JOURNAL OF DECISION MAKING AND HEALTHCARE
Volume 1 (2024), No. 1, 45–56
https://doi.org/10.69829/jdmh-024-0101-ta04
Tulipa Opera Scholarum

SCALABLE CLUSTERING OF LINES

AKSHAY SHARMA[1], DEBDAS GHOSH[1,*], AND WITOLD PEDRYCZ[2]

[1] *Department of Mathematical Sciences, Indian Institute of Technology (BHU), Varanasi, Uttar Pradesh 221005, India*
[2] *Department of Electrical and Computer Engineering, University of Alberta, Alberta T6G 1H9, Canada*

Abstract. We introduce a scalable algorithm for clustering lines and affine subspaces of fixed dimensions via data reduction using a coreset approximating scheme. A key characteristic of the proposed algorithm is exploiting a scatter-then-aggregate scheme for data partitioning, which allows to maintain a constant coreset approximation size. This key feature enables a novel algorithm which streams as well as distributes workload. We study how the proposed algorithm scales in a practical setting and compare it with other clustering methods. Results of experiments executed on the PARAM supercomputer confirm that the time complexity of algorithm is log-linear in problem size and decreases linearly as the number of processors increase. Source code for the algorithm, the experiments, and other potential applications is provided.

Keywords. Distributed Optimization, k-means, Clustering, Coresets.

1. Introduction

1.1. **Background.** Clustering is a semi-century-old problem [7] for partitioning observations into clusters. One of the industry-wide used algorithms for clustering is k-means, where each of the n observations belongs to one of the k clusters with the nearest mean. k-medians clustering is a variation of the k-means algorithm, where instead of calculating the mean for each cluster to determine its centroid, one calculates the median. This results in reducing total error over all the clusters. It is widely used in statistics and data mining. A natural extension to this problem, called k-median for lines, is replacing points with lines (or higher-dimensional structures such as vector spaces or sets). A key characteristic of clustering algorithms for higher dimensional spaces is their time and space complexity. Hence, existing clustering techniques that provide exact solutions are rendered moot when the size of the problem is large.

With a trade-off between clustering accuracy and tractability, data reduction or summarization can advocate a beneficial approach. By reducing a large dataset in a problem-dependent sense, we can use existing algorithms so that they take up significantly less time and space in memory.

1.2. **Literature review.** There have been some developments in finding k-means and k-medians for lines and other higher dimensional spaces in the past decade. Gao et al. [13] found a collection of Euclidean balls that cover a set of lines and coin the usage of such class of clustering to incomplete data. Perets [27] proved that the k-median problem for lines is NP-complete for a non-constant k. In the ensuing years, heuristics were developed such as approximating the solution on the original data. More prominently, Feldman et al. [11] provided a data reduction algorithm allowing one to use exact algorithms on a reduced dataset. Recently, Marom et al. [20] discussed a streaming solution for k-means clustering of lines. Statman et al. [30] shared a projective clustering algorithm for fixed dimensional

*Corresponding author.
E-mail addresses: akshay.sharma.mat16@iitbhu.ac.in (A. Sharma), debdas.mat@iitbhu.ac.in (D. Ghosh), and wpedrycz@ualberta.ca (W. Pedrycz)
Accepted: June 09, 2024.

affine subspaces using r-Lipschitz function as a distance measure. Jubran et al. [16] solved the problem of k-means clustering of sets using coresets.

Line clustering algorithms have several important applications, especially in the field of computer vision. Ommer et al. [23] provided a multi-scale object detection method using line clustering. Bazin et al. [6] reported clustering parallel lines to find vanishing points, which is an important vision task. Many researchers have developed Hausdorff distance based k-means variants. Zhou et al. [36] proposed a clustering algorithm for spacial data mining using extended Hausdorff distance. Poggenhans et al. [28] used line clustering to improve road feature detection. Recently, Li et al. [17] used clustering of 3D lines to classify fracture lines.

1.3. **Motivation and work done.** Although there have been numerous developments in clustering or classifying multi-dimensional data (such as vectors, geometric lines, etc.), hardly any of them work on scale, i.e., they fail to be directly deployed on multiple machines, possibly in a distributed setting. The primary causes of this inability to scale are the volume of communication overheads—both the size and frequency of data shared—and the inability to separate computationally intensive workloads into smaller tasks. With the explosion of data, there has been a recent trend of developing k-means clustering of such multi-dimensional data. For instance, Marom et al. [20] provided a streaming algorithm for clustering big data lines. There are additional benefits of providing a polynomial time solution to the k-lines clustering problem. Since most big data algorithms necessitate a space complexity linear in problem size, the proposed solution can be used to cluster vast amounts of line data. Also, with the advent of cloud computing, it is now possible to cluster and classify data on devices with limited computational resources in real time.

