# Coresets for Vector Summarization with Applications to Network Graphs

Dan Feldman[1] Sedat Ozer[2] Daniela Rus[2]

## Abstract

We provide a deterministic data summarization algorithm that approximates the mean $\bar{p} = \frac{1}{n}\sum_{p\in P} p$ of a set $P$ of $n$ vectors in $\mathbb{R}^d$, by a weighted mean $\tilde{p}$ of a *subset* of $O(1/\varepsilon)$ vectors, i.e., independent of both $n$ and $d$. We prove that the squared Euclidean distance between $\bar{p}$ and $\tilde{p}$ is at most $\varepsilon$ multiplied by the variance of $P$. We use this algorithm to maintain an approximated sum of vectors from an unbounded stream, using memory that is independent of $d$, and logarithmic in the $n$ vectors seen so far. Our main application is to extract and represent in a compact way friend groups and activity summaries of users from underlying data exchanges. For example, in the case of mobile networks, we can use GPS traces to identify meetings; in the case of social networks, we can use information exchange to identify friend groups. Our algorithm provably identifies the *Heavy Hitter* entries in a proximity (adjacency) matrix. The Heavy Hitters can be used to extract and represent in a compact way friend groups and activity summaries of users from underlying data exchanges. We evaluate the algorithm on several large data sets.

## 1. Introduction

The wide-spread use of smart phones, wearable devices, and social media creates a vast space of digital footprints for people, which include location information from GPS traces, phone call history, social media postings, etc. This is an ever-growing wealth of data that can be used to identify social structures and predict activity patterns. We wish to extract the underlying social network of a group of mobile users given data available about them (e.g. GPS traces, phone call history, news articles, etc.) in order to identify and predict their various activities such as meetings, friend groups, gathering places, collective activity patterns, etc. There are several key challenges to achieve these capabilities. First, the data is huge so we need efficient methods for processing and representing the data. Second, the data is multi-modal heterogeneous. This presents challenges in data processing and representation, but also opportunities to extract correlations that may not be visible in a single data source. Third, the data is often noisy.

We propose an approach based on coresets to extract underlying connectivity information while performing data summarization for a given a large data set. We focus our intuition examples and evaluations on social networks because of their intuitive nature and access to data sets, although the method is general and applies to networks of information in general. Our approach works on streaming datasets to represent the data in a compact (sparse) way. Our coreset algorithm gets a stream of vectors and approximates their sum using small memory. Essentially, a coreset $C$ is a significantly smaller portion (a scaled subset) of the original and large set $D$ of vectors. Given $D$ and the algorithm $A$, where running algorithm $A$ on $D$ is intractable due to lack of memory, the task-specific coreset algorithm efficiently reduces the data set $D$ to a coreset $C$ so that running the algorithm $A$ on $C$ requires a low amount of memory and the result is provable approximately the same as running the algorithm on $D$. Coreset captures all the important vectors in $D$ for a given algorithm $A$. The challenges are computing $C$ fast and proving that $C$ is the right scaled subset, i.e., running the algorithm on $C$ gives approximately the same result as running the algorithm on $D$.

More specifically, the goal of this paper is to suggest a way to maintain a sparse representation of an $n \times d$ matrix, by maintaining a sparse approximation of each of its rows. For example, in a proximity matrix associated with a social network, instead of storing the average proximity to each of $n$ users, we would like to store only the $N \ll n$ largest entries in each row (known as "Heavy Hitters"), which correspond to the people seen by the user most often. Given an unbounded stream of movements, it is hard to tell which are the people the user met most, without maintaining a counter to each person. For example, consider the special case $N = 1$. We are given a stream of pairs $(i, val)$ where $i \in \{1, \cdots, n\}$ is an index of a counter, and $val$ is a real number that represents a score for this counter. Our goal is

[1] University of Haifa, Israel, [2] CSAIL, MIT. Correspondence to: Dan Feldman <dannyf.post@gmail.com>.

*Proceedings of the 34th International Conference on Machine Learning*, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

to identify which counter has the maximum average score till now, and approximate that score. While that is easy to do by maintaining the sum of $n$ scores, our goal is to maintain only a constant set of numbers (memory words). In general, we wish to have a provable approximation to each of the $n$ accumulated scores (row in a matrix), using, say, only $O(1)$ memory. Hence, maintaining an $n \times n$ matrix would take $O(n)$ instead of $O(n^2)$ memory. For millions of users or rows, this means 1 Gigabytes of memory can be stored in RAM for real-time updates, compared to millions of Gigabytes. Such data reduction will make it practical to keep hundreds of matrices for different types of similarities (sms, phone calls, locations), different types of users, and different time frames (average proximity in each day, year, etc). This paper contributes the following: (1) A compact representation for streaming proximity data for a group of many users; (2) A coreset construction algorithm for maintaining the social network with error guarantees; (3) An evaluation of the algorithm on several data sets.

**Theoretical Contribution:** These results are based on an algorithm that computes an $\varepsilon$-coreset $C$ of size $|C| = O(1/\varepsilon)$ for the mean of a given set, and every error parameter $\varepsilon \in (0, 1)$ as defined in Section 2. Unlike previous results, this algorithm is deterministic, maintains a weighted subset of the input vectors (which keeps their sparsity), can be applied on a set of vectors whose both cardinality $n$ and dimension $d$ is arbitrarily large or unbounded. Unlike existing randomized sketching algorithms for summing item frequencies (1-sparse binary vectors), this coreset can be used to approximate the sum of arbitrary real vectors, including negative entries (for decreasing counters), dense vectors (for fast updates), fractions (weighted counter) and with error that is based on the variance of the vectors (sum of squared distances to their mean) which might be arbitrarily smaller than existing errors: sum/max of squared/non-squared distances to the origin ($\ell_2/\ell_1/\ell_\infty$).

### 1.1. Solution Overview

We implemented a system that demonstrates the use and performance of our suggested algorithm. The system constructs a sparse social graph from the GPS locations of real moving smart-phone users and maintains the graph in a streaming fashion as follows.

**Input stream:** The input to our system is an unbounded stream of (real-time) GPS points, where each point is represented in the vector format of $(time, userID, longitude, latitude)$. We maintain an approximation of the average proximity of each user to all the other $n - 1$ users seen so far, by using space (memory) that is only logarithmic in $n$. The overall memory would then be near-linear in $n$, in contrast to the quadratic $O(n^2)$ memory that is needed to store the exact average proximity vector for each of the $n$ users. We maintain a dynamic array of the $n$ user IDs seen so far and assume, without loss of generality, that the user IDs are distinct and increasing integers from 1 to $n$. Otherwise we use a hash table from user IDs to such integers. In general, the system is designed to handle any type of streaming records in the format $(streamID, v)$ where $v$ is a $d$-dimensional vector of reals, to support the other applications. Here, the goal is to maintain a sparse approximation to the sum of the vectors $v$ that were assigned to each stream ID.

