Marigold: Efficient k-means Clustering in High Dimensions

Kasper Overgaard Mortensen
Aarhus University
km@cs.au.dk

Fatemeh Zardbani
Aarhus University
fatemeh.zardbani@cs.au.dk

Mohammad Ahsanul Haque
Aarhus University
mah@cs.au.dk

Steinn Ymir Agustsson
Aarhus Univeristy
agustsson@phys.au.dk

Davide Mottin
Aarhus University
davide@cs.au.dk

Philip Hofmann
Aarhus University
philip@phys.au.dk

Panagiotis Karras
Aarhus University
piekarras@gmail.com

ABSTRACT

How can we efficiently and scalably cluster high-dimensional data? The k-means algorithm clusters data by iteratively reducing intra-cluster Euclidean distances until convergence. While it finds applications from recommendation engines to image segmentation, its application to high-dimensional data is hindered by the need to repeatedly compute Euclidean distances among points and centroids. In this paper, we propose Marigold (k-means for high-dimensional data), a scalable algorithm for k-means clustering in high dimensions. Marigold prunes distance calculations by means of (i) a tight distance-bounding scheme; (ii) a stepwise calculation over a multiresolution transform; and (iii) exploiting the triangle inequality. To our knowledge, such an arsenal of pruning techniques has not been hitherto applied to k-means. Our work is motivated by time-critical Angle-Resolved Photoemission Spectroscopy (ARPES) experiments, where it is vital to detect clusters among high-dimensional spectra in real time. In a thorough experimental study with real-world data sets we demonstrate that Marigold efficiently clusters high-dimensional data, achieving approximately one order of magnitude improvement over prior art.

PVLDB Reference Format:
Kasper Overgaard Mortensen, Fatemeh Zardbani, Mohammad Ahsanul Haque, Steinn Ymir Agustsson, Davide Mottin, Philip Hofmann, and Panagiotis Karras. Marigold: Efficient k-means Clustering in High Dimensions. PVLDB, 16(7): 1740 - 1748, 2023.
doi:10.14778/3587136.3587147

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/AU-DIS/scalable-kmeans.

1 INTRODUCTION

Data in very high-dimensional spaces arises in the health sciences, astronomy, physics, finance, surveillance, bioinformatics, and via the *datafication* of human society [24, 25, 29]. To analyze such data, we need to tailor data science methods for high-dimensional spaces.

A prominent data science method used to automatically discover groupings in a raw data distribution and as an initial step before further analysis is *k-means clustering* by Lloyd's algorithm [30]. As an unsupervised method, it does not necessitate any annotations on the raw data as input; it partitions the data into k disjoint groupings, or *clusters*, such that each data record is closer to its allocated cluster's *center* or *representative*, by a given distance function, than to others. The distance function is usually *Euclidean distance* [16, 47], forming a Voronoi/Dirichlet tessellation among cluster centers [41].

While k-means clustering has been extensively applied and enhanced in terms of choosing the value of k [6, 26, 33, 39, 43], initializing cluster centers [44, 45, 48], and handling many small clusters [9], such methods evade scaling up the *exact* k-means solution to very high dimensions. The main bottleneck is calculating distances from each data record to each candidate cluster center in each iteration. For instance, a method focused on high-dimensional data records deals with data of only up to 100 features per document [9]. Some methods aim to eschew calculating high-dimensional distances [16, 18, 46], yet fully calculate distances that pass a filtering step. To our knowledge, no previous work aims to calculate high-dimensional distances only *partially* for the sake of time-efficiency without affecting the k-means clustering result.

In this paper we introduce Marigold: a pruning-intensive reformulation of Lloyd's algorithm that uses bounds based on the triangle inequality, as in [16, 18], and, unprecedentedly, a lightweight pre-processing and a tight double-bounding scheme along with a stepwise distance calculation over an *energy-concentrating* transformation to trim high-dimensional distance calculations during iterations. Marigold delivers the exact result of Lloyd's algorithm in at least one order of magnitude less time.

Motivating application. Marigold addresses a real-world need in time-critical Angle-Resolved Photoemission Spectroscopy (ARPES) condensed-matter physics experiments, where the high-dimensional spectra of a solid's electronic structure need to be clustered in real time, and even repetitively under modification of experimental parameters. In this application, clustering in reduced dimensionality can miss important physical properties of materials, while standard Lloyd's takes a prohibitive amount of time.