Another benefit of research in this direction is the ability to work with incomplete data. Each data point with incomplete or missing information can be modeled as a line or fixed dimensional affine subspace. Such incomplete data points are encountered on a daily basis in various machine learning, computer vision, and digital communication tasks and are mostly discarded, which may introduce bias and affect the veracity of the results. As the size of data increases, even well-known statistical imputation measures fail to provide a real-time approximation. In fact, replacing missing data with substituted values fails miserably when the degree of incompleteness improves [35]. However, assuming that such data points belong to a common subspace, data reduction techniques coupled with subspace clustering turn out to be an effective solution [29].

Since such clustering is essential to so many fields, there are algorithms that solve k-mean and k-median problems for lines problem for a fixed dimension [27, 17] or for fixed centers (say $k = 1, 2$) [13, 27]. Since the problem stems from k-means for points, the EM algorithm, RANSAC, and Lloyd's algorithm [4, 24] were used primarily. Recently there has been a shift to several heuristics—block diagonalization, convolutional neural networks, deep learning—to solve variants of the problem such as clustering subspaces [15, 19, 26], matrix completion [34, 22] and classification of incomplete data [18, 31].

2. Preliminaries

The following list provides symbols being used throughout the article:

- L is the set of lines: $\{l : l \subset \mathbb{R}^d\}$.
- n is the number of lines: $|L| = n$.
- d denotes the line dimensionality, i.e., $l \in \mathbb{R}^d$.
- m is the reduced set (or coreset) size.
- $k \in \mathbb{Z}_+$ denotes the desired number of centers/means.
- P is the set of k-mean points.
- ϵ is the maximum permissible error.

- $w : \mathbb{R}^d \to \mathbb{R}_+$ is the map of lines to a weight.
- p represents the number of parallel processors.
- T_s is the time taken for the serial algorithm.
- T_p is the time taken for the parallel algorithm.
- S denotes the speed-up metric for the parallel algorithm.
- E is the efficiency metric for the parallel algorithm.

Unless otherwise mentioned, all the sets are assumed to be a subset of $\mathbb{R}^d$—the d-dimensional Euclidean space. As with any other clustering algorithm, we begin by defining the distance metric used in our algorithm.

Definition 2.1. (*Manhattan distance*). The Manhattan distance or l_1 norm between two vectors p and q in a d-dimensional vector space with a fixed Cartesian coordinate system is the sum of the lengths of the projections of the line segment between the points onto the coordinate axes. Formally,

$$l_1(p, q) = \|p - q\|_1 = \sum_{i=1}^{d} |p_i - q_i|.$$

Definition 2.2. (*Cost*). The cost/distance between a point $x \in \mathbb{R}^d$ and set of points $P \subset \mathbb{R}^d$ is defined as

$$D_2(x, P) = \inf_{p \in P} \|x - p\|_2,$$

where $\|\cdot\|_2$ denotes the l_2 norm. This notion of cost can be extend to a line $l \subset \mathbb{R}^d$ by

$$D_2(l, P) = \inf_{x \in l} D_2(x, P).$$

Definition 2.3. (*Manhattan cost*). The Manhattan cost between a point $x \in \mathbb{R}^d$ and set of points $P \subset \mathbb{R}^d$ is defined by

$$D_1(x, P) = \inf_{p \in P} \|x - p\|_1,$$

where $\|\cdot\|_1$ denotes the l_1 norm. This cost, too can be extend to a line $l \subset \mathbb{R}^d$ by

$$D_1(l, P) = \inf_{x \in l} D_1(x, P).$$

It is worth pointing out that the Manhattan cost is useful in computing the k-median.

Definition 2.4. (*Weighted set*). A weighted set of lines is a pair $L' = (L, w)$ where L is a set of lines in $\mathbb{R}^d$ and $w : L \to (0, \infty)$ is a function that maps every $l \in L$ to $w(l) \geq 0$, called the weight of l.

Definition 2.5. (*Coreset* [3]). Let $P \subset \mathbb{R}^n$ be a set of points. A set of points $Q \subset P$ is called coreset of the original set P if Q approximates the shape of P in the sense that a smaller set size makes a complex problem tractable using a computationally inefficient, polynomial-time algorithm. The solution for Q is then translated to an approximate solution to the original point set P.