**Proximity matrix:** We also maintain an (non-sparse) array $pos$ of length $n$ that stores the current location of each of the $n$ users seen so far. That array forms the current $n^2$ pairs of proximities $prox(u, v)$ between every pair of users and their current locations $u$ and $v$. These pairs correspond to what we call the *proximity matrix* at time $t$, which is a symmetric adjacency $n \times n$ matrix of a social graph, where the edge weight of a pair of users (an entry in this matrix) is their proximity at the current time $t$. We are interested in maintaining the sum of these proximity matrices over time, which, after division by $n$, will give the average proximity between every two users over time. This is the *average proximity matrix*. Since the average proximity matrix and each proximity matrix at a given time require $O(n^2)$ memory, we cannot keep them all in memory. Our goal is to maintain a *sparse approximation* version of each of row in the average proximity matrix which will use only $O(\log n)$ memory. Hence, the required memory by the system will be $O(n \log n)$ instead of $O(n^2)$.

**Average proximity vector:** The average proximity vector is a row vector of length $n$ in the average proximity matrix for each of the $n$ users. We maintain only a sparse approximation vector for each user, thus, only the non-zeroes entries are kept in memory as a set of pairs of type $(index, value)$. We define the proximity between the current location vectors $u, v \in \mathbb{R}^3$ of two users as: $prox(u, v) := e^{-\text{dist}(u,v)}$.

**Coreset for a streamed average:** Whenever a new record $(time, userID, longitude, latitude)$ is inserted to the stream, we update the entries for that user as his/her current position array $pos$ is changed. Next, we compute that $n$ proximities from that user to each of the other users. Note that in our case, the proximity from a user to himself is always $e^0 = 1$. Each proximity $prox_j$ where $1 \leq j \leq n$ is converted to a sparse vector $(0, \cdots, 0, prox_j, 0, \cdots, 0)$ with one non-zero entry. This vector should be added to the average proximity vector of user $j$, to update the $j$th entry. Since maintaining the exact average proximity vector will take $O(n)$ memory for each of the $n$ users, we instead add this sparse vector to an object ("coreset") that maintains an approximation to the average proximity vector of user $j$.

Our problem is then reduced to the problem of maintaining a sparse average of a stream of sparse vectors in $\mathbb{R}^n$, using $O(\log n)$ memory. Maintaining such a stream for each of the $n$ users seen so far, will take overall memory of $O(n \log n)$ as desired, compared to the exact solution that requires $O(n^2)$ memory. We generalize and formalize this problem, as well as the approximation error and its practical meaning, in Section 2.

### 1.2. Related Work

As mobile applications become location-aware, the representation and analysis of location-based data sets become more important and useful in various domains (Wasserman, 1980; Hogan, 2008; Carrington et al., 2005; Dinh et al., 2010; Nguyen et al., 2011; Lancichinetti & Fortunato, 2009). An interesting application is to extract the relationships between mobile users (in other words, their social network) from their location data (Liao, 2006; Zheng, 2011; Dinh et al., 2013). Therefore, in this paper, we use coresets to represent and approximate (streaming) GPS-based location data for the extraction of the social graphs. The problem in social network extraction from GPS data is closely related to the frequency moment problem. *Frequency approximation* is considered the main motivation for streaming in the seminal work of (Alon et al., 1996a), known as the "AMS paper", which introduced the streaming model.

Coresets have been used in many related applications. The most relevant are coresets for $k$-means; see (Barger & Feldman, 2015) and reference therein. Our result is related to coreset for 1-mean that approximates the mean of a set of points. A coreset as defined in this paper can be easily obtained by uniform sampling of $O(\log d \log(1/\delta))/\varepsilon^2$ or $O(1/(\delta\varepsilon))$ points from the input, where $\delta \in (0, 1)$ is the probability of failure. However, for sufficiently large stream the probability of failure during the stream approaches 1. In addition, we assume that $d$ may be arbitrarily large. In (Barger & Feldman, 2015) such a coreset was suggested but its size is exponential in $1/\varepsilon$. We aim for a deterministic construction of size independent of $d$ and linear in $1/\varepsilon$.

A special case of such coreset for the case that the mean is the origin (zero) was suggested in (Feldman et al., 2016), based on Frank-Wolfe, with applications to coresets for PCA/SVD. In this paper we show that the generalization to any center is not trivial and requires a non-trivial embedding of the input points to a higher dimensional space.

Each of the above-mentioned prior techniques has at least one of the following disadvantages: (1) It holds only for positive entries. Our algorithm supports any real vector. Negative values may be used for deletion or decreasing of counters, and fraction may represent weights. (2) It is randomized, and thus will always fail on unbounded stream. Our algorithm is deterministic. (3) It supports only $s = 1$ non-zero entries. Our algorithm supports arbitrary number of non-zeroes entries with only linear dependency of the required memory on $s$. (4) It projects the input vectors on a random subspace, which diminishes the sparsity of these vectors. Our algorithm maintains a small weighted subset of the vectors. This subset keeps the sparsity of the input and thus saves memory, but also allows us to learn the representative indices of points (time stamps in our systems) that are most important in this sense.

The most important difference and the main contribution of our paper is the error guarantee. Our error function in (1) is similar to $\|\bar{p} - \hat{p}\|_\ell \le \varepsilon \|\bar{p}\|_q$ for $\ell = 2$ on the left hand side. Nevertheless, the error on the right hand side might be significantly smaller: instead of taking the sum of squared distances to the origin (norm of the average vector), we use the variance, which is the sum of squared distances to the mean. The later one is always smaller, since the mean of a set of vectors minimized their sum of squared distances.