Proceedings of the VLDB Endowment, Vol. 16, No. 7 ISSN 2150-8097.
doi:10.14778/3587136.3587147

2 BACKGROUND

A *clustering* partitions a set of N-dimensional points $\mathbf{X}$ to disjoint sets or *clusters*. In k-means, a cluster $\mathbf{X}_i \subseteq \mathbf{X}$ is represented by a *centroid* $\mathbf{c}_i$ being the mean of the points therein:

$$\mathbf{c}_i = \frac{1}{|\mathbf{X}_i|} \sum_{\mathbf{x} \in \mathbf{X}_i} \mathbf{x}$$

The *compactness* of a cluster i is the sum of the squared Euclidean distances among points in a cluster and its centroid:

$$\sum_{\mathbf{x} \in \mathbf{X}_i} d(\mathbf{x}, \mathbf{c}_i)^2 = \sum_{\mathbf{x} \in \mathbf{X}_i} \|\mathbf{x} - \mathbf{c}_i\|^2 = \sum_{\mathbf{x} \in \mathbf{X}_i} \sum_{j=1}^{N} (\mathbf{x}_j - \mathbf{c}_{ij})^2 \quad (1)$$

The k-means problem is to find a clustering C of k clusters that minimizes total compactness.

Algorithm 1 LLOYD $(\mathbf{X}, k)$

```
1: $C \leftarrow$ SAMPLE$(k, \mathbf{X})$                  ▷ sample $k$ points from $\mathbf{X}$
2: while not $converged$ do
3:     for all $\mathbf{x}$ in $\mathbf{X}$ do               ▷ assign points to centroids
4:         $\alpha[\mathbf{x}] \leftarrow \arg\min_{\mathbf{c}}\{d(\mathbf{x}, \mathbf{c})\}$
5:     $converged \leftarrow$ RECALCULATE $(\mathbf{X}, C, \alpha[\mathbf{X}])$     ▷ Algorithm 2
6: return $\alpha[\mathbf{X}]$
```

Algorithm 2 RECALCULATE $(\mathbf{X}, C, \alpha[\mathbf{X}])$

```
1: $C_{old} \leftarrow C$
2: $converged \leftarrow$ true
3: for all $\mathbf{c}$ in $C$ do                  ▷ recalculate centroid positions
4:     $\mathbf{c} \leftarrow$ MEAN $(\{\mathbf{x} \in \mathbf{X} | \alpha[\mathbf{x}] = \mathbf{c}\})$          ▷ new centroid
5:     $div[\mathbf{c}] \leftarrow d(\mathbf{c}, \mathbf{c}_{old})$               ▷ centroid divergence
6:     if $div[\mathbf{c}] > \epsilon$ then $converged \leftarrow$ false
7: return $converged, div[C]$
```

2.1 Lloyd's algorithm

As the k-means problem is **NP**-hard [30], it is addressed by heuristics; *Lloyd's algorithm* [17, 30], starting with a set of k centroids, repetitively assigns each point to the *nearest* centroid's cluster by Euclidean distance and recalculates each centroid as the mean of its members' positions until convergence. In effect, it repetitively computes $O(kn)$ distances between points and centroids; this computation becomes prohibitive as dimensionality N grows [32]. Algorithm 1 presents the pseudocode, while Algorithm 2 presents the subroutine that recalculates centroid positions until they converge.

Several enhancements on Lloyd's algorithm have been proposed: choosing an appropriate value of k [6, 33] by the Elbow method [26], gap statistics [43], or the Silhouette method [39]; selecting initial cluster centers [44, 48] by *k-means++* [45]; addressing the problem with many small clusters by nearest-neighbor search from cluster centers, rather than from observation points [9]; avoiding distance computations by utilizing the triangle inequality [16, 18]. We review these triangle-inequality-based methods next.

