An Effective Coreset Compression Algorithm for Large Scale Sensor Networks

Dan Feldman
MIT
Computer Science and AI Lab,
32 Vassar Street,
Cambridge, Massachusetts
dannyf@csail.mit.edu

Andrew Sugaya
MIT
Computer Science and AI Lab,
32 Vassar Street,
Cambridge, Massachusetts
asugaya@csail.mit.edu

Daniela Rus
MIT
Computer Science and AI Lab,
32 Vassar Street,
Cambridge, Massachusetts
rus@csail.mit.edu

ABSTRACT

The wide availability of networked sensors such as GPS and cameras is enabling the creation of sensor networks that generate huge amounts of data. For example, vehicular sensor networks where in-car GPS sensor probes are used to model and monitor traffic can generate on the order of gigabytes of data in real time. How can we compress streaming high-frequency data from distributed sensors? In this paper we construct *coresets* for streaming motion. The coreset of a data set is a small set which approximately represents the original data. Running queries or fitting models on the coreset will yield similar results when applied to the original data set.

We present an algorithm for computing a small coreset of a large sensor data set. Surprisingly, the size of the coreset is independent of the size of the original data set. Combining map-and-reduce techniques with our coreset yields a system capable of compressing in parallel a stream of $O(n)$ points using space and update time that is only $O(\log n)$. We provide experimental results and compare the algorithm to the popular Douglas-Peucker heuristic for compressing GPS data.

Categories and Subject Descriptors

H.3.1 [**INFORMATION STORAGE AND RETRIEVAL**]:

General Terms

Algorithms

Keywords

Linear Simplification, Streaming, Coresets, GPS, Douglas-Peucker

1. INTRODUCTION

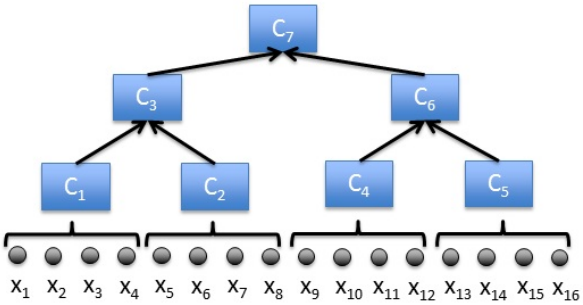
IPSNŠ12, April 16–20, 2012, Beijing, China.

Field-deployed sensor networks are collecting massive amounts of data in real time in applications ranging from environmental systems [18] to traffic [1] to city-scale observation systems [27]. In this paper we describe an algorithm that receives as input the data stream generated by a sensor network and produces as output a much smaller data set that approximates the original data with guaranteed bounds. The smaller data set can be used for faster, real-time processing. The results of computing on the smaller set are guaranteed to approximate the same computation on the original set within specified bounds.

We are motivated by traffic applications using vehicular sensor networks such as the network of 16,000 taxis in Singapore. Even in one hour, the GPS devices installed in the taxis in Singapore generate time-stamped GPS location triples (time, latitude, longitude) that require approximately 40MB. If we add acceleration, imaging, and status data, the amount of information is significantly larger. Yet we wish to collect and process this data in real time, in order to accurately predict city-scale congestion and traffic patterns. This data can also enable geo-location analysis, to identify the set of places visited by a particular device. But given a stream of GPS traces, how do we decide which points are critical for identifying the location of the vehicle in a human-readable way, such as "National University of Singapore" or "Starbucks"? One option is to pose a Google query for every data point. However, Google has a 2500 daily cap on queries, and running such a query for every GPS trace would be impractical.

By learning the critical points contained within the data and summarizing the data stream, we compress the original data and represent it using a much smaller set that ultimately enables much faster processing for a large set of applications.

If we store or send all the data from a given fielded sensor network, analyzing the data in real time is significantly harder. Most of the analysis tools today are based on data mining algorithms (e.g, MATLAB, Weka, SPSS, SPlus, R). They can only handle blocks of static data on the order of a few gigabytes, that fit in the internal memory (RAM). A small number of applications, for example IBM infosphere [7] and Apache Mahout [25]) support larger data sets for a few very specific model fitting heuristics, usually without quality guarantees. The user of such applications has to follow the constraints of the database and programming language used by the application. Spatiotemporal data mining applications face storage-space problems as well as problems



(a) *Tree for coreset construction*

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 enumerated 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 parallel construction of C_3 and C_6, finally resulting in C_7.

in efficiently accessing the motion data. For example, assuming that a GPS point takes 12 bytes and a GPS point is generated (i.e., sampled) every second for 24 hours a day, 10M cellular subscribers will generate a daily volume of over 9600 gigabytes.

Coresets.

We propose to use coresets as a way of approximating large data sets.

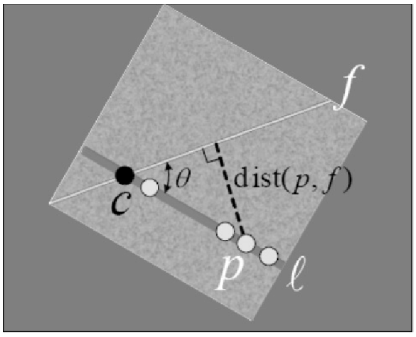
The existence and construction of coresets has been investigated for a number of problems in computational geometry (such as k-means and k-median) in many recent papers (cf. surveys in [13, 3]). Here we demonstrate how these techniques from computational geometry can be lifted to the realm of sensor networks. As a by-product of our analysis, we also provide a solution to the open question on the possibility of compressing GPS data.

More specifically, the input to the coreset algorithm described in this paper is a constant $\varepsilon > 0$ and a set P of n points in $\mathbb{R}^d$ (representing n signals from d sensors) that can be approximated by a k-spline or k segments. Our algorithm returns a coreset of $O(k)$ points (independent of n) such that that the Hausdorff Euclidean distance from P to any given query set of points is preserved up to an additive error of ε (Theorem 5.1 Corollary 5.2).

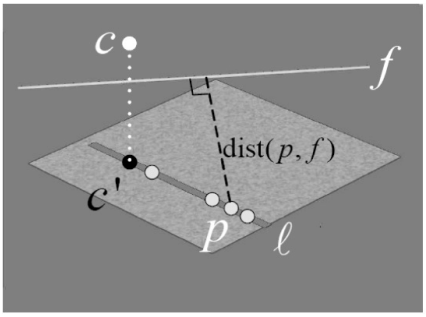
To our knowledge, this is the first type of compression that provides a guarantee on the approximation error for any query and not for a specific family of queries (k-points, k-lines, etc.).

Streaming and parallel computation.

One major advantage of coresets is that they can be constructed in parallel, as well as in a streaming setting where data points arrive one by one. For streaming data it is impossible to remember the entire data set due to memory constraints. The key insight is that coresets satisfy certain composition properties, described in Section 9 and Fig. 1(a).



(a) $f \cap \ell \neq \emptyset$



(b) $f \cap \ell = \emptyset$

Figure 2: (*left*) $\mathrm{dist}(p, f) = \sin\theta \cdot \mathrm{dist}(p, c)$, Hence, c, weighted by $\sin\theta$, replaces f for points on ℓ. (*right*) $\mathrm{dist}(p, f) = \sin\theta \cdot \mathrm{dist}(p, c)$, for any pair ($\ell$, f) of lines in $\mathbb{R}^d$, where c is a point on the line that spans the shortest distance between ℓ and f, placed at distance $\mathrm{dist}(\ell, f)/\sin\theta$ from the point $c' \in \ell$, nearest to f, and θ is the angle between the (orientations of the) lines ℓ and f (a routine exercise in stereometry).

Experimental results and application.

We describe and analyze the coreset algorithm and evaluate it on several real data sets. We also show that coresets can be used to geo-locate vehicular networks efficiently. Using the GPS data streams from in-car networked devices (e.g. smart phones or customized sensor network probes), we demonstrate a sensor data processing application that generates a readable textual description (known as reverse geocoiting) for the node's current position using traffic landmarks retrieved from Google maps, e.g. "Elm street", "NUS", "Starbucks", etc.

This GPS-to-text service enables automatic logging and reporting. Databasing the output allows new types of text queries, searches, and data mining algorithms that were unsuitable for the original raw GPS points.