## 2. Problem Statement

The input is an unbounded stream vectors $p_1, p_2, \cdots$ in $\mathbb{R}^d$. Here, we assume that each vector has one non-zero entry. In the social network example, $d$ is the number of users and each vector is in the form $(0, \cdots, 0, prox_j, 0, \cdots, 0)$, where $prox_j$ is the proximity between the selected user and user $j$. Note that for each user, we independently maintain an input stream of proximities to each of the other users, and the approximation of its average. In addition, we get another input: the error parameter $N$, which is related to the memory used by the system. Roughly, the required memory for an input stream will be $O(N \log n)$ and the approximation error will be $\varepsilon := 1/N$. That is, our algorithms are efficient when $N$ is a constant that is much smaller than the number of vectors that were read from stream, $2 < N \ll n$. For example, to get roughly 1 percents of error, we have $\varepsilon = 0.01$, and the memory is about $100n$, compared to $n^2$ for the exact average proximity.

The output $\hat{p}$ is an $N$-sparse approximation to the average vector in the stream over the $n$ vectors $p_1, \cdots, p_n$ seen so far. That is, an approximation to the centroid, or center of mass, $\bar{p} = \frac{1}{n}\sum_i p_i$. Here and in what follows, the sum is over $i \in \{1, \cdots, n\}$. Note that even if $s = 1$, the average $\bar{p}$ might have $n \gg N$ non-zero entries, as in the case where $p_i = (0, \cdots, 0, 1, 0, \cdots, 0)$ is the $i$th row of the identity matrix. The sparse approximation $\hat{p}$ of the average vector $\bar{p}$ has the following properties: (1) The vector $\hat{p}$ has at most $N$ non-zero entries. (2) The vector $\hat{p}$ approximates the vector of average proximities $\bar{p}$ in the sense that the (Euclidean) distance between the two vectors is $\mathrm{var}/N$ where $\mathrm{var}$ is the variance of all the vectors seen so far in the stream. More formally, $\|\bar{p} - \hat{p}\|_2 \le \varepsilon \mathrm{var}$, where $\varepsilon = 1/N$ is the error, $\bar{p} = \frac{1}{n}\sum_{i=1}^n p_i$ is the average vector in $\mathbb{R}^d$ for the $n$ input vectors, and the variance is the sum of squared distances to the average vector.

**Distributed and parallel computation.** Our system supports distributed and streaming input simultaneously in a "embarrassingly parallel" fashion. E.g., this allows multi-

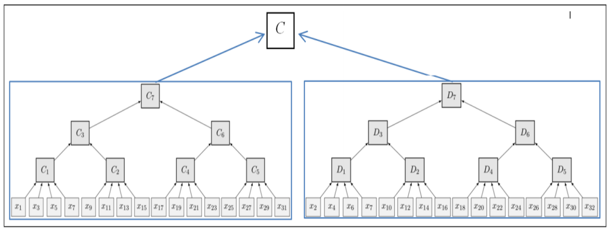

$C$
$C_7$
$C_3$
$C_6$
$C_1$
$C_2$
$C_4$
$C_5$
$x_1$ $x_3$ $x_5$ $x_7$ $x_9$ $x_{11}$ $x_{13}$ $x_{15}$ $x_{17}$ $x_{19}$ $x_{21}$ $x_{23}$ $x_{25}$ $x_{27}$ $x_{29}$ $x_{31}$
$D_7$
$D_3$
$D_6$
$D_1$
$D_2$
$D_4$
$D_5$
$x_2$ $x_4$ $x_6$ $x_7$ $x_{10}$ $x_{12}$ $x_{14}$ $x_{16}$ $x_{18}$ $x_{20}$ $x_{22}$ $x_{24}$ $x_{26}$ $x_{28}$ $x_{30}$ $x_{32}$

*Figure 1.* Coreset computation of streaming data that is distributed among $M = 2$ machines. The odd/even vectors in the stream (leaves) are compressed by the machine on the left/right, respectively. A server (possibly one of these machine) collects the coreset $C_7$ and $D_7$ from each machine to obtain the final coreset $C$ of the $n = 32$ vectors seen so far. Each level of each tree stores at most one coreset in memory, and overall of $O(\log n)$ coresets.

ple users to send their streaming smart-phone data to the cloud simultaneously in real-time. There is no assumption regarding the order of the data in user ID. Using $M$ nodes, each node will have to use only $1/M$ fraction of the memory to $(\log n)^{O(1)}/M$ that is used by one node for the same problem, and the average insertion time for a new point will be reduced by a factor of $M$ to $(\log n)^{O(1)}/M$.

Parallel coreset computation of unbounded streams of distributed data was suggested in (Feldman & Tassa, 2015), as an extension to the classic merge-and-reduce framework in (Bentley & Saxe, 1980; Har-Peled, 2006). We apply this framework on our off-line algorithm to handle streaming and distributed data (see Section 3).

**Generalizations.** Above we assume that each vector in the stream has a single non-zero entry. To generalize that, we now assume that each vector has at most $s$ non-zeroes entries and that these vectors are weighted (e.g. by their importance). Under these assumptions, we wish to approximate the weighted mean $\bar{p} = \sum_{i=1}^{n} u_i p_i$ where $u = (u_1, \cdots, u_n)$ is a weight vector that represents distribution, i.e., $u_i \geq 0$ for every $i \in [n]$. Then, our problem is formalized as follows.

**Problem 1** *Consider an unbounded stream of real vectors $p_1, p_2, \cdots$, where each vector is represented only by its non-zero entries, i.e., pairs $(entryIndex, value) \in \{1, 2, 3, \ldots\} \times \mathbb{R}$. Maintain a subset of $N \ll n$ input vectors, and a corresponding vector of positive reals (weights) $w_1, w_2, \cdots, w_N$, where the sum $\hat{p} := \sum_{i=1}^{N} w_i p_i$ approximates the sum $\bar{p} = \sum_{i=1}^{n} p_i$ of the $n$ vectors seen so far in the stream up to a provably small error that depends on its variance $\mathrm{var}(p) := \sum_{i=1}^{n} \|p_i - \bar{p}\|_2^2$. Formally, for an error parameter $\varepsilon$ that may depend on $N$,*

$$\|\bar{p} - \hat{p}\| \leq \varepsilon \mathrm{var}(p). \tag{1}$$

We provide a solution for this problem mainly by proving Theorem 1 for off-line data, and turn it into algorithms for streaming and distributed data as explained in Section 3.

# 3. New Coreset Algorithms

In this section we first describe an algorithm for approximating the sum of $n$ streaming vectors using one pass. The algorithm calls its off-line version as a sub-procedure. We then explain how to run the algorithm on distributed *and* unbounded streaming data using $M$ machines or parallel threads. The size of the weighted subset of vectors that are maintained in memory, and the insertion time per new vector in the stream are logarithmic on the number $n$ of vectors in the stream. Using $M$ machines, the memory and running time per machine is reduced by a factor of $M$.

