an effective compression of Bob's signal for storage in a database. Fig. 5(b) is a visualization of the coreset of Bob's signal. See Section 4 for details on the coreset construction.

Compressing the coresets. Since Bob will continually be uploading new data points to the server, we require some method of calculating a coreset in a streaming setting (rather than recalculating the coreset every time new points come in). To do this, we compress the coresets that have already been computed together while reading the streaming signal. Fig. 6 outlines this approach. The result is that a constant-sized coreset for Bob's data can be maintained using only $O(\log^2 n)$ time per point insertion.

Using the same ideas from the streaming model, a non-parallel coreset construction can be transformed into a parallel one (that we run on our network). This allows us to merge trajectory information from many different users and analyze group activity patterns as well as those of a single individual.

3.2 Textual Description of Activity

Trajectory clustering on coresets. While the n GPS points of Bob's (and most of our) everyday GPS signal can be partitioned into $k \ll n$ routes that we use over time, we usually visit these routes more than once. That is, our routes correspond to only $m \ll k$ different projections on the (latitude, longitude) space (geographic locations), which we call *motion patterns*.

For extracting the motion patterns from the GPS signal, we approximate the signal by k-segments (as above) with the further constraint that the k-segments consist of only m motion patterns. This k segments are called the (k, m)-*segment mean*, and approximation algorithms to solve it were previously suggested in [14]. Both the extraction of the k routes over time and their clustering to m routes is done simultaneously by our trajectory clustering algorithm. This trajectory clustering algorithm is applied to the coreset and outputs the estimated semantic trajectories of the user. Fig. 5(c) shows an example output.

Trajectories as database tables. The output of our clustering algorithm consists of two database tables. The first table has fields "begin time," "end time," "route code," "direction," and "average speed." This table partitions the signal into $k \ll n$ estimated linear routes (rows in the table) that the user took. Since routes are usually repeated over time and we want to use this information, a route code (integer index from 1 to $m \ll k$) is saved instead of the coordinates of the route. A second table consists of a list of these m routes. Its fields are "route code," "begin point," and "end point." That is, each route is represented by its two endpoints, which are (longitude, latitude) pairs.

From numeric tables to text. Each route code that is output by the clustering algorithm consists of (latitude, longitude) coordinates in space. In order to turn these coordinates into human-readable and searchable text, we use *reverse geo-coding* which turns the coordinates into their textual description ("Starbucks," "Elm Street," etc.) and assigns this text to the coordinates. We obtain this information from external services: Yelp's reviews [3] for businesses and Google Maps [21] for other locations.

From text to activities. Searching on the reverse geo-

coding text alone would not allow us to find activities or, more generally, words that are related to a place but do not appear in its description. For example, we want the query "sandwich" to also return "Subway" (the sandwich store). We use Latent Semantic Analysis (LSA) [20] to construct a "semantic space." Intuitively, we extract j kinds of activities (topics, concepts) from a database of text about the locations Bob visits, where each activity is a linear combination of words in the corpus. More generally, instead of assigning just the geo-coding of the user's location, we can add additional text words such as activity based on speed ('driving'), an SMS that the user entered at this place, or even a snapshot that the user took in some appropriate feature space.

In the traditional usage of LSA, the corpus itself is now projected on the semantic space. We use LSA differently: we use reverse geo-coding on each GPS point of Bob's routes codes as described above and project the corresponding vector of terms on our semantic j-dimensional space. This yields j numbers for each GPS point which represent its correlation with each of the j activities. We add these j correlations as association fields for each GPS point's entry in the database. See Section 4 for more details.

3.3 User Interface

After Bob sends his GPS points to the server via our iPhone or Android application, he can browse his data by logging to our web site using the iPhone or any other internet browser at "http://moovector.com/diary/". Figures 1 and 2 show snapshots from the web application.

Diary. After logging in, Bob gets the screen that appears in Fig. 2. The map on the left corresponds to GPS coordinates that are sent to a "Google Maps" object [21]. To the right of the map is a list of visited places, sorted by visiting time.

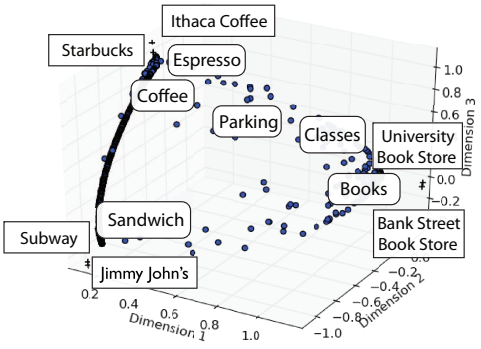


Figure 7: The $j = 3$ dimensional space that is produced by running LSA on Yelp's reviews database and corresponding projections of terms (words in the reviews) and businesses (based on reverse geocoding of GPS points). Only specific terms have been labeled to reduce clutter. Terms related to a given business are closer in angle (relative to the origin) to the business. For example, *sandwich* is near "Jimmy John's" and "Subway".