2. OUR TECHNIQUE

The most related technique to the coreset construction in this paper is the coreset for the k-line center by Agarwal et al. [4]. In the k-line center the input is a set of points P in $\mathbb{R}^d$ and the output is a set S^* of k-lines that minimizes the maximum distance $D(P, S^*)$ between every point in P to its closest line in S^*. Agarwal et al. proved that there is a (core)set $C \subseteq P$ of size $|C| = 2^{O(kd)}/\varepsilon^d$ such that $D(C, S) \geq (1 - \varepsilon)D(P, S)$ for every (query) set S of k-lines.

An approximation to the k-line center of P can then be obtained by computing the k-line center of the small set C. However, they did not suggest an efficient construction for this coreset. This is essentially the same problem that we encounter in the simple construction of Section 4.3. Still, they provide an algorithm that takes time $O(n \log(n) \cdot |C|)$ and computes a $(1+\varepsilon)$ approximation to the k-line center of P. The fact that a small coreset C exists was used only in the analysis.

Similarly to the inefficient construction of [4], our algorithm first projects P onto a few segments, and then scans the projected points from left to right. However, we were able to construct the coreset efficiently using bi-criteria approximation (Section 4.1). It is straightforward to plug our bi-criteria technique to the second part of the paper of Agarwal and obtain the first efficient ($O(n)$ time) construction of coresets for k-lines. In particular, this coreset yields an improved algorithm for computing a $(1+\varepsilon)$ multiplicative approximation for the k-line center in only $O(n)$ time.

Unlike the case of our paper that deals with k-segments and k-splines, the Hausdorff distance between a point and a line is infinite; see Definition 2. Also, when k-lines queries are replaced by k-segments queries, there is no similar coreset C as described above; see [17]. Instead, we use a small additive error in the definition of our coreset (Definition 4.1). This also allows the coreset to approximate every query (and not just k-lines or k-segments queries).

Another difference between our coresets and coresets for k-lines (as in [4]) is that there is a lower bound of 2^k for coresets that provides a $(1+\varepsilon)$ approximation for distances to any k-lines [17]. This bound is impractical for our applications, where usually $k > 100$. However, we were able to construct coresets of size *linear* in k for our problem, using the same relaxation of ε-additive error that was described above.

Our construction uses the fact that, after projecting a set of points on a line ℓ, every query line f can be replaced by a weighted point c; see Fig. 2. While this is trivial for lines on the plane (the case $d = 2$, Fig. 2(a)), it is less intuitive for the case $d \geq 3$ (Fig. 2(b)). We use the fact that when the angle between the two lines f and ℓ is $\theta = \varepsilon$, then the distance from every point $p \in \ell$ to f is similar to the (unweighted) distance to some fixed point $c \in \mathbb{R}^d$, up to a multiplicative factor of $O(\varepsilon)$.

One of the unique advantages of our coreset is its provable ability to handle streaming and parallel data. To this end, we had to use a different approach than those used by existing algorithms. For example, if P is a set of points on a single segment, existing algorithms will usually choose only the two endpoints of the segment (e.g. [2]). While this makes sense for static input, when new points are added to P, the distribution of the original points on the segment might be necessary for sub-dividing the segment, or merging it with a new one. Indeed, our algorithm adds such representatives to the output coreset, even if they are all lying on the same line.

Our contribution.

Our coreset is significantly different from previous compression techniques. (i) It guarantees both a threshold ε *and* a small coreset size that depends on ε; (ii) the construction is more involved, and based on global optimization, rather than on local relations between input points; and (iii) our coreset C is a set of points, not segments. In fact, the first step of our coreset construction computes a set of only $O(k/\varepsilon)$ segments that are provably close to P (See Lemma 4.2), but the final coreset C is a subset of P. This allows us to replace P with C while using existing algorithms that accept points as input, as in the database techniques above. For example, a road map can be used together with C in order to compute the final segments or trajectory that will approximate P. This problem of compressing the data in a way that will allow us to handle constraints such as road-maps was suggested as an open problem in [8].

To our knowledge, our coresets are the first that support the merge-and-reduce model with bounds on both error and space. In particular, we didn't find other parallel (distributed) computing algorithms for compressing trajectories with similar bounds.

Open problems addressed in this paper.

Abam et al. recently stated [2] that an obvious question is whether we can have a streaming algorithm that can approximate S using exactly k-segments (and not $\beta = 2k$ as in [2]). We answer this question in the affirmative by computing the optimal k-spline of our streaming coreset; see Section 8. In [2] it is also stated that the authors couldn't apply the merge-and-reduce technique on their coreset. They suggest the open problem of constructing such a coreset, which we answer in the affirmative in this paper.

Streaming heuristics such as Douglas-Peucker Heuristic (DPH), unlike [2], do not assume monotone/convex input, and provide bounds either on the running time or on the approximation error ε. Streaming simplification is considered a "challenging issue" and "important topic of future work" [8]. Our coreset provides the first bound on both the error and the update time/space simultaneously.

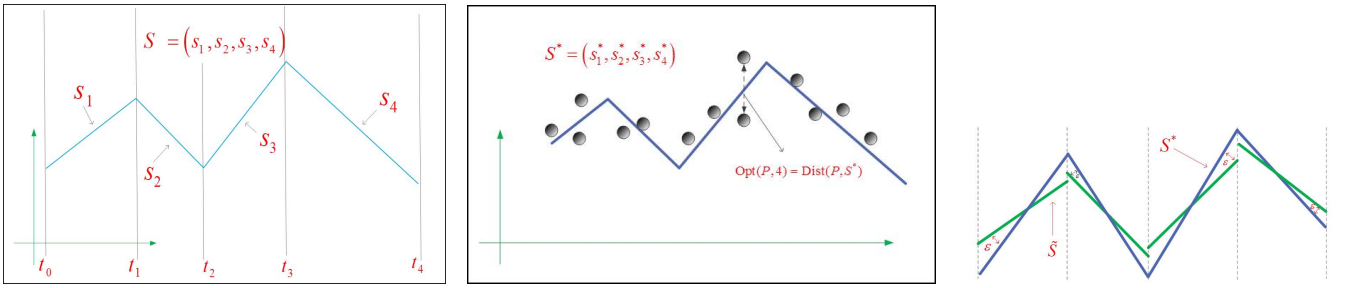
Abam et al. provided the first provable linear simplification streaming $\alpha = O(1)$-approximation algorithm, under the monotone/convex assumption, using $O(k^2)$ space. They suggested a second open problem for reducing this (sometimes impractical) size to $O(k)$, and the possibility of compression using the merge-and-reduce technique. Indeed, our coresets are of size $O(k)$ for every constant $\alpha = \varepsilon > 0$. They are suitable for the merge-and-reduce technique. In addition, our coresets yield the first algorithm that supports parallel (distributed) computing for line simplification. Other compressions that use the merge-and-reduce technique fail either in the merge or the reduce part of the method.

In [8] it was suggested to investigate the issues that arise when the uncertainty of the location technology (e.g. GPS) is combined with the ε-error due to the simplification. Indeed, the error of our coreset depends on ε, but also on the additional parameter k that corresponds to the optimal linear k-simplification of the input path. Unlike with existing heuristics that require the adjustment of the error parameter during the streaming, since our error depends on the current optimal solution, there is no need to adjust it.

3. K-SPLINE CENTER

Let P be a set of points in $\mathbb{R}^d$, where $d \geq 1$ is constant and every point $p \in P$ is of the form $p = (x_1, \cdots, x_{d-1}, t)$. The first coordinates $(x_1, \cdots, x_{d-1})$ represent outputs (real numbers) from $d-1$ sensors at time t. We denote the last coordinates of a point $p = (x_1, \cdots, x_{d-1}, x_d)$ by $p(t) := x_d$.

We call a set $S \subseteq \mathbb{R}^d$ a *k-spline* if it is the union of k segments $\overline{s_0 s_1}, \overline{s_1 s_2}, \ldots, \overline{s_{k-1} s_k}$ for some $s_0, \cdots, s_k \in \mathbb{R}^d$,



(a) 4-*Spline* (b) *Optimal* 4-*spline* (c) ε-*Rotation of the optimal spline*

Figure 3: k-Splines

where $s_0(t) < \cdots < s_k(t)$. The segment $\overline{s_{i-1}s_i}$ is called the *ith segment* of S, for $i \in \{1, \ldots, k\}$. See Fig. 3(a).