2.2 Applying the triangle inequality

To prune the $O(kn)$ point-to-centroid Euclidean distance computations, Elkan [16] leverages the triangle inequality in a metric space, by which, for any point $\mathbf{x}$ and centroids $\mathbf{c}_i$ and $\mathbf{c}_j$, it holds that:

$$d(\mathbf{c}_i, \mathbf{c}_j) \le d(\mathbf{x}, \mathbf{c}_i) + d(\mathbf{x}, \mathbf{c}_j) \quad (2)$$

$$d(\mathbf{x}, \mathbf{c}_i) \le d(\mathbf{c}_i, \mathbf{c}_j) + d(\mathbf{x}, \mathbf{c}_j) \quad (3)$$

$$d(\mathbf{x}, \mathbf{c}_j) \le d(\mathbf{c}_i, \mathbf{c}_j) + d(\mathbf{x}, \mathbf{c}_i) \quad (4)$$

The following lemma follows directly from Equations (2) and (3):

LEMMA 2.1. $d(\mathbf{x}, \mathbf{c}_j) \ge |d(\mathbf{x}, \mathbf{c}_i) - d(\mathbf{c}_i, \mathbf{c}_j)|$

By Lemma 2.1, if we know $d(\mathbf{x}, \mathbf{c}_i)$ and $d(\mathbf{c}_i, \mathbf{c}_j)$, we can lower-bound $d(\mathbf{x}, \mathbf{c}_j)$. Further, if $d(\mathbf{c}_i, \mathbf{x}) \le \frac{d(\mathbf{c}_i, \mathbf{c}_j)}{2}$, then, from Equation (2) it follows that $d(\mathbf{x}, \mathbf{c}_i) \le \frac{1}{2}\left[d(\mathbf{x}, \mathbf{c}_i) + d(\mathbf{x}, \mathbf{c}_j)\right]$, which implies that $d(\mathbf{x}, \mathbf{c}_i) \le d(\mathbf{x}, \mathbf{c}_j)$. Formally:

LEMMA 2.2. *If* $d(\mathbf{x}, \mathbf{c}_i) \le \frac{d(\mathbf{c}_i, \mathbf{c}_j)}{2}$, *then* $d(\mathbf{x}, \mathbf{c}_i) \le d(\mathbf{x}, \mathbf{c}_j)$

By Lemma 2.2, if $\mathbf{c}_i$'s distance from $\mathbf{x}$ is at most half its distance from any other centroid, then $\mathbf{c}_i$ is nearest to $\mathbf{x}$. Thus, $\mathbf{x}$ may be re-assigned from $\mathbf{c}_i$ to $\mathbf{c}_j$ only if $d(\mathbf{x}, \mathbf{c}_i) > d(\mathbf{c}_i, \mathbf{c}_j)/2$; in effect, these lemmata prune recomputing point-to-centroid distances using inter-centroid distances and prior point-to-centroid distances.

Hamerly [18] proposed two further conditions to eschew distance computations. The first condition utilizes a lower bound ℓ_H on the distance to a point's nearest *non-assigned* centroid. At any iteration, if an upper bound for the distance to a point $\mathbf{x}$'s currently nearest centroid derived by Lemma 2.2 is no more than ℓ_H, then the nearest centroid to $\mathbf{x}$ remains unchanged. The second condition uses centroid-to-centroid distances as in Lemma 2.2.

2.3 Other related work

We accelerate LLOYD's algorithm in high dimensions by pruning its calculations. Other works, which do not address the same problem, render LLOYD more robust via *seeding*, accelerate it in low dimensions via *indexing*, and provide guarantees on top of a presumed approximation algorithm via *dimensionality reduction*. These goals are orthogonal to ours as we explain in the following.

Seeding. k-means++ [45] boosts the stability of LLOYD through a smart initialization, extended to a stream setting [2]; k-means|| [4] allows for parallelization of k-means++ via oversampling. These are orthogonal works, as they cater to initialization only.

Index-based. Some techniques accelerate k-means by indexing [35]. Others propose using cosine similarity and use an inverted index to assign points to centroids [9]. However, such works require an additional $O(n \log n)$ pre-computation for index building, while indexing in high dimensions is problematic in itself; thus, unfortunately, such methods do not address high-dimensional k-means.