## 3.1. Streaming and Distributed Data

**Overview.** The input to Algorithm 1 is a $stream$ provided as a pointer to a device that sends the next input vectors in a stream that consists of $n$ vectors, upon request, For example, a hard drive, a communication socket, or a web-service that collects information from the users. The second parameter $\varepsilon$ defines the approximation error. The required memory grows linearly with respect to $1/\varepsilon$.

Algorithm 1 maintains a binary tree whose leaves are the input vectors, and each inner node is a coreset, as in the left or right hand side of Fig. 1. However, at most one coreset in each level of the tree is actually stored in memory. In Line 1, we initialize the current height of this tree. Using $\log(n)/\varepsilon$ vectors in memory our algorithm returns an $O(\varepsilon)$-coreset, but to get exactly $\varepsilon$-coreset, we increase it in Line 2 by a constant factor $\alpha$ that can be find in the proof of Theorem 1. In Lines 3-14, we read batches (sets) of $O(\log(n)/\varepsilon)$ vectors from the stream and compress them. The last batch may be smaller. Line 4 defines the next batch $P$. Unlike the coresets in the nodes of the tree, we assume that the input vectors are unweighted, so, Line 5 defines a weight 1 for each input vector. Line 6 reduce the set $P$ by half to the weighted coreset $(S, w)$ using Algorithm 2 (the off-line coreset construction.) Theorem 1 guarantees that such a compression is possible.

In Lines 8–12 we add the new coreset $(S, w)$ to the lowest level $\ell$ of the binary tree, if it is not assigned to a coreset already, i.e., $S_\ell$ is not empty. Otherwise, we merge the new coreset $(S, w)$ with the level's coreset $S_\ell$, mark the level as empty from coresets ($S_\ell \leftarrow \emptyset$), and continue to the next higher level of the tree until we reach a level $\ell$ that is not assigned to a coreset , i.e., $S_\ell = \emptyset$. Line 13 handle the case where we reach to the root of the tree, and a new (top) level is created. In this case, only the new root of the tree contains a coreset.

When the streaming is over, in Lines 15–16 we collect the active coreset in each of the $O(\log n)$ tree levels (if it has one) and return the union of these coresets.

**Parallel computation on distributed data.** In this model each machine has its own stream of data, and computes its coreset independently and in parallel to the other machines, as described above. Whenever we wish to get the coreset for the union of streaming vectors, we collect the current coreset from each machine on a single machine or a server. Since each machine sends only a coreset, the communication is also logarithmic in $n$. For $M \geq 2$ machines and a single input stream, we send every $i$th point in the stream to the $i$th machine, for every $i$ between 1 and $M$. For example, if $M = 2$, the odd vectors will be sent to the first machine, and every second (even) vector will be sent to the second machine. Then, each machine will compute a coreset for its own unbounded stream; See Fig 1.

### 3.2. Off-line data reduction

Algorithm 2 is called by Algorithm 1 and its variant in the last sub-section for compressing a given small set $P$ of vectors in memory by computing its coreset. As in Line 10 of Algorithm 1, the input itself might be a coreset of another set, thus we assume that the input has a corresponding weight vector $u$. Otherwise, we assign a weight 1 for each input vector, i.e., $u = (1, \cdots, 1)$. The output of Algorithm 1 is an $\varepsilon$-coreset $(S, w)$ of size $|S| = O(1/\varepsilon)$ for a given $\varepsilon \in (0, 1)$. While the number $n$ of input vectors can be of an arbitrary size, Algorithm 2 always passes an input set of $n = 2|S|$ points to get output that is smaller by half.

**Overview of Algorithm 2:** In Line 1 the desired mean $E_u$ that we wish to approximate is computed. Lines 2–4 are used for adding an extra dimension for each input vector later. In Lines 4-6 we normalize the augmented input, by constructing a set $q_1, \cdots, q_n$ of unit vectors with a new set of weights $s_1, \cdots, s_n$ whose mean is $\sum_i s_i q_i = (0, \cdots, 0, x/v)$. We then translate this mean to the origin by defining the new set $H$ in Line 7.

The main coreset construction is computed in Lines 9–13 on the normalized set $H$ whose mean is the origin and its vectors are on the unit ball. This is a greedy, gradient descent method, based on the Frank-Wolfe framework (Feldman et al., 2016). In iteration $i = 1$, we begin with an arbitrary input point $c_1$ in $H$. Since $c_1$ is a unit vector, its distance from the mean of $H$ (origin) is 1. In Line 11– 12 we compute the farthest point $h_2$ from $c_1$, and approximates the mean using only $c_1$ and the new point $h_2$. This is done, by projecting the origin on the line segment through $c_1$ and $h_2$, to get an improved approximation $c_2$ that is closer to the origin. We continue to add input points in this manner, where in the $i$th iteration another input point is selected for the coreset, and the new center is a convex combination of $i$ points. In the proof of Theorem 1 it is shown that the distance to the origin in the $i$th iteration is $\alpha/i$, which yields an $\varepsilon$-approximation after $\beta = O(\alpha/\varepsilon)$ iterations.

The resulting center $c_\beta$ is spanned by $\beta$ input vectors. In Line 11 we compute their new weights based on their distances from $c_\beta$. Lines 14–16 are used to convert the weights of the vectors in the normalized $H$ back to the original input set of vectors. The algorithm then returns the small subset of $\beta$ input vectors with their weights vector $w$.

**Algorithm 1:** STREAMING-CORESET$(stream, \varepsilon)$

```
Input:   An input stream of $n$ vectors in $\mathbb{R}^d$.
         an error parameter $\varepsilon \in (0, 1)$
Output:  An $\varepsilon$-coreset $(S, w)$ for the set of $n$ vectors;
         see Theorem 1.
1  Set $max \leftarrow 0$
2  Set $\alpha$ to be a sufficiently large constant that can be
   derived from the proof of Theorem 1.
3  while $stream$ is not empty do
4      Set $P \leftarrow$ next $\lceil 2\alpha \ln(n)/\varepsilon \rceil$ input vectors in
       $stream$
5      Set $u \leftarrow (1, \cdots, 1)$ where $u$ has $|P|$ entries.
6      Set $(S, w) \leftarrow$ CORESET$(P, u, \varepsilon/(\alpha \ln(n))$
7      Set $\ell \leftarrow 1$
8      while $S_\ell \neq \emptyset$ and $\ell \leq max$ do
9          Set $S_\ell \leftarrow S \cup S_\ell$
10         Set $(S, w) \leftarrow$ CORESET$(S_\ell, w_\ell, \varepsilon)$
11         Set $S_\ell \leftarrow \emptyset$
12         Set $\ell \leftarrow \ell + 1$
13     if $\ell > max$ then
           Set $max \leftarrow \ell$
14     Set $(S_\ell, w_\ell) \leftarrow (S, w)$
15 Set $S \leftarrow \bigcup_{i=1}^{max} S_i$ and $w \leftarrow (w_1, w_2, \cdots, w_{max})$
16 return $(S, w)$
```

## 4. Correctness

In this section we show that Algorithm 1 computes correctly the coreset for the average vector in Theorem 1.