The *regression distance* $\text{dist}_R(\{p\}, S)$ between a point $p \in \mathbb{R}^d$ and a k-spline S is the Euclidean distance between p and S along the last coordinate axis, i.e,

$$\text{dist}_R(\{p\}, S) := \begin{cases} \|p - s\| & \text{if } \exists s \in S \text{ s.t. } p(t) = s(t) \\ \infty & \text{otherwise,} \end{cases}.$$

where $\|\cdot\|$ denotes the Euclidean norm. The regression distance between a set $P \subseteq \mathbb{R}^d$ and a k-spline S is the maximum regression distance between a point in P to S, i.e,

$$\text{dist}_R(P, S) := \max_{p \in P} \text{dist}_R(\{p\}, S).$$

A *k-spline center* $S^* = S^*(P, k)$ of P is a k-spline that minimizes the regression distance to P among all the possible k-splines in $\mathbb{R}^d$:

$$\min_S \text{dist}_R(P, S) = \text{dist}_R(P, S^*). \tag{1}$$

See Fig. 3(b).

The k-spline $\tilde{S} = \tilde{S}(P, k)$ is an *ε-rotation* of a k-spline center, if there is a k-spline center S^* of P such that, for every $i \in \{1, \cdots, k\}$, the ith segment of $\tilde{S}$ is a rotation of the ith segment of S^* by an angle ε around some point $s_i \in S_i$. See Fig. 3(c).

4. ε-CORESETS

4.1 Overview of construction

Let P be a set of points in $\mathbb{R}^d$ for some constant integer $d \geq 1$ and let $\varepsilon > 0$. In Section 4.3 we prove that a small ε-coreset C exists for P under the assumption that it can be approximated by an ε-rotation of its k-line center S^* for some integer $k \geq 1$. See Fig 4 for a sketch of the final coreset construction. See Fig 5 for example run on a real set of 5000 GPS points.

The set C is computed by first constructing a dense set T of $O(k/\varepsilon)$ segments around every one of the k segments in S^*. Then, we project every point of P onto its nearest segment in T. We prove that the resulting set C' (which contains n points) has a small Hausdorff distance to P. Since C' is contained in T, we can now scan the projected points on each segment t of T from left to right and select a representative point from every ε^2-fraction of t. The union of representatives is denoted by C''. Note that every point $p'' \in C''$ is a projection of some point $p \in P$ on T. Our output ε-coreset C is the union of points in P whose projection is in C''. We prove that for every such input set P, the resulting set C is a small ε-coreset with size independent of n.

The above construction is inefficient, since we assume that the optimal k-spline S^* was already computed. In Section 5 we will replace this assumption by a rough and fast approximation to S^*, called (α, β) or bi-criteria approximation.

4.2 Necessary assumptions

We define $\text{dist}_H(A, B)$ as the Hausdorff Euclidean distance between a pair of sets $A, B \subseteq \mathbb{R}^d$:

$$\text{dist}_H(A, B) := \max\left\{ \max_{p \in P} \min_{p' \in C} \|p - p'\|, \max_{p' \in C} \min_{p \in P} \|p' - p\| \right\}. \tag{2}$$

DEFINITION 4.1 (ε-CORESET). *An* ε-coreset *for* P *is a set* $C \subseteq P$ *such that*

$$\text{dist}_H(P, C) \leq \varepsilon.$$

By scaling the points of P, we can see that in general an ε-coreset for P must contain all the points of P. Hence, we will add the assumption that P is not an arbitrary set of points, because it can be roughly approximated by its k-spline center for some $k \geq 1$.

Formally, let $S^* = S^*(P, k)$ denote a k-spline center of P. We assume that

$$\text{dist}_R(P, S^*) \leq c \tag{3}$$

for some constant $c \in (0, \infty)$ (that is independent of n). Hence, it suffices to prove that

$$\text{dist}_H(P, C) \leq \varepsilon \cdot \text{dist}_R(P, S^*) \tag{4}$$

to conclude that C is an $c\varepsilon$-coreset. By applying the construction with ε/c instead of ε we obtain an ε-coreset.

Using assumption (3) we compute in the next section a set C (denoted by C') that satisfies (4). The resulting ε-coreset C' is still large (of size $n = |P|$), but is contained in only $O(k/\varepsilon)$ segments. Unfortunately, it is impossible to compute a *small* ε-coreset C', even if (3) holds for a small constant c. For example, if P is a set of points on a line, then $\text{dist}_H(P, S^*(P, k)) = 0$ and we must have $C' = P$. Therefore, we replace $S^*(P, k)$ with its ε-rotation $\tilde{S} = \tilde{S}(P, k)$ in our assumption (3); see Section 3 for definition of $\tilde{S}$. Using the new assumption and the fact that C' is contained in a few segments we prove in Theorem 5.1 and Corollary 5.2 that we can always compute a small ε-coreset C for P in $O(n)$ time.

4.3 Inefficient Construction Algorithm

Let P be a sequence of n points in $\mathbb{R}^d$, $\varepsilon > 0$ and $k \geq 1$. In order to clarify the algorithm and its analysis, we first prove that a small ε-coreset C exists. For a point $p \in P$, and a segment S_i of a k-spline S, we say that S_i serves p if there is $s \in S_i$ such that $p(t) = s(t)$. That is, the last coordinates of s and p are identical. For a set $Y \subseteq \mathbb{R}^d$, we define $\text{dist}(p, Y) = \min_{y \in Y} \|p - y\|$, and for a set $X \subseteq \mathbb{R}^d$ we define $\text{dist}(X, Y) = \min_{x \in X} \text{dist}(x, Y)$. For simplicity, we first describe the construction when P is a set of points on the plane (i.e., $d = 2$).

Step 1: Constructing C'.

Let $S^* = S^*(P, k)$ denote a k-spline center of P. Although we don't know how to compute S^*, such an optimum exists. Fix a segment S_i of S^*, and let P_i denote the points of P that are served by S_i. Let T_i denote an ε-grid of segments around S_i. More formally, T_i is the union of $\lceil 2/\varepsilon \rceil$ parallel segments, each of length $|S_i| + \text{dist}(P, S^*)$, such that the distance from a point $p \in P$ to its closest segment t in T_i is ε than its distance to S_i:

$$\text{dist}(p, T_i) \leq \varepsilon \text{dist}(p, S_i). \tag{5}$$

Let p' be the projection of $p \in P_i$ onto its closest segment in T_i. Let $C'_i = \{p' \mid p \in P\}$ be the union of these points, $T = \bigcup_{1 \leq i \leq k} T_i$ and $C' = \bigcup_{1 \leq i \leq k} C_i$

Step 2: Constructing C.

Let $t \subseteq T_i$ denote one of the segments of T_i that was constructed in Step 1, that contains at least one point from C', i.e., $C' \cap t \neq \emptyset$. Let p'_L, p'_R denote the leftmost and rightmost points from C' on t, respectively. Partition the segment $\overline{p'_L, p'_R} \subseteq t$, into $r = \lceil 10/\varepsilon \rceil$ equal sub-segments $t_1, \cdots, t_r$. For every such sub-segment t_j, $1 \leq j \leq r$ that contains at least one point from C', pick a single point $p'_j \in C' \cap t_j$. We call p'_j the *representative* of every $p' \in C' \cap t_j$. Let $C'_t = \left\{p'_j \mid 1 \leq j \leq r, t_j \cap C' \neq \emptyset\right\}$ be the union of these representatives on t. Let $C'' = \bigcup_t C'_t$ where the union is over all the segments of T. Recall that every point $p' \in C'' \subseteq C'$ is the projection of some $p \in P$ on t. Let $C = \{p \in P \mid p' \in C''\}$ be the final output set of the construction.

