SN Computer Science (2020) 1:215
https://doi.org/10.1007/s42979-020-00227-7

ORIGINAL RESEARCH

A Comparative Study of the Some Methods Used in Constructing Coresets for Clustering Large Datasets

Nguyen Le Hoang[1,2] · **Le Hong Trang**[1,2] · **Tran Khanh Dang**[1,2]

Received: 6 May 2020 / Accepted: 12 June 2020 / Published online: 27 June 2020

Abstract
Coresets can be described as a compact subset such that models trained on coresets will also provide a good fit with models trained on full data set. Using coresets, we can scale down a big data to a tiny one to reduce the computational cost of a machine learning problem. In recent years, data scientists have investigated various methods to create coresets with many techniques and approaches, especially for solving the problem of clustering large datasets. In this paper, we make comparisons among four state-of-the-art algorithms: ProTraS by Ros and Guillaume with improvements, Lightweight Coreset by Bachem et al. Adaptive Sampling Coreset by Feldman et al. and a native Farthest-First-Traversal-based coreset construction. We briefly introduce these four algorithms and compare their performances to find out the benefits and drawbacks of each one.

Keywords Machine learning · Big data · Coreset · Clustering · Farthest-first-traversal · Sampling

Introduction

The development of technology such as smart city and IoT has lead to the rapid increasing of data. Consequently, machine learning has to face new hard situation: solving problems for data that have big amount in volume, variety, velocity and veracity. Many methods have been proposed for several years to deal with big data. One of the simplest way is depending on infrastructure and hardware, but few people can afford this. Another option is finding suitable algorithms to reduce the computational complexity from the input size that may contain millions or billions of data points. The idea of finding a relevant subset from original data to decrease the computational cost brings scientists to the concept of coreset, which was first applied in geometric approximation by Agarwal et al. [1, 2]. The problem of coreset constructions for k-median and k-means clustering was then stated and investigated by Har-Peled et al. [17, 18, 19]

In recent years, many coreset construction algorithms have been proposed for a wide variety of clustering problems [10, 12, 13, 22]. These research always try to find good algorithms that create samples that are more correct or being faster. Even though there are many investigations about this, two approaches that fascinates us are the sampling-based methods and farthest-first-traversal-based algorithms. In each techniques, there are also various researches. In this paper, we focus on four state-of-the-art researches as follows

- The first one is ProTraS algorithm which is proposed in 2018 by Ros and Guillaume [29]. In this paper, we use the version with improvement proposed in [31].
- The second coreset method is an idea based on native Farthest-First-Traversal algorithm in [32].
- The third coreset construction is Adaptive Sampling by Feldman et al. [14].
- The last one is the Lightweight Coreset by Bachem et al. [7]. The lightweight coreset defines a new type of coreset, the lightweight form, but the samples created for this type can be considered as a coreset which provide a good fit with full data set.
- Besides, we also use Uniform or Naive Sampling as baseline for comparison.

This article is part of the topical collection "Future Data and Security Engineering 2019" guest edited by Tran Khanh Dang.

✉ Tran Khanh Dang
khanh@hcmut.edu.vn

[1] Ho Chi Minh City University of Technology (HCMUT), 268 Ly Thuong Kiet, District 10, Ho Chi Minh City, Vietnam

[2] Vietnam National University Ho Chi Minh City (VNU-HCM), Linh Trung Ward, Thu Duc District, Ho Chi Minh City, Vietnam

The remaining of this paper is organized as follows. In "Background and Related Works", we discuss some related works and background used in this paper. In "Coreset Construction Methods Used for Comparisons", we introduce briefly about these four coreset construction methods. We do experiments and comparisons using relative error in "Comparative Results" then have discussions about the advantages as well as the disadvantages of each method. We end this paper by the conclusion in "Conclusions" of this paper.

Background and Related Works

k-Median and *k*-Means Clustering

- *k-means clustering* is a popular method, originally from signal processing, for cluster analysis in data mining. The standard algorithm was first proposed by Lloyd of Bell Labs in 1957, and was published later in [21]. The algorithm was then developed by Inaba et al. [24], Vega et al. [33], Matousek [25], or the *k*-Means++ by Author and Vassilvitskii in [4].
- *k-median clustering* is a variation of *k*-means where instead of calculating the mean for each cluster to determine its centroid, one instead calculates the median. There are also plenty research about this algorithm such as Arora [3], Charikar et al. [9], etc.

In this paper, we refer *k*-clustering for both *k*-median and *k*-means. The k-clustering problems can be stated as follows:

Let $X \subset \mathbb{R}^d$, the *k*-clustering problems are to find $Q \subset \mathbb{R}^d$ with $|Q| = k$ such that these functions are minimized

$$\phi_X(Q) = \sum_{x \in X} d(x, Q) = \sum_{x \in X} \min_{q \in Q} ||x - q|| \tag{1}$$

$$\phi_X(Q) = \sum_{x \in X} d(x, Q)^2 = \sum_{x \in X} \min_{q \in Q} ||x - q||^2 \tag{2}$$

Equations (1) and (2) are for k-median and k-means respectively.

Coresets for *k*-Median and *k*-Means Clustering

If the data set X mentioned above is big enough, it is hard and expensive to solve these *k*-median and *k*-means problems properly. Therefore, instead of solving on X, one of the classical techniques is the extraction of small amount information from the given data, and performing the computation on this extracted subset. However, in many circumstances, it is not easy to find this most relevant subset. Consequently, attention has shifted to developing approximation algorithms. The goal now is to compute an $(1 + \varepsilon)$-approximation subset, for some $0 < \varepsilon < 1$. The framework of coresets has recently emerged as a general approach to achieve this goal [2].

The definition of coresets for *k*-median and *k*-means can be stated as:

Definition 1 Coresets for k-median and k-means clustering. Let $\varepsilon > 0$, the weighted set C is a (k, ε)- coreset of X if for any $Q \subset \mathbb{R}^d$ of cardinality at most k

$$|\phi_X(Q) - \phi_C(Q)| \leq \varepsilon \phi_X(Q); \tag{3}$$

this also equivalent to

$$(1 - \varepsilon)\phi_X(Q) \leq \phi_C(Q) \leq (1 + \varepsilon)\phi_X(Q). \tag{4}$$

Coreset Construction Methods Used for Comparisons

Many coreset construction algorithms have been proposed in recent years for clustering problems. As a result, various approaches have been investigated such as exponential grids [11], bounding points [19] or dimension reduction [15]. In this paper, we focus on two other techniques that have been used in recent researches: farthest-first-traversal-based algorithms and sampling-based methods.