The mode of transportation (walking, driving, or flying) is based on simple thresholds on the distances between the GPS points. The user can browse through the diary using the arrows above the text, or by pressing the date box and choosing a new date from a pop-up calender. The interface includes a resolution slider that allows users to control the number of points shown on the map and a time frame slider for changing the start/end time of the descriptions.

Query search.

In addition to viewing his diary, Bob can input text queries on the GPS data in free form ("Where did I last buy books?"). The output is a list of relevant locations that he has visited. When Bob submits such a text query search on the top of the screen, the terms in the query are translated into a (sparse) vector of code words. This vector is projected on the semantic space and replaced by its (dense) semantic j-vector of activities. On the semantic space, the application computes the similar locations (closest vectors) to the query and returns them.

In Fig. 1 we see the results after typing the query "restaurants" in the search text box. The word "restaurant" is projected onto the semantic space that was constructed off-line, as explained in the previous paragraph, and is replaced by a vector in this space. This vector assigns high weights for words in English that have strong correlation to "restaurant" in Yelp reviews. Each business in Yelp also corresponds to a point in the semantic space. The output of the query are the four businesses that correspond to the four points which are closest to the query point. Note that none of the restaurants, except for the last, contain the word "restaurant" in their names.

Summary of visited places. When Bob presses the "Summary" button, the output results are aggregated according to locations rather than time; see Fig. 2. The sum of visited hours is displayed for every place in the list. Based on this list, a pie-chart shows the fraction of time spent in each place. The fraction is with respect to the time interval that is defined above the chart, which can be modified using a slider bar.

Summary of activities. Each Yelp business also contains a category of related activity in the Corpus: "Food", "Entertainment", etc. When Bob checks the "Collapse activities" check box, our system aggregates the visited businesses in the previous paragraph according to their activities and shows them in a list or a pie-chart. See Fig 2. If the user checks the "Collapse activities" box while in the "Diary" tab, a similar aggregation appears as text on the diary.

3.4 Server Architecture

We use a Twisted web server to handle incoming http requests, which compose 100% of the external communications of this server. Twisted gives us the flexibility to add asynchronous communication in the future. These http requests are sent to the Django web framework, which processes the data from requests and returns query results. The Django function takes in arguments (such as GPS data) from the http requests and publishes them to the messaging backbone, ØMQ, which is a lightweight transport layer that has libraries written in many languages. This data published to ØMQ is then received by subscribers, which include scripts that perform analyses on the data. The algorithms used for these analyses were implemented in MATLAB and Python.

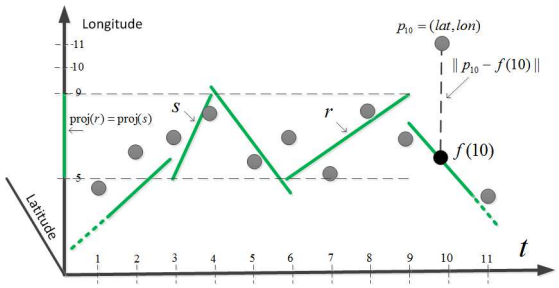


Figure 8: A signal of $n = 11$ GPS points $p_1, \ldots, p_{11}$ at times $t = 1, \cdots, 11$, and a k-segment function $f(t)$ where $k = 5$. The squared distance $\|p_{10} - f(10)\|^2$ between the point p_{10} and $f(10)$ is the error of approximating p_{10} by f. The projection of f on the (latitude, longitude) space yields only $m = 4$ projected segments, since the two segments s and r have the same projection (the same route with two different speeds). Hence, f is also a $(5, 4)$-segment.

4. ALGORITHMS

In this section we summarize the algorithms for clustering and learning the "big data" signals that are used by iDiary. Due to lack of space, we only give the main results and omit the proofs, which can be found in [14]. Our algorithms support both the parallel computation model and the streaming model, which we define as follows:

DEFINITION 4.1 (PARALLEL STREAMING ALGORITHM). *We call an algorithm* Parallel and Streaming *if, with high probability, it can process on-line a signal of n points using:*

- $O(\log^2 n)$ *space (memory)*
- $O(\log^2 n)$ *update time per point insertion*
- $O(n/M \cdot \log^2 n)$ *overall running time using $M \geq 1$ processors*

Linear Simplification. We solve the following k-segment mean in order to get a linear routes description from a GPS signal (see Figs. 4 and 8).

DEFINITION 4.2 (k-SEGMENT MEAN). *For an integer $k \geq 1$, a* k-segment *is a piecewise linear function $f : \mathbb{R} \to \mathbb{R}^d$ of k segments. Given a signal (sequence of points) $p_1, \cdots, p_n$ in $\mathbb{R}^d$, the* k-segment mean *minimizes the* error $\sum_{t=1}^{n} \|f(t) - p_t\|^2$ *among every possible k-segment f. Here, $\|x - y\|$ is the Euclidean distance between x and y in $\mathbb{R}^d$.*

We use the $(1+\varepsilon)$-approximation parallel streaming algorithm from [14] to solve the k-segment mean.