Let $D^n$ denote all the possible distributions over $n$ items, i.e., $D$ is the unit simplex

$$D^n = \{(u_1, \cdots, u_n) \in \mathbb{R}^n \mid u_i \geq 0 \text{ and } \textstyle\sum_{i=1}^n u_i = 1\}.$$

Given a set $P = \{p_1, \cdots, p_n\}$ of vectors, the mean of $P$ is $\frac{1}{n} \sum_{i=1} p_i$. This is also the expectation of a random vector that is chosen uniformly at random from $P$. The sum of variances of this vector is the sum of squared distances to the mean. More generally, for a distribution $u \in S$ over the vectors of $P$, the (weighted) mean is $\sum_{i=1}^n u_i p_i$, which is the expected value of a vector chosen randomly using the distribution $u$. The variance $\text{var}_u$ is the sum of weighted squared distances to the mean. By letting $N = 1/\varepsilon$ in the following theorem, we conclude that there is always a sparse distribution $w$ of at most $1/\varepsilon$ non-zeroes entries, that yields an approximation to its weighted mean, up to an $\varepsilon$-fraction of the variance.

**Theorem 1 (Coreset for the average vector)** *Let $u \in D^n$ be a distribution over a set $P = \{p_1, \cdots, p_n\}$ of $n$ vectors in $\mathbb{R}^d$, and let $N \geq 1$. Let $(S, w)$ denote the output of a call to* CORESET$(P, u, 1/N)$*; see Algo-*

**Algorithm 2:** CORESET$(P, u, \varepsilon)$

```
Input:   A set $P$ of vectors in $\mathbb{R}^d$,
         a positive weight vector $u = (u_1, \cdots, u_n)$,
         an error parameter $\varepsilon \in (0, 1)$
Output:  An $\varepsilon$-coreset $(S, w)$ for $(P, u)$
Set $E_u \leftarrow \sum_{i=1}^{n} u_i p_i$
Set $x \leftarrow \sum_{j=1}^{n} u_j \|p_j - E_u\|$
Set $v \leftarrow \sum_{j=1}^{n} u_j \|(p_j - E_u, x)\|$
for $i \leftarrow 1$ to $n$ do
    Set $q_i \leftarrow \frac{(p_i - E_u, x)}{\|(p_i - E_u, x)\|}$
    Set $s_i \leftarrow \frac{u_i \|(p_i - E_u, x)\|}{v}$
Set $H \leftarrow \{q_i - (0, \cdots, 0, x/v) \mid i \in [n]\}$
Set $\alpha \leftarrow$ a sufficiently large constant that can be
    derived from the proof of Theorem 1.
Set $c_1 \leftarrow$ an arbitrary vector in $H$
for $i \leftarrow 1$ to $\beta := \lceil \alpha/\varepsilon \rceil$ do
    $h_{i+1} \leftarrow$ farthest point from $c_i$ in $H$
    $c_{i+1} \leftarrow$ the projection of the origin on the
        segment $\overline{c_i, h_{i+1}}$
Compute $w' = (w'_1, \cdots, w'_\beta) \in S^\beta$ such that
    $c_\beta = \sum_{i=1}^{\beta} w'_i h_{i+1}$
for $i \leftarrow 1$ to $\beta$ do
    $w''i \leftarrow \frac{v w'_i}{(p_i - E_u, x)}$
    $w_i \leftarrow \frac{w''_i}{\sum_{j=1}^{\beta} w''_j}$
$w \leftarrow (w_1, \cdots, w_\beta)$
$S \leftarrow \{v_1, \cdots, w_\beta\}$
return $(S, w)$
```

*rithm 1. Then $w \in D^n$ consists $O(N)$ non-zero entries, such that the sum $\bar{p} = \sum_{i=1} u_i p_i$ deviates from the sum $\hat{p} = \sum_{i=1}^{n} w_i p_i$ by at most a $(1/N)$-fraction of the variance* $\text{var}_u = \sum_{i=1}^{n} u_i \|p_i - \bar{p}\|_2^2$*, i.e.,* $\|\bar{p} - \hat{p}\|_2^2 \leq \frac{\text{var}_u}{N}$.

By the $O(\cdot)$ notation above, it suffices to prove that there is a constant $\alpha > 0$ such that $N \geq \alpha$ and

$$\|E_u - E_w\|_2^2 \leq \frac{\alpha \text{var}_u}{N}, \tag{2}$$

where $E_u = \bar{p}$ and $E_w = \hat{p}$. The proof is constructive and thus immediately implies Algorithm 1. Indeed, let

$$x = \textstyle\sum_j u_j \|p_j - E_u\|, \text{ and } v = \sum_j u_j \|(p_j - E_u, x)\|.$$

Here and in what follows, $\|\cdot\| = \|\cdot\|_2$ and all the sums are over $[n] = \{1, \cdots, n\}$. For every $i \in [n]$ let

$$q_i = \tfrac{(p_i - E_u, x)}{\|(p_i - E_u, x)\|}, \text{ and } s_i = \tfrac{u_i \|(p_i - E_u, x)\|}{v}.$$

Hence,

$$\begin{aligned}\sum_i s_i q_i &= \frac{1}{v} \sum_i u_i (p_i - E_u, x) \\ &= \frac{1}{v} \left( \sum_i u_i p_i - \sum_m u_m E_u, \sum_k u_k x \right) \\ &= \frac{1}{v} \left( \sum_i u_i p_i - \sum_j u_j p_j, x \right) = \left( 0, \cdots, 0, \frac{x}{v} \right).\end{aligned} \tag{3}$$

