The Single Pixel GPS: Learning Big Data Signals from Tiny Coresets

Dan Feldman, Cynthia Sung, Daniela Rus
Massachusetts Institute of Technology
Cambridge, MA, USA

ABSTRACT

We present algorithms for simplifying and clustering patterns from sensors such as GPS, LiDAR, and other devices that can produce high-dimensional signals. The algorithms are suitable for handling very large (e.g. terabytes) streaming data and can be run in parallel on networks or clouds. Applications include compression, denoising, activity recognition, road matching, and map generation.

We encode these problems as (k, m)-*segment* mean problems. Formally, we provide $(1+\varepsilon)$-approximations to the k-*segment* and (k, m)-*segment* mean of a d-dimensional discrete-time signal. The k-*segment mean* is a k-piecewise linear function that minimizes the regression distance to the signal. The (k, m)-*segment mean* has an additional constraint that the projection of the k segments on $\mathbb{R}^d$ consists of only $m \leq k$ segments. Existing algorithms for these problems take $O(kn^2)$ and $n^{O(mk)}$ time respectively and $O(kn^2)$ space, where n is the length of the signal.

Our main tool is a new *coreset* for discrete-time signals. The coreset is a smart compression of the input signal that allows computation of a $(1 + \varepsilon)$-approximation to the k-segment or (k, m)-segment mean in $O(n \log n)$ time for arbitrary constants ε, k, and m. We use coresets to obtain a parallel algorithm that scans the signal in one pass, using space and update time per point that is polynomial in $\log n$. We provide empirical evaluations of the quality of our coreset and experimental results that show how our coreset boosts both inefficient optimal algorithms and existing heuristics. We demonstrate our results for extracting signals from GPS traces. However, the results are more general and applicable to other types of sensors.

Categories and Subject Descriptors

H.3 [**INFORMATION STORAGE AND RETRIEVAL**]: Clustering

ACM SIGSPATIAL GIS'12, Nov. 6-9, 2012. Redondo Beach, CA, USA

General Terms

Algorithms, Theory, Experimentation

Keywords

Line Simplification, Big Data, Coresets, Streaming, Douglas Peucker, Trajectories Clustering

1. INTRODUCTION

Today, most smartphones carry multiple sensors that can be used to retrieve the trajectories, directions, locations, etc., of the user. We are interested in robustly identifying locations despite the error that is intrinsic in the GPS signal. Specifically, we consider two problems: (1) Given a large GPS trace, how do we robustly identify the critical points in this trace that indicate a change in semantics for the trace (such as turning from one road to another)? (2) Given multiple GPS traces, how do we compute the points that have been visited multiple times across these trajectories?

Although the sampling frequency of a signal may be high, it usually contains a great deal of data redundancy. This is because unlike random noise signals, signals from sensors usually satisfy:

1. The underlying data has simple geometrical structure.
2. Patterns in the data are repeated over time.

For example, we expect that a signal of n GPS points collected over a user's life will contain only $k \ll n$ trajectories of this user, and that those can usually be described by simple shapes (roads, streets). Also, typical trajectories should come from a set of $m \ll k$ places or roads in space (such as work, home and roads in between). This paper exploits these two properties in order to learn signals, compress them, and reveal their hidden semantic patterns.

For simplicity, we discuss GPS-signals ($d = 2$), although our results hold for general d-dimensional signals with only linear dependency on the dimension d. For example, the iPhone 4 produces signals from an accelerometer and gyroscope, an ambient light sensor, a moisture sensor, and a proximity sensor [11]. Other relevant signals include those from LiDAR, video, and audio.

2. THE OPTIMIZATION PROBLEMS

The sensor signal properties described in Section 1 gives rise to the following two optimization problems, which we address in this paper.

2.1 Linear Simplification

The first property states that the n $(x, y$,time) GPS-points in a signal can be approximated over time by some $k \ll n$ simple shapes, such as linear (3-dimensional) segments. Since GPS-points usually contain Gaussian noise [31], the maximum likelihood estimation (MLE) method [24] implies that the originating (without noise) signal that most likely generated our noisy one is the one that minimizes its sum of squared distances from the points. The Gaussian noise assumption holds for most sensors.

In that case, the optimization problem of interest is to compute the *k-segment mean*, the k-piecewise linear function over time that minimizes the cost (sum of squared regression distances) to the points among all possible k-segments. These types of problems are known as line simplification.

2.2 Signal Clustering

The second property states that the k segments in the signal simplification will repeat themselves, and that each segment in the signal is actually a translation in time of one of $m \ll k$ 2-dimensional segments in space. Then, the (k, m)-segment mean is a k-segment that minimizes the sum of squared regression distances to the input GPS-points, with the additional constraint that the projection of the k segments on (x, y)-space (i.e, their geographical path, ignoring time and speed) is a set of only $m \ll k$ (two-dimensional) segments. In particular, the k-segment mean is the (k, k)-segment mean of P.

2.3 Potential Applications

Our algorithms take as input a stream of n GPS-points and output an approximation to the k-segment and (k, m)-segment mean of the signal. From this output we compute the following pair of tables, which represent the semantic trajectories behind the signal:

1. The actual trajectory in time, containing $k \ll n$ rows, with fields begin time, end time, and pattern ID.
2. The $m \ll k$ segment patterns in space, with fields pattern ID, begin point, end point. Here, a point is a (latitude, longitude) location in space.

This output pair of tables is the main step for many powerful applications in learning and predicting GIS data and its semantic meaning, such as determining user activities and community behavior. We list some of them below.

Compression. While the sampling frequency of sensors can be high, storage on mobile devices and computers is bounded. In addition, transmitting data to a server is expensive, slow, and may be impossible due to communication problems. The ability to save on-line only the semantic trajectories that are represented by the signal, rather than high resolution GPS-points, mitigates these problems. Even for small datasets when storage of the signals is not an issue, analyzing the signals usually involves non-trivial optimization problems. Applying the algorithms on small semantic representations of the signals reduces their running time, which usually depends on the input signal size. This idea will be formalized as coresets.

De-noising. Consider the GPS-signal of a static user. Since GPS-error is Gaussian, the mean of the sampled point will eventually converge to the user's true location, by the law of large numbers. However, this usually takes too long in practice. The problem is even worse when the user is moving and the period of sampled time is small. On the other hand, if a user visits the same location several times, even for a quick visit, the points that correspond to these visits will be assigned the same location id. Computing the average of all the points in this cluster would then allow prediction of the correct location of the user with higher probability.

Map Generation and Matching. Navigation devices use pre-built maps for planning. The algorithms in this paper can generate such maps from the GPS-traces of cars on the street.In fact, map symbols other than road structure can be extracted from our tables. For example, when several users spend time at the same location and then increase speed, then this location may be a traffic signal. We can also generate associations such as: people that visit place a and b will usually visit place c. Such rules have natural applications in marketing and homeland security.

The output maps can also be used to perform map-matching, where the goal is to find which road a vehicle is on based on its GPS-location. The naïve approach of assigning the vehicle to the nearest road in a map usually fails because of noise (see [28]). Instead, we can match every pattern ID to a road in the map. While related papers use local consecutive sets of GPS-points, our linear simplification is based on both global optimization and clustering and may improve results.

