

Supplementary Materials for “SparseKmeans: Efficient K-means Clustering For Sparse Data”

Khoi Nguyen Pham Dang*

He-Zhe Lin*

khoinguyen.phamdang@mbzuai.ac.ae

b07902028@csie.ntu.edu.tw

Mohamed bin Zayed University of Artificial Intelligence
Abu Dhabi, UAE

Chih-Jen Lin

National Taiwan University

Taipei, Taiwan

Mohamed bin Zayed University of Artificial Intelligence
Abu Dhabi, UAE
cjlin@csie.ntu.edu.tw

A STOPPING CONDITION OF K-MEANS ALGORITHM

SparseKmeans follows the implementation in scikit-learn to stop the iteration when the new centroids are close enough to their corresponding old ones. Specifically, the procedure is terminated when

$$\sum_{j=1}^K \|c_j - \hat{c}_j\|^2 \leq \tau \times \frac{1}{n} \sum_{k=1}^n \sigma_k^2, \quad (\text{A.1})$$

where c_j 's and \hat{c}_j 's are current and previous centroids, τ is a pre-defined parameter, and σ_k^2 is the variance of the k -th component in each x_i , respectively.

In SparseKmeans, we store the current centroids and previous centroids as two matrices C and \hat{C} :

$$C = \begin{bmatrix} - & c_1^T & - \\ - & c_2^T & - \\ & \vdots & \\ - & c_K^T & - \end{bmatrix} \text{ and } \hat{C} = \begin{bmatrix} - & \hat{c}_1^T & - \\ - & \hat{c}_2^T & - \\ & \vdots & \\ - & \hat{c}_K^T & - \end{bmatrix}.$$

To calculate $\|c_j - \hat{c}_j\|^2$ in (A.1), we first take the subtraction between C and \hat{C} :

$$C - \hat{C} = \begin{bmatrix} c_1^T - \hat{c}_1^T \\ c_2^T - \hat{c}_2^T \\ \vdots \\ c_K^T - \hat{c}_K^T \end{bmatrix},$$

and then calculate the squared norms of $C - \hat{C}$

$$\begin{bmatrix} \|c_1^T - \hat{c}_1^T\|^2 \\ \|c_2^T - \hat{c}_2^T\|^2 \\ \vdots \\ \|c_K^T - \hat{c}_K^T\|^2 \end{bmatrix}. \quad (\text{A.2})$$

Summing all elements in (A.2), we have the left-hand side of (A.1).

B K-MEANS++ INITIALIZATION

The initialization of c_j 's are often by choosing a subset of K samples $C \subseteq \{x_1, \dots, x_m\}$. By default, scikit-learn uses K-means++ initialization [1] to determine C . Compared to randomly choosing K points as the initial centroids, K-means++ effectively reduces the number of iterations for convergence and minimizes the risk of poor clustering results.

*Both authors contributed equally to this research.

B.1 The Procedure of K-means++

K-means++ sequentially selects the initial points in a more optimal way by prioritizing points that are far from the previously chosen centroids. The detailed procedure is given as follows.

- (1) Uniformly choose c_1 from $\{x_1, \dots, x_m\}$ and let $C = \{c_1\}$.
- (2) Initialize a squared distance vector $\mathbf{d} \in \mathbb{R}^m$ with $d'_i = \|x_i - c_1\|^2$. Through the whole procedure, d'_i records the squared distance between x_i and the nearest point in C , i.e.,

$$d'_i = \min_{c \in C} \|x_i - c\|^2. \quad (\text{B.1})$$

We define the “potential” with respect to C to be the summation of all d'_i .

- (3) For $j = 2, \dots, K$, do the following to select c_j .
 - (a) Sample $L = 2 + \log_2 K$ candidate points from $\{x_1, \dots, x_m\}$ using the distribution

$$\Pr(x_i) = \frac{d'_i}{\sum_{i'=1}^m d'_{i'}}. \quad (\text{B.2})$$

Let $S = \{s_1, \dots, s_L\}$ denote the sampled candidates. From (B.2), we see that a point that is further from C is more likely to be selected. Besides, according to (B.1), we have $d'_i = 0$ for any $x_i \in C$, so it is guaranteed that the candidate set S does not include any point in C .¹

- (b) To find the “best” candidate to be c_j , for each $s \in S$, we first calculate the potential with respect to $C \cup \{s\}$, then choose the one that yields the least potential, i.e.,

$$c_j \in \arg \min_{s \in S} \sum_{i=1}^m \min_{c \in C \cup \{s\}} \|x_i - c\|^2 \quad (\text{B.3})$$

$$= \arg \min_{s \in S} \sum_{i=1}^m \min\{d'_i, \|x_i - s\|^2\}. \quad (\text{B.4})$$

- (c) Update $C \leftarrow C \cup \{c_j\}$ and

$$d'_i \leftarrow \min\{d'_i, \|x_i - c_j\|^2\}, i = 1, \dots, m. \quad (\text{B.5})$$

B.2 The Concept of K-means++

The core idea of the loop in Step-3 of the K-means++ procedure is that, given that x_i whose $d'_i > 0$ is a candidate and is chosen as a new centroid, the updated potential would be reduced. To clarify, from (B.5), the selected x_i has the updated $d'_i = 0$, while the squared distances between other points and the nearest centroids do not increase. By our sampling scheme, as long as $\sum_{i'=1}^m d'_{i'} > 0$, some

¹However, there is an exception when the number of distinct points in $\{x_1, \dots, x_m\}$ is less than K . See Section B.2 for details.