Farthest-First-Traversal (FFT)-Based Algorithm

In computational geometry, the FFT of a metric space is a set of points selected sequently; after the first point is chosen arbitrarily, each next successive point is located as the farthest one from the set of previously selected points. The first use of the FFT was by Rosenkrantz et al. [30] in connection with heuristics for the traveling salesman problem. Then, Gonzalez [16] used it as part of a greedy approximation algorithm for the problem of finding k clusters that minimize the maximum diameter of a cluster. Later, Arthur and Vassilvitskii [4] use a FFT-like algorithm to propose *k*-means++ algorithm.

Algorithm 1 Farthest-First-Traversal algorithm

Require: dataset X with $|X| = n$
Ensure: $C \subset X$ with $|C| = m \leq n$
1: Pick $C = \{x_0\}$
2: **while** $(|C| < m)$ **do**
3: Find $T = \{t | \min d(t, x), \forall x \in C\}$
4: Find $x_* = \max(T)$
5: $C = C \cup \{x_*\}$
6: **end while**
7: **return** C

In 2018, Ros and Guillaume proposed DENDIS [27], DIDES [28] and ProTraS [29] which are based on FFT algorithm. These are iterative algorithms based on the hybridization of distance and density concepts. They differ in the

priority given to distance or density, and in the stopping criterion defined accordingly. After these proposal methods, there are plenty of improvements for coreset constructions that are also based on FFT. Two state-of-the-art methods are proposed in [31] and in [32]. We use these two for comparisons in next chapter.

ProTraS Post-Processing Improvement

Authors in [31] proposed an improvement for ProTraS. The idea of this improvement is the post-processing task of ProTraS by replacing a representative in the sample obtained by ProTraS by the center of the group represented by it. Thereby, objects located at the boundary side of clusters will be replaced by interior ones of those. The new obtained sample, thus, has separated clusters. This helps to improve the quality of clustering process. This algorithm is described in Algorithm 2.

Algorithm 2 ProTraS post-processing improvement [31]

Require: $P = \{x_i\}$, for $i = 1, 2, \ldots, n$, a tolerance $\epsilon > 0$.
Ensure: A sample S and D'^*.
1: Call ProTraS for P and ϵ to obtain $S = \{y_j\}$ and $P(y_j)$.
2: $S' = \emptyset$.
3: **for all** $y_j \in S$ **do**
4: $y_k^* = \arg\min_{y_k \in P(y_j)} \sum_{y_l \in P(y_j)} d(y_k, y_l)$.
5: $S' = S' \cup \{y_k^*\}$.
6: **end for**
7: Form D^* the reordered matrix corresponding to S'.
8: Apply iVAT on D^* to obtain D'^* and produce $I(D'^*)$.
9: **return** S and D'^*.

FFT-Based with Pre-processing Improvement

ProTraS provides a good method to build coresets, but it still has some drawbacks. To overcome these problems, authors in [32] use a native FFT with pre-processing strategies to create a coreset. The first one is to find a specific initial point instead of randomly and the second one is a technique to reduce the computational complexity to make the construction much more faster. This algorithm is described in Algorithm 3.

Algorithm 3 FFT-based with pre-processing improvement [32]

Require: $X = \{x_i\}; i = 1, 2, ..., n$; and m
Ensure: $C = \{c_j\}; T(c_j), j = 1, 2, ..., m$
1: Pick an initial pattern $x_{init} \in X$
2: $C = \{c_1 = x_{init}\}; T(c_1) = \{c_1\}$
3: **while** $(|C| < m)$ **do**
4: **for** $(x_i \in X \setminus C)$ **do**
5: Find $d_{near}(x_i) = \min_{c \in C} d(x_i, c)$
6: $T(c_k) = T(c_k) \cup \{x_i\}$
7: **end for**
8: $MaxWD = 0$
9: **for** $(c_j \in C)$ **do**
10: Find $d_{max}(c_j) = d(x_{max}(c_j), c_j) = \max_{x_t \in T(c_j)} d(x_t, c_j)$
11: $p_j = |T(c_j)| * d_{max}(c_j)$
12: **if** $(p_j > MaxWD)$ **then**
13: $MaxWD = p_j; c_* = c_j$
14: **end if**
15: **end for**
16: $x_* = x_{max}(c_*)$
17: $C = C \cup \{x_*\}; T(c_{new}) = \{x_*\}$
18: **end while**
19: **return** $C; T(y_j), j = 1, 2, ..., m$

Sampling-Based Methods

Sampling is a definition from Statistics. This is the selection of a subset of individuals from a population or original set according to a specific probability or distribution. The process of finding coresets based on sampling is quite simple and fast. However, it requires a lot of mathematical proofs and theorems behind. Like other approaches, there are many researches for sampling-based coreset constructions. In this paper, along with naive or uniform sampling as the baseline for comparisons, we use two most effective and well-proved methods: the Adaptive Sampling [14] and the Lightweight Coreset [7].

Adaptive Sampling

This algorithm is proposed in [14]. The key idea is to build an approximate solution (sample set, C) and to use it to bias the random sampling. The first step is achieved by an iterative algorithm that samples a small number of points, and removes half of the data set X closest to the sampled points. In the second step, the sampling is biased with probabilities, for each point in X , which are roughly proportional to their squared distance to C. This algorithm is widely used in many researches and is described in Algorithm 4.