Since $(s_1, \cdots, s_n) \in D^n$ we have that the point $p = \sum_i s_i q_i$ is in the convex hull of $Q = \{q_1, \cdots, q_n\}$. By applying the Frank-Wolfe algorithm as described in (Clarkson, 2005) for the function $f(s) = \|As\|$, where each row of $A$ corresponds to a vector in $Q$, we conclude that there is $w' = (w'_1, \cdots, w'_n) \in D^n$ that has at most $N$ non-zero entries such that

$$\|\textstyle\sum_i (s_i - w'_i) q_i\|^2 = \left\| \sum_i s_i q_i - \sum_j w'_j q_j \right\|^2 = \|p - q\|^2 \leq \frac{1}{N}. \tag{4}$$

For every $i \in [n]$, define

$$w''_i = \frac{v w'_i}{\|(p_i - E_u, x)\|} \quad \text{and} \quad w_i = \frac{w''_i}{\sum_j w''_j}.$$

We thus have:

$$\|E_u - E_w\|^2 = \left\| \textstyle\sum_i u_i p_i - \sum_j w_j p_j \right\|^2 = \|\textstyle\sum_i (u_i - w_i) p_i\|^2 \tag{5}$$

$$= \|\textstyle\sum_i (u_i - w_i)(p_i - E_u, x)\|^2 \tag{6}$$

$$= v^2 \left\| \textstyle\sum_i \left( \frac{u_i \|(p_i - E_u, x)\|}{v} - \frac{w_i \|(p_i - E_u, x)\|}{v} \right) q_i \right\|^2 \tag{7}$$

$$= v^2 \left\| \textstyle\sum_i \left( s_i - \frac{(w''_i / \sum_j w''_j) \cdot \|(p_i - E_u, x)\|}{v} \right) q_i \right\|^2 \tag{8}$$

$$= v^2 \left\| \textstyle\sum_i \left( s_i - \frac{w'_i}{\sum_j w''_j} \right) q_i \right\|^2, \tag{9}$$

where (5) is by the definitions of $E_u$ and $E_w$, (6) follows since $\sum_i u_i = \sum_i w_i = 1$ and thus $\sum_i u_i y = \sum_j u_j y$ for every vector $y$, (8) follows by the definitions of $w_i$ and $q_i$, and (9) by the definition of $w'_i$. Next, we bound (9). Since for every two reals $y, z$,

$$2yz \leq y^2 + z^2 \tag{10}$$

by letting $y = \|a\|$ and $z = \|b\|$ for $a, b \in \mathbb{R}^d$,

$$\begin{aligned}\|a + b\|^2 &\leq \|a\|^2 + \|b\|^2 + 2\|a\|\|b\| \\ &\leq \|a\|^2 + \|b\|^2 + (\|a\|^2 + \|b\|^2) = 2\|a\|^2 + 2\|b\|^2.\end{aligned} \tag{11}$$

By substituting $a = \sum_i (s_i - w'_i) q_i$ and $b = \sum_i (w'_i - \frac{w'_i}{\sum_j w''_j}) q_i$ in (11), we obtain

$$\left\| \sum_i \left( s_i - \frac{w'_i}{\sum_j w''_j} \right) q_i \right\|^2 \leq 2 \left\| \sum_i (s_i - w'_i) q_i \right\|^2 \tag{12}$$

$$+ 2 \left\| \sum_i \left( w'_i - \frac{w'_i}{\sum_j w''_j} \right) q_i \right\|^2. \tag{13}$$

**Bound on (13):** Observe that

$$\left\|\sum_i \left(w'_i - \frac{w'_i}{\sum_j w''_j}\right) q_i\right\|^2 = \left\|\sum_i w'_i \left(1 - \frac{1}{\sum_j w''_j}\right)\right\|^2$$
$$= \left(1 - \frac{1}{\sum_j w''_j}\right)^2 \cdot \left\|\sum_i w'_i q_i\right\|^2. \tag{14}$$

Let $\tau = \frac{v}{\sqrt{N}x}$. By the triangle inequality

$$v = \sum_j u_j \left\|(p_j - E_u, x)\right\| \leq \sum_j u_j \left\|p_j - E_u\right\| + x = 2x. \tag{15}$$

By choosing $c > 16$ in (2) we have $N \geq 16$, so

$$\tau \leq \tfrac{2}{\sqrt{N}} \leq \tfrac{1}{2}. \tag{16}$$

Substituting $a = -\sum_i s_i q_i$ and $b = \sum_i (s_i - w'_j) q_j$ in (11) bounds the right expression of (14) by

$$\left\|\sum_i w'_i q_i\right\|^2 \leq 2\left\|\sum_i s_i q_i\right\|^2 + 2\left\|\sum_i (s_i - w'_j) q_j\right\|^2$$
$$\leq \frac{2x^2}{v^2} + \frac{2}{N} = \frac{2(1+\tau^2)}{\tau^2 N}, \tag{17}$$

where the last inequality follows from (3) and (4). For bounding the left expression of (14), note that

$$(1 - \sum_j w''_j)^2 = \left(1 - \sum_j w'_j \cdot \frac{v}{\|(p_j - E_u, x)\|}\right)^2 \tag{18}$$
$$\leq \left\|(0, \cdots, 0, 1) - \sum_j w'_j \cdot \frac{(\frac{v}{x}(p_j - E_u), v)}{\|(p_j - E_u, x)\|}\right\|^2$$
$$= \frac{v^2}{x^2}\left\|\left(0, \cdots, 0, \frac{x}{v}\right) - \sum_j w'_j \cdot \frac{(p_j - E_u, x)}{\|(p_j - E_u, x)\|}\right\|^2$$
$$= \frac{v^2}{x^2}\left\|\sum_i (s_i - w'_i) q_i\right\|^2 \leq \frac{v^2}{N x^2} = \tau^2,$$

where (18) follows since $\|b\|^2 \leq \|(a,b)\|^2$ for every pair $a, b$ of vectors, and the last inequality is by (3) and (4). Hence, $\sum_j w''_j \geq 1 - \tau$, and

$$\left(1 - \tfrac{1}{\sum_j w''_j}\right)^2 = \left(\tfrac{1 - \sum_i w''_i}{\sum_j w''_j}\right)^2 \leq \tfrac{\tau^2}{(1-\tau)^2}.$$

Combining (14) and (17) bounds (13) by

$$2\left\|\sum_i \left(w'_i - \frac{w'_i}{\sum_j w''_j}\right) q_i\right\|^2 = 2\left(1 - \frac{1}{\sum_j w''_j}\right)^2 \cdot \left\|\sum_i w'_i q_i\right\|^2$$
$$\leq \frac{2\tau^2}{(1-\tau)^2} \cdot \frac{2(1+\tau^2)}{\tau^2 N} = \frac{4(1+\tau^2)}{N(1-\tau)^2}.$$