THEOREM 4.3 ([14]). *Let P be a signal in $\mathbb{R}^d$, $k \geq 1$ and $\varepsilon > 0$. There is a parallel streaming algorithm that returns a $(1+\varepsilon)$-approximation to the k-segment mean of P (see Definition 4.2).*

Semantic Compression. Given a signal P, an integer $k \geq 1$ and $\varepsilon > 0$, a coreset of P ε-approximates its fitting error by any query k-segment, as formally defined below.

Algorithm 1: CORESET(P, k, ε)

Input: A sequence $P = p_1, \cdots, p_n \in \mathbb{R}^d$, an integer $k \geq 1$, and $\varepsilon \in (0, 1)$.
Output: A (k, ϵ)-coreset (C, w) for P with high probability

Compute the k-segment mean f of P `/* See Theorem 4.3 */`
Set $P' \leftarrow \{f(t) \mid p_t \in P\}$, the projection of P on f
foreach $p_t \in P$ **do**
  $s(p) \leftarrow \frac{\|f(t)-p_t\|^2}{\sum_{p \in P} \|f(t)-p_t\|^2)}$
Pick a non-uniform random sample T of $\lceil 16dk/\varepsilon^2 \rceil$ points from P, where for every $p \in P$ and $q \in T$ we have $p = q$ with probability $s(p)$
$C \leftarrow P' \cup T \cup \{f(t) \mid p_t \in T\}$
for *each* $p_t \in T$ **do**
  $w(p) \leftarrow \frac{1}{|T| \cdot s(p)}$
  $w(f(p_t)) \leftarrow -w(p)$
for *each* $p \in P'$ **do**
  $w(p) \leftarrow 1$
return (C, w).

DEFINITION 4.4 ((k, ε)-CORESET). *Let $p_1, \cdots, p_n \in \mathbb{R}^d$ be a signal. Let $k \geq 1$ be an integer and $\varepsilon > 0$. A sequence of points $C = c_1, \cdots, c_m \in \mathbb{R}^d$ with a weight function $w : C \to \mathbb{R}$ is a* (k, ε)-coreset *for this signal if:*

$$(1-\varepsilon)\sum_{t=1}^{n}\|f(t) - p_t\|^2 \leq \sum_{t=1}^{m} w_t\|f(t) - c_t\|^2 \leq$$
$$(1+\varepsilon)\sum_{t=1}^{n}\|f(t) - p_t\|^2.$$

We use Algorithm 1 and its implementation from [14] to construct a (k, ε)-coreset for any given signal. To begin, the algorithm computes an approximation to the k-segment mean f of P as a rough approximation of the signal. However, with only the k-segment mean, points that are far away will be approximated badly. We therefore add a sample of points to the approximation. In particular, we sample points that are far from f with high probability and those that are closer with low probability. See Fig 4 and Fig. 9 for examples. Since the sample is small and the k-segment mean can be represented by its $O(k) = O(1)$ endpoints, the coreset can be represented using constant space. The correctness of the algorithm was proved in [14].

THEOREM 4.5 ([14]). *Let P be a signal in $\mathbb{R}^d$, and let $\varepsilon > 0$, $k \geq 1$ be constants. Let (C, w) be the output of a call to* CORESET(P, k, ε) *(Algorithm 1). Then the following holds:*

- *(C, w) is a (k, ε)-coreset for P, with high probability*
- CORESET *is a Parallel Streaming algorithm*
- *(C, w) can be stored using constant space*

Trajectory Clustering. A (k, m)-segment is a k-segment in time that corresponds to only m projected segments in space (see Fig. 8). Similarly to the k-segment mean, the (k, m)-segment mean of a signal minimizes the sum of squared distances to the signal, under this additional constraint. Unfortunately, the problem of computing the (k, m)-segment

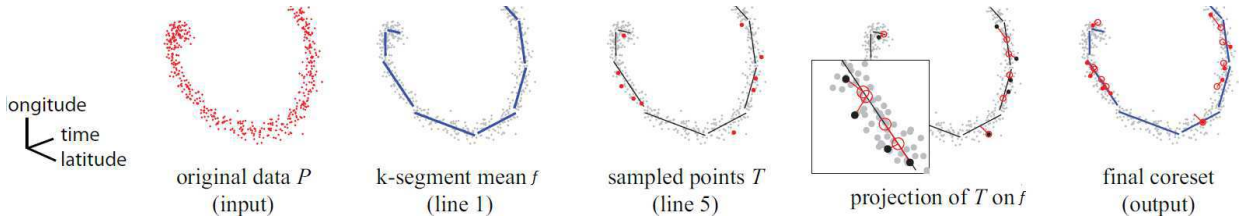


Figure 9: The output coreset in Algorithm 1 consists of the k-segment mean f of P, together with sample of points T of positive weight, and their projection $f(T)$ with negative weights. At each step, the new entities are shown in color (segments in blue, points in red) and pre-existing structures are shown in light gray

.

Algorithm 2: EM-KM-MEAN$(P, k, m, i_{\text{end}})$

Input: A signal $P \subseteq \mathbb{R}^d$, two integers $k \geq m \geq 1$, and number i_{end} of iterations.
Output: A (k, m)-segment $S_{i_{\text{end}}}$.