\mathbf{x}_i with $d'_i > 0$ are selected as the L candidates. In the most extreme situation, only one \mathbf{x}_i with $d'_i > 0$ exists, and all L candidates are the same \mathbf{x}_i .

However, it turns out that there is a scenario in which the sampling procedure is no longer valid. In the case where $\sum_{i'=1}^m d'_{i'} = 0$, meaning that all the squared distances between points and their nearest centroids are 0, we are unable to select any of them as a candidate. In other words, every \mathbf{x}_i is the same as some $\mathbf{c}_j \in \mathcal{C}$, which indicates that the number of distinct points is less than K . This situation occurs if $m < K$, or $m \geq K$ but there are some duplicated points. As the procedure can no longer find K centroids, the number of predefined clusters should be reduced to match the number of distinct data points.

B.3 Implementation Details

The most time-consuming part in the above procedure is Step-3b, which requires $\|\mathbf{x}_i - \mathbf{s}\|^2$ for all $\mathbf{s} \in S$ and \mathbf{x}_i 's. Similar to (4), the pairwise squared distance matrix in $\mathbb{R}^{m \times L}$ is

$$\begin{bmatrix} \|\mathbf{x}_1\|^2 - 2\mathbf{x}_1^T \mathbf{s}_1 + \|\mathbf{s}_1\|^2 & \dots & \|\mathbf{x}_1\|^2 - 2\mathbf{x}_1^T \mathbf{s}_L + \|\mathbf{s}_L\|^2 \\ \|\mathbf{x}_2\|^2 - 2\mathbf{x}_2^T \mathbf{s}_1 + \|\mathbf{s}_1\|^2 & \dots & \|\mathbf{x}_2\|^2 - 2\mathbf{x}_2^T \mathbf{s}_L + \|\mathbf{s}_L\|^2 \\ \vdots & \ddots & \vdots \\ \|\mathbf{x}_m\|^2 - 2\mathbf{x}_m^T \mathbf{s}_1 + \|\mathbf{s}_1\|^2 & \dots & \|\mathbf{x}_m\|^2 - 2\mathbf{x}_m^T \mathbf{s}_L + \|\mathbf{s}_L\|^2 \end{bmatrix}, \quad (\text{B.6})$$

which needs the the following matrix product similar to (C.3)

$$XS^T = X \begin{bmatrix} | & | & \dots & | \\ \mathbf{s}_1 & \mathbf{s}_2 & \dots & \mathbf{s}_L \\ | & | & \dots & | \end{bmatrix}. \quad (\text{B.7})$$

However, unlike the centroid matrix C in (C.3) where the density greatly varies, S is always very sparse because each \mathbf{s} is just a data point. Thus, we always apply a sparse-sparse product for calculating (B.7).

C IMPLEMENTATION DETAILS OF LLOYD'S CLUSTER ASSIGNMENT

In Section 2.1.1, we mentioned that we need to compute the following pairwise distance matrix in $\mathbb{R}^{m \times K}$

$$\begin{bmatrix} \|\mathbf{x}_1\|^2 - 2\mathbf{x}_1^T \mathbf{c}_1 + \|\mathbf{c}_1\|^2 & \dots & \|\mathbf{x}_1\|^2 - 2\mathbf{x}_1^T \mathbf{c}_K + \|\mathbf{c}_K\|^2 \\ \|\mathbf{x}_2\|^2 - 2\mathbf{x}_2^T \mathbf{c}_1 + \|\mathbf{c}_1\|^2 & \dots & \|\mathbf{x}_2\|^2 - 2\mathbf{x}_2^T \mathbf{c}_K + \|\mathbf{c}_K\|^2 \\ \vdots & \ddots & \vdots \\ \|\mathbf{x}_m\|^2 - 2\mathbf{x}_m^T \mathbf{c}_1 + \|\mathbf{c}_1\|^2 & \dots & \|\mathbf{x}_m\|^2 - 2\mathbf{x}_m^T \mathbf{c}_K + \|\mathbf{c}_K\|^2 \end{bmatrix}.$$

In Lloyd's cluster assignment, for each i , all we need is to find

$$\ell_i \in \arg \min_{j=1, \dots, K} \|\mathbf{x}_i - \mathbf{c}_j\|^2,$$

i.e., finding the cluster j that yields the least $\|\mathbf{x}_i - \mathbf{c}_j\|$ among the i -th row of the above matrix. Therefore, we may omit calculating the shared $\|\mathbf{x}_i\|^2$ terms within each row; instead, we can only compute the following matrix

$$\begin{bmatrix} -2\mathbf{x}_1^T \mathbf{c}_1 + \|\mathbf{c}_1\|^2 & \dots & -2\mathbf{x}_1^T \mathbf{c}_K + \|\mathbf{c}_K\|^2 \\ -2\mathbf{x}_2^T \mathbf{c}_1 + \|\mathbf{c}_1\|^2 & \dots & -2\mathbf{x}_2^T \mathbf{c}_K + \|\mathbf{c}_K\|^2 \\ \vdots & \ddots & \vdots \\ -2\mathbf{x}_m^T \mathbf{c}_1 + \|\mathbf{c}_1\|^2 & \dots & -2\mathbf{x}_m^T \mathbf{c}_K + \|\mathbf{c}_K\|^2 \end{bmatrix}. \quad (\text{C.1})$$