**Bound on (12):** Plugging the last inequality, (4) and (12) in (9) yields

$$\|E_u - E_w\|^2 = v^2 \left\|\sum_i \left(s_i - \frac{w'_i}{\sum_j w''_j}\right) q_i\right\|^2$$
$$\leq 2v^2\left(\left\|\sum_i (s_i - w'_i) q_i\right\|^2 + \left\|\sum_i \left(w'_i - \frac{w'_i}{\sum_j w''_j}\right) q_i\right\|^2\right)$$
$$\leq \frac{2v^2}{N}\left(1 + \frac{2(1+\tau^2)}{(1-\tau)^2}\right) \leq \frac{\alpha v^2}{N}, \tag{19}$$

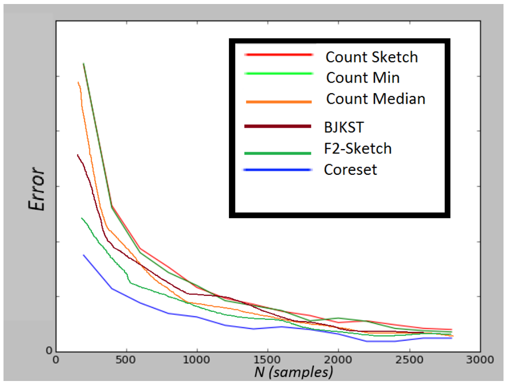


*Figure 2.* Given a set of vectors from a standard Gaussian distribution, the graph shows the $\ell_2$ error (y-axis) between their sum and their approximated sum using only $N$ samples (the $x$-axis) based on Count Sketch (Charikar et al., 2004), Count Min (Cormode & Muthukrishnan, 2005a), Count Median (Cormode & Muthukrishnan, 2005b), BJKST (Bar-Yossef et al., 2002), F2-Sketch (Alon et al., 1996b), and our coreset.

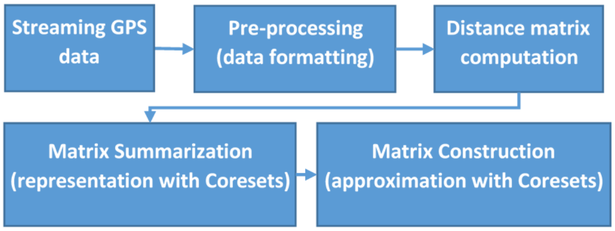

Streaming GPS data
Pre-processing (data formatting)
Distance matrix computation
Matrix Summarization (representation with Coresets)
Matrix Construction (approximation with Coresets)

*Figure 3.* The overview of our designed system to extract and represent social networks is given.

for a sufficiently large constant $\alpha$, e.g. $\alpha = 3$, where in the last inequality we used (16). Since $v \leq 2x$ by (15) we have

$$v \leq 2x = 2\sum_j u_j \|p_j - E_u\|$$
$$= \sum_j 2 \cdot \sqrt{\sqrt{\mathrm{var}_u}\sqrt{u_j}} \cdot \frac{\sqrt{u_j}\,\|p_j - E_u\|}{\sqrt{\sqrt{\mathrm{var}_u}}}$$
$$\leq \sum_j \left(\sqrt{\mathrm{var}_u}\, u_j + \frac{u_j \|p_j - E_u\|^2}{\sqrt{\mathrm{var}_u}}\right)$$
$$= \sqrt{\mathrm{var}_u} + \frac{1}{\sqrt{\mathrm{var}_u}}\sum_j u_j \|p_j - E_u\|^2 = 2\sqrt{\mathrm{var}_u},$$

where in the second inequality we used (10). Plugging this in (19) and replacing $N$ by $4\alpha N = O(N)$ in the proof above, yields the desired bound

$$\|E_u - E_w\|^2 \leq \tfrac{\alpha v^2}{N} \leq \tfrac{4\alpha \cdot \mathrm{var}_u}{N}.$$

## 5. Experimental Results

We implemented the coreset algorithms in 1 and 2. We also implemented a brute force method for determining the social network that considers the entire data. We used this method to derive the ground truth for the social network for small scale data. Our system's overview is given in Figure 3 and explained in Section 1.1. The coreset algorithm computes the heavy hitters by approximating the sum of

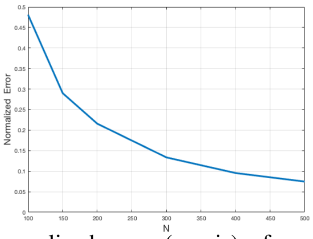


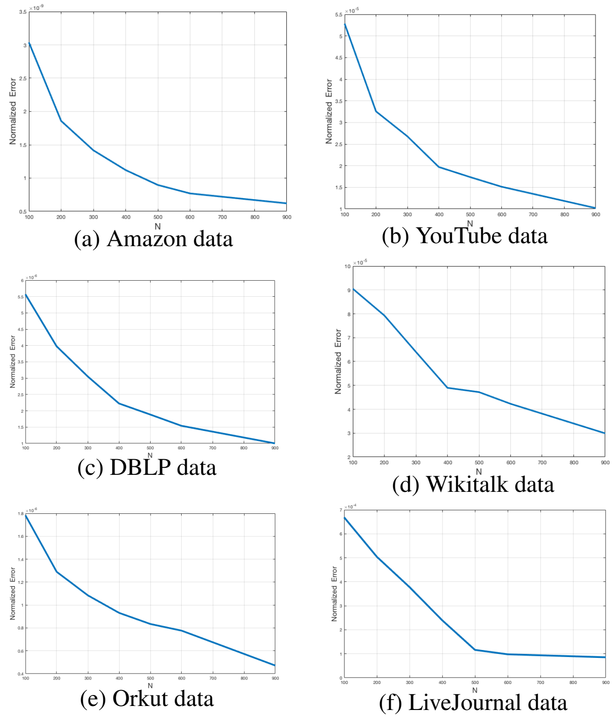
*Figure 4.* The normalized error (y-axis) of proximities for the GPS traces taxi-drivers in the New York City Cab dataset using only $N$ samples (N increases along x-axis).

the columns of the proximity matrix as explained in Section 1.1. In this section, we have used different data sets from three different sources: In our first experiment, we compared our coreset algorithm's error with other sketch algorithms (Charikar et al., 2004; Cormode & Muthukrishnan, 2005a). The second dataset is the New York City Cab dataset[1] and the third data set is from Stanford[2] and includes six different graph-based data sets. In all the figures shown in this section, the x axis shows the coreset size (N) and the y axis represents the normalized error value $(Error * mean(var(p_i)^2)/mean(norm(p_i)))$, where $Error = \|p_i - \bar{p}\|_2$. In our experiments, we ran N iterations and wrote down the empirical error $\epsilon$.