`/* Initialization: */`
Compute the k-segment mean f_0 of P
`/* See Theorem 4.3 */`
$Q \leftarrow k$ points in $\mathbb{R}^4$ that corresponds to the concatenated k vertices of f
Compute the m-means (points) clustering $Q_1, \cdots, Q_m$ of Q (using EM)
for $j \leftarrow 1$ *to* m **do**
  $P_{1,j} \leftarrow$ the points of P whose projection is on a segment $\overline{p,q}$ where $p, q \in Q_j$
for $i \leftarrow 1$ *to* i_{end} **do**
  `/* Maximization step: */`
  for $j \leftarrow 1$ *to* m **do**
    $a_{i,j} \leftarrow$ the 1-segment mean of $P_{i,j}$
  $A_i \leftarrow \{a_{i,1}, \cdots, a_{i,m}\}$
  `/* Expectation step: */`
  $f_i \leftarrow$ The optimal (k, m)-segment f of P whose projection is A_i
  for $j \leftarrow 1$ *to* m **do**
    $P_{i+1,j} \leftarrow$ points whose projections are closest to the jth projected segment of f_i, among its m projected segments

mean of a signal is NP-hard [14] if m is part of the input (not a constant). We use the practical algorithm from [14] that takes time linear in m, and uses the expectation-maximization technique for the clustering. The cost decreases in every iteration and the solution converges to a local minimum, as in the EM-version of k-means [10]; see Alg. 2.

Geo-Text Search. In order to obtain a semantic space of Yelp's reviews, we first number every one of the m English terms in the corpus (Yelp). Every document in the corpus is then translated into a sparse vector in $\mathbb{R}^m$ whose ith entry is the frequency of the ith term in the document. We decrease this number for words that appear too often (known as "stop words") in the corpus, such as "the" and "is," using a simple formula called tf-idf (term frequency-inverse document frequency), which is the number of appearance in the document divided by number of appearance in the corpus for a given word. From the n filtered document vectors, we construct an $n \times m$ term-document matrix A, and compute the Singular Value Decomposition (SVD) of A. This is a factorization $A = UDV^T$, where $U \in \mathbb{R}^{n \times m}$ and $V \in \mathbb{R}^{m \times m}$ are orthogonal matrices and $D \in \mathbb{R}^{m \times m}$ is a diagonal matrix with non-decreasing diagonal entries. The base of our j-dimensional semantic space is then defined to be the first j columns of V. See Fig. 7 for our results for $j = 3$.

Recall that our algorithms return selected points from the GPS signal, which are translated to text descriptions using reverse geo-coding; see Section 3.2. Each description of s terms is converted to an s-sparse vector $u \in \mathbb{R}^m$. We project u on our semantic space to get a dense mixture $u' = V^T u \in \mathbb{R}^j$ of j topics, which intuitively tells us how the query is related to each of the main j topics in our corpus (Yelp).

When users enter a query of h terms (corresponding to an h-sparse vector $q \in \mathbb{R}^m$), we project q to obtain the semantic representation $q' = V \cdot q \in \mathbb{R}^j$. In order to find the locations that are most similar to the query q, we define the semantic relevancy between a query and a location as the correlation between their mixture of topics. Mathematically, this is the cosine of the angle between the vectors u and q,

$$\text{relevancy}(u, q) := \frac{|u' \cdot q'|}{\|u'\| \|q'\|}.$$

For a given query q we then return the location u that maximizes this similarity.

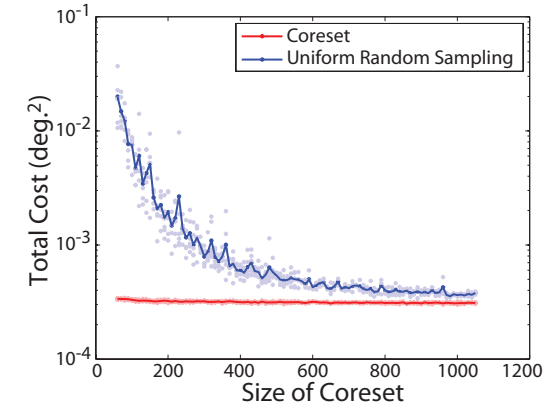
5. EXPERIMENTAL RESULTS

We have implemented the iDiary system according to Fig. 3 and collected data from 6 users and a fleet of taxis using our smartphone application for a period of a year, giving us about 9GB of data. In this section, we illustrate the system capabilities using several small pilot studies.