Learning Social Networks and Communities. From our output tables, we can extract a sparse matrix A whose (i, j) entry is the number of times that user i visited place ID j. During recent decades, many algorithms have been suggested for learning such matrices. For example, applying PCA on this matrix reveals correlations between users and places. Running clustering algorithms on the rows/columns of A will allow us to find clusters of users, places, and unique people (outliers) (see [23] for a survey and algorithms).

GPS-text mining. We have implemented a new system called iDiary [12], that allows text search of users' GPS data based on the algorithms in this paper. 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. In iDiary, we apply reverse geociting on the output tables using existing APIs such as Google Maps. These services translate (latitude, longitude) coordinates into textual descriptions (exact address, places, business name, zip code, etc.) on which we can run text queries. However, the service does not de-noise the data and provides only a limited number of queries. We run queries once on the m rows of the location table, rather than on the large set of noisy input points, and then run all future queries on the local tables, allowing us to serve unlimited numbers of queries with greater accuracy.

3. RELATED WORK

3.1 Line Simplification

Line simplification is a common approach to compressing and denoising continuous signals and has already been well studied for purposes such as cartography [14] and digital graphics [5]. Streaming heuristics for this problem is suggested in [8]. Whereas many algorithms, optimal, approximate, and heuristics, have been proposed for the problem

of a maximum-distance error function (see [22] and references therein), the problem of line simplification for a sum-of-squared-distances error has largely been unexplored. To our knowledge, no algorithms with provable error bounds have been proposed since Bellman's dynamic program [7], which yields an optimal solution in $O(kn^2)$ time and $O(kn^2)$ space, although several heuristics exist that find locally optimal partitions using grid search [26] or recursive partitioning of intervals [21]. None the existing sum-of-squared-distances algorithms can run on streams of GPS data. In this paper we suggest the first $(1+\varepsilon)$-approximation for this problem that takes $O(n)$ time for arbitrary constant $\epsilon > 0$.

3.2 Signal Clustering

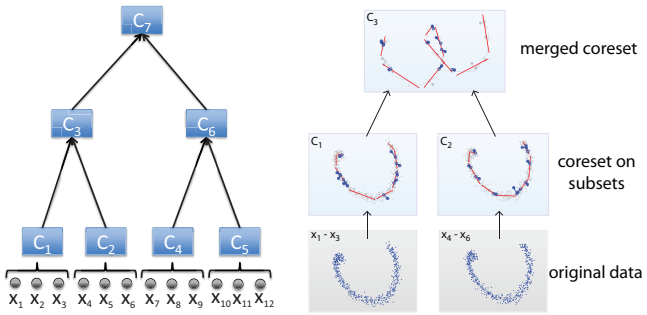
The problem of signal clustering is significantly harder, and even the heuristics for solving it are few. The main challenge of the signal clustering problem is that unlike the line simplification problem, where we cluster to the same segment only consecutive points in time, here we wish to cluster sets of GPS-points from arbitrary time intervals. We observe that the projection of the (n, m)-segment mean on $\mathbb{R}^d$ is the solution to the classic k-means clustering problem (where k is replaced by m) of computing a set of k centers (points) that minimizes the sum of squared distances from every input point in $\mathbb{R}^d$ to its closest center. This also implies that the (k, m)-segment mean, unlike the k-segment mean, is NP-hard, even for a $(1+\varepsilon)$-approximation when $\varepsilon < 1$ and m is not constant.

Unlike k-means, however, we could not find any algorithm with provable bound on its error. Actually, to our knowledge, this is the first paper that suggests a formal definition of the problem. There exists a body of signal clustering work [9, 25, 32] whose goal is to translate a time signal into a map of commonly traversed paths. In particular, Sacharidis et al. [13] process GPS streams online to generate common motion paths and identify "hot" paths. However, the majority of this work either suggest an algorithm without defining any optimization problem, or suggest quality functions based on a ad-hoc tuning variables whose meaning are not clear. Most also use a maximum-distance cost function. Furthermore, since the aim of these works are to identify trajectory patterns in space, the temporal information in the original data is lost. No current work in trajectory clustering uses the results as a method of compression for the original input trajectory. Other semantic compression techniques for signals can be found at [4, 30, 10]. Techniques for semantic annotation of trajectories can be found in [6, 27].

4. CORESETS

To solve the k-segment and (k, m)-segment mean efficiently for massive streams of points using distributed computing and existing inefficient optimization algorithms, we use coresets. A coreset is a small semantic compression of an original signal P, such that every k-segment has the same cost with respect to the original signal as to the coreset, up to $(1+\varepsilon)$-multiplicative error, for a given $\varepsilon > 0$. Coresets provide:

Shorter running time. An (possibly inefficient) optimization algorithm for a problem such as the k- or (k, m)-segment mean can be applied on a small coreset and yield a $(1+\varepsilon)$ approximation to the corresponding mean of the original (large) signal much faster. This includes constrained version of these optimization problems. In particular, for non-constant m the (k, m)-segment mean problem is NP-hard, but a small coreset for the problem can be computed in linear time. We thus provide EM-algorithm for this problem that converges to local optimum, and using the coreset are able to run it for many more iterations and initial seeds.



(a) Coreset tree (b) Construction of C_3

Figure 1: (a) Tree construction for generating coresets in parallel or from data streams. 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. [16]

Streaming computing. In the streaming setting, GPS points arrive one-by-one and it is impossible to remember the entire data set due to memory constraints. Using a merge-and-reduce technique [16], we can apply inefficient and non-streaming algorithms on coresets to obtain efficient streaming algorithms. This approach is usually used in the context of coreset and we summarize it in Fig. 1.

Parallel computing. Our coreset construction is embarrassingly parallel [17] and can be easily run on a network or a cloud. We partition the data into sets and compute coresets for each set independently. We then merge pairs of coresets to get the final coreset for the entire data set. Construction of a single coreset can also involve parallel computation for faster running time. This is because it involves mainly Euclidean distance calculations, an operation that is known to be suitable for parallel processing (see Line 5 of Algorithm 1 Line 1 of Algorithm 2).

It has been proven that in general, coresets of size $o(n)$ do not exist for the line simplification problem [2, 19], although under certain conditions, coresets have been constructed for the *maximum* distance cost function. For example, Abam et al. [2] provided the first provable line simplification streaming algorithm for signals that are monotone; unfortunately, GPS signals are typically not monotone, and their results cannot be used here. In [16], to overcome the $o(n)$ lower bound above, a different approximation function was used: ε-angular rotation from the optimal k-segment. However, this approximation allows large errors for GPS points that are approximated by long segments (such as high-ways), which is undesirable. The size of the coreset in [16] is also exponential in the dimension d of the signal, unlike the coreset in this paper.

In this paper, we construct the first small coresets that

deals with sum of squared distances, which is more suitable for GPS points as noted in Section 2.1. To avoid the $o(n)$ lower bound we used the natural assumption that the GPS-signals are discrete-time, i.e, sampled at constant frequency or a small number of frequencies. This is the first sub linear size coreset that provides a $(1+\varepsilon)$-approximation for the k-segment mean problem of discrete signals, or in general, for non-monotone signals.

5. OUR RESULTS

Our algorithms receive as input a discrete signal P of $p_1, \cdots, p_n$ of n points in $\mathbb{R}^d$ that is processed on-line (in one pass) using:

- $\log^{O(1)} n$ space (memory)
- $\log^{O(1)} n$ update time per point insertion
- $n/M \cdot \log^{O(1)} n$ overall runtime for $M \geq 1$ processors

In the above notation, we assumed that all input parameters other than n are fixed. More generally, the dependency of space and time on k is linear, and on d is polynomial.