LEMMA 4.2. *Let P be a set of points in $\mathbb{R}^d$, $k \geq 1$ and $\varepsilon \in (0, 1)$. There is a set C' that is contained in $O(k/\varepsilon)$ segments such that*

$$\text{dist}_H(P, C') \leq \varepsilon \text{dist}_R(P, S^*(P, k)).$$

PROOF. We use the construction in the beginning of this section and its notation. Let $C_i = C' \cap P_i$ for every i, $1 \leq i \leq k$. We have

$$\text{dist}_H(P, C') \leq \max_{1 \leq i \leq k} \text{dist}_H(P_i, C_i)$$

Put i, $1 \leq i \leq k$. Then

$$\begin{aligned}
&\text{dist}_H(P_i, C_i) \\
&\leq \max\left\{\max_{p \in P_i} \min_{q \in C_i} \|p - q\|, \max_{q \in C_i} \min_{p \in P} \|q - p\|\right\} \\
&\leq \max\left\{\max_{p \in P_i} \left\|p - p'\right\|, \max_{p' \in C_i} \left\|p' - p\right\|\right\} \\
&\leq \max_{p \in P_i} \left\|p - p'\right\|.
\end{aligned}$$

Let $p \in P_i$. By (5),

$$\begin{aligned}
\left\|p - p'\right\| &= \text{dist}(p, T_i) \leq \varepsilon \text{dist}(p, S_i) \\
&\leq \varepsilon \max_{p \in P_i} \text{dist}(p, S_i) \\
&\leq \varepsilon \text{dist}_R(P_i, S_i) \leq \varepsilon \text{dist}_R(P, S^*).
\end{aligned} \tag{6}$$

Combining the last inequalities yields

$$\begin{aligned}
\text{dist}_H(P, C') &\leq \max_{1 \leq i \leq k} \text{dist}_H(P_i, C_i) \\
&\leq \max_{1 \leq i \leq k} \max_{p \in P_i} \left\|p - p'\right\| \\
&\leq \varepsilon \text{dist}_R(P, S^*).
\end{aligned}$$

Since $C' \subseteq T$, the last inequality proves the lemma. □

LEMMA 4.3. *Let P be a set of points in $\mathbb{R}^d$, $k \geq 1$ and $\varepsilon > 0$. There is a set $C \subseteq P$ of size $|C| = O(k/\varepsilon^3)$ such that*

$$\text{dist}_H(P, C) \leq \varepsilon \text{dist}_R(P, \tilde{S}(P, k)).$$

PROOF. We use the construction and notation from the beginning of this section, and prove that

$$\text{dist}_H(P, C) \leq 10\varepsilon \text{dist}_R(P, \tilde{S}(P, k)) \tag{7}$$

By replacing ε with $\varepsilon/10$ in the construction, this would prove the lemma.

Using the triangle inequality,

$$\begin{aligned}
&\text{dist}_H(P, C) \leq \\
&\text{dist}_H(P, C') + \text{dist}_H(C', C'') + \text{dist}_H(C'', C).
\end{aligned}$$

By (6) and the definition of S^*,

$$\begin{aligned}
\text{dist}_H(P, C') + \text{dist}_H(C'', C) &\leq 2 \max_{p \in P} \left\|p - p'\right\| \\
&\leq 2\varepsilon \text{dist}_R(P, S^*) \\
&\leq 2\varepsilon \text{dist}_R(P, \tilde{S}).
\end{aligned}$$

For every $p' \in C'$ let p'' denote its representative in C'' (as defined in the construction of C''). Hence,

$$\begin{aligned}
&\text{dist}_H(C', C'') \\
&= \max\left\{\max_{q' \in C'} \min_{q'' \in C''} \left\|q' - q''\right\|, \max_{q'' \in C''} \min_{q' \in C'} \left\|q'' - q\right\|\right\} \\
&\leq \max_{p' \in C'} \left\|p' - p''\right\|.
\end{aligned}$$

Combining the last three inequalities yields

$$\text{dist}_H(P, C) \leq 2\varepsilon \text{dist}_R(P, \tilde{S}) + \max_{p' \in C'} \left\|p' - p''\right\|. \tag{8}$$

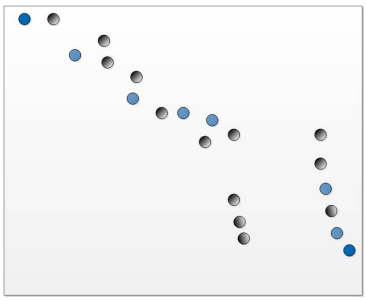
Next, we compute a bound for $\|p - p''\|$.

Let $p \in P$. Suppose that $p' \in C'$ is the projection of p on the segment $t \subseteq T_i$. Let P'_L and P'_R denote, respectively, the leftmost and rightmost point of $C' \cap t$. By construction,

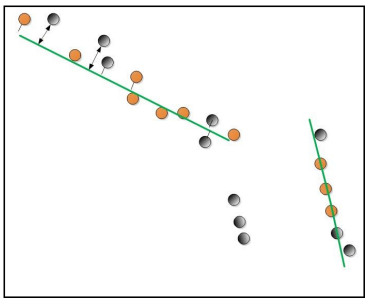
$$\left\|p' - p''\right\| \leq \varepsilon^2 \left\|p'_R - p'_L\right\|. \tag{9}$$

We now bound $\|p'_R - p'_L\|$.

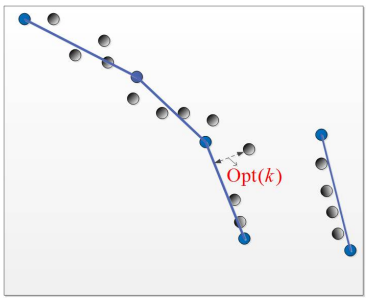
Recall that every segment of T_i, including t, is parallel to S_i. Let $\tilde{S}_i$ denote the corresponding ε-rotation of S_i in $\tilde{S}$. Let ℓ_i denote the line on the plane that contains $\tilde{S}_i$. Let ℓ denote the line that contains t. Let x denote the intersection point between ℓ and ℓ_i. The farthest point from x in $C' \cap t$



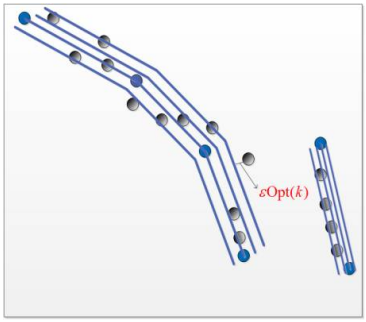
(a) *Pick a random sample from the input points*



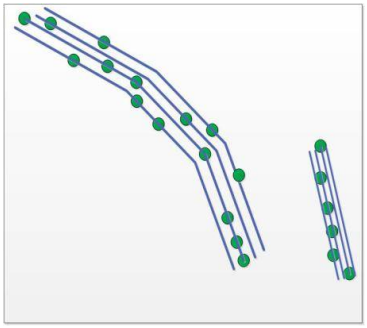
(b) *Approximate sample by a k-spline. Remove* $\sim n/2$ *closest points to the k-spline and goto step (a) till no points left.*



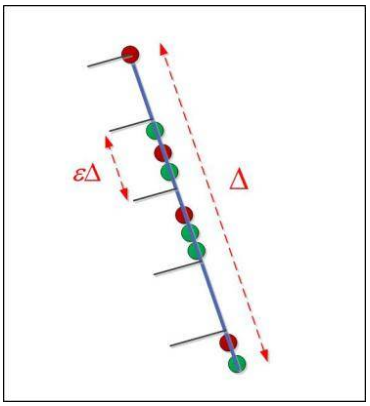
(c) *Take the union of k-splines from all the* $O(\log n)$ *iterations*



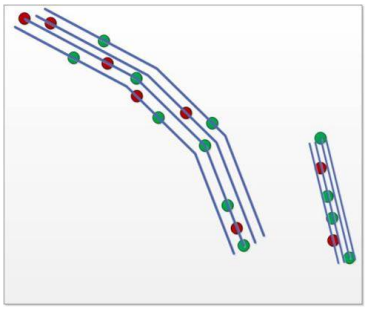
(d) *Construct* $O(1/\varepsilon^d)$ *grid of segments around every segment of the k-splines*



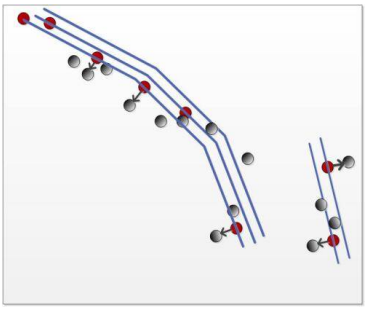
(e) *Project input points on the grids*



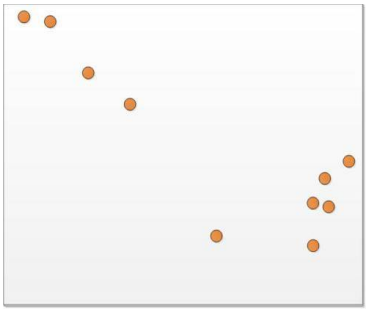
(f) *Partition each grid's segment into equal* $O(1/\varepsilon)$ *sub-segments*



(g) *Pick a representative point from each sub-segment*

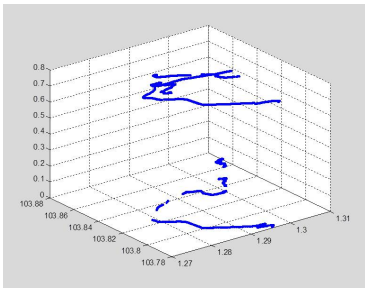


(h) *Find the corresponding input point for every representative*

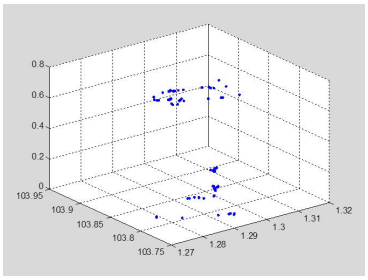


(i) *Return the union of corresponding points*

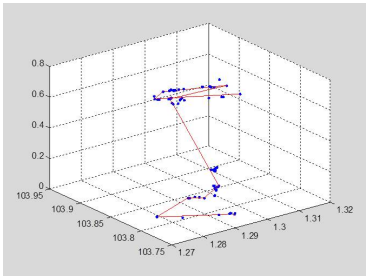
Figure 4: Algorithm for constructing ε-coreset that approximates every k-spline.