Coresets are widely used in computational geometry. Many natural geometric optimization problems have coresets that approximate an optimal solution to within a factor of $1+\epsilon$, that can be found quickly (in linear time or near-linear time), and that have size bounded by a function of $1/\epsilon$ independent of the input size, where ϵ is an arbitrary positive number or usually an error bound. The independence between data cardinality and coreset size allows PCA [10], k-means clustering, or training an SVM [33] on big data. Barger et al. [5] propose methods to find coresets for big sparse data. Feldman provides a comprehensive survey [9] of designing coresets such that there is a provable tradeoff between their size and approximation error.

Remark 2.6. Note that coresets are usually weighted sets.

2.1. **Problem formulation.** The exact form of k-means and k-medians problems are given below.

Definition 2.7. (*k-means for lines*). A set of k centers P^* is called k-means for the set L if P^* minimizes the sum of squared distances over every line in L and its nearest center. Formally, P^* is a solution to

$$\begin{array}{ll} \text{minimize} & \Sigma_{l\in L} D_2(l, P) \\ \text{subject to} & P \in \mathbb{R}^d, w : \mathbb{R}^d \to \mathbb{R}_+. \end{array}$$

Definition 2.8. (*k-medians for lines*). A set of k centers P is called k-medians for the set L if it minimizes the sum of Manhattan distance over every line in L and its nearest center. Formally, P is a solution to

$$\begin{array}{ll} \text{minimize} & \Sigma_{l\in L} D_1(l, P) \\ \text{subject to} & P \in \mathbb{R}^d, w : \mathbb{R}^d \to \mathbb{R}_+. \end{array}$$

The above two problems are readily solvable on commer2009multicial solvers, assuming the set of lines L is small enough to be stored in memory. If L is exceedingly large (possibly an infinite stream of lines), the idea is to use data reduction, essentially constructing a coreset and then applying the approximate k-means approach.

Definition 2.9. (*Approximate k-means for lines*). Let L' be a weighted coreset of L. A set of k centers $\widetilde{P}$ is called approximate k-means for the set L if $\widetilde{P}$ minimizes the sum of squared distances over every line in L' and its nearest center. Formally, $\widetilde{P}$ is the solution to

$$\begin{array}{ll} \text{minimize} & \Sigma_{l\in L'} w(l) D_2(l, P) \\ \text{subject to} & P \in \mathbb{R}^d, w : \mathbb{R}^d \to \mathbb{R}_+. \end{array}$$

We note that there is a cost-to-running time trade-off in the exact algorithms. With a slight compromise in cost, we can find an approximation of the set P^* given in Definition 2.7. Moreover, Marom et al. [20] proved that a weighted subset of L, which is $O(\log n)$ in size, can effectively approximate the set P^* up to a relative cost inaccuracy of ϵ.

3. The Algorithm

This section illustrates the distributed algorithm, which has a quasilinear[1] time complexity in terms of n and which scales well in a high-performance computing environment. If the set of lines L is very large in size, input data partitioning turns out to be a suitable decomposition technique. Depending upon the number of processors, set L is split into smaller subsets L_i. Coresets C_i are generated from lines L_i using the streaming algorithm mentioned below. Finally, all the coresets are aggregated into a single coreset.

The psuedo-code for finding approximate k-means for lines is given in Algorithm 1. In the pseudo-code, CORESET(L_i, k, m) denotes the streaming algorithm to generate a corset, provided by Marom et al. [20], that gives an approximate solution to the k-means problem. The algorithm can be integrated into architecture supporting a message-passing interface [12]. Also, SEND and RECEIVE are schedules used for inter-process communication by the distributed computing architecture. The task dependency graph of the algorithm is demonstrated in Figure 22.

Given that the running time of CORESET(L_i, k, m) algorithm does not depend upon the distribution of lines chosen, a static partitioning of data is suitable for the problem—constant batch sizes and batch-processor mapping is computed a priori. Although communications, while merging the results, have an $m \log n$ time complexity, merging coresets have an $O(m^2)$ time complexity. Hence, a dynamic mapping scheme can cause unnecessary communication overheads pronounced as m increases.

[1] An algorithm is said to run in quasilinear or log-linear time if the time complexity of the algorithm, denoted by $T(n)$, is $T(n) = O(n \log^k n)$ for some positive constant k.