Algorithm 4 Adaptive Sampling [14]

Require: Data set $D, \varepsilon, \delta, k$
Ensure: Coreset $C = \{(\gamma(x_1), x_1), ..., (\gamma(x_{|C|}), x_{|C|})\}$
1: $D' \leftarrow D; B \leftarrow \emptyset$;
2: **while** $(|D'| > 10dk \ln(1/\delta))$ **do**
3: Sample set S of $\beta = 10dk \ln(1/\delta)$ points uniformly at random from D';
4: Remove $\lceil |D'|/2 \rceil$ points $x \in D'$ closest to S (i.e. minimizing $dist(x, S)$) from D';
5: Set $B \leftarrow B \cup S$;
6: **end while**
7: $B \leftarrow B \cup D'$;
8: **for** each $b \in B$ **do**
9: $D_b \leftarrow$ the points in D whose closest point in B is b. Ties broken arbitrarily;
10: **end for**
11: **for** each $b \in B$ and $x \in D_b$ **do**
12: $m(x) \leftarrow \lceil \frac{5}{|D_b|} + \frac{dist(x,B)^2}{\sum_{x' \in D} dist(x',B)^2} \rceil$;
13: **end for**
14: Pick a non-uniform random sample C of $10\lceil dk|B|^2 \ln(1/\delta)/\varepsilon^2 \rceil$ points from D, where for every $x' \in C$ and $x \in D$, we have $x' = x$ with probability $\frac{m(x)}{\sum_{x' \in D} m(x')}$;
15: **for** each $x' \in C$ **do**
16: $\gamma(x') \leftarrow \frac{\sum_{x \in Dm(x)}}{|C|.m(x')}$;
17: **end for**
18: **return** C

Lightweight Coresets

In inequality (3) of coreset definition, the right term $\varepsilon\phi_X(Q)$ allows the approximation error to scale with the quantization error as well as to include both the additive and multiplicative errors. Bachem et al. [5, 6, 8, 7] interpret and split these errors that lead to the definition of Lightweight Coresets as follows

Definition 2 Lightweight Coresets for k-clustering. Let $\varepsilon > 0$ and $k \in \mathbb{N}$. Let $X \subset \mathbb{R}^d$ be a set of points with mean

$\mu(X)$. The weighted set C is an (ε, k)-lightweight coreset of X if for any set $Q \subset \mathbb{R}^d$ of cardinality at most k

$$|\phi_X(Q) - \phi_C(Q)| \leq \frac{\varepsilon}{2}\phi_X(Q) + \frac{\varepsilon}{2}\phi_X(\{\mu(X)\}) \tag{5}$$

In inequality (5), the $\frac{\varepsilon}{2}\phi_X(Q)$ term allows the approximation error to scale with the quantization error and constitutes the multiplicative part; while the $\frac{\varepsilon}{2}\phi_X(\{\mu(X)\})$ term scales with the variance of the data and corresponds to the additive approximation error term that is invariant of the scale of the data.

Even though there are differences in definitions between Coresets and Lightweight Coresets, Bachem et al. [7] have shown that as we decrease ε, the true cost of the optimal solution obtained on the lightweight coreset approaches the true cost of the optimal solution on the full data set in an additive manner.

Construction of Lightweight Coresets The construction is based on importance sampling. Let $q(x)$ be any probability distribution on X and Q any set of k centers in $\mathbb{R}^d$.The quantization error can be approximated by sampling m points from X using $q(x)$ and assigning them weights inversely proportional to $q(x)$. The $q(x)$ is defined as follows

$$q(x) = \frac{1}{2}\frac{1}{|X|} + \frac{1}{2}\frac{d(x, \mu(X))^2}{\sum_{x' \in X} d(x', \mu(X))^2} \tag{6}$$

The construction for lightweight coreset is described as in Algorithm 5

Algorithm 5 Lightweight Coreset [7]

Require: Set of data points X, coreset size m
Ensure: Lightweight Coreset C
1: $\mu \leftarrow$ mean of X
2: **for all** $x \in X$ **do**
3: $q(x) \leftarrow \frac{1}{2}\frac{1}{|X|} + \frac{1}{2}\frac{d(x,\mu)^2}{\sum_{x' \in X} d(x',\mu)^2}$
4: **end for**
5: $C \leftarrow$ sample points from where each point is sampled with probability and has weight $w_x = \frac{1}{m \cdot q(x)}$
6: **return** lightweight coreset C

Comparative Results

In this section, we do comparisons among these five methods

- Uniform Sampling: a naive approach to coreset constructions which is based on uniform sub-sampling of the data. This may be regarded as the baseline since it is commonly used in practice.
- ProTraS post-processing improvement: this is described in Algorithm 2. The idea is based on ProTraS with the post-processing to improve the correctness [31].
- FFT-based with pre-processing improvement in Algorithm 3 [32].
- Adaptive Sampling: method to construct coresets based on Gaussian Mixture Models and is described in Algorithm 4 [14], [23].
- Lightweight Coreset: the method mentioned in Algorithm 5. The idea is to perform importance sampling where data point is sampled with probability $\frac{1}{2}$ uniformly at random or with probability $\frac{1}{2}$ proportional to its squared distance to the mean of the data [7].

Data for Experiment

We use 15 datasets from data clustering repository of the computing school of Eastern Finland University [34], and from GitHub clustering benchmark [35]. These datasets are described in Table 1. We display some data examples in Fig. 1

Experiment Setup

Since these five algorithms need different input parameters, ProTraS and algorithm 2 need the value of ε while the others need sample size as input. Therefore, we first run ProTraS with post-processing in algorithm 2 for $\varepsilon = 0.1$ and $\varepsilon = 0.2$, then we use the sample size from results of this first step and used as the input parameter for the Uniform Sampling, FFT-based with pre-processing improvement in Algorithm 3 [32], Adaptive Sampling in Algorithm 4 [14] and Lightweight Coreset in Algorithm 5 [7].

The experiment for each data set is described as follows:

1. Step 1. Use k-means++ [4] to cluster the full data set
2. Step 2. Generate coreset by improved ProTraS
   (a) Step 2.1. Apply ProTraS algorithm to full data set
   (b) Step 2.2. Apply algorithm 2 to the sample from step 2.1
   (c) Denote m as the sample size of the coreset received from step 2.2
3. Step 3. Generate samples by FFT-based construction in algorithm 3
4. Step 4. Generate samples by Uniform Sampling with size m
5. Step 5. Generate samples by Adaptive Sampling in algorithm 4 with size m
6. Step 6. Generate samples for Lightweight Coreset with size m by algorithm 5
7. Step 7. Use k-means++ to solve the k-means clustering problem on each subsample.
8. Step 8. We measure the elapsed time and compute the relative error for each method and subsample size compared to the full solution from step 1.