This only requires the norm of \mathbf{c}_j 's

$$\begin{bmatrix} \|\mathbf{c}_1\|^2 \\ \|\mathbf{c}_2\|^2 \\ \vdots \\ \|\mathbf{c}_K\|^2 \end{bmatrix}, \quad (\text{C.2})$$

and the matrix multiplication

$$XC^T = \begin{bmatrix} - & \mathbf{x}_1^T & - \\ - & \mathbf{x}_2^T & - \\ \vdots & \vdots & \vdots \\ - & \mathbf{x}_m^T & - \end{bmatrix} \begin{bmatrix} | & | & \dots & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \dots & \mathbf{c}_K \\ | & | & \dots & | \end{bmatrix}. \quad (\text{C.3})$$

D DETAILS OF ELKAN'S CLUSTER ASSIGNMENT

D.1 The Maintenance of $u(\mathbf{x}_i)$ and $l(\mathbf{x}_i, \mathbf{c}_j)$

- For each \mathbf{x}_i in cluster- ℓ_i , $u(\mathbf{x}_i)$ is an upper bound of $\|\mathbf{x}_i - \mathbf{c}_{\ell_i}\|$. Since the centroids are updated after the cluster assignment, we need to update $u(\mathbf{x}_i)$ correspondingly. Suppose $\hat{\mathbf{c}}_{\ell_i}$ the centroid for cluster- ℓ_i before the centroid updates and $\hat{u}(\mathbf{x}_i)$ is an upper bound of $\|\mathbf{x}_i - \hat{\mathbf{c}}_{\ell_i}\|$. To update $u(\mathbf{x}_i)$ from $\hat{u}(\mathbf{x}_i)$, by the triangle inequality, we have

$$\|\mathbf{x}_i - \hat{\mathbf{c}}_{\ell_i}\| + \|\hat{\mathbf{c}}_{\ell_i} - \mathbf{c}_{\ell_i}\| \geq \|\mathbf{x}_i - \mathbf{c}_{\ell_i}\|.$$

Therefore, after centroid updates, $u(\mathbf{x}_i)$ can be obtained via

$$u(\mathbf{x}_i) \leftarrow \hat{u}(\mathbf{x}_i) + \|\mathbf{c}_{\ell_i} - \hat{\mathbf{c}}_{\ell_i}\|. \quad (\text{D.1})$$

- For each \mathbf{x}_i and any $j = 1, \dots, K$, $l(\mathbf{x}_i, \mathbf{c}_j)$ is a lower bound of $\|\mathbf{x}_i - \mathbf{c}_j\|$. Similar to the situation in $u(\mathbf{x}_i)$, since \mathbf{c}_j is changed after the centroid updates, we need to update $l(\mathbf{x}_i, \mathbf{c}_j)$ for all i . Suppose $\hat{\mathbf{c}}_j$ is the centroid of cluster- j before the centroid updates and $\hat{l}(\mathbf{x}_i, \hat{\mathbf{c}}_j)$ is a lower bound of $\|\mathbf{x}_i - \hat{\mathbf{c}}_j\|$. By the triangle inequality and the definition of $\hat{l}(\mathbf{x}_i, \hat{\mathbf{c}}_j)$, we have

$$\|\mathbf{x}_i - \mathbf{c}_j\| \geq \|\mathbf{x}_i - \hat{\mathbf{c}}_j\| - \|\hat{\mathbf{c}}_j - \mathbf{c}_j\| \geq \hat{l}(\mathbf{x}_i, \hat{\mathbf{c}}_j) - \|\hat{\mathbf{c}}_j - \mathbf{c}_j\|.$$

Therefore, after centroid updates, $l(\mathbf{x}_i, \mathbf{c}_j)$ can be updated by

$$l(\mathbf{x}_i, \mathbf{c}_j) \leftarrow \max \left\{ \hat{l}(\mathbf{x}_i, \hat{\mathbf{c}}_j) - \|\mathbf{c}_j - \hat{\mathbf{c}}_j\|, 0 \right\}. \quad (\text{D.2})$$

D.2 Elkan's K-means Assignment in the Original Work [2]

In the main paper, we give the sketch of Elkan's K-means assignment in the original work of [2]. Here we provide a detailed version in Algorithm D.1. In particular, there are two additional settings:

- (1) (Line 3 to Line 4) To avoid the for-loop over j in Algorithm 1, Elkan [2] considers the following condition

$$u(\mathbf{x}) \leq \frac{1}{2} \min_{j: j \neq \ell_i} \|\mathbf{c}_j - \mathbf{c}_{\ell_i}\|. \quad (\text{D.3})$$

If (D.3) holds, then condition (6) must hold for all $j \neq \ell_i$, which means \mathbf{x} must remain in its original cluster.