Algorithm 1: KLINES(L, k, m)

Input:
Set L of lines in $\mathbb{R}^d$ of size n
$k \geq 1$, number of centers
$m \geq 1$, coreset size
$p \geq 1$, number of processors

Output: A weighted coreset P^*, solution to the problem in Definition 2.9

Split the set L into p subsets L_i such that $\bigcup_{i=1}^{p} L_i = L$
Distribute the sets L_i among p processors
Do parallel: Rank i
  $C_i \leftarrow$ CORESET(L_i, k, m)
  for $d \leftarrow 1$ **to** $\log_2 p$ **by** 1 **do**
    k $\leftarrow 1$
    while $k \leq p$ **do**
      $R \leftarrow k$ // receiver processor
      $S \leftarrow k + 2^d$ // sender processor
      if *i is R* **then**
        $C' \leftarrow$ RECEIVE(S)
        $C_i \leftarrow C_i \bigcup C'$ // doubles the set size
        $C_i \leftarrow$ CORESET(C_i, k, m) // halves the set size
      end
      if *i is S* **then**
        SEND(C_i, R)
      end
      $k \leftarrow k + 2^d$
    end
  end
end
return C_1 // Rank 1 process has final result

4. Theoretical Results

In this section, we share the analytical expressions of distributed algorithm time complexity, communication overhead, and scalability metrics. Firstly, we note that the serial solution provided by Marom et al. [20] is

$$T_s = O(nm^3d^2k \log n \log k \log m) + n \log nm^3dk^{O(k)}. \tag{4.1}$$

The derivation of the expression of T_s is as follows. The CORESET(L,m,k) algorithm discussed in Marom et al. [20] provides a weighted coreset (compressed set) by using three inner algorithms, viz.,

(i) CENTROID-SET (L): It provides the closest points between a set of lines that runs in $O(n^2d^3)$ time.
(ii) BI-CRITERIA-APPROXIMATION (L, m): It provides a bi-criterion (α, β) approximation of the k-mean of L in $O(nd^2k \log(k) + m^2 \log n)$ time.
(iii) LINES-SENSITIVITIES (L, b, k): It creates a sensitivity (and weight) function for the set of lines which runs in $O(nd^2k \log(k) \log n) + ndk^{O(k)}$ time.

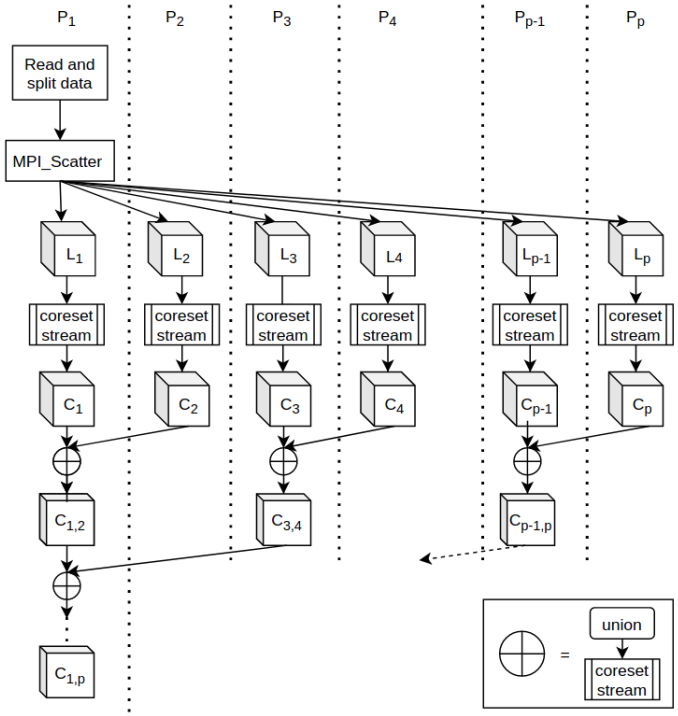

FIGURE 1. The figure shows the task dependency graph of Algorithm 1. Initially, the set of lines L are sampled and distributed randomly to avoid bias in clustering. Since the subset sizes are known a priori, we use a group communication, not a P2P communication, to distribute them using SCATTER command of MPI. Note that the $\bigoplus$ reduction operator preserves the size of the coresets, which is in $O(m^3)$.

Assuming $d << n$, the cost of CENTROID-SET(L) is almost constant time since it is executed on subsets of L, which are progressively smaller in size. Finally, from Algorithm 4, CORESET(L,m,k), given in [20] its clear that the total time complexity is $O(nd^2k\log(k)\log n) + ndk^{O(k)}$.

Moreover, each iteration of CORESET(L_i, k, m) requires space complexity $O(m^3)$ during streaming. Since coresets are merged and then halved, the total memory required for coresets is $O(m^3)$ during aggregation.

Next, before providing any theoretical guarantee of the distributed algorithm, we need to find an analytical expression of running time, T_p, of the distributed solution in terms of problem size n and number of processors, p.