Table 1 Datasets for experiments

Data ID	Data name	Size	No. clusters
D1	Flame	240	2
D2	Jain	373	2
D3	Aggregation	788	7
D4	R15	600	15
D5	D31	3100	31
D6	Unbalance	6500	8
D7	A1	3000	20
D8	A2	5250	35
D9	A3	7500	50
D10	S1	5000	15
D11	S2	5000	15
D12	S3	5000	15
D13	S4	5000	15
D14	t4.8k	8000	6
D15	Birch3	100,000	100

Since the experiments of uniform sampling and lightweight coresets are randomized, we run them 20 times with different random seeds and compute sample averages.

All experiments were implemented in Python and run on an Intel Core i7 machine with 8-2.8GHz processors and 16 GB memory.

Results and Discussion

In the experiments, we compare five coreset construction methods:

- Uniform sampling as the baseline, denoted as "Uniform"
- ProTraS with the post-processing improvements, denoted as "iProTraS"
- FFT-based coreset with pre-processing improvement, denoted as "sizeFFT"
- Adaptive Sampling, denoted as "Adaptive"
- Lightweight Coreset, denoted as "lwCoreset"

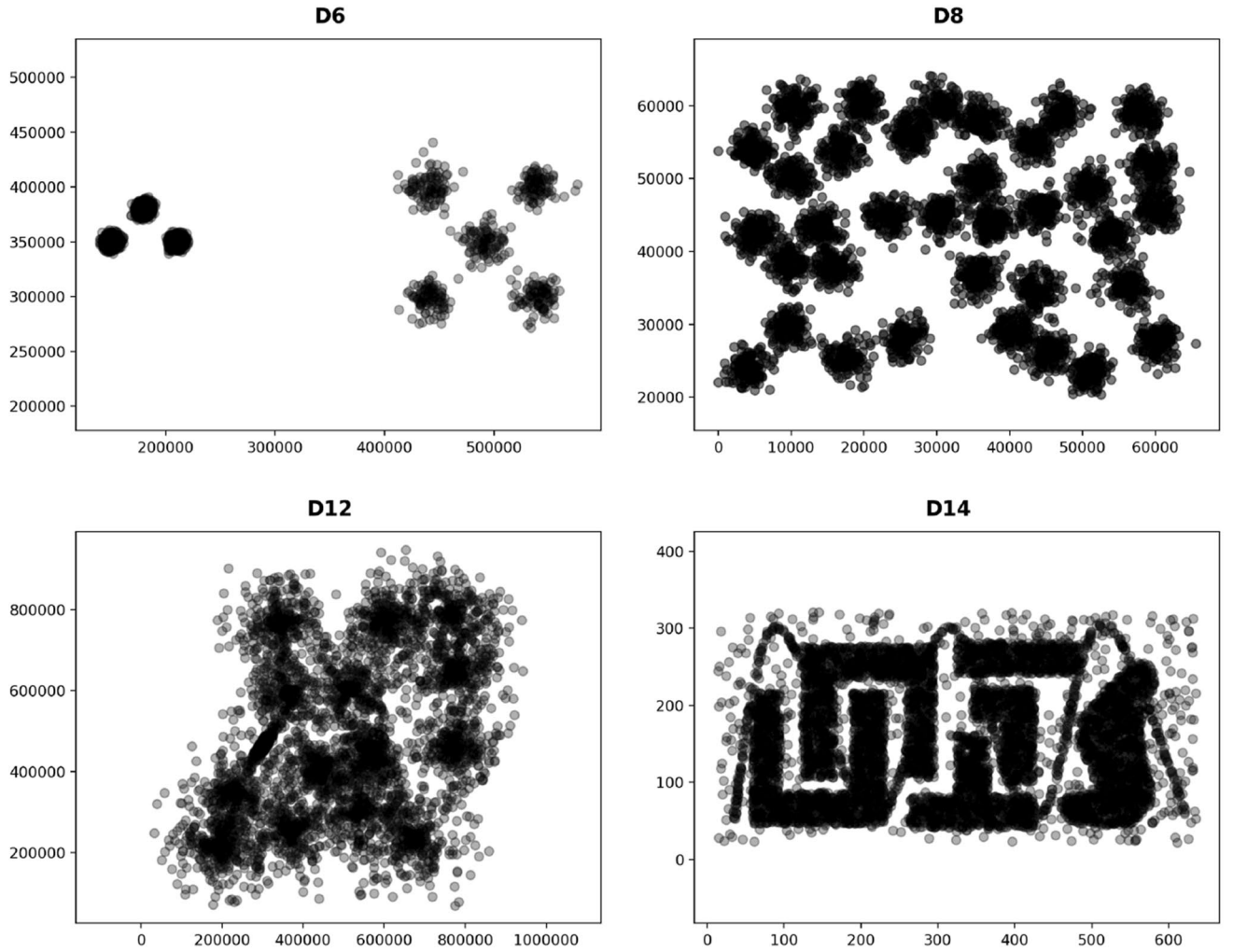

Fig. 1 Some datasets for experiments

We use relative error as the measurement for correctness and the run-time comparison. The results are expressed as follows:

- The relative errors are shown in Table 2 for $\epsilon = 0.1$ and in Table 3 for $\epsilon = 0.2$. For this type of measurement, smaller means better.
- The time comparison is shown in Table 4 for $\epsilon = 0.1$ and in Table 5 for $\epsilon = 0.2$. This measurement is estimated in second (s) unit. Here, smaller means faster.
- Figures 2, 3 and 4 show the relations between relative error and subsample sizes.

Discussions

- All Figs. 2,3 and4 show that the relative errors decrease for all methods as the sample size is decreased. More accurate coresets we can receive if we get more points.
- In most cases, Uniform Sampling creates the high error values, it means Uniform Sampling is the worst coreset construction. This is understandable since Uniform Sampling is the simplest and naivest method. However, this is the fastest method.
- Adaptive Sampling creates coresets having low error in most cases, especially if the clusters are well separated.