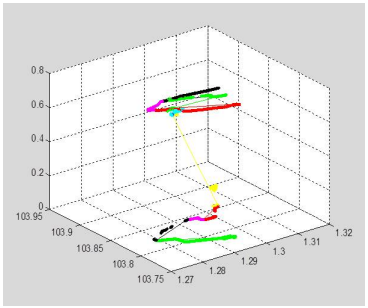
(a) *Input:* $n = 5000$ *points.*



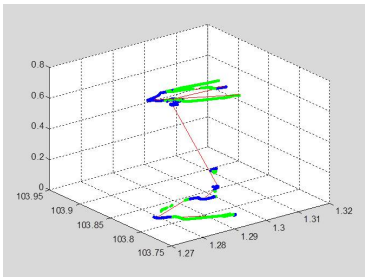
(b) *Pick a random sample of* $10k = 100$ *points.*



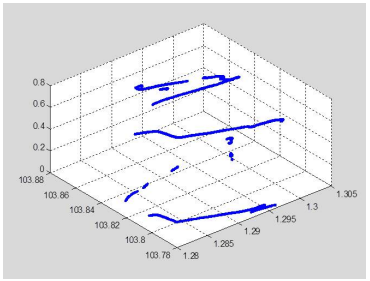
(c) *Compute optimal* $k = 10$ *spline for sample*



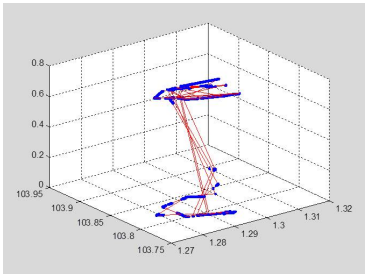
(d) *Compute distances from the* n *input points to the spline. Here, different colors mean different clusters.*



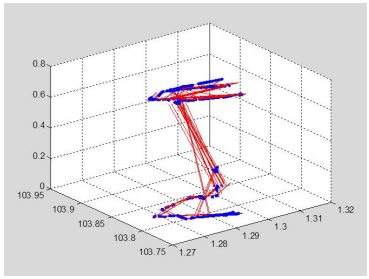
(e) *Remove closest* $n/2$ *points to spline.*



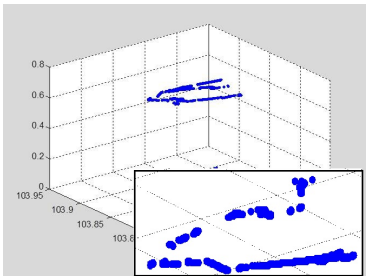
(f) *Repeat from step (b) on remaining* $n/2$ *points, until* $n < 10k$



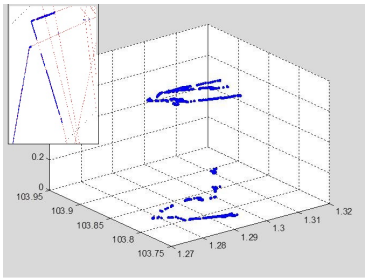
(g) *Take the union of* 50 *segments that were computed in Step (c) on parallel to all* 5 *iterations .*



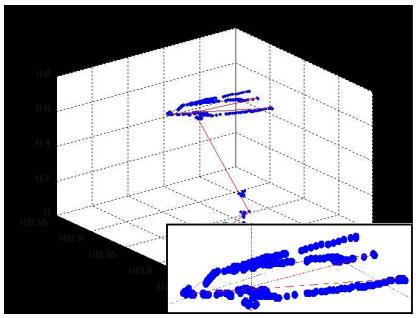
(h) *Extend to* 80 *segments that are the* 50 *segments*



(i) *Project every point onto its nearest segment*



(j) *Pick* 5 *points from each segment*



(k) *Output corresponding input points*

Figure 5: Example coreset construction C **for** $n = 5000$ **input points, and any** $k = 10$ **spline, where** $|C| = 300$**.**

is either P'_R or P'_L. Hence,

$$\begin{aligned}\left\|p'_R - p'_L\right\| &\leq \left\|P'_L - x\right\| + \left\|x - P'_R\right\| \\ &\leq 2\max\left\{\left\|P'_L - x\right\|, \left\|P'_R - x\right\|\right\} \\ &\leq 2\max_{q'\in C'}\left\|q' - x\right\|. \end{aligned} \tag{10}$$

We denote by $\theta(\ell')$ the sinus of the angle between ℓ and a given line ℓ'. Hence, $\theta(\ell_i) = \sin(\varepsilon) \geq 2\varepsilon/\pi \geq \varepsilon/2$. Since $p' \in \ell$, we thus have

$$\text{dist}(p', \ell_i) = \sin(\theta(\ell_i))\left\|p' - x\right\| \geq \frac{\varepsilon}{2}\left\|p' - x\right\|.$$

That is,

$$\left\|p' - x\right\| \leq 2\text{dist}(p', \ell_i)/\varepsilon. \tag{11}$$

Since $\tilde{S}_i \subseteq \ell_i$, we have

$$\begin{aligned}\text{dist}(p', \ell_i) &\leq \text{dist}(p', \tilde{S}_i) \leq \left\|p' - p\right\| + \text{dist}(p, \tilde{S}_i) \\ &\leq \text{dist}_H(P, C') + \text{dist}_R(P, \tilde{S}).\end{aligned}$$

By Lemma 4.2, $\text{dist}_H(P, C') \leq \varepsilon\text{dist}_R(P, S^*)$. By the assumption $\varepsilon < 1$ and the definition of S^*, the last inequality implies $\text{dist}_H(P, C') \leq \text{dist}_R(P, \tilde{S})$. Combining the last inequalities with (11) yields

$$\begin{aligned}\left\|p' - x\right\| &\leq 2\text{dist}(p', \ell_i)/\varepsilon \\ &\leq \frac{2}{\varepsilon}(\text{dist}_H(P, C') + \text{dist}_R(P, \tilde{S})) \\ &\leq \frac{4\text{dist}_R(P, \tilde{S})}{\varepsilon}.\end{aligned}$$

By plugging the last inequality in (9) and (10), we obtain

$$\begin{aligned}\left\|p' - p''\right\| &\leq \varepsilon^2\left\|p'_R - p'_L\right\| \\ &\leq 2\varepsilon^2\max_{q'\in C'}\left\|q' - x\right\| \\ &\leq 8\varepsilon\text{dist}_R(P, \tilde{S}).\end{aligned}$$

By (8), this proves (7) as

$$\begin{aligned}\text{dist}_H(P, C) &\leq 2\varepsilon\text{dist}_R(P, \tilde{S}) + \max_{p'\in C'}\left\|p' - p''\right\| \\ &\leq 10\varepsilon\text{dist}_R(P, \tilde{S}).\end{aligned}$$

□

5. EFFICIENT CONSTRUCTION

The above construction is inefficient since it assumes that we already computed a k-spline center S^* of P, which we don't know how to do in time near-linear in n. In Section 5, we prove that C can be constructed efficiently (in $O(n)$ time). The k-spline S^* can be replaced in our construction by a rough approximation S called bi-criteria approximation or (α, β)-approximation for S^*. The distance from P to S is larger by a multiplicative constant factor α than its distance to S^*. Still, we couldn't find any algorithm in literature that computes such a constant factor approximation for P in near-linear time. Hence, we add a relaxation that S can contain $\beta = O(k\log n)$ segments instead of k segments. We use random projections to obtain a simple algorithm that computes such an (α, β)-approximation S for the k-line center of P in $O(n)$ time as follows.