Our results are as follows.

A (k,ε)-coreset. The size of the coreset for P is $O(k/\varepsilon^2)$ for any input integer $k \geq 1$ and error bound $\varepsilon > 0$. The coreset construction is randomized and succeeds with high (arbitrarily small constant) probability. All other algorithms are deterministic, but since we apply them on the coreset, they also succeed with high probability.

$(1+\varepsilon)$-approximation to the k-segment mean. The algorithm is based on running the existing optimal algorithm on the coreset of P.

$(1+\varepsilon)$-approximation to the (k,m)-segment mean. The overall running time for a fixed $\varepsilon > 0$ is $O(n/M) + 2^{O(m)}$ using an approximation algorithm on the coreset.

EM-Algorithm for the (k,m)-segment mean. We run an iterative (k,m)-segment mean algorithm on the coreset of P in $km \log^{O(1)} n$ time per iteration. The algorithm is based on the expected-maximization (EM) technique and each iteration returns a better approximation to the (k,m)-segment mean of the input signal. The algorithm converges to a local minimum.

Experimental Results. We implement our coreset and algorithms above and present experimental results on several datasets that demonstrate: (a) the quality and running time of the coreset constructions as a function of their size and construction time, (b) benchmarks of our algorithms and their compression ratio compared to existing heuristics, (c) new streaming capabilities and better performance for existing heuristics using our coresets.

Note on choosing k and m Since the cost function is reduced with both k and m, and since both of them are integers between 1 to n, the values of k and m can be automatically calibrated in $O(\log^2 n)$ runs of the algorithm via binary search or hill climbing techniques for finding the "elbow" (zero second derivative) of the cost function [3].

6. PROBLEM STATEMENT

We are concerned with compressing an input *discrete-time signal*, which is a set $P = \{p_1, \cdots, p_n\}$ of ordered points in $\mathbb{R}^d$, where p_i is a point that was sampled at time i for $i = 1, \cdots, n$. A crucial observation is that if all the points are lying on a line, then we can represent the signal exactly just by its endpoints p_1 and p_n. We use this observation to compress the projection of a signal on its k-segment mean.

DEFINITION 6.1 (k-SEGMENT MEAN). *For an integer $k \geq 1$, a* k-segment *in $\mathbb{R}^d$ is a piecewise linear function $f : \mathbb{R} \to \mathbb{R}^d$ of k linear segments. The* fitting cost *of f for a signal P in $\mathbb{R}^d$ is the sum of squared distances to its k segments,*

$$\text{cost}(P, f) := \sum_{p_t \in P} \|f(t) - p_t\|^2.$$

Here, $\|x - y\|$ is the Euclidean distance between the x and y in $\mathbb{R}^d$. The k-segment mean *f^* of P minimizes $\text{cost}(P, f)$ over every k-segment f in $\mathbb{R}^d$.*

For obtaining fast and practical algorithms, we will aim only for α-approximation algorithms whose cost equals to the cost of the k-segment mean f^* of P, up to a multiplicative constant factor α, i.e,

$$\text{cost}(P, \tilde{f}) \leq \alpha \cdot \text{cost}(P, f^*), \text{ and } \tilde{f} \text{ is a } k\text{-segment}$$

In order to decide which points should be represented in the coreset, we usually need some rough approximation of the k-segment mean. Unfortunately, existing results do not provide even α-approximation that can be computed in time sub-cubic in n, for any constant α. Instead, we provide an algorithm that computes the following weaker approximation for the k-segment mean for constructing the coreset in $O(n)$ time.

DEFINITION 6.2 ((α,β)-APPROXIMATION). *For $\alpha, \beta > 0$, an (α,β)-approximation for the k-segment mean of P is a βk-segment $b : \mathbb{R} \to \mathbb{R}^d$ whose cost equals to the cost of the k-segment mean f^* of P, up to a multiplicative constant factor α, i.e,*

$$\text{cost}(P, b) \leq \alpha \cdot \text{cost}(P, f^*), \text{ and } b \text{ is a } \beta k\text{-segment}$$

Note that we compare b to the optimal k-segment mean f^*, and not to the (βk)-segment mean of P. Unlike α-approximations, there are (α,β)-approximation for $\alpha < 1$. For example, using $n-1$ segments ($\beta = (n-1)/k$) we can trivially get zero cost ($\alpha = 0$) for P by connecting every pair of consecutive points in P by a segment.

In order to store an efficient representation of a signal as a coreset, we introduce weighted signals, where more important points are given larger weights. Unlike existing coresets, in order to approximate k-segments we had to introduce negative weights.

DEFINITION 6.3 ((k,ε)-CORESET). *Let $k \geq 1$ be an integer and $\varepsilon > 0$. A weighted signal (C, w) where $w : C \to \mathbb{R}$ is a* (k,ε)-coreset *for a discrete signal P if for every k-segment f in $\mathbb{R}^d$,*

$$(1-\varepsilon)\text{cost}(P, f) \leq \text{cost}_w(C, f) \leq (1+\varepsilon)\text{cost}(P, f).$$

where

$$\text{cost}_w(C, f) := \sum_{c_t \in C} w_c \cdot \|f(t) - c_t\|^2.$$

To exploit the data redundancy of sequences of points that are repeated over time we define the (k,m)-segment mean.

DEFINITION 6.4 ((k,m)-SEGMENT MEAN). *For a pair of integers $k \geq m \geq 1$, a* (k,m)-segment *is a k-segment f in $\mathbb{R}^d$ whose projection $\{f(t) \mid t \in \mathbb{R}\}$ is a set of only m segments*

in $\mathbb{R}^d$. Given a signal P in $\mathbb{R}^d$, its (k,m)-segment mean f^* *minimizes* $\text{cost}(P,f)$ *among every possible (k,m)-segment f in $\mathbb{R}^d$.*

7. (α,β)-APPROXIMATION

The exact k-segment mean can be computed in $O(kn^2)$ time as described in [7]. Using the map-and-reduce technique, we will need to apply this algorithm (as part of the coreset construction) only on small data sets of size $k\log^{O(1)} n$, so the total running time will be $k^3\log^{O(1)} n$.

However, in practice and in our experiments the value of k may be large, so we wish to compute the coreset in time that is linear in both n and k. To this end, we suggest an (α,β)-approximation for the k-segment mean that takes only $O(n)$ time, shown in Algorithm 1. Informally, given k and the signal P, we partition P into $4k$ consecutive subsequences and find the 1-segment mean of each. Then we remove the k subsequences that have the lowest cost 1-segments means and iterate on the remaining $\frac{3}{4}$ of the signal. The output of the algorithm is the 1-segment means of the subsequences we removed at each iteration, for a total of $O(k\log n)$ segments.

The algorithm uses the following algorithm for finding the 1-segment mean as a sub-routine.