- (2) To further reduce the number of distance calculations, we update $u(\mathbf{x}_i)$ to be the tightest upper bound $\|\mathbf{x}_i - \mathbf{c}_{\ell_i}\|$ upon seeing the first \mathbf{c}_j with $\|\mathbf{x}_i - \mathbf{c}_j\|$ may need to be calculated. Let us

Algorithm D.1: The original version of Elkan’s cluster assignment in [2].

```

1 Compute  $D \in \mathbb{R}^{K \times K}$  with  $D_{j,j'} = \|\mathbf{c}_j - \mathbf{c}_{j'}\|/2$ 
2 for  $i = 1, \dots, m$  do
3   if  $u(\mathbf{x}_i) \leq \min_{j \neq \ell_i} D_{j,\ell_i}/2$  then
4     continue
5    $bound\_tight \leftarrow \text{False}$ 
6   for  $j = 1, \dots, K$  do
7     if  $j \neq \ell_i$  and  $u(\mathbf{x}_i) > D_{j,\ell_i}/2$  and  $u(\mathbf{x}_i) > l(\mathbf{x}_i, \mathbf{c}_j)$ 
8       then
9         // filter clusters that  $\mathbf{x}_i$  must not belong to
10        if  $bound\_tight$  is  $\text{False}$  then
11           $d_i \leftarrow \|\mathbf{x}_i - \mathbf{c}_{\ell_i}\|$ 
12           $u(\mathbf{x}_i) \leftarrow d_i, l(\mathbf{x}_i, \mathbf{c}_{\ell_i}) \leftarrow d_i$ 
13           $bound\_tight \leftarrow \text{True}$ 
14          Go to Line 7
15         $l(\mathbf{x}_i, \mathbf{c}_j) \leftarrow \|\mathbf{x}_i - \mathbf{c}_j\|$ 
16        if  $u(\mathbf{x}_i) > \|\mathbf{x}_i - \mathbf{c}_j\|$  then
17           $\ell_i \leftarrow j, u(\mathbf{x}_i) \leftarrow \|\mathbf{x}_i - \mathbf{c}_j\|$ 

```

Algorithm E.1: Algorithm for Handling Empty Clusters

```

1  $filled\_cluster \leftarrow 0$ 
2  $k \leftarrow 1$ 
3 while  $filled\_cluster < E$  do
4    $\mathbf{x} \leftarrow$  the point  $\mathbf{x}_i$  with the  $k$ -th largest  $d_i$ 
5   if the size of the cluster that  $\mathbf{x}_i$  belongs to  $> 1$  then
6     Reassign  $\mathbf{x}_i$  to an empty cluster
7      $filled\_cluster \leftarrow filled\_cluster + 1$ 
8    $k \leftarrow k + 1$ 

```

assume j^* is the first cluster satisfying neither (6) nor (7). Then the for-loop over j can be illustrated as follows:

$$\underbrace{1, \dots, (j^* - 1)}_{bound_tight=False}, \quad \underbrace{j^*}_{\text{Line 8 invoked}}, \quad \underbrace{(j^* + 1), \dots, K}_{bound_tight=True}. \quad (\text{D.4})$$

We use a flag $bound_tight$ to record if $u(\mathbf{x}_i) = \|\mathbf{x}_i - \mathbf{c}_{\ell_i}\|$ or not. For $j = 1, \dots, j^* - 1$, this flag remains its default value False because either condition (6) or (7) holds and $\|\mathbf{x}_i - \mathbf{c}_{\ell_i}\|$ is not needed. Therefore, we do not update $u(\mathbf{x}_i)$ to be the tight bound $\|\mathbf{x}_i - \mathbf{c}_{\ell_i}\|$. When $j = j^*$, we need both $\|\mathbf{x}_i - \mathbf{c}_{\ell_i}\|$ and $\|\mathbf{x}_i - \mathbf{c}_j\|$ to determine whether \mathbf{x}_i should stay in cluster- ℓ_i or go to cluster- j^* . Under this situation, we update $u(\mathbf{x}_i)$ to $\|\mathbf{x}_i - \mathbf{c}_{\ell_i}\|$ and go to Line 7 to recheck the conditions. Then, from $j = j^* + 1, \dots, K$, we maintain $u(\mathbf{x}_i) = \|\mathbf{x}_i - \mathbf{c}_{\ell_i}\|$ in Line 15 while ℓ_i may be changed. Since we use the smallest possible value of $u(\mathbf{x}_i)$, the number of distance calculations can be reduced.

E ADDITIONAL IMPLEMENTATION DETAILS FOR UPDATING CENTROIDS

After assigning each sample to its nearest cluster ℓ_i , some clusters may end up empty, making it impossible to update \mathbf{c}_j . To address this, scikit-learn selects the points with the largest distances $d_i = \|\mathbf{x}_i - \mathbf{c}_{\ell_i}\|$ and reassigns them to the empty clusters. This explains the need for maintaining the vector \mathbf{d} , as noted in Section 3.1. Nevertheless, this strategy introduces potential issues: if all points in a clusters are reassigned to fill other empty clusters, the original cluster itself becomes empty. In the worst case (though we have not constructed such an example), this process could repeat indefinitely. We propose Algorithm E.1 to address this issue. Similar to scikit-learn, our approach begins by reassigning points with the largest d_i to the empty clusters. However, if a point is the only remaining member of its original cluster, we skip it and move to the next candidate. The key question is whether this procedure always provides enough points for reassignment.

Suppose there are E empty clusters and \bar{K} clusters that each contain only a single distinct point (possibly with duplicates). According to our K-means++ initialization strategy in Section B.2, the number of distinct points is at least K . Therefore, the remaining $(K - E - \bar{K})$ clusters together must contain at least $(E - \bar{K})$ distinct points. Among these $(E - \bar{K})$ distinct points, at most $(K - E - \bar{K})$ points coincide with their cluster centroids. Consequently, there are at least

$$(E - \bar{K}) - (K - E - \bar{K}) = E$$

points with $d_i > 0$ available for reassignment to those E empty clusters.