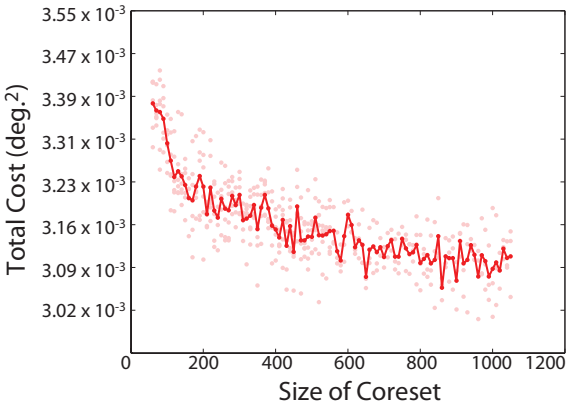
5.1 GPS to text for iDiary

We tested our system's ability to translate GPS data into a list of visited businesses. For these experiments, we used trajectory data collected by a single user over a period of 5.5 hours. The resulting data set consisted of 20,031 points during which the user made 14 stops at 6 locations. The destinations and routes were pre-designed (to generate ground truth for evaluation) and the user spent about 15 minutes at each stop. For textual description of the activity, we used the businesses and reviews in the Yelp academic dataset [3].

Coreset Quality. We tested the quality of the coreset by computing the total cost of the optimal 30-segment mean of the coreset and comparing it to the optimal 30-segment

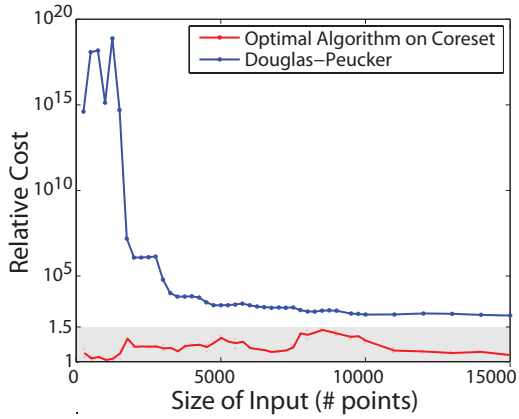


(a) Vs. Uniform Sampling

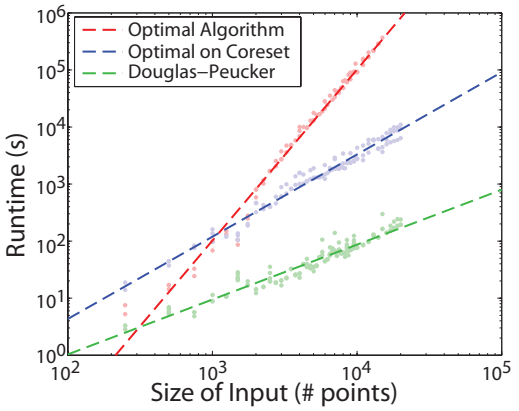


(b) Close-up of (a)

Figure 10: Comparison of absolute error resulting from running the optimal 30-segment mean algorithm on coresets of varying sizes versus uniform random sampling. Individual trial runs are shown as lightened points, and the mean over trials as solid lines.



(a) Vs. Douglas-Peucker



(b) Runtime

Figure 11: (a) Comparison of relative error resulting from running the optimal k-segment mean algorithm on a coreset versus Douglas-Peucker, as compared to the optimal k-segment mean of the entire input. Individual trials are shown as lightened points and the mean over trials as a solid line. Note that because of the large difference in performance between the optimal algorithm and Douglas-Peucker, there is a change in scale between the gray shaded portion of the graph and the white. (b) Comparison of runtimes for the optimal algorithm, the optimal algorithm on a coreset, and Douglas-Peucker.

mean of a equivalently sized uniform random sample of the input. The cost was the sum of squared distances between the 30-segment mean of the compressions and the optimal 30-segment mean on the original dataset. We maintained f to contain 50 segments and varied the size of the non-uniform random sample T. For comparison with the uniform random sample, we took the size of the coreset to be $50+|T|$. For all tests, we used the streaming procedure outlined in Section 4, processing the data in blocks of 1000 points.

The results, shown in Fig. 10, demonstrate that both the coreset and uniform random sampling yield improvements in cost as size increases, and the two methods appear to be converging to the same error value. However, for small sizes, using the coreset yields significantly better results than the uniform random sample.

We also compared our k-segments mean approximation to Douglas-Peucker [9], a commonly-used heuristic for finding the k-segment mean. For this test, we found the optimal k-segments mean on varying-sized subsets of the user's trajectory and computed the relative cost, the ratio of the cost of the approximation to the optimal cost. A relative cost of 1 indicates that the result had the same cost as the optimal solution. For all trials, we used a coreset of 50 segments and 100 samples.

The results in Fig. 11 show that computing the optimal k-segments mean of the coreset yields a solution much closer to the optimal in terms of cost than using Douglas-Peucker. In addition, the runtimes for these trials (Fig. 11) show that using the coreset-based approximation is only about a constant factor slower than Douglas-Peucker. The power fits to the data (dotted lines) indicate a runtime that is approximately linear in the size of the input for both approximation schemes, but cubic for the optimal algorithm. The majority of the time for the coreset approximation was spent constructing the coreset itself.