LEMMA 7.1. *Let P be a signal of n points in $\mathbb{R}^d$. The 1-segment mean of P can be computed in $O(n)$ time.*

PROOF. Let $u,v\in\mathbb{R}^d$, and let $\ell:\mathbb{R}\to\mathbb{R}^d$ be a 1-segment $\ell(t)=u+vt$ which minimizes

$$\text{cost}_w(P,\ell)=\min_{u,v\in\mathbb{R}^d}\sum_{p_t\in P} w(p)\|u+vt-p_t\|^2. \qquad (1)$$

By letting $X^T := [u \mid v]$, for an appropriate matrices A,P_+,B and N we have that

$$\text{cost}_w(P,\ell)=\|AX-P_+\|_F-\|BX-N\|_F, \qquad (2)$$

where $\|\cdot\|_F$ denotes the Frobenius norm (sum of squared entries). Taking the derivative of the right-hand side of (2) with respect to X yields $X=M^{-1}Y$, where $M=A^TA-B^TB$ and $Y=P_+^TA-N^TB$. Hence, computing X which correspond to the 1-segment mean of P takes $O(n)$ time. □

THEOREM 7.2. *Let P be a discrete signal of n points in $\mathbb{R}^d$, and $k\geq 1$ be an integer. Let f be the output of the algorithm* BICRITERIA*(P,k) (Algorithm 1). Then f is an (α,β)-approximation for the k-segment mean of P, with $\alpha,\beta=O(\log n)$. The running time of the algorithm is $O(n)$.*

PROOF. Running time: One-quarter of the points of P are removed in every iteration so there are $O(\log n)$ iterations. In every iteration, we compute the 1-segment mean of k sets, which takes $O(n)$ time using the previous lemma. Hence, the overall running time is $\sum_i n(1-1/4)^i=O(n)$.

Correctness: In every one of the $O(\log n)$ iterations we add k segments to the output, which yields $\beta=O(\log n)$. In order to bound α, observed that if we partition P into $4k$ segments, half of these segments will not contain any of the $2k$ endpoints of the k-segment mean f^* of P. Hence, the cost c of these segments to f^* is at least their sum of costs to their 1-segment mean. Since our algorithm chooses the k segments with the minimum sum of costs to their 1-segment mean, this sum is less than c. By repeating this argument recursively for the $\alpha=O(\log n)$ iterations we obtain a cost that is at most α times the optimal k-segment mean of P. □

Algorithm 1: BICRITERIA(P,k)

Input: A discrete signal P of n points in $\mathbb{R}^d$, and an integer $k\geq 1$.
Output: (α,β)-approximation b to the k-segment of P, where $\alpha,\beta=O(\log n)$.

$i\leftarrow 1$; $P_1\leftarrow P$; $F\leftarrow\emptyset$
while $|P_i|>4k$ **do**
  `/* Partition` P_i `into` $4k$ `consecutive sequences */`
  for $j\leftarrow 1$ *to* $4k$ **do**
    $P_{i,j}\leftarrow\{p_{(j-1)(4n/k)+1},\cdots,p_{jn/4k}\}$
    $f_{i,j}\leftarrow$ A 1-segment mean of $P_{i,j}$
    `/* see Lemma 7.1 for a suggested algorithm */`
  $P_{i,j_1},\cdots,P_{i,j_k}\leftarrow$ The k sets $P_{i,j}$ with the smallest $\text{cost}(P_{i,j},f_{i,j})$ over $j=1,\cdots,4k$
  $P_{i+1}\leftarrow P_i\setminus(P_{i,j_1}\cup\cdots\cup P_{i,j_k})$
  $F\leftarrow F\cup\{f_{i,j_1},\cdots,f_{i,j_k}\}$
  $i\leftarrow i+1$
$f_i\leftarrow 4k$-segment whose endpoints are the points of P_i.
$F\leftarrow F\cup\{f_i\}$
$b\leftarrow k\cdot|F|$-segment corresponding to the segments in F
return b

8. CORESET

Algorithm 2: CORESET(P,k,ε,δ)

Input: A discrete signal $P=\{p_1,\cdots,p_n\}$ in $\mathbb{R}^d$, an integer $k\geq 1$, and $\varepsilon,\delta\in(0,1)$.
Output: With probability at least $1-\delta$, a (k,ε)-coreset (C,w) for P.

Compute an (α,β)-approximation $\tilde{f}$ for the k-segment mean of P, where $\alpha,\beta=O(\log n)$
`/* See Theorem 7.2 for suggested algorithm. */`
for $i\leftarrow 1$ *to* n **do**
  Set $\sigma(p_i)\leftarrow\dfrac{\|p_i-\tilde{f}(i)\|^2}{\sum_{i=1}^n\|p_i-\tilde{f}(i)\|^2}$
  `/* The dominator is` $\text{cost}(P,\tilde{f})$ `(Def. 6.1) */`
$c\leftarrow$ sufficiently large constant that is determined in the proof of Theorem 8.1.
Pick a non-uniform random sample S of $10c\alpha^2k\ln(1/\delta)/\varepsilon^2$ points from P, for every $q\in S$ and $p\in P$ we have $q=p$ with probability $\sigma(p)$
Set $P'\leftarrow\left\{\tilde{f}(i)\mid 1\leq i\leq n\right\}$ `/* project` P `on` $\tilde{f}$ `*/`
Set $S'\leftarrow\left\{\tilde{f}(s)\mid p_s\in S\right\}$ `/* project` S `on` $\tilde{f}$ `*/`
Set $C\leftarrow P'\cup S\cup S'$
for *each* $p_s\in S$ **do**
  $w(p_s)\leftarrow\frac{1}{|S|\cdot\sigma(p_s)}$
  $w(\tilde{f}(s))\leftarrow -w(p_s)$
for *each* $p\in P'$ **do**
  $w(p)\leftarrow 1$
return (C,w)

We use the bicriteria approximation from the previous section to compute a coreset of size that depends polynomi-

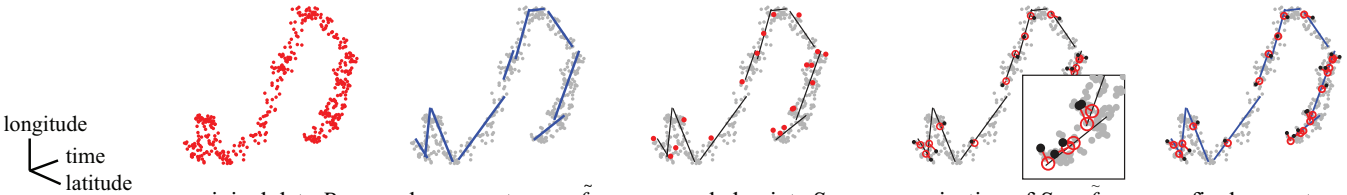


Figure 2: The coreset construction in Algorithm 2. At each step, the new entities are shown in color (segments in blue, points in red) and pre-existing structures are shown in gray

ally on $\log n$. In the following corollary we will show how to construct a coreset of size independent of n.