Sampling. Sampling methods [21, 40] select a fixed amount of points to cluster. However, such techniques still require many samples when clusters are small. Kumar et al. [27] samples a fixed number of points to devise a $1/\epsilon$-approximate solution linear in n and N but, unfortunately, exponential in k. Our proposal are orthogonal to, and may be used along with, such sampling methods.

Dimensionality reduction. Several works study k-means on dimensionality reduced compared to the inherent dimensionality of the data, and eventually independent thereof [5, 7, 12], leading to the currently best $(1+\epsilon)$-approximation with $O(\log(k/\epsilon)/\epsilon^2)$ dimensions in [31]. However, such methods typically assume a pre-existing γ-approximation of the optimal k−*means* solution, yet resort to the Lloyd's algorithm to obtain a good solution in practice [8]. Further, the assumed reduced dimensionality is still high in practice; it depends on $1/\epsilon^2$, which corresponds at least to $N > 10^4$ [8] to achieve a 10% deviation from the optimal solution, even assuming a γ-approximation algorithm were available. Our work is orthogonal and complementary to such approaches, which still require running Lloyd's algorithm in a significant number of dimensions; thus, improving the scalability of Lloyd's algorithm would provide a direct benefit on real-world data science applications and *also* enhance the usability of such theoretical advances.

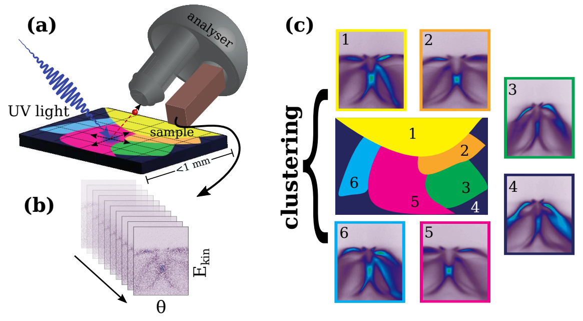


Figure 1: k-means for nanoARPES: *(a)* a light pulse illuminates a position on the sample surface to emit electrons of variant energy and angle; *(b)* a high-dimensional photoemission intensity image represents electronic structure at each position; *(c)* k-means detects electronic structure areas.

3 APPLICATION DOMAIN

The need to scale k-means to high-dimensional data arises in Angle-Resolved Photo-Emission Spectroscopy (ARPES), an experimental method in condensed-matter physics that studies the electronic structure of materials [37]. In an ARPES experiment, electrons are excited from a sample by means of ultraviolet light exploiting the photoelectric effect. We focus on *spatially-resolved* ARPES, also called nanoARPES due to its high resolution [19, 38]: as Figure 1 shows, a finely focused light spot is raster-scanned across a sample to yield, *at each position*, a photoemission intensity image, or *spectrum*, that records photoemission intensity as a function of electron energy E_{kin} and emission angle θ; such spectra encode the material's *band structure* and its electronic properties.

The development of photon sources and detectors has increased the measurable parameter space, data dimensionality, and acquisition rate. Machine learning methods have proven effective at spotting differences in ARPES data [24, 34, 36], yet they resort to dimensionality reduction that risks missing essential features. As the cost of acquisition remains high and the time of access to the required light sources such as synchrotrons or free-electron lasers is limited, physicists need to quickly extract clusters from a stack of photoemission images at high dimensionality to make real-time decisions on how to proceed during an experimental data-collection session; to this end, we aim at accelerating k-means clustering on high-dimensional data as those arising in nanoARPES.

4 MARIGOLD

While Elkan's and Hamerly's methods [16, 18] reduce distance calculations, their runtime is still prohibitive on high data dimensionality. After all, by those methods, once a distance passes the filtering step, it is fully calculated. Arguably, a method should prune distance calculations even *after* a first filtering step, i.e., perform *partial* distance calculations with *early abandonment*. Here we devise Marigold, a method that utilizes a hierarchical energy-concentrating feature transformation to gradually calculate distances over the feature hierarchy, on top of triangle-inequality-based pruning. We first show how to exploit the triangle inequality utilizing both of [16, 18] and then illustrate the stepwise calculation of distances.