Overall our experiments confirm that compression via core-

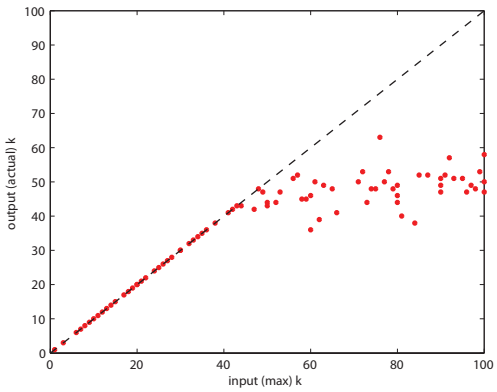


Figure 12: Output k value for multiple runs of the EM-(k, m)-mean algorithm with m held constant at 15. The dotted line shows the relationship $k_{out} = k_{in}$. The output k levels off to some constant even as the input k continues to increase.

set provides outputs of quality vastly better than currently-used heuristics without a large increase in runtime.

Trajectory Clustering for Detecting Movement Repetitions. We implemented the EM algorithm in Section 4 for use in our system. An issue of interest when applying algorithms for k-segment mean and (k, m)-segment mean is what the values of k and m should be. We found that for m small relative to k, the output was not always an exact (k, m)-segments approximation but rather had a smaller k than the permitted value (see Fig. 12). In addition, this output k value leveled off approximately to a constant despite further increases in the input k. This indicates that for a given data set and m value, the EM algorithm will find the optimal k naturally. Therefore, a user need not choose a k value but may merely choose m and, inputting a sufficiently large k, leave determination of true k to the algorithm.

Extracting Business Names. To extract business names from the GPS points, we used the following procedure. For each point in the dataset, businesses within 64 meters were found. We used 64 meters as the radius because the iPhone GPS is generally accurate up to this distance. Once these businesses were found, the recall and precision were calculated. If the point's timestamp was within 15 minutes of a start time and the business that was actually visited at the given GPS point appears on the list of businesses, then recall was 100% for that point and the precision was one divided by the number of businesses on the list. These values for recall and precision were also used if the timestamp was after 15 minutes of a start time and the last business visited was not on the list of businesses. Otherwise, the recall and precision for that point was 0.

Average recall and precision were found for each dataset, shown in Table 1. To verify that the correct stop points appear in our compressed representation of the data, we performed this experiment on the original dataset, on the points in the coreset, and on the endpoints of the k segments in the (k, m)-mean. The recall and precision are not significantly different between the three datasets, and the precision is consistently low on all of the datasets.

Recall for all the datasets is over 50%, which means the system correctly recalls the true location over half the time.

Dataset	# of Points	Recall	Precision	Runtime
Raw Data	5526	59%	8%	103 sec
Coreset	679	65%	8%	14 sec
Clusters	60	63%	8%	1 sec

Table 1: Results of the end-to-end experiment. The number of points, recall, precision, and runtime are shown for each representation of the dataset.

The poor recall is due to inaccurate GPS measurements from the phone. We expect that as a user visits the same location more often, the endpoints of the (k, m)-segment mean will converge (by the law of large numbers) to the true location that the user visited. We therefore suspect that as we collect longer data sets, the recall should improve.

The low precision is due to two sources of error. First, the inaccuracy of the phone GPS forces us to report all businesses within a 64 meter radius, of which there are large number. Second, business locations in the Yelp academic dataset are not accurate. Businesses that are not placed at their true location in the dataset or are placed in the same location of another landmark can greatly decrease the precision. This means that the precision can be increased with better phone GPS technology and further analysis on the user's history to predict where the user would go.

The significant reduction in computation shows that semantically compressing the data to make it easier to store and analyze is not only feasible, but effective. Both the number of points and time taken can be reduced by a factor of 100 while maintaining similar results. The computational complexity of the function performed is $O(n)$ time. If the key algorithm required to deliver the desired results has running time greater than $O(n)$ time, the effect of applying our coreset methodology is even greater.

5.2 Diary Search

In this section, we discuss experiments for correlating businesses with terms and querying GPS data for activities.

Searchable Businesses. We experimented with LSA on a matrix A of tf-idf values generated using the entire Yelp academic dataset [3]. In our case, a "document" was all reviews corresponding to a specific business. For initial tests, the dataset was limited to two book stores ("University Book Store" and "Bank Street Book Store"), two sandwich places ("Subway" and "Jimmy John's"), and two coffee shops ("Starbucks" and "Ithaca Coffee Company"). The resulting A matrix contained elements for 4799 terms and 6 businesses.

We performed the geo-text search procedure outlined in Section 4 using $k = 3$. A visualization of the resulting matrices U and V is shown in Fig. 7. We performed three search queries using the search terms *coffee*, *books*, and *sandwich*. We expect *coffee* to have a high relevancy value with the two coffee shops, *books* to have a high relevancy value with the two book stores, and *sandwich* to have a high relevancy value with the two sandwich shops.