Table 2 Experiment results—relative error values with $\epsilon = 0.1$

DataID	$\epsilon = 0.1$					
	Size	Uniform	iProTraS	sizeFFT	Adaptive	lwCoreset
D1	166	0.0021	0.0007	0.0016	0.0085	0.0128
D2	108	0.0265	0.0272	0.0195	0.0920	0.0852
D3	202	0.0524	0.0256	0.0478	0.1724	0.2205
D4	184	0.0681	0.5659	0.1327	0.2583	0.1080
D5	851	0.0337	0.0247	0.0282	0.4168	0.0261
D6	176	0.1815	0.0369	0.0411	0.3125	0.0711
D7	261	0.1180	0.2564	0.0912	0.1219	0.1704
D8	315	0.0754	0.3161	0.1017	0.4756	0.1063
D9	341	0.3194	0.4441	0.4850	0.2781	0.3290
D10	237	0.5862	0.0241	0.0852	0.3217	0.6890
D11	327	0.1780	0.1596	0.1137	0.3021	0.1513
D12	422	0.0466	0.1804	0.0795	0.2176	0.0581
D13	448	0.1020	0.3030	0.0865	0.1349	0.1967
D14	1532	0.0084	0.0153	0.0093	0.0241	0.0552
D15	424	0.4662	0.5454	0.4736	0.6185	0.4869

Table 3 Experiment results—relative error values $\epsilon = 0.2$

DataID	$\epsilon = 0.2$					
	Size	Uniform	iProTraS	sizeFFT	Adaptive	lwCoreset
D1	90	0.0509	0.0246	0.0227	0.0435	0.0852
D2	56	0.0662	0.0136	0.0376	0.1196	0.1696
D3	130	0.2217	0.0030	0.0038	0.1163	0.1396
D4	97	0.4483	0.5308	0.4921	0.3876	0.5478
D5	329	0.2440	0.2821	0.5486	0.5327	0.0173
D6	89	0.3412	0.1132	0.2417	0.2132	0.1877
D7	97	0.7396	0.7936	0.6148	0.8216	0.7588
D8	116	0.5928	0.4947	0.4184	0.5719	0.3260
D9	119	1.4656	1.5380	1.1527	0.9756	1.2776
D10	96	0.7154	1.5268	0.9174	0.9910	1.5437
D11	120	0.6766	0.2340	0.3167	0.8013	0.6150
D12	155	0.4842	0.2953	0.3210	0.4448	0.3965
D13	166	0.2308	0.2725	0.2263	0.2546	0.1273
D14	1126	0.0562	0.0300	0.0238	0.0617	0.0341
D15	153	0.1144	0.8543	0.7910	0.9102	0.8621

Table 4 Experiment results—runtime (in second(s)) of each sample

DataID	$\epsilon = 0.1$					
	Size	Uniform	iProTraS	sizeFFT	Adaptive	lwCoreset
D1	166	0.0004	88	90	0.0010	0.0006
D2	108	0.0003	50	46	0.0017	0.0007
D3	202	0.0004	533	498	0.0019	0.0006
D4	184	0.0003	319	287	0.0021	0.0006
D5	851	0.0005	129040	128682	0.0109	0.0009
D6	176	0.0003	3091	2991	0.0068	0.0010
D7	261	0.0003	4290	4176	0.0128	0.0008
D8	315	0.0003	12776	12948	0.0037	0.0009
D9	341	0.0004	23068	22871	0.0216	0.0012
D10	237	0.0004	5804	5862	0.0163	0.0009
D11	327	0.0003	14343	14290	0.0129	0.0010
D12	422	0.0003	29844	30174	0.0114	0.0009
D13	448	0.0004	32860	30857	0.0138	0.0009
D14	1532	0.0007	1928116	1939418	0.3620	0.0015
D15	424	0.0028	559332	557980	0.1756	0.0116

Table 5 Experiment results—runtime (in second(s)) of each sample

DataID	$\epsilon = 0.2$					
	Size	Uniform	iProTraS	sizeFFT	Adaptive	lwCoreset
D1	90	0.0003	18	25	0.0029	0.0004
D2	56	0.0002	10	13	0.0012	0.0003
D3	130	0.0003	164	153	0.0104	0.0004
D4	97	0.0003	59	48	0.0018	0.0004
D5	329	0.0003	8300	8189	0.0401	0.0006
D6	89	0.0004	500	487	0.0076	0.0009
D7	97	0.0002	294	276	0.0025	0.0006
D8	116	0.0003	806	781	0.0048	0.0007
D9	119	0.0002	1234	1327	0.0062	0.0010
D10	96	0.0003	501	528	0.0046	0.0008
D11	120	0.0003	895	814	0.0061	0.0007
D12	155	0.0003	1789	1625	0.0107	0.0007
D13	166	0.0003	1983	1879	0.0085	0.0009
D14	1126	0.0006	775071	772109	0.2547	0.0012
D15	153	0.0011	32183	31087	0.1670	0.0095

In cases of nearly overlap clusters (D11, D12, D13), this method creates worse coresets.

- Lightweight Coreset performs well in most cases, it is also very fast; in fact, it is just slower than Uniform Sampling and faster than other methods. However, lightweight coreset rarely creates a sample with low error. In well-separated clusters, this method is not as good as Adaptive Sampling but it is clearly better in some other cases.
- It is obviously that the improved ProTraS is much more slower than the other. ProTraS is built based on the farther-first-traversal algorithm in which the points are selected sequently while the uniform sampling, adaptive sampling and lightweight coreset all are based on sampling method which is extremely fast. However, this method creates coresets with very low errors.
- The FFT-based algorithm creates the best coresets in most cases, but also the slowest algorithms. This algorithm takes a very long run-time, nearly the same of the improved ProTraS.