Algorithm 3 Triangle-based k-means $(\mathbf{X}, k)$

```
1:  $C \leftarrow$ Sample$(k, \mathbf{X})$                              ▷ initial cluster centers
2:  $\alpha[\mathbf{X}] = \emptyset$                                    ▷ null assignment
3:  $\ell_E(\mathbf{X}, C) = \ell_H(\mathbf{X}) = near[C] = \mathbf{0}$  ▷ initialize lower bounds
4:  $u_E(\mathbf{X}) = \infty$                                         ▷ initialize upper bounds
5:  while not $converged$ do                                            ▷ main loop
6:      for all $\mathbf{x}$ in $\mathbf{X}$ do                         ▷ assign points to centroids
7:          if $u_E(\mathbf{x}) > \max\{near[\alpha[\mathbf{x}]], \ell_H(\mathbf{x})\}$ then   ▷ Hamerly
8:              $\ell_E(\mathbf{x}, \alpha[\mathbf{x}]) \leftarrow d(\mathbf{x}, \alpha[\mathbf{x}])$   ▷ Elkan's LB
9:              $u_E(\mathbf{x}) \leftarrow d(\mathbf{x}, \alpha[\mathbf{x}])$                   ▷ Elkan's UB
10:             for all $\{\mathbf{c} \in C | \mathbf{c} \neq \alpha[\mathbf{x}]\}$ do              ▷ Elkan check
11:                 if $u_E(\mathbf{x}) > \max\{\ell_E(\mathbf{x}, \mathbf{c}), d(\alpha[\mathbf{x}], \mathbf{c})/2\}$ then
12:                     $\ell_E(\mathbf{x}, \mathbf{c}) \leftarrow d(\mathbf{x}, \mathbf{c})$        ▷ calculate dist.
13:                     if $d(\mathbf{x}, \mathbf{c}) < u_E(\mathbf{x})$ then
14:                         $\alpha[\mathbf{x}] \leftarrow \mathbf{c}$
15:                         $u_E(\mathbf{x}) \leftarrow d(\mathbf{x}, \mathbf{c})$              ▷ Elkan's UB
16:     $converged, div[C] \leftarrow$ Recalculate $(\mathbf{X}, C, \alpha[\mathbf{X}])$     ▷ Alg. 2
17:     if not $converged$ then
18:         UpdateBounds $(\mathbf{X}, C, \ell_E, u_E, \ell_H, div, near)$                  ▷ Alg. 4
19: return $\alpha[\mathbf{X}]$
```

Algorithm 4 UpdateBounds $(\mathbf{X}, C, \ell_E, u_E, \ell_H, div, near)$

```
1:  for all $\mathbf{x}$ in $\mathbf{X}$ do                             ▷ update bounds
2:      for all $\mathbf{c}$ in $C$ do                                  ▷ Elkan's LB
3:          $\ell_E(\mathbf{x}, \mathbf{c}) \leftarrow \max\{0, \ell_E(\mathbf{x}, \mathbf{c}) - div[\mathbf{c}]\}$
4:      $u_E(\mathbf{x}) \leftarrow u_E(\mathbf{x}) + div[\alpha[\mathbf{x}]]$          ▷ Elkan's UB
5:      $\ell_H(\mathbf{x}) \leftarrow \min\{\ell_E(\mathbf{x}, \mathbf{c} \neq \alpha[\mathbf{x}])\}$   ▷ Hamerly's LB
6:  for all $\mathbf{c}$ in $C$ do
7:      $near[\mathbf{c}] \leftarrow \frac{1}{2}\min_{\mathbf{c}' \neq \mathbf{c}}\{d(\mathbf{c}, \mathbf{c}')\}$   ▷ nearest other centroid
8:  return $\ell_E, u_E, \ell_H, near$
```

4.1 Applying the triangle inequality

Both Elkan [16] and Hamerly [18] exploit the triangle inequality to contain distance calculations, yet their methods have yet to be combined. In Algorithm 3, we put together all opportunities, derived from [16] and [18], to avoid distance calculations using bounds derived from the triangle inequality, while assigning data points to centroids. Algorithm 4 presents the bound update subroutine. We use the following bounds to eschew distance calculations:

Elkan's lower bound $\ell_E(\mathbf{x}, \mathbf{c})$ bounds the distance from $\mathbf{x}$ to centroid $\mathbf{c}$ as the latter moves in one iteration so that $d(\mathbf{x}, \mathbf{c}) \geq \ell_E(\mathbf{x}, \mathbf{c})$.