F DATA SETS

This section describes the preprocessing method applied to the four data sets across all experiments. For binary classification data set `Url`,² we directly use the data instances for clustering. Other data sets including `Amazon-670K`,³ `Wiki-500K`⁴ and `Amazon-3M`⁵ are multi-label data sets, in which each instance can be associated with several labels. In the context of this work, to avoid confusion with clustering results, which is called as labels in the main paper, we refer to the original labels of these multi-label data sets as meta-labels. Instead of clustering on data instances, we preprocessed the data to obtain a set of representations for meta-labels. These representations are then normalized and used as input for the K-means clustering algorithms. To clarify, let \mathbf{z}_i be the sparse representation of i^{th} instance, m be the total number of meta-labels and \mathbf{x}_j be the representation of j^{th} meta-label. To obtain a meta-label representation, we compute the sum over all data instances corresponding to that meta-label as follows:

$$\mathbf{x}_j = \frac{\tilde{\mathbf{x}}_j}{\|\tilde{\mathbf{x}}_j\|}. \quad (\text{F.1})$$

²https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary/url_combined_no_rmalized.bz2

³https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multilabel/Amazon-670K_tfidf_train_ver2.svm.bz2

⁴<https://drive.google.com/open?id=1bGecCagh8zaDV0ZNGsgF0QtwjAm0Afk>

⁵<https://drive.google.com/open?id=187vt5vAkG12mS2WOMZ2Qv48YKsjNbQv4>

in which

$$\tilde{x}_j = \sum \{z_i \mid z_i \text{ is associated with } j^{\text{th}} \text{ meta-label}\}. \quad (\text{F.2})$$

G A COMPREHENSIVE STUDY AND BENCHMARK ON THE MAJOR OPERATIONS OF K-MEANS ALGORITHM

In this section, we comprehensively study the optimization of the five operations mentioned in Section 3.2, including the optimized storage format for the matrices and the way to do the operation. We compare each setting in a real scenario by extracting matrices used in K-means clustering on the Wiki-500K data set. We run all experiments for 10 times on an Intel(R) Core(TM) i7-6900K CPU @ 3.20GHz machine using eight threads.

G.1 Operation (I): XC^T

In Section 2.1.1, we mentioned that the density of C may change across iterations. To find out a suitable setting under varying densities, we start with benchmarking XC^T using four matrices with different densities:

- $C_1 \in \mathbb{R}^{K \times n}$, where $K = 1500$, $n = 2, 381, 304$, and the density of C is $\text{nnz}(C)/(Kn) \approx 2.9\%$.
- $C_2 \in \mathbb{R}^{K \times n}$, where $K = 1000$, $n = 2, 381, 304$, and the density of C is $\text{nnz}(C)/(Kn) \approx 3.7\%$.
- $C_3 \in \mathbb{R}^{K \times n}$, where $K = 500$, $n = 2, 381, 304$, and the density of C is $\text{nnz}(C)/(Kn) \approx 5.7\%$.
- $C_4 \in \mathbb{R}^{K \times n}$, where $K = 100$, $n = 2, 381, 304$, and the density of C is $\text{nnz}(C)/(Kn) \approx 14.8\%$.

We compare the running time under different storage formats of C and the way to conduct the multiplication.

- We consider four possible storage formats for the matrix C based on a cross combination of dense/sparse and row/column-major settings. A sparse format stores only the non-zero entries in C , while a dense format stores all Kn entries regardless of value. In terms of layout, row-major (respectively, column-major) means the entries are stored in a row-wise (respectively, column-wise) order. This combination results in four formats: CSR (Compressed Sparse Row), CSC (Compressed Sparse Column), FullR (dense row-major), and FullC (dense column-major).
- As for computing the product, we compare three settings based on GraphBLAS (GB) and Intel Math Kernel Library (MKL). Since the result of XC^T is necessary for forming the typically dense matrix (C.1), it must be ultimately stored in a dense format. In GraphBLAS, this can be achieved in two ways, depending on when the conversion to dense is performed:
 - GB-NORMAL: Compute XC^T in GraphBLAS, store it in sparse format, and then convert it to dense.
 - GB-ACCUM: Use the GraphBLAS accumulation operator to obtain a dense result directly. In this case, we first initialize a dense zero matrix $M \in \mathbb{R}^{m \times K}$, and the operator updates it as

$$M \leftarrow M + XC^T,$$

yielding the desired dense output.

For MKL, we consider the following setting similar to GB-NORMAL:

- MKL-NORMAL: XC^T is computed using MKL and assigned to a sparse matrix.

Table 1 shows the running time under each configuration. Based on the results, we have the following observations.

- Computing the product in a “row-major \times row-major” setting is consistently the fastest. We speculate that this is because GraphBLAS internally generates results row by row. For example, to compute the first row of the product, it requires accessing the first row of X with all rows of C^T . Storing both matrices in row-major order therefore reduces memory access overhead.
- As C becomes denser, “CSR \times FullR” demonstrates more advantages than “CSR \times FullR”, regardless of using GB-NORMAL or GB-ACCUM. The reason might be that, as $\text{nnz}(C)$ grows, “sparse \times sparse” operations lead to more overheads compared to “sparse \times dense” operations.
- If we further compare GB-NORMAL and GB-ACCUM using “CSR \times FullR”, we see that GB-ACCUM is much faster. This is because the resulting XC^T is almost fully dense. In this case, by initializing a dense output matrix and using the “accumulation mode”, GraphBLAS assumes a dense output and applies a more efficient kernel for calculating the matrix product.