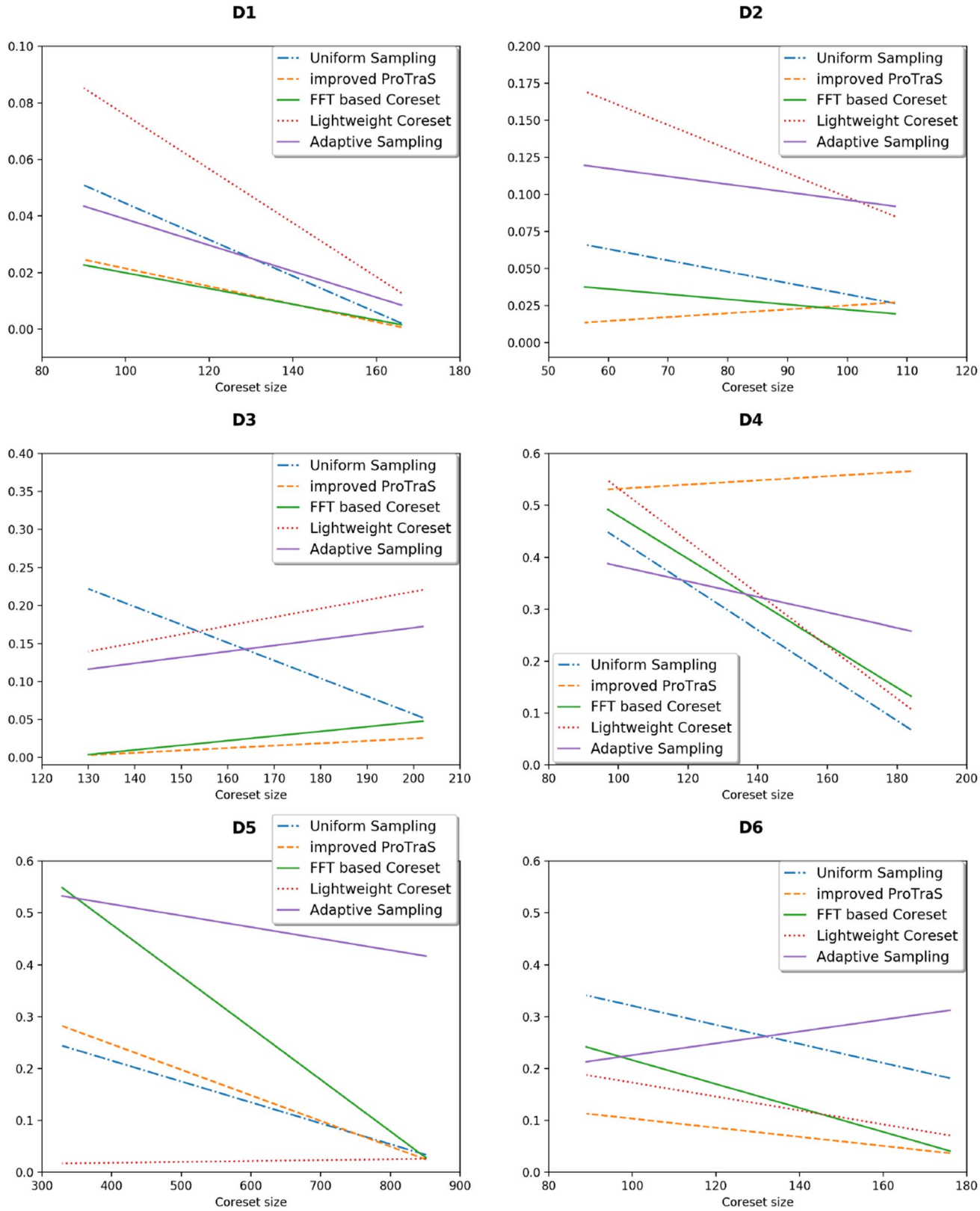

Fig. 2 Relative error in relation to subsample size

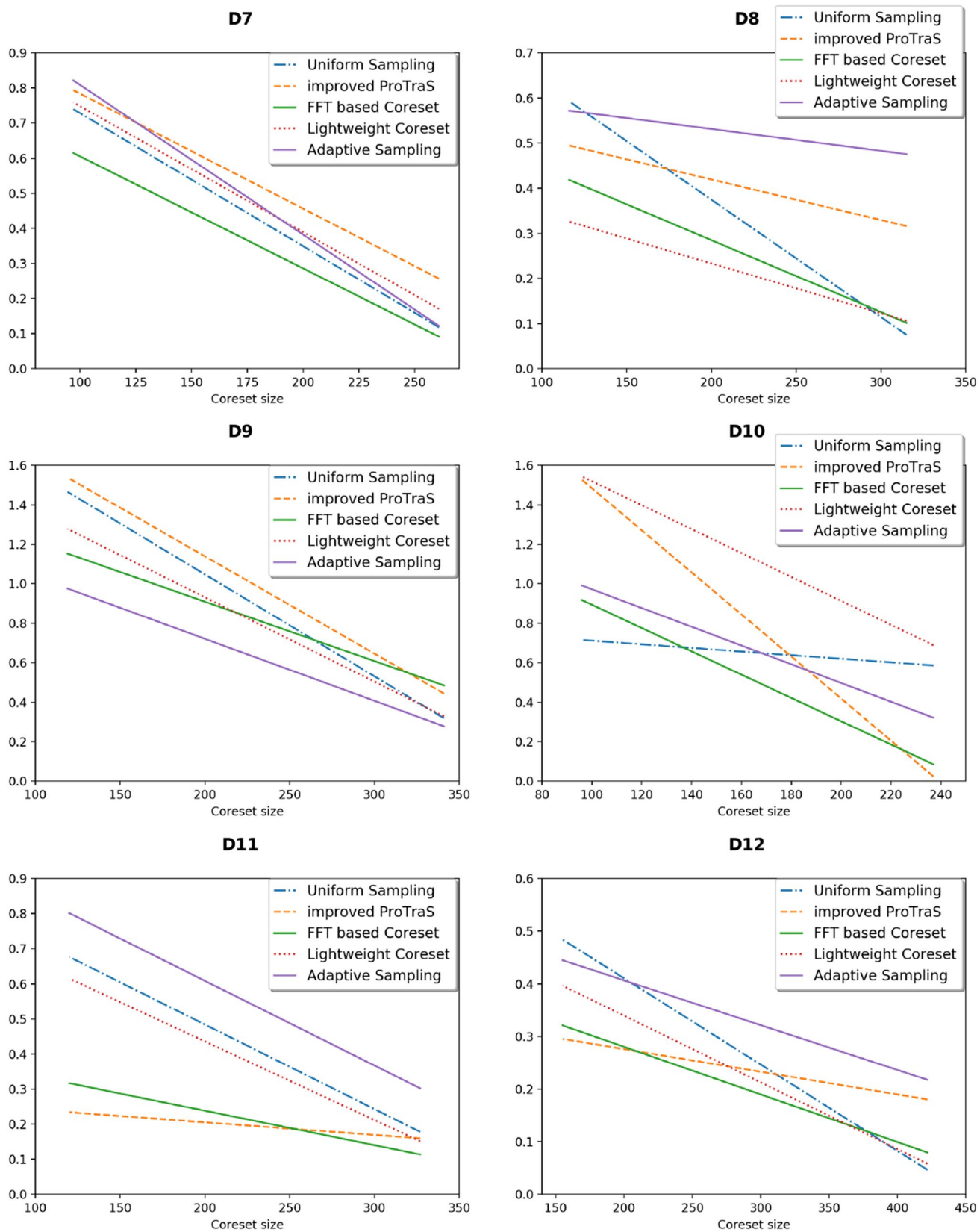

Fig. 3 Relative error in relation to subsample size

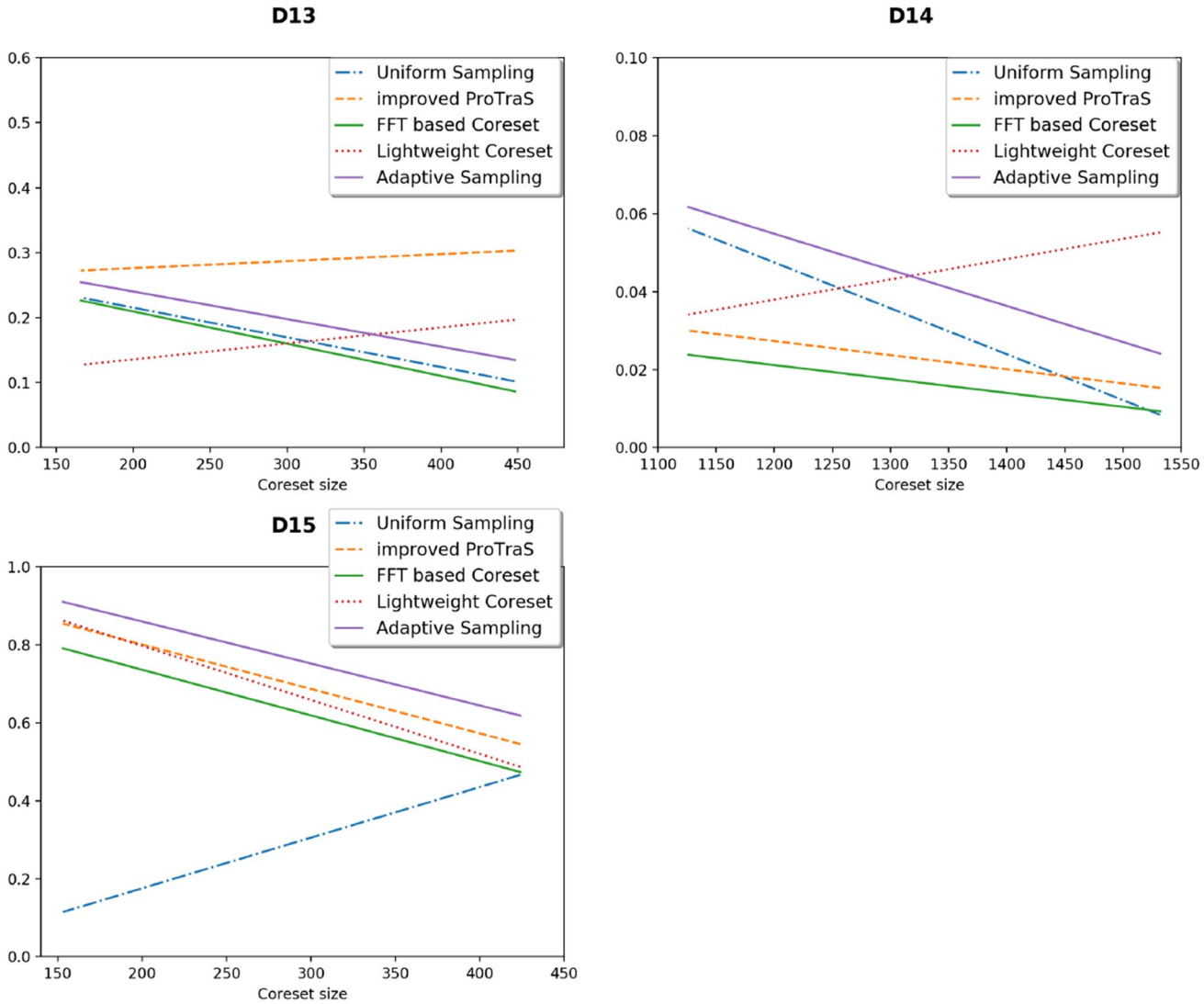

Fig. 4 Relative error in relation to subsample size

The three sampling-based methods (uniform, adaptive and lightweight coreset) create coresets with very high errors in some cases and also create coresets with very low errors sometimes. However, the average errors of sampling-based methods seem to be good enough to use in reality.

In most cases, the improved ProTraS and the FFT-based algorithm seem to have similar low relative errors and slow run-time. Unlike sampling-class methods which can yield results very fast, the FFT based algorithms (improved ProTraS and native FFT-based with preprocessing) creates coresets by checking point by point in full data set and need to calculate many distances during runtime. These ones take a lot of time but the result is extremely useful since the coresets from these methods have the lowest errors in most cases.

Conclusions

In this paper, we introduce and compare four state-of-the-art coreset constructions, the ProTraS algorithm [29] with post-processing improvement [31], FFT-based coreset with pre-processing improvement [32], Adaptive Sampling [14], and Lightweight Coreset [7] and we use relative errors to compare these methods along with uniform sampling as the baseline.

Even though FFT-based class methods and its improvement defeat other methods in the experiments, the speed or runtime is a big concern when comparing to other sampling-based ones. This method needs a lot of computation to create coresets, that is why it is very slow.

On the other hand, the sampling-based class constructions complete all experiments at a glance; however, the correctness of the created sample is still a big problem. Since this is sampling-based method, we need to check that the result is good enough or not.

Finally, each method mentioned in this paper has its own advantages and disadvantages. The options 'Slow but more accuracy' or 'Fast but less correct' will be weighed before applying any of these algorithms in practice.

Acknowledgements This research is funded by a project with the Department of Science and Technology, Ho Chi Minh City, Vietnam (contract with HCMUT No. 42/2019/HD-QPTKHCN, dated 11/7/2019). We would like to thank the FDSE 2019 organizing committee and paper reviewers for suggesting corrections and improvements. The audience at our presentation session in FDSE 2019 conference also offered constructive feedbacks for the paper.