THEOREM 8.1. *Let P be a signal in $\mathbb{R}^d$, $\delta, \varepsilon \in (0, 1.2)$ and $k \geq 1$ be constants where $\delta > \varepsilon/2$. Let (C, w) be the output of a call to* CORESET$(P, k, \varepsilon, \delta)$ *(Algorithm 2). Then, with probability at least $1 - \delta$, the pair (C, w) is a (k, ε)-coreset for P. The coreset (C, w) can be computed in $O(n)$ time.*

PROOF. We use the notation and variables that are defined in Algorithm 2. For every $p_i \in P$ we denote by p'_i its corresponding projection in P'. Fix a k-segment f in $\mathbb{R}^d$. The error of approximating the weighted distances to S from C rather than P is:

$$\begin{aligned}
&\left| \sum_{p_i \in P} \|f(i) - p_i\|^2 - \sum_{p_i \in C} w(p_i)\|f(i) - p_i\|^2 \right| \\
&= \sum_{p_i \in P} \left(\|f(i) - p_i\|^2 - \|f(i) - p'_i\|^2 \right) \qquad (3) \\
&\quad - \sum_{p_i \in C} w(p_i)\left(\|f(i) - p_i\|^2 - \|f(i) - p'_i\|^2 \right).
\end{aligned}$$

Denote the contribution of $p_i \in P$ to the desired cost as

$$\Delta(p_i) := \|f(i) - p_i\|^2 - \|f(i) - p'_i\|^2.$$

Let Q denote the points in P such that $|\Delta(p_i)| > \varepsilon f^2(p)$. With probability at least $1-\delta$, we have $\sigma(p) \geq \delta/(10|S|)$ for every $p \in S$. Suppose that this event indeed occurs. Hence,

$$w(p) = \frac{1}{|S| \cdot \sigma(p)} \leq \frac{10}{\delta}.$$

Summing this over the points of $P \setminus Q$ yields

$$\begin{aligned}
&\left| \sum_{p_i \in P \setminus Q} \Delta(p_i) - \sum_{p_i \in C \setminus Q} w(p_i)\Delta(p_i) \right| \\
&\leq \left| \sum_{p_i \in P \setminus Q} \varepsilon f^2(p) \right| + \left| \sum_{p_i \in C \setminus Q} \frac{10}{\delta} \cdot \varepsilon f^2(p) \right| \\
&\leq \frac{10\varepsilon \text{cost}(P, f)}{\delta} \leq 5\varepsilon \text{cost}(P, f).
\end{aligned}$$

It is left to bound the error for the points in Q.

Using the framework from [15] for constructing coresets, we only need to bound the total sensitivity T of the points in Q to the family of k-segments in $\mathbb{R}^d$ and the VC-dimension v of this family. The desired sample $|S|$ then should be $cv \ln(1/\delta) T^2/\varepsilon^2$ as proved in [15]. Intuitively, the VC-dimension corresponds to the number of parameters needed to define a k-segment. It is easy to verify that the VC-dimension of the family of k-segments is $O(k)$; see the formal proof for the family of k lines in [15].

Suppose that we are given a query k-segment of distance at most 1 to each query point, and that the sum of squared distances is *cost*. By Hoeffding inequality, approximating the sum of squared distances from the n points to this k-segment by a uniform random sample of size $O(1/\varepsilon^2)$ would yield an additive error of $O(\varepsilon n)$, with high probability. The theory of PAC-learning generalizes this results for approximating simultaneously *every* k-segment, using $O(v/\varepsilon^2)$ points. The result in [15] generalizes this result to obtain ε*cost* additive error, i.e, $(1 + \varepsilon)$ multiplicative error, by re-scaling each input point p by a weight that it proportional to its sensitivity $1/s(p)$, and duplicate it in number of copies that is proportional to $s(p)$. While the total cost to the query segment is the same, the maximum contribution of a specific point is bounded. Taking a uniform sample from such a set of duplicated point is of course the same as taking a non-uniform sample from the original set.

The sensitivity of a point $p \in Q$ is a number $s(p) \in [0, 1]$ of how important it is to take $p \in Q$ (and its projection p') in the coreset. For example, if there is a k-segment f that is close to all other points but far from p, then we must choose p in the coreset, otherwise f will not be approximated well.

Let $\Delta'(p) = \Delta(p)$ if $p \in Q$ and 0 otherwise. Since we wish to approximate $\sum_{p \in P} \Delta'(p)$ with an additive error $\varepsilon \text{cost}(P, f)$, the sensitivity of $p \in P$ is

$$s(p) := \max_f \frac{|\Delta'(p)|}{\text{cost}(P, f)},$$

where the maximum is over every k-segment f in $\mathbb{R}^d$. We now bound this sensitivity. Since $s(p) = 0$ for $p \in P \setminus Q$, we assume $p \in Q$.

Using the triangle inequality, it can be proved that for every $\tau \in (0, 1/2)$

$$\Delta(p) \leq \frac{3\tilde{f}^2(p)}{\tau} + 2\tau f^2(p).$$

See [20]. Hence,

$$s(p) \leq \frac{3\tilde{f}^2(p)}{\tau \text{cost}(P, f)} + \frac{2\tau f^2(p)}{\text{cost}(P, f)}.$$

Since $p \in Q$ we have $\Delta(p_i)/\varepsilon > f^2(p)$. Hence, for $\tau < \varepsilon/4$,

$$s(p) \leq \frac{3\tilde{f}^2(p)}{\tau \text{cost}(P, f)} + \frac{2\tau|\Delta(p_i)|}{\varepsilon \text{cost}(P, f)} \leq \frac{3\tilde{f}^2(p)}{\tau \text{cost}(P, f)} + \frac{s(p)}{2}.$$

So, for $\tau > \varepsilon/6$

$$s(p) \leq \frac{6\tilde{f}^2(p)}{\tau \text{cost}(P, f)} \leq \frac{6\alpha\tilde{f}^2(p)}{\tau \text{cost}(P, \tilde{f})} \leq \frac{36\alpha\tilde{f}^2(p)}{\varepsilon \text{cost}(P, \tilde{f})}.$$

Summing this over every $p \in P$ yields that the total sensitivity is $T = \sum_{p \in Q} s(p) \leq \frac{36\alpha}{\varepsilon}$. □

COROLLARY 8.2. *A (k, ε)-coreset for P of size $O(k)$ can be constructed in $O(n)$ expected time for arbitrary small constant $\varepsilon > 0$.*

PROOF. We use ideas from [20]. First we compute a $(k, 1/2)$-coreset C of size $O(\log^2 n)$ by applying Algorithm 2 with $\varepsilon = 1/2$. Using C and the Bellman's optimal algorithm [7], we compute the optimal k-segment of C. This yields an (α, β)-approximation f^* for the k-segment of P with $\alpha = \beta = O(1)$. We then compute a new coreset C' using Algorithm 2, using f^* as the (α, β)-approximation in the first line of the algorithm. □

By applying the optimal algorithm of Bellman [7] that takes time $O(kn^2)$ on the coreset C of P instead of P, we obtain a $(1+\varepsilon)$-approximation for the k-segment mean of P.

THEOREM 8.3. *Let P be a discrete signal of n points in $\mathbb{R}^d$, and $k \geq 1$ be an integer. A $(1+\varepsilon)$-approximation to the k-segment mean of P can be computed, with high probability, in $O(n + k^3)$ time for arbitrary small $\varepsilon > 0$.*

9. (K, M)-SEGMENT MEAN

Similarly to the approach of the previous section, we first suggest a $(1+\varepsilon)$-approximation non-streaming algorithm for the (k, m)-segment mean that takes $n^{O(m)}$ time. Then we apply the algorithm on our coreset from the previous section to reduce the space to $O(\log n)$ and the running time to $O(n) + 2^{O(m)}$. The algorithm is simply based on exhaustive search and due to space limitation we leave the proof for the full version.

THEOREM 9.1. *Let P be a signal of n points in $\mathbb{R}^d$, $k \geq m \geq 1$ be integers, and $\varepsilon > 0$ be an arbitrary small constant. Then a $(1+\varepsilon)$-approximation to the (k, m)-segment mean of P can be computed in $n^{O(m)}$ time.*

Applying the algorithm from the proof of Theorem 9.1 on the coreset of size $O(k)$ from Theorem 8.1 yields the following theorem.