The procedure was tested 1000 times for each term and the average computation time was .007 seconds for the query of a given term, regardless of the term. The actual relevancy matrices u are shown in Table 2 for each of the three search terms. Each of the search terms has a high relevancy value with the associated two businesses. Also, the other four businesses that should not be associated with the term have

	Books	Coffee	Sandwich
University Book Store	0.500	0.012	-0.007
Bank Street Book Store	0.500	-0.001	0.001
Starbucks	0.004	0.493	0.011
Ithaca Coffee Company	-0.012	0.506	-0.026
Jimmy John's	-0.003	-0.022	0.501
Subway	-0.001	0.000	0.498

Table 2: Relevancy values of a business (row) given a search term (column). Relevancy values are derived from the multiplication of matrix V with the query vector. Relevancy values greater than 0.1 are highlighted in gray.

query	Starbucks	Bank Street Book Store
1st	starbucks (0.310)	books (0.384)
2nd	coffee (0.175)	bookstore (0.180)
3rd	ithaca (0.059)	children's (0.035)

Table 3: Relevancy values of the 3 highest ranking terms in response to the business queries "Starbucks" and "Bank Street Book Store." Each cell contains the term, with the relevancy value in parentheses.

a low relevancy value. This confirms that using LSA can successfully perform a search query to retrieve businesses associated with a search term.

The results from this experiment show that it is possible to use LSA to make businesses searchable by term, which allows for a diary created by a coreset of GPS locations to become searchable. The same tests were performed on the entire Yelp academic dataset (157796 terms, 6898 businesses) with $k = 100$ with similar results. For example, searching businesses related to *sandwich* resulted in "Subway," "Jolly Roger Cafe," and "Jimmy John's" with relevancy values of 0.018, 0.016, and 0.015 respectively. The lower relevancy values come from the greater number of terms and businesses, which when normalized, leads to lower values.

Activity Recognition. For activity recognition, we used LSA in the reverse direction; that is, we used businesses as search terms and retrieved the most relevant terms. We performed a search for activities associated with the businesses "Starbucks" and "Bank Street Book Store." The search was performed 1000 times for each business and the average computation time was .008 seconds for the query of a given business, regardless of the business. The top three relevant terms for each of the two business searches are shown in Table 3. The top results (those with relevancy value > 0.1) are terms that are clearly semantically associated with the business that was searched. For example, "Bank Street Book Store" resulted in the top results *books* and *bookstore*. By using the relevancy values found by LSA, we are successfully able to determine what terms are associated with a business.

The results from this test reveal that using LSA allows us to find terms that are most relevant to a given business. This allows for more semantic activity recognition in which we can predict not only where people have visited based on their GPS data but also what activities they performed while at those locations. As with the previous experiment, this one was also performed on the larger dataset with promising results. Searching for terms relevant to "Starbucks" leads to *coffee* as the top hit, with a relevancy value of 0.341.

Queries on GPS Data. For this experiment, we used two different datasets. The first dataset contained 46,478 GPS points collected by one user over 6 months. The second dataset contained 115,562 GPS points collected by 5 different users, including the user from the first dataset, over 6 months. The data was compressed into a coreset, but trajectory clustering was not performed. While the theoretical bounds guarantee that our system is scalable, additional fine tuning and techniques might be needed in future research for large community of users.

	5 users	1 user
books	Harvard Book and Binding Harvard Book Store	MIT Press Bookstore The MIT Coop
food	Cafe Spice Cafe China	Cafe Spice 65 Cafe China
subway	Subway Sola Cafe and Catering	Subway Quiznos

Table 4: Results of the end-to-end diary search results for each of three queries on the two different datasets. The top two results are shown for each query.

We performed queries using the search interface in Fig. 1. In the search bar, a user was specified using the format of "user:*name*" with search terms. Three different types of search terms were used on each of the datasets. These included a full question ("Where did I last buy books?"), a noun ("food"), and a name of a business ("Subway").

Table 4 shows the results of performing the queries on the data from all 5 users, as well as the results for a single user. The search results in each case are sorted by relevancy and the top two results are shown.

Both of the queries pertaining to books return bookstores as results. Likewise, both of the queries pertaining to food return restaurants, and the two queries about Subway both return Subway as the most relevant result. From these results, we conclude that this system is functional and successfully allows for a user to search her history of activities.

6. CONCLUSION

In this paper we consider the creation of a text searchable diary system. We show how GPS data can be mapped to geo-referenced locations and activities described as text. We describe efficient coreset-based algorithms that achieve semantic data compression to cope with large volumes of data generated by phone sensors. We use the coreset algorithm to identify critical semantic points along a user's trajectory. We then show how reverse geo-coding, trajectory clustering, and signal latent semantic analysis coupled with reviews repositories such as Yelp can be used to go from GPS location to business description to a more abstract description of the user's activity. We present experimental results of our system and algorithms in small pilot studies.

We believe this work provides first steps towards the exciting vision of creating text searchable data bases from sensor data. The key insight is to semantically compress the data into small number of critical points that can be mapped to text by reverse geo-coding. This creates a human-understandable and text-searchable description. Much work remains to be done, especially in the areas of applying scalable information retrieval techniques to accommodate large sensor data sets, and using other external text resources (beyond Yelp) to make sense of the data. Our goal for the near future is more extensive data collection and evaluation of the system we have begun to create.