Line 8 of Algorithm 3 resets $\ell_E(\mathbf{x}, \alpha[\mathbf{x}])$ to the distance from $\mathbf{x}$ to its assigned centroid in each iteration (bar the first one), $d(\mathbf{x}, \alpha[\mathbf{x}])$, as soon as that distance is calculated. Thereafter, in Line 11, if the distance from $\mathbf{x}$ to its currently assigned centroid $\alpha[\mathbf{x}]$ exceeds both $\ell_E(\mathbf{x}, \mathbf{c})$ for a centroid $\mathbf{c}$ and the bound derived from Lemma 2.2, we proceed to check whether $\mathbf{x}$ is closer to $\mathbf{c}$ than $\alpha[\mathbf{x}]$. To do so, Line 12 calculates $d(\mathbf{x}, \mathbf{c})$ and also resets $\ell_E(\mathbf{x}, \mathbf{c})$ accordingly. In each iteration, Line 3 of Algorithm 4 subtracts from each $\ell_E(\mathbf{x}, \mathbf{c})$ the divergence of $\mathbf{c}$, $div[\mathbf{c}]$, according to Lemma 2.1.

Elkan's upper bound $u_E(\mathbf{x})$ bounds the distance from $\mathbf{x}$ to its currently assigned centroid $\alpha[\mathbf{x}]$, i.e., $d(\mathbf{x}, \alpha[\mathbf{x}]) \le u_E(\mathbf{x})$. In Line 7 of Algorithm 3, if $u_E(\mathbf{x})$ does not exceed a bound derived from Lemma 2.2 and Hamerly's lower bound (discussed below) on the distance from $\mathbf{x}$ to any other centroid $\mathbf{c}$, then $\mathbf{x}$ remains closest to $\alpha[\mathbf{x}]$ in that iteration, hence we eschew calculating distances from $\mathbf{x}$. Lines 9 and 15 reset $u_E(\mathbf{x})$ to the distance from $\mathbf{x}$ to its new centroid. In each iteration, Line 4 of Algorithm 4 adds to each $u_E(\mathbf{x})$ the divergence of $\alpha[\mathbf{x}]$, $div[\alpha[\mathbf{x}]]$, following the triangle inequality.

Hamerly's lower bound $\ell_H(\mathbf{x})$ bounds the distance from $\mathbf{x}$ to any centroid $\mathbf{x}$ is *not* assigned to; if $u_E(\mathbf{x}) \le \ell_H(\mathbf{x})$, then $\mathbf{x}$ may remain with its current centroid $\alpha[\mathbf{x}]$. Line 7 of Algorithm 3 compares $\ell_H(\mathbf{x})$, along with the bound derived from Lemma 2.2, to $u_E(\mathbf{x})$, to decide on this matter, as discussed in the context of $u_E(\mathbf{x})$ above. In each iteration, Line 5 of Algorithm 4 adjusts each $\ell_H(\mathbf{x})$ to the minimum $\ell_E(\mathbf{x}, \mathbf{c})$ among centroids other than $\alpha[\mathbf{x}]$.

4.2 Leveraging stepwise distance calculations

Pruning by the triangle inequality reduces the *number* of Euclidean distance calculations in each iteration. However, it leaves the cost of each such calculation intact: once a distance calculation passes the pruning stage, it has to be executed at full cost. Unfortunately, this cost grows prohibitively with dimensionality. More drastic measures are needed to discard or contain distance calculations.

We might use *dimensionality reduction* [42] hoping to obtain a data set that preserves the characteristics of the original. However, such reduction incurs *information loss*, which may severely affect the clustering result in critical applications where clustering aims to discover unknown data properties, such as nanoARPES which aims to unveil the electronic structure of materials [38]. Methods with approximation guarantees [8] typically require high dimensions to render their guarantees meaningful and are applied on top of the Lloyd heuristic anyway. Therefore, an ability to scale up Lloyd to high dimensions is imperative even to use such methods.