THEOREM 9.2. *Let P be a discrete signal in $\mathbb{R}^d$, $\varepsilon > 0$ and $k \geq m \geq 1$ be a pair of constant integers. A $(1+\varepsilon)$-approximation to the (k, m)-segment mean of P can be computed in $O(n) + 2^{O(m)}$ time.*

Practical Heuristic for large m.

THEOREM 9.3. *Let P be a signal in $\mathbb{R}^d$, and $i_{\text{end}} \geq 1$, $k \geq m \geq 1$ be integers. Then the running time of Algorithm 3 is $O(n) + O(i_{\text{end}})$. Moreover, the cost of the output (k, m)-segment reduces with i_{end} and converges to a local minimum with $i \to \infty$.*

10. EXPERIMENTAL RESULTS

Running algorithms on our experimental data sets, such as Bellman's optimal k-segment mean, would require literally Terabytes of RAM and years of computation time. Using our coreset, we were able to run such algorithms on millions of GPS-points in only few hours on our laptop, using naïve MATLAB implementations.

Algorithm 3: KM-SEGMENT$(P, k, m, i_{\text{end}})$

Input: A signal $P = \{p_1, \cdots, p_n\} \subseteq \mathbb{R}^d$, two integers $k \geq m \geq 1$, and number i_{end} of iterations.
Output: A (k, m)-segment $f_{i_{\text{end}}}$.

```
/* Initialization:                                              */
```
$f \leftarrow$ A $(1+\varepsilon)$-approximation to the k-segment mean of P `/* See Theorem 8.3 */`
$Q \leftarrow$ The set of concatenated k pairs $(q_1 \mid q_2), \ldots, (q_{2k-1} \mid q_k) \in \mathbb{R}^{2d}$ of the segments endpoints of $\{f(i) \mid 1 \leq i \leq n\}$
$\{Q_1, \cdots, Q_m\} \leftarrow$ The partition of Q into its m-means (points) clusters
for $j \leftarrow 1$ *to* m **do**
  $P_{1,j} \leftarrow$ the points of P that are on a segment $\overline{p, q} \in Q_j$
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

In this section, we compare the performance of these algorithms in terms of quality of the signal approximation, runtime, and storage cost. More precisely, given a specific constraint on the size k and m of the output signal, we measure the error cost (sum of squared distances) to the original signal, and the running time of the algorithm. We measured all times in seconds on a 4-core Intel Xeon CPU running at 2.67GHz with 6GB of RAM.

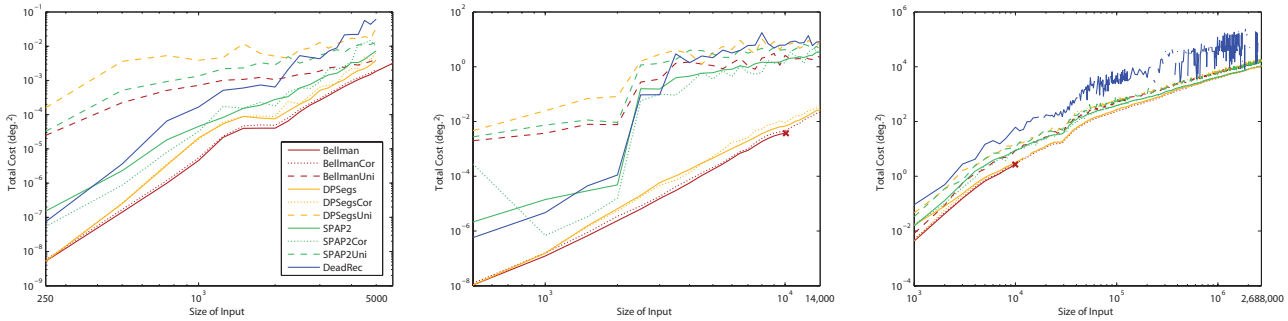
Our results demonstrate large reductions in storage size by using the proposed algorithms without much loss of quality. We further find that applying these algorithms, both optimal and heuristic, to our coreset yields significant speedup. Our (k, m)-algorithm produces maps that approximate the input trajectory better than existing trajectory clustering algorithms.

Datasets. We used three sets of GPS trajectories for our experiments: (DRL) a trajectory, consisting of 5,811 points, pre-planned and executed by the authors,(SIGSPATIAL) the 14,436-point training dataset provided to participants in the ACM SIGSPATIAL Cup 2012 [1], and (CAB) 2,688,000 points collected by 121 taxicabs in San Francisco [29].

10.1 Compression

In the first set of experiments, we compare the storage-quality tradeoff that comes with approximating a signal by line segments. We ran k-segment and (k, m)-segment mean algorithms on the DRL dataset and compared the quality of the approximation for various sizes of output.

Algorithms. The algorithms we tested are: optimal Bellman [7], MATLAB's SPAP2 (from the curve fitting tool-



(a) Total Cost, DRL (b) Total Cost, SIGSPATIAL (c) Total Cost, CAB

Figure 3: Mean cost of Bellman's, Douglas-Peucker, spap2, and dead-reckoning algorithms for the (a) DRL, (b) SIGSPATIAL, and (c) CAB datasets. Data points were processed in batches of 1000, and 3 trial runs were performed. Algorithms run on entire input are shown as solid lines, on coresets (COR) as dotted lines, and on uniform random samples (UNI) as dashed lines. Note that Bellman's was not run on the entire SIGSPATIAL or CAB datasets for lack of memory. The sizes of the coreset were (DRL) $k = 100, |S| = 100$ and (SIGSPATIAL, CAB) $k = 200, |S| = 100$. 30-segment means were calculated on the DRL dataset, and 100-segment means on SIGSPATIAL and CAB. Three tests were performed for each algorithm and dataset.

Size, kB		Total Cost, deg.^{2}			
(% of orig)	(k,m)	Bellman	DPSegs	SPAP2	DeadRec
0.2 (0.14)	0.2667	0.3530	0.3900	0.5033	2.1738
0.4 (0.29)	0.0293	0.0331	0.0820	0.1156	0.8364
0.8 (0.57)	0.0069	0.0075	0.0152	0.0261	0.2519
1.0 (0.72)	0.0048	0.0048	0.0093	0.0176	0.1528
1.5 (1.08)	0.0022	0.0017	0.0039	0.0058	0.0592
2.0 (1.43)	0.0012	0.0010	0.0020	0.0033	0.0290
2.5 (1.79)	0.0007	0.0006	0.0012	0.0024	0.0168

Table 1: Comparison of size (in bytes) of the k- or (k,m)-segments representation and their mean quality. The original total size of the dataset (DRL) was 139.4 kB.

box), and the popular Douglas-Peucker (DPSegs) [14] and dead-reckoning (DeadRec) [18] heuristics. To improve their results, we modified spap2 and DPSegs to support disconnected segments. We also changed the cost function of DPSegs from maximum distance to sum of squared distances. Our version of DPSegs consists of the following steps: Beginning with a partitioning that consists of the entire data set, 1) for the partition with the highest cost, compute the optimal 2-segment mean, 2) divide that partition into two parts according to the result, and 3) repeat until the dataset has been partitioned into k. Of these four algorithms, only DeadRec can inherently be applied in a streaming context.