Based on the analysis, for C with higher density, using “CSR \times FullR” under GB-ACCUM is more preferred. Therefore, we heuristically set a density threshold 0.05 for C to dynamically select the way to conduct XC^T :

- If the density of C from the previous iteration is below 0.05, we use “CSR \times CSR” and GB-NORMAL.
- Otherwise, we use “CSR \times FullR” and GB-ACCUM.

G.2 Operation (III): $\mathbf{x}_i^T \mathbf{c}_{\ell_i}$

Our new design of Elkan’s method in Algorithm 2 requires calculating m inner products between each \mathbf{x}_i and its closest centroid \mathbf{c}_{ℓ_i} . To efficiently access \mathbf{c}_{ℓ_i} in the centroid matrix, C must be stored with a row-major format. Therefore, the remaining question is whether C should be in a sparse or dense format.

G.2.1 Theoretical Analysis. Let us start from analyzing the theoretical complexity of computing

$$\begin{aligned} \mathbf{x}_i^T \mathbf{c}_{\ell_i} &= \sum_{k=1}^n x_{i,k} \cdot c_{\ell_i,k} \\ &= \sum_{k: x_{i,k} \neq 0} x_{i,k} \cdot c_{\ell_i,k}. \end{aligned} \quad (\text{G.1})$$

- If C is stored as a dense matrix, for each $x_{i,k} \neq 0$, we can directly access $c_{\ell_i,k}$ in constant time. Therefore, calculating (G.1) requires $\text{nnz}(\mathbf{x}_i)$ times memory accesses to $c_{\ell_i,k}$ and $\text{nnz}(\mathbf{x}_i)$ times of scalar products.
- If C is stored in a sparse format, then we need at most $\text{nnz}(c_{\ell_i})$ times of element accesses to traverse all non-zero elements in c_{ℓ_i} , followed by $\text{nnz}(\mathbf{x}_i)$ times of scalar products.

According to (2), $\text{nnz}(c_{\ell_i}) \geq \text{nnz}(\mathbf{x}_i)$. As a result, using a dense format for C yields less memory accesses and better theoretical complexity.

G.2.2 GraphBLAS Runtime Benchmark. To acquire the m inner products at a time, we utilize the “mask operations” supported in GraphBLAS. For such operations, GraphBLAS selectively calculates certain entries of a matrix product by providing a boolean mask

Table 1: Running time (in seconds) of XC_1^T , XC_2^T , XC_3^T and XC_4^T . We terminate the timing if a setting takes more than one hour and mark “> 1 hr” in these cells. Cells marked with “-” indicate that a segmentation fault happens when using the MKL library.

| Method | X | C^T | Time for XC_1^T | Time for XC_2^T | Time for XC_3^T | Time for XC_4^T |
|------------|-----|-------|-------------------|-------------------|-------------------|-------------------|
| GB-NORMAL | CSR | CSR | 389.03 | 273.57 | 156.86 | 60.00 |
| | CSR | CSC | 390.51 | 281.30 | 160.31 | 245.42 |
| | CSR | FullR | 1192.65 | 778.98 | 275.66 | 43.71 |
| | CSR | FullC | 3022.12 | 1964.15 | 826.42 | 142.95 |
| | CSC | CSR | 388.73 | 273.47 | 157.44 | 60.47 |
| | CSC | CSC | 385.07 | 283.96 | 160.82 | 245.33 |
| | CSC | FullR | 1136.83 | 690.66 | 273.49 | 43.54 |
| | CSC | FullC | 2838.94 | 1714.99 | 827.31 | 143.19 |
| GB-ACCUM | CSR | CSR | 389.03 | 273.32 | 157.26 | 59.94 |
| | CSR | CSC | > 1hr | > 1hr | 3,092.87 | 1,814.20 |
| | CSR | FullR | 481.37 | 283.56 | 132.59 | 19.67 |
| | CSR | FullC | 2773.73 | 1709.22 | 839.07 | 144.91 |
| | CSC | CSR | 388.49 | 273.30 | 157.39 | 60.04 |
| | CSC | CSC | > 1hr | > 1hr | 3,093.00 | 1,816.75 |
| | CSC | FullR | 449.86 | 284.33 | 132.83 | 19.93 |
| | CSC | FullC | 2775.12 | 1706.82 | 839.10 | 144.67 |
| MKL-NORMAL | CSR | CSR | 441.30 | 319.71 | 198.31 | 87.48 |
| | CSR | CSC | 451.93 | 330.46 | 207.23 | 103.33 |
| | CSR | FullR | - | - | 114.78 | 17.70 |
| | CSR | FullC | 1473.70 | 1012.90 | 487.57 | 90.08 |
| | CSC | CSR | 439.40 | 319.58 | 197.11 | 87.19 |
| | CSC | CSC | 459.45 | 333.03 | 207.63 | 104.14 |
| | CSC | FullR | - | - | 115.36 | 19.73 |
| | CSC | FullC | 1542.62 | 1022.73 | 483.99 | 89.79 |

matrix. In SparseKmeans, we compute the m inner products by applying a mask $M \in \{0, 1\}^{m \times K}$ on the matrix product XC^T , where $M_{i,\ell_i} = 1$ for all i , and $M_{i,j} = 0$ for all $j \neq \ell_i$.