First, we pick a small uniform random sample Q of $O(k/\varepsilon)$ points from P. Next, we compute the k-spline center $S^*(Q, k)$ of the small set Q using an existing inefficient optimal algorithm for spline approximation, and remove from P the $|P|/2$ points that are closest to $S^*(Q, k)$. We then repeat this algorithm recursively until P is empty. The output of the algorithm is the union of the computed k-splines.

The algorithm runs using at most $O(\log n)$ iterations, and thus outputs $O(\log n)$ k-splines. This is straightforward from PAC-learning theory, and the technique is also popular for sensor networks applications [16]. The size of the final coreset is $O(\log n)$. In order to have a coreset of size independent of n, we compute a k-spline approximation S on our existing coresets, and repeat the construction with S instead of using the $O(\log n)$ splines from the bi-criteria approximation. De-randomization and straightforward generalization for other distance functions can be obtained using the framework of [13] with our observations.

THEOREM 5.1. *Let P be a set of n points in $\mathbb{R}^d$. Let $\varepsilon \in (0, 1)$ and $k \geq 1$. Then a set $C \subseteq P$ of size $|C| = O(k/\varepsilon^3)$ can be constructed in $O(n)$ time, such that*

$$\text{dist}_H(P, C) \leq \varepsilon\text{dist}_R(P, \tilde{S}(P, k)).$$

COROLLARY 5.2. *Let P be a set of n points in $\mathbb{R}^d$. Let $\varepsilon \in (0, 1)$ be a constant and $k \geq 1$ be an integer. If $\text{dist}_R(P, \tilde{S}(P, k) \in O(1)$ then an ε-coreset C for P can be computed in $O(n)$ time. That is,*

$$\text{dist}_H(P, C) \leq \varepsilon.$$

PROOF. Since $\text{dist}_R(P, \tilde{S}(P, k)) \leq c$ for some constant $c = O(1)$, we can replace ε with ε/c in Theorem 5.1 to obtain

$$\text{dist}_H(P, C) \leq (\varepsilon/c)\text{dist}_R(P, \tilde{S}(P, k)) \leq \varepsilon.$$

□

6. EXPERIMENT AND RESULTS

The data. We tested the practical compression ratio of our coreset construction by implementing it and running experiments using a public dataset of GPS traces [26]. This dataset contains mobility traces of taxi cabs in San Francisco, USA. It contains GPS coordinates of 500 taxis collected over 30 days in the San Francisco Bay Area.

The experiment. We applied the following procedure independently to every trace in the data set. The input set P was partitioned into two parts containing 10,000 traces each. We applied our coresets construction on each part to obtain two coresets of approximately 200 points. We then merged the two coresets and compressed the new set (of size 400) again, as shown in Fig 1(a). On the resulting coreset of 200 points, we then applied DPH with $k = 100$ (i.e, DPH approximates the coreset using 100 points, or 100-spline) to get a k-spline ($k = 100$) S_C which approximates the original set.

We then repeated the experiment, using DPH itself as the compression algorithm, as was done in [8]. That is, we partitioned the original 20k into two sets and applied DPH independently on every set using $k = 200$. We then applied DPH on the merged set of 400 points using $k = 200$. On the resulting set D of 200 points, we applied DPH using $k = 100$ to get a k-spline S_D.

The results. We computed the Hausdorff error between the original set P and the k-spline S_C that was obtained from our coreset. Similarly, we computed the error between

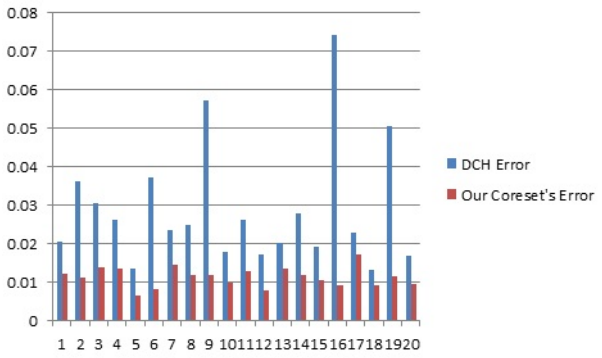


Figure 6: Comparison of experiments on traces of 500 taxi-cabs over a month. The y-axis is the Hausdorff distance from the original set to the k-spline approximation that was constructed on the compression. The x-axis is a sample of the first 20 taxi-cabs that had at least 20000 GPS traces.

P and the k-spline S_D that was constructed on the DPH-compressed set. The results for 20 such experiments on each taxi-cab are shown on Fig 6, where the y-axis is the error. Additional points would result in more merges that increase the comparison gap exponentially with the levels of the tree in Fig. 1(a).

The implementation. We implemented our algorithm in MATLAB [22]. We also used the official implementation of MATLAB for the DPH algorithm. DPH uses as input parameter the maximum allowed error ε instead of the number of desired segments k. We overcome this problem by applying binary search on the value of ε until DPH returned k lines.

For the bi-criteria approximation during the coreset construction, we used a random sample of 5 points, and connected them by a 5-spline S. We then removed half of the closest input points to S recursively as explained in Section 5 .We used 5 additional segments for the ε-grid around every segment of S. Sampling more input points and using a more involved algorithm for computing S, as in Section 5, will likely improve the result. We are in the process of testing this hypothesis. In addition, since we did not try to configure the parameters of the construction to obtain better results, another direction of our current work is to tune the parameters.

7. DISCUSSION IN THE CONTEXT OF RELATED WORK

The challenge of compressing trajectories semantically (also known as "line simplification") has been tackled from various perspectives: geographic information systems [11], databases, [10], digital image analysis [21], computational geometry [5], and especially in the context of sensor networks [6]. The input to this problem is a sequence P of n points that describes coordinates of a path over time. The output is a set Q of k points (usually subset of P) that approximates P. More precisely, the k-spline S that is obtained by connecting every two consecutive points in Q via a segment should be close to P according to some distance function. The set Q is sometimes called a coreset [2] since it is a small set that approximates P. However, our definition of coreset for this problem is significantly different (see Definition 4.1) and more similar to its original definition in computational geometry [3].

Several books have been written about the line simplification problem [15]. Yet, it seems that every discipline improves the solution with respect to some parameters, but deteriorates others. Our coreset was inspired by several previous techniques, aiming to formalize the trade-offs and suggest a unified solution that enjoys the good benefits of all previous ones.

Simple heuristics.

The oldest heuristic [8] for line simplification is the Douglas-Peucker heuristic (DPH) [11]. DPH gets an input threshold $\varepsilon > 0$ and returns a set Q that represents a k-spline S as defined above. DPH guarantees that the Euclidean distance from every $p \in P$ to S is at most ε. This is also the attractiveness of DPH, compared to other lossy data compression techniques such as wavelets [9]. DPH is very simple, easy to implement, and has a very good running time in practice [8]. The guaranteed ε-error allows us to merge two compressed sets Q_1 and Q_2 in the streaming model, while keeping the ε-error for $Q_1 \cup Q_2$.

We compared our experimental results to DPH not only because it is popular, but also because it "achieves near-optimal savings at a far superior performance" [8].

While DPH has a guaranteed ε-error, it suffers from serious space problems due to its local (ad-hoc, greedy) optimization technique. In particular, the size k of its output is unbounded, and might be arbitrarily larger than the smallest set $Q \subseteq P$ that obtained such an ε-error. While merging two sets preserves the error, it is not clear how to reduce the merged set again. The size of the compressed output will increase linearly with the input stream. Choosing a larger ε will result in too small or empty set Q for the first compressions. The worst case running time for the basic (and practical) implementation is $O(n^2)$. More modern versions of the DPH ([24]) appear to have similar pros and cons.

Approximation Algorithms.

Provable approximation algorithms from theoretical computer science and computational geometry seem to have opposite properties. They are based on much more involved global optimization algorithms with theoretical worst-case guarantees on the running time, error, and space. This is also their main disadvantage: the papers (at least in our context) usually do not contain experimental results (e.g. [2]), and it is not clear that an efficient implementation is possible due to problems such as numerical stability and hidden constants in the $O()$ notation. Few exceptions are recently available [12]. We run a popular heuristic (Douglas-Peucker) against itself, by simply running the heuristic once on P and once on C. Surprisingly, the running time improves and the approximation error is *reduced* on the representative set C as compared to the original P. This might be due to noise removal as a side effect of the coreset construction. In addition, since C is significantly smaller than P, we can apply the heuristic many more times, using different initial parameters, during a single competitive run on P. Overall, this technique combines provable guarantees with practical heuristics.