Results. The results are shown in Table 1. As an optimal algorithm, Bellman produces the best storage-to-cost ratio out of all k-segment algorithms. Interestingly, it also outperforms the (k,m)-segment mean EM algorithm for the larger output sizes. We expected the (k,m)-segment mean to yield better storage-to-cost ratios since the DRL dataset contains multiple repetitions of the same subtrajectory and suspect that the larger costs stem from the fact that Bellman is an optimal algorithm and KM-SEGMENT is not. Notice that KM-SEGMENT outperformed all of the k-segment mean heuristics.

10.2 Advantage of Coresets: k-Segment Mean

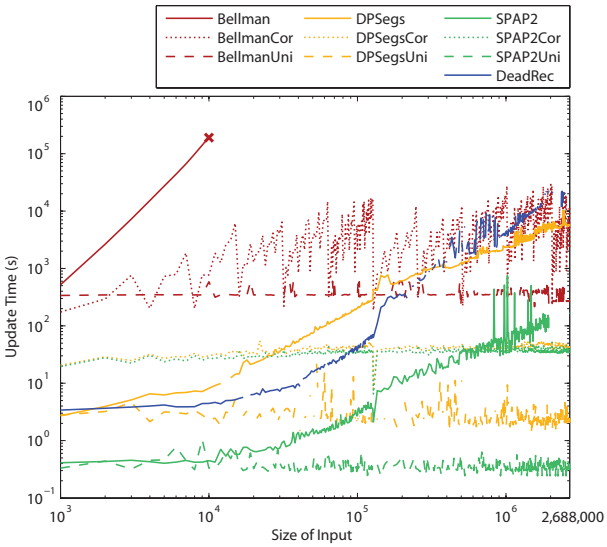
The experiments in Section 10.1 could be performed on the entire DRL dataset since it was relatively small. For larger data sets, the computation time becomes prohibitively large. In the following experiments, we tested k-segment mean algorithms on our proposed coreset to verify our claims of shorter running time without loss of quality.

Method. We compare the results of running k-segment mean algorithms on P compared to its coreset Cor and to a uniform random sampling Uni from P. We used $k = 30$ for the DRL dataset and $k = 100$ for the SIGSPATIAL and CAB datasets. The input to the algorithms was a stream of subsets of size 1000 from P. Coresets and uniform samples, each of size $S = 600$ points, were constructed from each batch. The algorithms that do not support streaming (all except DeadRec) were run on the whole set that was seen so far, on the coreset, and on the uniform random sample.

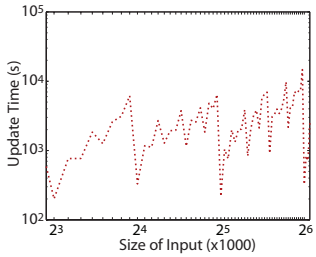
Results. The results are shown in Fig. 3. Update time for the trials, which were approximately the same for all datasets, are shown in Fig. 4. For Bellman's, Douglas-Peucker, and spap2, applying the algorithm to the coreset yields little cost difference from applying it to the input points. However, using the coreset allows a relatively constant update time with increasing input size, while the update time required to apply the algorithm directly to the input points increases with the total input size. Actually, for the coreset, update times vary with the coreset size. In Fig. 4(b) we see evidence of the updating mechanism of the streaming tree, where update time (coreset size) reaches a minimum when input sizes are powers of two, as described in Fig 1. The uniform random sample, which is chosen in sub-linear time, is significantly faster than the coreset, but the cost of the random sample is consistently higher than both using the entire input and using the coreset.

10.3 (k,m)-Segment Mean

Method. A (k,m)-segments approximation to the input trajectory can also be computed ona coreset, as described



(a)



(b) Close-up

Figure 4: Mean update time over three tests of Bellman's, Douglas-Peucker, spap2, and dead-reckoning algorithms on the input datasets. The plots looked similar for all three datasets.

in Section 9. We analyzed the quality and size of the output for coresets of the DRL and SIGSPATIAL datasets.

Results. Fig. 5 shows example maps created using the KM-SEGMENT algorithm as compared to computing only k-segments. We also compared the output maps to those created by the algorithm proposed by Lee et al. in [25]; this algorithm also clusters subtrajectories but approximates each cluster by a linear spline rather than a single segment and does not attempt to perform compression. Furthermore, the time information from the input trajectory is not preserved. We ran Lee et al.'s algorithm with distance weights all equal to 1. For comparison with the (k, m)-segments-mean and k-segments-mean, we used the the resulting number of clusters from running Lee et al. as m, and the number of partitions as k.

We see that a (k, m)-segment mean with $m < k$ will yield a larger cost than the corresponding k-segment mean, but will have a smaller storage size. Both algorithms performed much better than Lee et al. in terms of storage, since Lee et al. does not compress the data.

Qualitatively, we observe that trajectory clustering produces a cleaner map by decreasing variation in the directions of segments that overlap in space. We also observe that taking a global approach to the k-segments or (k, m)-segments approximation produces a map that approximates the the input trajectory much better than an approach such as Lee et al.'s, which makes local clustering decisions. In particular, our algorithms were able to better approximate areas of high curvature, such as the tail in the bottom right of the DRL trajectory or the square in the center of the SIGSPATIAL trajectory.

11. CONCLUSION

In this paper, we present a novel coreset for the k-segment mean and introduce the problem of finding a (k, m)-segment mean. By running existing algorithms on our coreset, we provide more efficient, streamable algorithms that yield high-quality semantic compressions of the original input data. We demonstrate large decreases in runtime for large datasets without significant loss of quality and produce maps from users' trajectories. In the future, we intend to extend our work to design coresets for the (k, m)-segments mean problem, with size dependent only on m. Our work has many potential applications in map generation and matching, activity recognition, and analysis of social networks.

12. ACKNOWLEDGEMENT

Support for this work has been provided in part by the Office of Naval Research under grant numbers ONR-MURI Award N00014-09-1-1051 and ONR-MURI Award N00014-09-1-1031, by the Singapore-MIT Alliance on Research and Technology under the Future of Urban Mobility project, and by Google. We are grateful for this support.

13. REFERENCES

[1] ACM SIGSPATIAL Cup training data set, url:http://depts.washington.edu/giscup/trainingdata, 2012.

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

[3] P. K. Agarwal and N. H. Mustafa. k-means projective clustering. In A. Deutsch, editor, *PODS*, pages 155–165. ACM, 2004.

[4] L. Alvares, V. Bogorny, B. Kuijpers, J. MACEDO, B. Moelans, and A. Vaisman. A model for enriching trajectories with semantic geographical information. 2007.

[5] T. Asano and N. Katoh. Number theory helps line detection in digital images. In *Proc. 4th Ann. Intl. Symposium on Algorithms and Computing*, pages 313–322, 1993.

[6] D. Ashbrook and T. Starner. Using gps to learn significant locations and predict movement across multiple users. *Personal and Ubiquitous Computing*, 7(5):275–286, 2003.

[7] R. Bellman. On the approximation of curves by line segments using dynamic programming. *Commun. ACM*, 4(6):284, 1961.

[8] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal*, 15(3):211–228, 2006.

[9] G. Chen, B. Chen, and Y. Yu. Mining Frequent Trajectory Patterns from GPS Tracks. In *2010 Intl. Conf. on Comp. Intel. and Soft. Eng.*, 2010.