Theorem 4.1. *The running time T_p of the distributed algorithm has an asymptotic time complexity of $O(\frac{n}{p}\log\frac{n}{p})$.*

Proof. From the equation (4.1), we know the time complexity of the serial algorithm CORESET(L_i, k, m). We can get the expression of the distributed Algorithm 1 as

$$T_p = O\left(\tfrac{n}{p}m^3d^2k\log\tfrac{n}{p}\log k\log m\right)\ \tfrac{n}{p}\log\tfrac{n}{p}m^3dk^{O(k)} + t_m,$$

where t_m denotes the time to merge the coresets. Note that each $\bigoplus$ operation (Steps 12 and 13 in Algorithm 1) takes time

$$t_{\bigoplus} = m + O\left(m^2\tfrac{n}{p}\log\tfrac{n}{p}\right),$$

which is the sum of its inner atomic operations, i.e., union and compression, respectively. Since m, coreset size, is kept fixed while merging and the aggregation operator forms a tree with a max depth

around $\log p$, the total time of merging is

$$t_m = \log p \left(m + O\left(m^2 \tfrac{n}{p} \log \tfrac{n}{p} \right) \right).$$

Hence,

$$T_p = O\left(\tfrac{n}{p} m^3 d^2 k \log \tfrac{n}{p} \log k \log m \right) + \tfrac{n}{p} \log \tfrac{n}{p} m^3 d k^{O(k)} + O\left(m^2 \tfrac{n}{p} \log p \log \tfrac{n}{p} \right).$$

□

Since we are equipped with expressions of T_s and T_p, we first prove that the proposed algorithm is cost-optimal.

Definition 4.2. (*Cost optimality.*) A parallel system is called cost-optimal if it solves a problem with a cost proportional to the execution time of the fastest known sequential algorithm on a single processor.

Theorem 4.3. *The solution discussed in Algorithm 1 is cost-optimal.*

Proof. We note that the best serial solution (so far) for the k-means for lines problem is given by Marom et al. [20], which has a time complexity of

$$T_s = O(n \log n).$$

Also, the total time taken by the parallel solution is

$$T_{tot} = pT_p = p\, O\left(\tfrac{n}{p} \log n \right) = O(n \log n).$$

Since both the time complexities are asymptotically equivalent, the parallel solution discussed in Algorithm 1 is cost-optimal. □

Another important factor that decides the effectiveness of a distributed algorithm is the total overhead.

Definition 4.4. (*Total overhead T_o*). The total overhead of a parallel system is the total time collectively spent by all the processing elements over and above that required by the fastest known sequential algorithm for solving the same problem on a single processing element.

Now we provide the expression of T_o for Algorithm 1.

Theorem 4.5. *The total overhead T_o of the distributed algorithm has a time complexity of $O(n \log \frac{n}{p})$.*

Proof. From Definition 4.1, we deduce that the time collectively spent by all the processing elements is

$$pT_p = O\left(nm^3 d^2 k \log \tfrac{n}{p} \log k \log m \right) + O\left(n \log \tfrac{n}{p} m^3 d k^{O(k)} \right) + O\left(m^2 n \log p \log \tfrac{n}{p} \right).$$

Hence,

$$T_o = pT_p - T_s = \; O(nm^3 d^2 k \log p \log k \log m) + O(n \log p m^3 d k^{O(k)}) + O(m^2 n \log p \log \tfrac{n}{p}).$$

□

Lastly, we share results regarding the most common performance metrics for evaluating parallel algorithms—speed-up and efficiency.

Definition 4.6. (*Speed-up S*). Speed-up is defined as the ratio of the time taken to solve a problem on a single processing element to the time required to solve the same problem on a parallel computer with p identical processing elements.

Definition 4.7. (*Efficiency E*). Efficiency is defined as the ratio of speedup to the number of processing elements p and is a measure of the fraction of time for which a processing element is usefully employed.

TABLE 1. Running time for the serial algorithm T_s and the parallel algorithm T_p for different number of processors. Note that there is almost a linear reduction on T_p as p increases.

n	T_s	$p=2$ T_p	$p=4$ T_p	$p=8$ T_p	$p=16$ T_p	$p=32$ T_p
512	2.398	1.99	1.287	1.24	1.267	1.497
2048	9.023	6.177	3.892	2.617	2.27	2.36
8192	36.007	21.965	11.76	6.917	4.812	3.735
32768	136.087	90.331	43.147	23.53	13.865	9.235
131072	605.8	336.9	170.3	90.1	50.57	31.43
262144	–	–	–	–	–	59.53