Compliance with Ethical Standards

Conflict of Interest The authors report no conflicts of interest.

References

1. Agarwal PK, Procopiuc CM, Varadarajan KR. Approximating extent measures of points. J ACM. 2004;51(4):606–35.
2. Agarwal PK, Procopiuc CM, Varadarajan KR. Geometric approximation via coresets. Comb Comput Geom. 2005;52:1–30.
3. Arora S. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. J Assoc Comput Mach. 1998;45(5):753–82.
4. Arthur D, Vassilvitskii S. k-Means++: the advantages of careful seeding. In: Proceedings of the eighteenth annual ACM-SIAM symposium on discrete algorithms, 2007, pp. 1027–1035.
5. Bachem O, Lucic M, Krause A. Coresets for nonparametric estimation—the case of DP-means. In: International Conference on Machine Learning (ICML), 2015.
6. Bachem O, Lucic M, Lattanzi S. One-shot coresets: the case of k-clustering. In: International conference on articial intelligence and statistics (AISTATS), 2018.
7. Bachem O, Lucic M, Krause A. Scalable and distributed clustering via lightweight coresets. In: International conference on knowledge discovery and data mining (KDD), 2018.
8. Bachem O, Lucic M, Krause A. Practical coreset constructions for machine learning. arXiv preprint, 2017.
9. Charikar M, O'Callaghan L, Panigrahy. Better streaming algorithms for clustering problems. In: Proceedings of the 35th annual ACM symposium on theory of computing, 2003, pp. 30–39.
10. Dang TK, Tran KTK. The meeting of acquaintances: a cost-efficient authentication scheme for light-weight objects with transient trust level and plurality approach. Secur Commun Netw. 2019;2019:1–18.
11. Frahling G, Sohler C. Coresets in dynamic geometric data streams. In: Proceedings of the thirty-seventh annual ACM symposium on theory of computing, pp. 209–217, STOC 2005. https://doi.org/10.1145/1060590.1060622
12. Feldman D, Monemizadeh M, Sohler C. A PTAS for k- means clustering based on weak coresets. In: Symposium on Computational Geometry (SoCG), ACM, 2007, pp 11–18.
13. Feldman D, Monemizadeh M, Sohler C, Woodruff DP, Coresets and sketches for high dimensional subspace approximation problems. In: Symposium on discrete algorithms (SODA), Society for Industrial and Applied Mathematics, pp. 630–649, 2010.
14. Feldman D, Faulkner M, Krause A, Scalable training of mixture models via coresets. In: Advances in neural information processing systems (NIPS), 2011, pp. 2142–2150
15. Feldman D, Schmidt M, Sohler C, Turning big data into tiny data: constant-size coresets for k-means, PCA and projective clustering. In: Symposium on discrete algorithms (SODA), Society for industrial and applied mathematics, pp. 1434–1453, 2013.
16. Gonzalez TF. Clustering to minimize the maximum inter-cluster distance. Theor Comput Sci. 1985;38:293–306.
17. Har-Peled S. Geometric approximation algorithms, vol. 173. Washington: American mathematical society Providence; 2011.
18. Har-Peled S, Kushal A. Smaller coresets for k-median and k-means clustering. In: Symposium on computational geometry (SoCG), ACM, pp. 126–134, 2005.
19. Har-Peled S, Mazumdar S, On coresets for k-means and k-median clustering. In: Symposium on theory of computing (STOC), ACM, pp. 291–300, 2004.
20. Hoang NL, Dang TK, Trang LH, A Comparative Study of the Use of Coresets for Clustering Large Datasets. In: LNCS 11814 Future Data and Security Engineering, pp. 45-55, 2019. https://doi.org/10.1007/978-3-030-35653-8
21. Lloyd SP. Least squares quantization in PCM. IEEE Trans Inf Theory. 1982;28:129–37.
22. Lucic M, Bachem O, Krause A, Strong coresets for hard and soft Bregman clustering with applications to exponential family mixtures. In: International conference on artificial intelligence and statistics (AISTATS), pp. 1–9, 2016.
23. Lucic M, Faulkner M, Krause A. Training mixture models at scale via coresets. J Mach Learn. 2017;18:1–25.
24. Inaba M, Katoh N, Imai H, Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering. In: Proceeding of 10th annual symposium on computational geometry, pp. 332–339, 1994.
25. Matousek J. On approximate geometric k-clustering. Discrete Comput Geometry. 2000;24:61–84.
26. Phan TN, Dang TK. A Lightweight Indexing Approach for Efficient Batch Similarity Processing with MapReduce. SN Comput Sci. 2020;1(1). https://doi.org/10.1007/s42979-019-0007-y.
27. Ros F, Guillaume S. DENDIS: a new density-based sampling for clustering algorithm. Expert Syst Appl. 2016;56:349–59.
28. Ros F, Guillaume S. DIDES: a fast and effective sampling for clustering algorithm. Knowl Inf Syst. 2017;50:543–68.
29. Ros F, Guillaume S. ProTraS: a probabilistic traversing sampling algorithm. Expert Syst Appl. 2018;105:65–76.
30. Rosenkrantz DJ, Stearns RE, Lewis PM II. An analysis of several Heuristics for the traveling salesman problem. SIAM J Comput. 1977;6:563–81.
31. Trang LH, Ngoan PV, Duc NV. A sample-based algorithm for visual assessment of cluster tendency (VAT) with large datasets. Future Data Secur Eng LNCS. 2018;11251:145–57.
32. Trang LH, Hoang NL, Dang TK. A farthest first traversal based sampling algorithm for k-clustering. In: 2020 14th international

conference on ubiquitous information management and communication (IMCOM), Taichung, Taiwan, 2020, pp. 1-6. https://doi.org/10.1109/IMCOM48794.2020.9001738
33. Vega WFdl, Karpinski M, Kenyon C, Rabani Y. Approximation schemes for clustering problems. In: Proceedings of the 35th annual ACM symposium on theory of computing, pp. 50–58, 2003.
34. https://cs.joensuu.fi/sipu/datasets. Accessed Jan 2020.
35. https://github.com/deric/clustering-benchmark. Accessed Jan 2020.