The optimal algorithm for simplifying $2d$-polygonal chain runs in $O(n^2)$ time for any Euclidean metrics and $O(n^{4/3+\delta})$

for L_1 and L_∞ metrics [5]. Sophisticated variations of DPH can be implemented in $O(n \log n)$ time [19] and even $O(n \log^* n)$ time [20]. Still, we couldn't find implementations for these algorithms, and DPH seems to have much more popularity and better running time in practice [8].

Abam, de Berg, Hachenberger and Zarei [2] recently suggested the provable construction of a coreset C for P under the streaming model. The set C is of size $O(k^2)$ and allows us to compute an $\alpha = O(1)$ ($\alpha > 2$) multiplicative-approximation to the optimal k-spline S of P using β-spline, where $\beta = 2k$. In our paper such an approximation is also used in the coreset construction and is called (α, β)-approximation; (see Section 4.1). The algorithms are non-trivial, interesting and followed by deep computational geometry proofs. However, this result holds *only if* S is xy-monotone or a convex linear function [2]. This assumption on S is unlikely to hold in many situations. The $O(k^2)$ space might also be infeasible for large values of k. While C can be computed in the streaming model, it cannot be computed in parallel.

Results that are similar or improve upon the results of Abam et al. can be obtained using our coreset as follows. The first step of our construction projects P onto $O(k/\varepsilon)$ segments with a provable small ε-error. The maximum distance from a point on a segment to a monotone path will be obtained by either the leftmost or rightmost point on the segment. Hence, selecting these two points from P in each segment to the coreset C suffices to approximate every monotone function. Since there are $O(k/\varepsilon)$ segments, the resulting coreset will be of size $O(k)$. Similarly, our coreset approximates convex paths (using simple observation from [17]). Unlike the coreset of Abam et al., our coreset also supports the parallel computation model.

Database techniques.

Popular database servers, such as MySQL, support spatial queries (such as: nearest road or station to a given trajectory). Unlike the previous two approaches, here we are interested in a data structure for compressing P that will be used to answer general queries, rather than just to compute a line simplification S for P. Our coreset (Definition 4.1) is inspired by this approach and different from all previous coresets: it guarantees that every input point in P has an ε-close representative in the coreset C. Using the triangle inequality, the error for *every* query set is bounded by ε, regardless of the specific type of query (k-points, k-segments, etc.).

The lack of locality in trajectories makes their compression harder than other databases (that represent more static data such as house locations or ages). Indeed, there are small coresets for $(1+\varepsilon)$ multiplicative approximation of P by k points for several distance functions [4], while there are lower bounds of 2^k for the size of such compression for k lines [17]. Unlike the case of points, covering P by k-lines is NP-hard [23] and bounding the complexity of the Voronoi diagram of k-lines is one of the main open problems in geometry.

Our coreset construction cannot be described as partitioning the space into simple shapes or cells and taking a single representative from each cell. The first step of our coreset construction *projects* the points onto *linear* objects, rather then compressing them. We prove that for a set of points projected onto the same segment, the distance to a given query segment or line can be represented as a distance to a point; see Fig 2. This observation allows us (in the second step) to partition *the segments* into cells and cluster the points on them, as in the database techniques.

8. APPLICATIONS

A small ε-coreset C for P can help us reduce the space and running time of many applications in the field of sensor networks.

Database queries.

Our coreset is a new type of compression that guarantees a small additive error for general types of queries, which makes it suitable for answering SQL or GIS spatial queries on the reduced coreset C as suggested in [10].

Suppose, for example, we require the nearest input point to a query curve that represents a road, or to a query point that represents a bus station. The property $\text{dist}_H(P, C) \leq \varepsilon$ of the coreset guarantee that every point in P has a close representative point in C. After computing the coreset C we can delete the original set P, and still be able to answer unbounded number of queries in $O(1)$ time using C. We give the following corollary.

COROLLARY 8.1. *Let P be a set of points. Suppose that C is an ε-coreset for P that was constructed using Theorem 5.1 for some constants $k \geq 1$ and $\varepsilon > 0$. Then, for every subset $Q \subset \mathbb{R}^d$, we have*

$$0 \leq \text{dist}_H(P, Q) - \text{dist}_H(C, Q) \leq \varepsilon.$$

Moreover, $\text{dist}_H(C, Q)$ can be computed in $O(1)$ time, if this is the time it takes to compute the distance from a single point $p \in P$ to Q.

PROOF. Since $C \subseteq P$ by Theorem 5.1, we have $\text{dist}_H(P, Q) - \text{dist}_H(C, Q) \geq 0$. Since the Hausdorff distance is a metric, it satisfies symmetry and the triangle inequality. Hence,

$$\begin{aligned}\text{dist}_H(P, Q) &\leq \text{dist}_H(P, C) + \text{dist}_H(C, Q) \\ &\leq \varepsilon + \text{dist}_H(C, Q) = \text{dist}_H(Q, C) + \varepsilon,\end{aligned}$$

where in the second deviation we used Theorem 5.1. Therefore $\text{dist}_H(P, Q) - \text{dist}_H(C, Q) \leq \varepsilon$.

The time it takes to compute $\text{dist}_H(C, Q)$ is $|C| \cdot t$ where t is the time it takes to compute $\text{dist}_H(\{p\}, Q)$ for a single point $p \in C \subseteq P$. By Theorem 5.1, we have $|C| = O(k/\varepsilon^3) = O(1)$. For $t = 1$ we obtain $t \cdot |C| = O(1)$. □

$(1+\varepsilon)$ *Approximations.*

In a vehicular network application, we may have a series of GPS points representing a vehicle traveling between several destinations. Instead of using all n GPS points on its path to represent the path, we would like to create the k-spline center in order to compress the data. The k-spline center $S^*(P, k)$ of a set P of n points is a set of k connected segments that approximates P. The set $S^*(P, k)$ is also called the line simplification of P; See Section 3 for formal definitions. The time it takes to compute $S^*(P, k)$ is near-quadratic in n [5] and impractical [8]. Instead of trying to improve this kind of optimization algorithms, we can apply the (possibly inefficient) algorithm on the small coreset C to get its k-spline center $S^*(C, k)$ in $O(1)$ time. In particular, if $\text{dist}_H(P, S^*(P, k)) \in O(1)$ then $S^*(C, k)$ is a $(1+\varepsilon)$ multiplicative approximation for the k-spline center *of the original set* P as proved in the following corollary.

COROLLARY 8.2. *Let P be a set of n points, and $k \geq 1$ be a constant integer. Let $S^* = S^*(P,k)$ be the k-spline center of P. Let C be an ε-coreset of P for some constant $\varepsilon > 0$. Then a k-spline S' can be computed in $O(1)$ time such that:*

(i)

$$\text{dist}_H(P,S') \leq \varepsilon + \text{dist}_H(P,S^*).$$

(ii) If $\text{dist}_H(P,S^*) \geq 1$ *then*

$$\text{dist}_H(P,S') \leq (1+\varepsilon)\text{dist}_H(P,S^*).$$

PROOF. *(i)* The k-line center $S' = S^*(C,k)$ of C can be computed in $O(|C|^3) = O(1)$ time using the algorithm in [5]. By the triangle inequality and the definition of S',

$$\begin{aligned}\text{dist}_H(P,S') &\leq \text{dist}_H(P,C) + \text{dist}_H(C,S') \\ &\leq \text{dist}_H(P,C) + \text{dist}_H(C,S^*)\end{aligned}$$

Since C is an ε-coreset of P, we have $\text{dist}_H(P,C) \leq \varepsilon$. Together with the previous inequality this proves Claim(i).
(ii) straightforward from (i). □

9. STREAMING AND PARALLEL COMPUTATION

In this section we show that our (static, off-line) coreset scheme suffices to solve the corresponding problem in parallel and distributed computing, and in a streaming setting where data points arrive one by one and remembering the entire data set is futile due to memory constraints. We use a map-and-reduce technique (popular today as "Google's map-and-reduce"). The key insight is that coresets satisfy certain composition properties. Similar properties have previously been used by [13] for streaming and parallel construction of coresets for geometric clustering problems such as k-median and k-means. We extend these results for Hausdorff distances as follows.