The analytical expressions for S and E of Algorithm 1 are found as follows. The speed-up S of Algorithm 1 is calculated by

$$\begin{aligned} S &= \frac{T_s}{T_p} \\ &= \frac{O(nm^3d^2k\log n\log k\log m)+n\log nm^3dk^{O(k)}}{\left\{O(\frac{n}{p}m^3d^2k\log\frac{n}{p}\log k\log m)+O(\frac{n}{p}\log\frac{n}{p}m^3dk^{O(k)})+O(m^2\frac{n}{p}\log p\log\frac{n}{p})\right\}} \\ &= \frac{O(n\log nk^{O(k)})}{\left\{O(\frac{n}{p}\log\frac{n}{p})+O(\frac{n}{p}\log\frac{n}{p}k^{O(k)})+O(\frac{n}{p}\log p\log\frac{n}{p})\right\}} \end{aligned}$$

and if k is small enough, above expression can be approximated as

$$S \approx \frac{O(n\log n)}{\left\{O(\frac{n}{p}\log\frac{n}{p})+O(n\log n)+O(\frac{n}{p}\log p\log\frac{n}{p})\right\}}.$$

The efficiency E of Algorithm 1 is calculated by

$$E = \frac{S}{p} = \frac{O(n\log n)}{\left\{O(n\log\frac{n}{p})+O(np\log n)+O(n\log p\log\frac{n}{p})\right\}}.$$

In the next section, the results confirm that $S \to p$ and $E \to 1$ for Algorithm 1 as p increases.

5. Experimental Studies and Applications

In this section, we share experiments that verify the theoretical results mentioned in the previous section. Also, we include two potential applications of k-means for line problems in machine learning and matrix approximation. The accompanying source code is open and available at [2]. For ease of use, we have created a Python package *klines*. The results are readily reproducible. The experiments were carried out on the PARAM Shivay Supercomputer at the Indian Institute of Technology (BHU), Varanasi, running a Linux – CentOS 7.6 – operating system with 40 parallel processing cores and a total of 384 GB of DDR4 memory.

5.1. **Scalability metrics.** This section illustrates computing k-means for OpenStreetMap (OSM) China road dataset [1]. The results illustrate a log-linear decrease in time as the number of processors increases and a linear increase in parallel computation time as the problem size increases. As an effect, the speed-up is less pronounced as p increases—attributable to communication overhead. Table 1 lists the average (mean of 5 readings) running time of the serial and parallel algorithm for different problem sizes and processor count, where coreset size is $m = 5$, number of centers required is $k = 3$, among other parameter settings ($d = 2$ and $\tau = 0.001$). Table 2 lists the performance metrics, S and E over the same configuration.

TABLE 2. Comparing two fundamental metrics, speed-up S and efficiency E, for different number of processors. Note that $S \to p$ and $E \to 1$ as n increases.

n	T_s	$p=2$		$p=4$		$p=8$		$p=16$		$p=32$	
		S	E	S	E	S	E	S	E	S	E
512	2.398	1.21	0.60	1.863	0.47	1.934	0.24	1.892	0.12	1.6	0.02
2048	9.023	1.46	0.73	2.318	0.58	3.447	0.43	3.975	0.25	3.823	0.12
8192	36.01	1.511	0.77	3.061	0.76	5.204	0.65	7.481	0.47	9.638	0.30
32768	136.09	1.639	0.82	3.152	0.79	5.779	0.72	9.809	0.61	14.726	0.46
131072	605.8	1.798	0.90	3.557	0.89	6.72	0.84	12.0	0.75	19.27	0.60

TABLE 3. Comparison of matrix completion algorithms with k-lines approach for different matrix sizes. Metrics used here are rank of the completion matrix and relative norm difference. Matrix dimensions are $n \times d$, where $d = 10$ fixed throughout.

Method	$n = 100$	$n = 1000$	$n = 10000$
	RMSE	RMSE	RMSE
kNN	0.094	0.110	0.109
Nuclear Norm [8]	0.081	–	–
SoftImpute [21]	0.821	0.091	0.098
k-lines (ours)	0.133	0.192	0.214

5.2. **Matrix completion.** Matrix completion is the task of filling in the missing entries of a partially observed matrix. It is an optimization problem where the objective is to fit a given matrix (the data) and an approximating matrix (the optimization variable):

$$\begin{aligned} &\text{minimize} \quad \operatorname{rank}(X) \\ &\text{subject to} \quad X_{ij} = M_{ij} \; \forall i, j \in E, \end{aligned}$$

where X is a low-rank matrix, M is the matrix we wish to recover, and E represents the set of exact values or observations. There are often constraints on the approximating matrix such as non-negativity constraint, minimizing $\|Y - X\|_F$, or enforcing Hankel structure, where $\|\cdot\|_F$ is the Frobenius norm[2]. Matrix completion has a lot of applications in machine learning, recommendation systems, natural language processing, and digital signal processing.