**Comparison to sketch algorithms:** Since the algorithms in (Charikar et al., 2004; Cormode & Muthukrishnan, 2005a) focused on selecting the entries at scalar level (i.e., individual entries from a vector), in this experiment, we generated a small scale synthetic data (standard Gaussian distribution) and compared the error made by our coreset implementation to four other sketch implementations. These sketch algorithms are: Count Sketch, Count Min, Count Median, BJKST and F2-Sketch (see Fig. 2). For the sketch algorithms, we used the code available at[3]. We plot the results in Fig. 2 where our Coresets algorithm showed better approximation than all other well known sketch techniques for all N values.

**Application on NYC data:** Here we applied our algorithm on the NYC data. The data contains the location information of 13249 taxi cabs with 14776616 entries. The goal here is showing how the error on Coreset approximation would change on real data with respect to N (we expect that the error would reduce with respect to $N$ as the theory suggests). This can be seen in Fig. 4, where x axis is the coreset size (N) and y axis is the normalized error.

**Application on Stanford Data Sets:** Here we apply our algorithm on six different data sets from Stanford: Amazon, Youtube, DBLP, LiveJournal, Orkut and Wikitalk data sets. We run the Coreset algorithm to approximate the total number of connectivities each node has. We computed the error for each of the seven different N values from [100, 200, 300, 400, 500, 600, 900] for each data set. We used the first 50000 entries from the Orkut, Live Journal, Youtube and Wiki data sets and the first 5000 entries from Amazon and DBLP data set. The results are shown in Figure 5. In the figures, y axis represents the normalized error. The results demonstrate the utility of our proposed method for summarization.



(a) Amazon data

(b) YouTube data

(c) DBLP data

(d) Wikitalk data

(e) Orkut data

(f) LiveJournal data

*Figure 5.* The normalized error (y-axis) of the coreset for network structure approximation is shown for four different datasets using only $N$ samples (N increases along x-axis).

## 6. Conclusion

In this paper we proposed a new coreset algorithm for streaming data sets with applications to summarizing large networks to identify the "heavy hitters". The algorithm takes a stream of vectors as input and maintains their sum using small memory. Our presented algorithm shows better performance at even lower values of non-zero entries (i.e., at higher sparsity rates) when compared to the other existing sketch techniques. We demonstrated that our algorithm can catch the heavy hitters efficiently in social networks from the GPS-based location data and in several graph data sets from the Stanford data repository.

## Acknowledgements

Support for this research has been provided in part by Ping An Insurance and NSFSaTC-BSF CNC 1526815. We are grateful for this support.

[1] https://publish.illinois.edu/dbwork/open-data/

[2] https://snap.stanford.edu/data/

[3] https://github.com/jiecchen/StreamLib/

## References

Alon, Noga, Matias, Yossi, and Szegedy, Mario. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 20–29. ACM, 1996a.

Alon, Noga, Matias, Yossi, and Szegedy, Mario. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pp. 20–29. ACM, 1996b.

Bar-Yossef, Ziv, Jayram, TS, Kumar, Ravi, Sivakumar, D, and Trevisan, Luca. Counting distinct elements in a data stream. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pp. 1–10. Springer, 2002.

Barger, Artem and Feldman, Dan. $k$-means for streaming and distributed big sparse data. *SDM'16 and arXiv preprint arXiv:1511.08990*, 2015.

Bentley, Jon Louis and Saxe, James B. Decomposable searching problems i. static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

Carrington, Peter J, Scott, John, and Wasserman, Stanley. *Models and methods in social network analysis*, volume 28. Cambridge university press, 2005.

Charikar, Moses, Chen, Kevin, and Farach-Colton, Martin. Finding frequent items in data streams. *Theoretical Computer Science*, 312(1):3–15, 2004.

Clarkson, K. L. Subgradient and sampling algorithms for $l_1$-regression. In *Proc. 16th Annu. ACM-SIAM Symp. on Discrete algorithms (SODA)*, pp. 257–266, 2005. ISBN 0-89871-585-7.

Cormode, Graham and Muthukrishnan, S. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005a.

Cormode, Graham and Muthukrishnan, S. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005b.

Dinh, Thang N, Xuan, Ying, Thai, My T, Park, EK, and Znati, Taieb. On approximation of new optimization methods for assessing network vulnerability. In *INFOCOM, 2010 Proceedings IEEE*, pp. 1–9. IEEE, 2010.

Dinh, Thang N, Nguyen, Nam P, and Thai, My T. An adaptive approximation algorithm for community detection in dynamic scale-free networks. In *INFOCOM, 2013 Proceedings IEEE*, pp. 55–59. IEEE, 2013.

Feldman, Dan and Tassa, Tamir. More constraints, smaller coresets: constrained matrix approximation of sparse big data. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15)*, pp. 249–258. ACM, 2015.

Feldman, Dan, Volkov, Mikhail, and Rus, Daniela. Dimensionality reduction of massive sparse datasets using coresets. In *NIPS*, 2016. URL http://arxiv.org/abs/1503.01663.

Har-Peled, Sariel. Coresets for discrete integration and clustering. In *FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science*, pp. 33–44. Springer, 2006.

Hogan, Bernie. Analyzing social networks. *The Sage handbook of online research methods*, pp. 141, 2008.

Lancichinetti, Andrea and Fortunato, Santo. Community detection algorithms: a comparative analysis. *Physical review E*, 80(5):056117, 2009.

Liao, Lin. *Location-based activity recognition*. PhD thesis, University of Washington, 2006.

Nguyen, Nam P, Dinh, Thang N, Xuan, Ying, and Thai, My T. Adaptive algorithms for detecting community structure in dynamic social networks. In *INFOCOM, 2011 Proceedings IEEE*, pp. 2282–2290. IEEE, 2011.

Wasserman, Stanley. Analyzing social networks as stochastic processes. *Journal of the American statistical association*, 75(370):280–294, 1980.

Zheng, Yu. Location-based social networks: Users. In *Computing with spatial trajectories*, pp. 243–276. Springer, 2011.