7. ACKNOWLEDGEMENTS

Support for this work has been provided in part by the Office of Naval Research under grant number ONR-MURI Award N00014-12-1-1000, Hon Hai/Foxconn Technology Group, and by the Future Urban Mobility project of the Singapore-MIT Alliance for Research and Technology (SMART) Center, with funding from Singapore's National Research Science Foundation. C.S. was supported by the Department of Defense through the National Defense Science & Engineering Graduate Fellowship Program. We are grateful. We also acknowledge Garry Wang for his programming support of the activity recognition module.

8. REFERENCES

[1] Foursquare,url:https://foursquare.com/.

[2] Techcrunch, url: http://techcrunch.com/2012/05/04/foursquare-gets-its-own-searchable-timeline-with-new-history-page/.

[3] Yelp academic dataset, url:http://www.yelp.com/academic_dataset.

[4] M. Abam, M. de Berg, P. Hachenberger, and A. Zarei. Streaming algorithms for line simplification. *Disc. & Comp. Geometry*, 43(3):497–515, 2010.

[5] J. Allan et al. Frontiers, challenges, and opportunities for information retrieval. Report, The Second Strategic Workshop on Information Retrieval in Lorne, February 2012.

[6] L. Bao and S. Intille. Activity recognition from user-annotated acceleration data. *Pervasive Computing*, pages 1–17, 2004.

[7] G. Chen, B. Chen, and Y. Yu. Mining frequent trajectory patterns from gps tracks. In *2010 Intl. Conf. on Comp. Intel. and Soft. Eng.* IEEE, 2010.

[8] Y. Chon, N. D. Lane, F. Li, H. Cha, and F. Zhao. Automatically characterizing places with opportunistic crowdsensing using smartphones. In *Proc. of the 2012 ACM Conf. on Ubiquitous Computing*, pages 481–490. ACM, 2012.

[9] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973.

[10] Q. Du, M. Emelianenko, and L. Ju. Convergence of the lloyd algorithm for computing centroidal voronoi tessellations. *SIAM J. on Num. Ana., pp. 102–119*, pages 102–119, 2006.

[11] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proc. 41th Ann. ACM Symp. on Theory of Computing (STOC)*, 2010.

[12] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. *Proc. of ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2013.

[13] D. Feldman, A. Sugaya, and D. Rus. An effective coreset compression algorithm for large scale sensor networks. In *The 11th Intl. Conf. on Info. Proc. in Sensor Networks*, pages 257–268, 2012.

[14] D. Feldman, C. Sung, and D. Rus. The single pixel gps: learning big data signals from tiny coresets. In *Proc. of the 20th Intl. Conf. on Advances in Geographic Information Systems*, pages 23–32. ACM, 2012.

[15] J. Frank, S. Mannor, and D. Precup. Generating storylines from sensor data. *Pervasive and Mobile Computing*, 2013.

[16] Z. Fu, W. Hu, and T. Tan. Similarity based vehicle trajectory clustering and anomaly detection. In *2005 IEEE Intl. Conf. on Image Processing (ICIP 2005)*, volume 2, pages II–602, 2005.

[17] M. Gonzalez, C. Hidalgo, and A. Barabási. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, 2008.

[18] J. G. Lee, J. Han, and K. Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proc. of the 2007 ACM SIGMOD Intl. Conf. on Mngmt of Data*, pages 593–604, 2007.

[19] L. Liao. *Location-based activity recognition.* PhD thesis, University of Washington, 2006.

[20] K. Lund and C. Burgess. Producing high-dimensional semantic spaces from lexical co-occurrence. *Behavior Research Methods, Instruments and Computers*, 28:203–208, 1996.

[21] C. Miller. A beast in the field: The google maps mashup as gis/2. *Cartographica*, 41(3):187–199, 2006.

[22] M. Mun, S. Reddy, K. Shilton, N. Yau, J. Burke, D. Estrin, M. Hansen, E. Howard, R. West, and P. Boda. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *Proc. of the 7th Intl. Conf. on Mobile Systems, Applications, and Services*, pages 55–68. ACM, 2009.

[23] S. Phithakkitnukoon, T. Horanont, G. Di Lorenzo, R. Shibasaki, and C. Ratti. Activity-aware map: Identifying human daily activity pattern using mobile phone data. *Human Behavior Understanding*, pages 14–25, 2010.

[24] S. Reddy, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. A framework for data quality and feedback in participatory sensing. In *Proc. of the 5th Intl. Conf. on Embedded Networked Sensor Systems*, pages 417–418. ACM, 2007.

[25] M. Siegler. Google latitude has 3 million active users. *TechCrunch.com, May*, 2010.

[26] Z. Yan, D. Chakraborty, C. Parent, S. Spaccapietra, and K. Aberer. Semitri: a framework for semantic annotation of heterogeneous trajectories. In *Proc. of the 14th Intl. Conf. on Extending Database Technology*, pages 259–270. ACM, 2011.

[27] M. Ye, D. Shou, W.-C. Lee, P. Yin, and K. Janowicz. On the semantic annotation of places in location-based social networks. In *Proc. of the 17th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*, pages 520–528. ACM, 2011.