Assuming that missing entries in an $n \times d$ partially observed matrix Y take real and continuous values, completion of Y, with a fixed number of missing values (say j) per row vector is effectively a k-means problem for j-dimensional affine spaces. Geometrically, each of the n incomplete row vectors represent either a line (one missing entry), plane (two missing entries) or some higher dimensional structure in $\mathbb{R}^d$.

There are variety of matrix completion (or imputation) algorithms such as kNN (nearest neighbor search), matrix factorization methods, SoftImpute [21], SVDImpute [32], Nuclear norm minimization [8] and the recently developed BiScaler method [14]. Table 3 compares k-lines approach with other algorithms for matrices of different dimensions. Note that nuclear norm minimization [8] results are omitted since it is very slow—impractical to compute—on a single RAM, explaining the obvious trade-off in accuracy vs time. Note that if $\|Y\|_2$ is the norm of the complete matrix, then the relative mean squared error, RMSE, is defined as

$$\text{RMSE} = \frac{\|Y - X\|_2}{\|Y\|_2}.$$

[2]Frobenius norm for an $m \times n$ matrix $A = (A_{ij})_{m \times n}$ is defined as $\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |A_{ij}|^2}$.

TABLE 4. Comparing clustering metrics, Homogeneity score H, Completeness score C, and V-measure (harmonic mean of H and C) for clustering of the two datasets. Metrics range from $[0, 1]$, and a value close to 1 desirable.

Metric	Iris		Synthetic	
	Scikit	Ours	Scikit	Ours
H	0.588	0.421	0.815	0.338
C	0.579	0.453	0.815	0.335
V-measure	0.583	0.434	0.815	0.337

5.3. **Clustering with missing values.** Clustering is the task of dividing data points into groups so that points in the same groups are more similar to each other than those in other groups. Over the years, many algorithms have been developed to perform clustering tasks on incomplete and noisy datasets. Minibatch variants of the k-means algorithm (often with a k-means++ [4] initialization step) have been developed.

The k-means for lines can also categorize large noisy or incomplete data by computing k-centers of the points in the data if there is at most a single missing entry in every data point. Table 4 compares results of k-means clustering between k-lines approach and the popular Scikit-learn [25] machine learning repository in Python. Apart from the well-known Iris dataset (metrics computed average of 20 iterations), a large synthetic dataset ($N = 10,000$) was created using Scikit-learn (metrics computed average of 5 iterations).

6. Conclusion and Future Work

We have provided a scalable algorithm for clustering of lines via coreset data reduction. The results of the experiments confirm that the time complexity of the algorithm is log-linear in problem size and decreases linearly as the number of processors increases. Despite the linear scalability, the algorithm does not take into account the distribution of lines while making the input partitions. It would be interesting to explore applications of coresets in fields that require processing big datasets (machine learning) or necessitate time/space-intensive processes (deep learning). Coresets might also be suitable for signal processing since the field lacks linear-time algorithms for applications such as computations of FFT or wavelets.

An important future development can be k-means for fixed dimensional affine subspaces, which can help in capturing multiple missing values in its applications. Also, the knowledge of the distribution of lines can be leveraged to partition lines unequally (say based on density) and produce proportionally sized coreset. Further, it would be interesting to explore task-based or hybrid partitioning, thus assigning specific tasks to special processors (GPUs, CPUs, TPUs, etc).

Statements and Declarations

The manuscript has no associated data. The accompanying source code for the results presented in this article is openly available at [2].

Acknowledgments

The support and the resources provided by PARAM Shivay Facility under the National Supercomputing Mission, Government of India at the Indian Institute of Technology (BHU), Varanasi, is gratefully acknowledged.

References

[1] Openstreetmap planet dump. https://planet.openstreetmap.org, 2015.

[2] Source code for the experiments. https://github.com/AKS1996/KLines, 2021.

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

[4] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[5] A. Barger and D. Feldman. Deterministic coresets for k-means of big sparse data. *Algorithms*, 13(4):92, 2020.

[6] J.-C. Bazin, Y. Seo, C. Demonceaux, P. Vasseur, K. Ikeuchi, I. Kweon, and M. Pollefeys. Globally optimal line clustering and vanishing point estimation in manhattan world. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 638–645. IEEE, 2012.

[7] P. S. Bradley, O. L. Mangasarian, and W. N. Street. Clustering via concave minimization. *Advances in Neural Information Processing systems*, 9:368–374, 1996.

[8] E. Candès and B. Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.

[9] D. Feldman. Introduction to core-sets: an updated survey. *preprint arXiv:2011.09384 [cs.LG]*, 2020.