OBSERVATION 9.1.

(i) Suppose that C_1 is an ε-coreset for P_1, and C_2 is a ε-coreset for P_2. Then $C_1 \cup C_2$ is an ε-coreset for $P_1 \cup P_2$.

(ii) Suppose C is an ε-coreset for P, and D is an ε-coreset for C. Then D is a 2ε-coreset for P.

PROOF. (i) $\begin{aligned}\text{dist}_H(C_1 \cup C_2, P_1 \cup P_2) &\leq \max\{\text{dist}_H(C_1,P_1), \text{dist}_H(C_2,P_2)\} \\ &\leq \max\{\varepsilon,\varepsilon\} = \varepsilon.\end{aligned}$

(ii) $\begin{aligned}\text{dist}_H(D,P) &\leq \text{dist}_H(D,C) + \text{dist}_H(C,P) \\ &\leq 2\varepsilon.\end{aligned}$

□

In the following, we review how to exploit these properties for streaming and parallel computation.

Streaming.

In the streaming setting, we wish to maintain a coreset over time, while keeping only a small subset of $O(\log n)$ coresets in memory (each of small size). The idea is to construct and save in memory a coreset for every block of consecutive points arriving in a stream.

When we have two coresets in memory, we can merge them (resulting in an ε-coreset via property (i)) and compress them by computing a single coreset from the merged coresets (via property (ii)) to avoid an increase in the coreset size.

Parallel computation.

Using the same ideas from the streaming model, a nonparallel coreset construction can be transformed into a parallel one. We partition the data into sets, and compute coresets for each set, independently, on different processors in a cluster (sensors, computer networks, cloud services, etc). We then (in parallel) merge (via property (i)) two coresets, and compute a single coreset for every pair of such coresets (via property (ii)). Continuing in this manner yields a process that takes $O(\log n)$ iterations of parallel computation. Fig. 1(a) illustrates this parallel construction.

GPU computation.

GPUs are installed today on most desktop computers and recently on smart phones (such as the Nexus One and iPhone). The GPU is essentially a set of dozens or hundreds of processors that are able to make parallel computations. However, the model of computation is very restrictive. In particular, non trivial linear algebra computations suffer from significant numerical problems. Our algorithm suggests a new paradigm to deal with such problems by computing the coreset on the GPU and using the CPU to compute the desired result on the coreset. As we explained in Section 5, most of the construction time of the coreset is spent on matrix multiplication (computing distances from points to segments) that can be computed on the GPU. Also, division operations (that usually cause the numerical problems) are applied by optimization algorithms only on the CPU.

10. CONCLUSION

In this paper we presented an algorithm for compressing the large data sets generated by sensor networks via coresets. Given a constant $\epsilon > 0$ and a set P of n points representing signals from sensors, our ϵ-coreset algorithm returns $O(k)$ points (independent of n) such that the Hausdorff Euclidean distance from P to any given query set of points is preserved up to an additive error of ϵ. Our compression algorithm departs from previous results by guaranteeing the approximation error for any query as compared to previous works that guarantee the solution quality only for subclasses of queries. Because our coreset algorithm is parallelizable, it can be used in off-line mode for historical data collected and databased, as well as in on-line mode for streamed data from sensors. To demonstrate the effectiveness of this coreset algorithm, we showed data from an experiment with GPS data collected from a vehicular network of taxis. We showed that the practical compression ratio of our coreset construction outperforms the Douglas-Peucker algorithm.

Our current results provide a theoretical step for enabling off-line and on-line computation on large-scale sensor networks. Much work remains to be done in order to provide practical solutions to in-network and off-line processing of large data sets. We are currently integrating our algorithms with sensor networks deployed in the field. This is a significant engineering challenge, which is important to undertake in order to close the loop from theory to practical applica-

tions. We are undertaking a more extensive experimental evaluation of this algorithm using data from a roving vehicular network of 15000 taxis, we are developing the georeferencing applications that take advantage of the potential benefits of our solution in this paper.

11. ACKNOWLEDGEMENT

Fig 1(a) was drawn by Andreas Krause and is taken from [14]. This research was supported in part by the Foxconn Company and the Future Urban Mobility project of the Singapore-MIT Alliance for Research and Technology (SMART) Center, with funding from Singapore's National Research Foundation. We are grateful for it.

12. REFERENCES

[1] Mobile millennium. Technical report.

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

[3] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximations via coresets. *Combinatorial and Computational Geometry - MSRI Publications*, 52:1–30, 2005.

[4] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. Approximation algorithms for k-line center. In *Proc. 10th Ann. European Symp. on Algorithms (ESA)*, volume 2461 of *Lecture Notes in Computer Science*, pages 54–63. Springer, 2002.

[5] P.K. Agarwal and K.R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete & Computational Geometry*, 23(2):273–291, 2000.

[6] J. Aslam, Z. Butler, F. Constantin, V. Crespi, G. Cybenko, and D. Rus. Tracking a moving object with a binary sensor network. In *Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 150–161. ACM, 2003.

[7] A. Biem, E. Bouillet, H. Feng, A. Ranganathan, A. Riabov, O. Verscheure, H. Koutsopoulos, and C. Moran. Ibm infosphere streams for scalable, real-time, intelligent transportation services. In *Proceedings of the 2010 international conference on Management of data*, pages 1093–1104. ACM, 2010.

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

[9] K. Chakrabarti, M. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. *The VLDB Journal*, 10(2):199–223, 2001.

[10] P. Cudre-Mauroux, E. Wu, and S. Madden. Trajstore: An adaptive storage system for very large trajectory data sets. In *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, pages 109–120. IEEE.

[11] 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: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[12] M. Feigin, D. Feldman, and Nir Sochen. From high definition image to low space optimization. In *Proc. 3rd Inter. Conf. on Scale Space and Variational Methods in Computer Vision (SSVM 2011)*, 2011.

[13] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proc. 41th Ann. ACM Symp. on Theory of Computing (STOC)*, 2011.

[14] Dan Feldman, Matthew Faulkner, and Andreas Krause. Scalable training of mixture models via coresets. In *Proc. Neural Information Processing Systems (NIPS)*, 2011.

[15] L. Forlizzi, R.H. Güting, E. Nardelli, and M. Schneider. *A data model and data structures for moving objects databases*, volume 29. ACM, 2000.

[16] S. Gandhi, S. Suri, and E. Welzl. Catching elephants with mice: sparse sampling for monitoring sensor networks. *ACM Transactions on Sensor Networks (TOSN)*, 6(1):1, 2009.

[17] S. Har-Peled. Coresets for discrete integration and clustering. *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, pages 33–44, 2006.

[18] J.K. Hart and K. Martinez. Environmental sensor networks: A revolution in the earth system science? *Earth-Science Reviews*, 78(3-4):177–191, 2006.

[19] J. Hershberger and J. Snoeyink. An o (n log n) implementation of the douglas-peucker algorithm for line simplification. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 383–384. ACM, 1994.

[20] J. Hershberger and J. Snoeyink. Cartographic line simplification and polygon csg formulć in o (n log* n) time. *Computational Geometry*, 11(3-4):175–185, 1998.

[21] J.D. Hobby. Polygonal approximations that minimize the number of inflections. In *Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 93–102. Society for Industrial and Applied Mathematics, 1993.

[22] U. Matlab. The mathworks. *Inc., Natick, MA*, 1992, 1760.

[23] N. Meggido and A. Tamir. Finding least-distance lines. *SIAM J. on Algebric and Discrete Methods*, 4:207–211, 1983.

[24] N. Meratnia and R.A. By. Spatiotemporal compression techniques for moving point objects. *Advances in Database Technology-EDBT 2004*, pages 561–562, 2004.

[25] S. Owen, R. Anil, T. Dunning, and E. Friedman. Mahout in action. *Online*, pages 1–90, 2011.

[26] Michal Piorkowski, Natasa Sarafijanovic-Djukic, and Matthias Grossglauser. CRAWDAD data set epfl/mobility (v. 2009-02-24). Downloaded from http://crawdad.cs.dartmouth.edu/epfl/mobility, February 2009.

[27] S. Reddy, M. Mun, J. Burke, D. Estrin, M. Hansen, and M. Srivastava. Using mobile phones to determine transportation modes. *ACM Transactions on Sensor Networks (TOSN)*, 6(2):13, 2010.