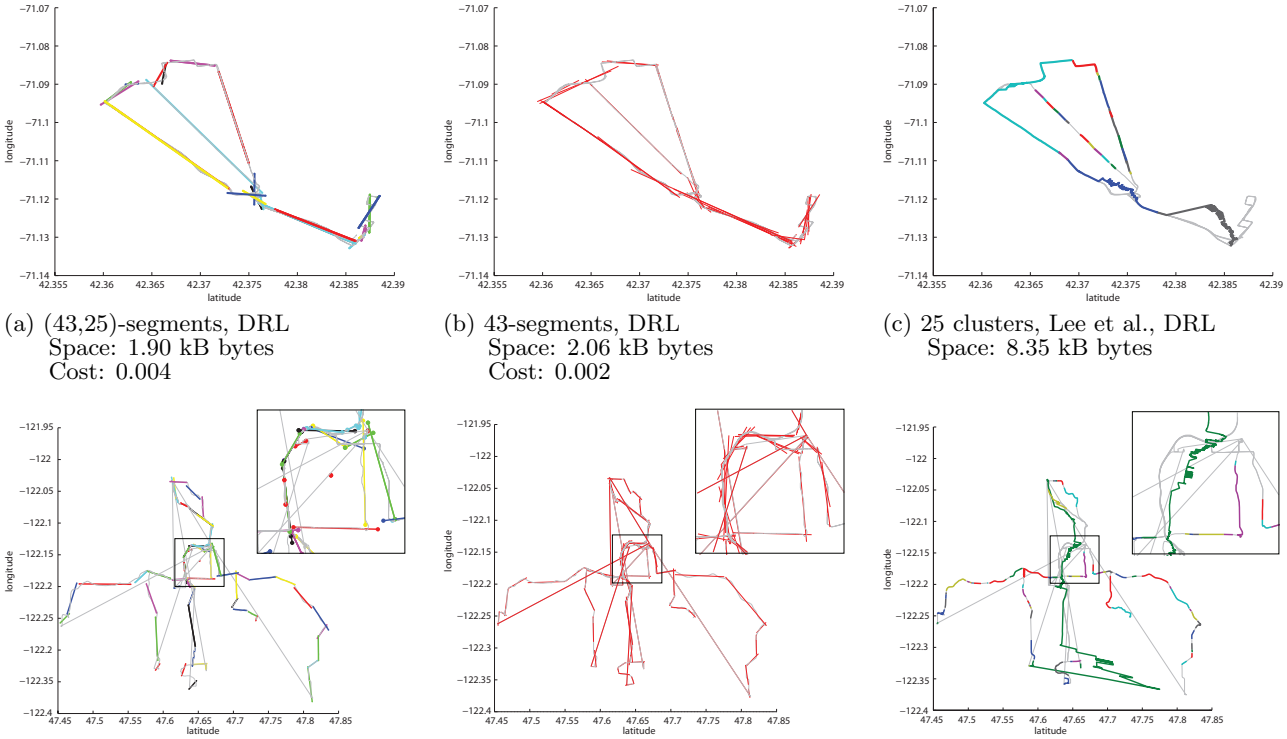
[10] M. Chen, M. Xu, and P. Franti. Compression of gps trajectories. In *Data Compression Conference (DCC), 2012*, pages 62–71. IEEE, 2012.

[11] S. Costello. iPhone sensors, url:http://ipod.about.com/od/ipodiphonehardwareterms/qt/iphone-sensors.htm.

[12] A. S. D. R. D. Feldman, C. Sung. My Long and Winding Road: From Big (GPS) Data to a Searchable Diary. Technical report, 2012.

[13] D. Sacharidis, et al. On-line discovery of hot motion paths. In *Proc. the 11th Intl. Conf. on Extending DB Technology*, pages 392–403, 2008.

[14] 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.



(a) (43,25)-segments, DRL
Space: 1.90 kB bytes
Cost: 0.004

(b) 43-segments, DRL
Space: 2.06 kB bytes
Cost: 0.002

(c) 25 clusters, Lee et al., DRL
Space: 8.35 kB bytes

(d) (91,70)-segments, SIGSPATIAL
Space: 4.82 kB
Cost: 0.193

(e) 91-segments, SIGSPATIAL
Space: 4.37 kB
Cost: 0.031

(f) 70 clusters,Lee et al.,SIGSPATIAL
Space: 27.31 kB

Figure 5: Example maps produced for (a-c) DRL and (d-f) SIGSPATIAL datasets using (DRL) $k = 43, m = 25$ and (SIGSPATIAL) $k = 91, m = 70$. For the algorithms that perform trajectory clustering, different clusters are indicated by different colors. The original trajectory is shown in light gray. The amount of space required to store the solution and, where appropriate, the cost of the solution are also given. (d-f) The box at in the upper-right corner shows a close-up of area near the center of the signal.

[15] 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.

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

[17] I. Foster. *Designing and building parallel programs*, volume 95. Addison-Wesley Reading, MA, 1995.

[18] G. Trajcevski, et al. On-line data reduction and the quality of history in moving objects databases. In *Proc. 5th ACM Intl. Workshop on Data Engineering for Wireless and Mobile Access*, pages 19–26, 2006.

[19] S. Har-Peled. Coresets for discrete integration and clustering. *Proc. 26th Intl. Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pages 33–44, 2006.

[20] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proc. 36th Ann. ACM Symp. on Theory of Computing (STOC)*, pages 291–300, 2004.

[21] D. M. Hawkins. Point estimation of the parameters of piecewise regression models. *Applied Statistics*, pages 51–57, 1976.

[22] N. Hönle, M. Grossmann, S. Reimann, and B. Mitschang. Usability analysis of compression algorithms for position data streams. In *Proc. 18th ACM SIGSPATIAL Intl. Conf. on Advances in Geographic Information Systems*, pages 240–249, 2010.

[23] J. Kleinberg, et al. The web as a graph: Measurements, models, and methods. *Computing and Combinatorics*, pages 1–17, 1999.

[24] S. Johansen and K. Juselius. Maximum likelihood estimation and inference on cointegration with applications to the demand for money. *Oxford Bulletin of Economics and statistics*, 52(2):169–210, 1990.

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

[26] P. M. Lerman. Fitting segmented regression models by grid search. *Applied Statistics*, pages 77–84, 1980.

[27] L. Liao, D. Fox, and H. Kautz. Extracting places and activities from gps traces using hierarchical conditional random fields. *The International Journal of Robotics Research*, 26(1):119–134, 2007.

[28] P. Newson and J. Krumm. Hidden Markov map matching through noise and sparseness. In *Proc. 17th ACM SIGSPATIAL Intl. Conf. on Advances in Geographic Information Systems*, GIS '09, pages 336–343, New York, NY, USA, 2009. ACM.

[29] M. Piorkowski, N. Sarafijanovic-Djukic, and M. Grossglauser. CRAWDAD data set epfl/mobility (v. 2009-02-24), url:http://crawdad.cs.dartmouth.edu/ epfl/mobility, Feb. 2009.

[30] K. Richter, F. Schmid, and P. Laube. Semantic trajectory compression: Representing urban movement in a nutshell. *Journal of Spatial Information Science*, (4):3–30, 2012.

[31] F. Van Diggelen. Innovation: Gps accuracy-lies, damn lies, and statistics. *GPS WORLD*, 9:41–45, 1998.

[32] J. Ying, W. Lee, T. Weng, and V. Tseng. Semantic trajectory mining for location prediction. In *Proc. 19th ACM SIGSPATIAL Intl. Conf. on Advances in Geographic Information Systems*, pages 34–43, 2011.