[10] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for k-means, PCA, and projective clustering. *SIAM Journal on Computing*, 49(3):601–657, 2020.

[11] D. Feldman and L. J. Schulman. Data reduction for weighted and outlier-resistant clustering. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1343–1354. SIAM, 2012.

[12] M. P. I. Forum. A message passing interface standard. *International Journal of Supercomputer Applications*, 8:3–4, 1994.

[13] J. Gao, M. Langberg, and L. J. Schulman. Clustering lines in high-dimensional space: Classification of incomplete data. *ACM Transactions on Algorithms*, 7(1):1–26, 2010.

[14] T. Hastie, R. Mazumder, J. D. Lee, and R. Zadeh. Matrix completion and low-rank svd via fast alternating least squares. *Journal of Machine Learning Research*, 16(1):3367–3402, 2015.

[15] P. Ji, T. Zhang, H. Li, M. Salzmann, and I. Reid. Deep subspace clustering networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[16] I. Jubran, M. Tukan, A. Maalouf, and D. Feldman. Sets clustering. In *International Conference on Machine Learning*, pages 4994–5005. PMLR, 2020.

[17] J. Li, S. Tang, H. Zhang, Z. Li, W. Deng, C. Zhao, L. Fan, G. Wang, J. Liu, P. Yin, G. Xu, L. Zhang, and P. Tang. Clustering of morphological fracture lines for identifying intertrochanteric fracture classification with hausdorff distance–based k-means approach. *Injury*, 50(4):939–949, 2019.

[18] W.-Y. Loh, J. Eltinge, M. J. Cho, and Y. Li. Classification and regression trees and forests for incomplete data from sample surveys. *Statistica Sinica*, 29(1):431–453, 2019.

[19] C. Lu, J. Feng, Z. Lin, T. Mei, and S. Yan. Subspace clustering by block diagonal representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(2):487–501, 2018.

[20] Y. Marom and D. Feldman. k-means clustering of lines for big data. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[21] R. Mazumder, T. Hastie, and R. Tibshirani. Spectral regularization algorithms for learning large incomplete matrices. *The Journal of Machine Learning Research*, 11:2287–2322, 2010.

[22] F. Monti, M. Bronstein, and X. Bresson. Geometric matrix completion with recurrent multi-graph neural networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[23] B. Ommer and J. Malik. Multi-scale object detection by clustering lines. In *2009 IEEE 12th International Conference on Computer Vision*, pages 484–491. IEEE, 2009.

[24] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy. The effectiveness of lloyd-type methods for the k-means problem. *Journal of the ACM*, 59(6):1–22, 2013.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, and D. Cournapeau. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] X. Peng, J. Feng, S. Xiao, W.-Y. Yau, J. T. Zhou, and S. Yang. Structured autoencoders for subspace clustering. *IEEE Transactions on Image Processing*, 27(10):5076–5086, 2018.

[27] T. Perets. *Clustering of Lines*. Open University of Israel Ra'anana, Israel, 2011.

[28] F. Poggenhans, A.-M. Hellmund, and C. Stiller. Application of line clustering algorithms for improving road feature detection. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2456–2461. IEEE, 2016.

[29] M. Soltanolkotabi, E. Elhamifar, and E. J. Candés. Robust subspace clustering. *Annals of Statistics*, 42(2):669–699, 2014.

[30] A. Statman, L. Rozenberg, and D. Feldman. Faster projective clustering approximation of big data. *preprint arXiv:2011.13476 [cs.DS]*, 2020.

[31] C. T. Tran, M. Zhang, P. Andreae, and B. Xue. Improving performance for classification with incomplete data using wrapper-based feature selection. *Evolutionary Intelligence*, 9(3):81–94, 2016.

[32] O. Troyanskaya, M. Cantor, G. Sherlock, P. Brown, T. Hastie, R. Tibshirani, D. Botstein, and R. B. Altman. Missing value estimation methods for dna microarrays. *Bioinformatics*, 17(6):520–525, 2001.

[33] M. Tukan, C. Baykal, D. Feldman, and D. Rus. On coresets for support vector machines. In *International Conference on Theory and Applications of Models of Computation*, pages 287–299. Springer, 2020.

[34] R. van den Berg, T. N. Kipf, and M. Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[35] D. W. van der Palm, L. A. van der Ark, and J. K. Vermunt. A comparison of incomplete-data methods for categorical data. *Statistical Methods in Medical Research*, 25(2):754–774, 2016.

[36] G. Zhou, B. Lin, and X. Ma. A spatial clustering algorithm for line objects based on extended Hausdorff distance. In *2013 21st International Conference on Geoinformatics*, pages 1–4. IEEE, 2013.