In Table 2, we compare the runtime performance of GraphBLAS with a handcrafted C-implementation for both dense and sparse C . We summarize the results as follows:

- We first compare the performance of the C-implementation against GraphBLAS by analyzing columns “(A) vs. (B)” and “(C) vs. (D)”. In all cases, the handcrafted C implementation consistently outperforms GraphBLAS, regardless of whether a dense or sparse format is used for matrix C . This is because our specialized C-implementation is designed to achieve the optimal theoretical complexity with minimal overheads, whereas GraphBLAS, as a general purpose library, incurs significant overheads.
- The results from the C-implementation (columns (A) and (C)) confirms the analysis in Section G.2.1, showing that using a dense format C is always faster.
- Comparing the GraphBLAS results (columns (B) and (D)), we observe that a sparse format for C , becomes advantageous when the matrix’s density is low ($< 5\%$). In these cases, using a sparse format significantly reduces the performance gap between GraphBLAS and our C-implementation, as the overhead of treating a sparse matrix as dense is substantial.

Table 2: Running time (in seconds) of calculating ten times of $x_i c_{\ell_i}^T$ for $i = 1, \dots, m$ with C-implementation and GraphBLAS. We set $n = 1,000,000$ and vary m and the density of X . The centroid matrix C is calculated based on (2).

| m | Density of X | Density of C | Dense C | | Sparse C | |
|-------|----------------|----------------|------------|--------|------------|--------|
| | | | C-code (A) | GB (B) | C-code (C) | GB (D) |
| 1,000 | 0.1% | 1.0% | 0.06 | 1.36 | 0.14 | 0.29 |
| 1,000 | 0.5% | 4.9% | 0.30 | 1.61 | 0.75 | 1.56 |
| 1,000 | 1% | 9.5% | 0.61 | 1.94 | 1.55 | 3.28 |
| 1,000 | 5% | 38.7% | 2.84 | 4.26 | 7.43 | 13.68 |
| 1,000 | 10% | 61.5% | 4.84 | 4.98 | 13.40 | 5.73 |
| 1,000 | 20% | 84.0% | 6.61 | 6.87 | 19.81 | 8.43 |
| 5,000 | 0.01% | 0.5% | 0.03 | 1.47 | 0.19 | 0.23 |
| 5,000 | 0.02% | 1.0% | 0.06 | 1.43 | 0.36 | 0.46 |
| 5,000 | 0.05% | 2.5% | 0.20 | 1.55 | 0.99 | 1.24 |
| 5,000 | 0.1% | 4.9% | 0.42 | 1.66 | 2.95 | 2.96 |
| 5,000 | 0.5% | 22.1% | 1.56 | 2.93 | 17.43 | 16.59 |
| 5,000 | 1% | 39.2% | 3.07 | 4.56 | 32.18 | 33.44 |
| 5,000 | 10% | 99.1% | 24.53 | 25.00 | 78.53 | 28.73 |

G.3 Operation (IV): $X_{S_j} \mathbf{c}_j$

Since this operation can be regarded as calculating $\mathbf{x}_i^T \mathbf{c}_j$ for $i \in S_j$, the theoretical analysis for the previous operation also suggests that using a dense \mathbf{c}_j yields to better theoretical complexity.

Table 3: Running time (in seconds) for benchmarking $X_{S_j}; c_j^T$ using both C-implementation and GraphBLAS. The experiment was conducted by randomly generating a mask for S_j ($j = 1, \dots, K$) with a specified density, repeated the calculation ten times, and recording the results. We set $n = 1,000,000$ while varying m and the density of X .

| (a) $m = 1,000$ | | | | | | | (b) $m = 1,000$ | | | | | | |
|-----------------|----------------|-----------------|-----------|-------|------------|-------|-----------------|----------------|-----------------|-----------|------|------------|-------|
| Density of X | Density of C | Density of mask | Dense C | | Sparse C | | Density of X | Density of C | Density of mask | Dense C | | Sparse C | |
| | | | C-code | GB | C-code | GB | | | | C-code | GB | C-code | GB |
| 0.1% | 1.0% | 0.1% | 0.003 | 1.57 | 0.009 | 0.09 | 0.01% | 0.5% | 0.1% | 0.003 | 2.53 | 0.015 | 1.56 |
| | | 1% | 0.05 | 2.35 | 0.10 | 0.31 | | | 1% | 0.03 | 2.30 | 0.13 | 0.25 |
| | | 10% | 0.47 | 2.67 | 0.96 | 1.90 | | | 10% | 0.25 | 2.30 | 1.32 | 0.12 |
| | | 20% | 1.27 | 4.46 | 2.49 | 4.61 | | | 20% | 0.66 | 3.50 | 3.47 | 3.13 |
| | | 50% | 2.36 | 5.56 | 6.53 | 11.42 | | | 50% | 1.37 | 5.04 | 8.67 | 8.84 |
| | | 100% | 3.13 | 6.96 | 12.61 | 22.74 | | | 100% | 2.19 | 5.81 | 18.42 | 17.00 |
| 1% | 9.5% | 0.01% | 0.002 | 0.14 | 0.01 | 0.71 | 0.1% | 4.8% | 0.01% | 0.003 | 0.47 | 0.015 | 0.99 |
| | | 0.1% | 0.04 | 0.96 | 0.12 | 1.44 | | | 0.1% | 0.03 | 1.83 | 0.14 | 1.04 |
| | | 1% | 0.41 | 1.58 | 1.27 | 2.26 | | | 1% | 0.27 | 1.83 | 1.44 | 1.45 |
| | | 10% | 2.25 | 3.19 | 9.88 | 3.61 | | | 10% | 1.87 | 3.33 | 13.55 | 3.36 |
| 10% | 61.5% | 0.01% | 0.03 | 0.19 | 0.07 | 0.27 | 1% | 39.1% | 0.01% | 0.022 | 0.95 | 0.21 | 0.73 |
| | | 0.1% | 0.24 | 1.19 | 0.76 | 1.76 | | | 0.1% | 0.22 | 2.38 | 2.12 | 2.07 |
| | | 1% | 1.09 | 2.88 | 7.69 | 5.10 | | | 1% | 1.60 | 3.73 | 13.71 | 3.04 |
| | | 10% | 8.61 | 13.32 | 76.63 | 28.96 | | | 10% | 5.84 | 7.99 | 110.64 | 17.78 |
| | | | | | | | | | | | | | |

| (c) $m = 10,000$ | | | | | | | |
|------------------|----------------|-----------------|-----------|-------|------------|-------|--|
| Density of X | Density of C | Density of mask | Dense C | | Sparse C | | |
| | | | C-code | GB | C-code | GB | |
| 0.01% | 1.0% | 0.01% | 0.004 | 1.58 | 0.009 | 0.08 | |
| | | 0.1% | 0.008 | 2.27 | 0.05 | 0.15 | |
| | | 1% | 0.06 | 2.32 | 0.47 | 0.46 | |
| | | 10% | 0.51 | 2.80 | 4.47 | 3.61 | |
| | | 20% | 1.35 | 4.63 | 13.31 | 8.38 | |
| | | 50% | 2.72 | 6.10 | 33.30 | 20.90 | |
| 0.05% | 4.8% | 0.01% | 0.005 | 1.76 | 0.02 | 1.17 | |
| | | 0.1% | 0.03 | 2.26 | 0.25 | 1.71 | |
| | | 1% | 0.27 | 2.48 | 2.45 | 1.86 | |
| | | 10% | 1.92 | 3.25 | 23.63 | 2.30 | |
| | | 20% | 4.30 | 6.94 | 67.19 | 5.06 | |
| | | 50% | 6.81 | 10.80 | 163.32 | 10.01 | |
| 0.1% | 9.5% | 0.01% | 0.006 | 1.67 | 0.04 | 1.24 | |
| | | 0.1% | 0.05 | 2.28 | 0.53 | 2.08 | |
| | | 1% | 0.50 | 2.67 | 4.96 | 2.24 | |
| | | 10% | 2.64 | 4.21 | 45.87 | 3.55 | |
| 1% | 30.0% | 0.01% | 0.04 | 1.60 | 0.72 | 1.87 | |
| | | 0.1% | 0.44 | 1.47 | 6.04 | 2.47 | |
| | | 1% | 2.30 | 3.22 | 36.74 | 5.40 | |
| | | 10% | 11.23 | 13.65 | 336.35 | 34.69 | |

Table 4: Running time (in seconds) of B_1X and B_2X .

| Method | B | X | Time for B_1X | Time for B_2X |
|------------|------------|------------|-----------------|-----------------|
| GB-NORMAL | <u>CSR</u> | <u>CSR</u> | 4.88 | 4.04 |
| | CSR | CSC | 4.83 | 4.05 |
| | CSC | CSR | 4.80 | 4.09 |
| | CSC | CSC | 4.78 | 4.04 |
| GB-ACCUM | CSR | CSR | 5.01 | 4.70 |
| | CSR | CSC | 4.99 | 4.72 |
| | CSC | CSR | 4.97 | 4.74 |
| | CSC | CSC | 5.02 | 4.73 |
| MKL-NORMAL | CSR | CSR | 10.25 | 6.47 |
| | CSR | CSC | 10.30 | 6.47 |
| | CSC | CSR | 46.08 | 46.02 |
| | CSC | CSC | 45.88 | 46.07 |

G.4 Operation (V): BX

To benchmark the multiplication (9), we dump X from the Wiki-500K data set and the following two matrices B_1 and B_2 under $K = 100$ and $K = 500$. As shown in (10), each column of B contains one non-zero entry, so the density of B is always $1/K$.

- $B_1 \in \mathbb{R}^{K \times m}$, where $K = 100$, $m = 500,091$, and the density of B_1 is 1%.
- $B_2 \in \mathbb{R}^{K \times m}$, where $K = 500$, $m = 500,091$, and the density of B_2 is 0.2%.

For the data formats, since B is rather sparse, so we consider CSR and CSC. As for the way to compute BX , we also consider GB-NORMAL, GB-ACCUM and MKL-NORMAL as in Section G.1. Details on running time could be found in Table 4.

REFERENCES

- [1] David Arthur and Sergei Vassilvitskii. 2007. k-means++: the advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. 1027–1035.
- [2] Charles Elkan. 2003. Using the triangle inequality to accelerate k-means. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning (ICML)*. 147–153.