Nevertheless, dimensionality reduction methods typically rest on an *energy-preserving* transform that unreels the data in a hierarchy of progressively finer levels of resolution [22] and concentrates energy in lower-order coefficients. Full precision is available at the *finest* level of resolution, yet cluster assignments can be decided at *coarser* levels, without reading all features. We employ this property of a multi-resolution transform and a tight bounding scheme to calculate distances à la carte, as required to decide cluster membership; therewith we accelerate Lloyd while preserving its output.

We use the *Discrete Cosine Transform* (DCT), extensively used in image compression [11, 15]. Let $[F]_2$ be an $M \times N$ matrix representing a 2-dimensional data set and $[G]_2$ be the matrix of its 2-dimensional DCT coefficients. Element (u, v) of $[G]_2$ is [28]:

$$g(u,v) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)\cos\left[\frac{(m+\frac{1}{2})u\pi}{M}\right]\cos\left[\frac{(n+\frac{1}{2})v\pi}{N}\right] \quad (5)$$

while each element is normalized by $\frac{2k_u k_v}{\sqrt{MN}}$ with $k_0 = {}^1\!/\!\sqrt{2}$, $k_{i\neq 0} = 1$.

DCT preserves Euclidean distance on the transformed space [3] over DCT coefficient vectors $\mathbf{x}$ of raw data points; thus, the Euclidean distance $d(\mathbf{x}, \mathbf{c})$ of a point $\mathbf{x}$ from a centroid $\mathbf{c}$ is:

$$d(\mathbf{x},\mathbf{c}) = \sqrt{\sum x_i^2 + c_i^2 - 2x_i c_i} = \sqrt{\sum x_i^2 + \sum c_i^2 - 2\sum x_i c_i} \quad (6)$$

where the summation is over *all relevant* DCT (u, v) pairs, identified by i. For any point $\mathbf{x}$ and centroid $\mathbf{c}$, we may pre-compute $\sum x_i^2$ and $\sum c_i^2$ terms. However, we should still compute the dot-product term $\sum x_i c_i$ on the fly. Still, we can bound this term using the Cauchy-Schwarz inequality [23] as follows:

$$\left(\sum x_i c_i\right)^2 \le \sum x_i^2 \sum c_i^2 \Leftrightarrow \sum x_i c_i \le \sqrt{\sum x_i^2 \sum c_i^2} \quad (7)$$

From Equations (6) and (7) it follows that:

$$\begin{aligned} d(\mathbf{x},\mathbf{c})^2 &\ge \sum x_i^2 + \sum c_i^2 - 2\sqrt{\sum x_i^2 \sum c_i^2} \\ d(\mathbf{x},\mathbf{c})^2 &\le \sum x_i^2 + \sum c_i^2 + 2\sqrt{\sum x_i^2 \sum c_i^2} \end{aligned} \quad (8)$$

All terms in Equation (8) involve sums of squares over the dimensions of $\mathbf{x}$ and $\mathbf{c}$, which we only need to compute *once*. We utilize these bounds to calculate point-to-centroid distances in *stepwise* fashion, as in [23]: in each step, we derive exact distances up to an increasing number of DCT features, the *known* part, along with upper and lower bounds, by Equation (8), for the diminishing remainder, *unknown* part. While doing so, we discard from consideration, for each point, centroids whose lower bound gets *higher* than the lowest upper bound among other centroids. Starting with DCT term $(0, 0)$, in level ℓ we extend the known DCT features from $4^{\ell-1}$ to 4^{ℓ}, $\ell > 0$, spanning a square-shaped area from $(0, 0)$ to $(2^{\ell}, 2^{\ell})$ in 2d, doubling the square side in each step. We pre-compute sums of $\sum_{\ell'=\ell}^{L} 4^{\ell'} = \frac{4^{L+1}-4^{\ell}}{3}$ squared feature values in the range from 4^{ℓ} to 4^{L}, for each ℓ, from the 2-dimensional DCT vector $\mathbf{x}$ of each data point in each such area, yielding a list $\mathbf{x}_{sq}$ of size $O(\log N)$.

Algorithm 5 Stepwise $(\mathbf{X}, k)$

```
1:  $L \leftarrow \log_4 N$                                  ▷ levels for stepwise distances
2:  $\mathbf{X} \leftarrow$ Transform $(\mathbf{X})$                       ▷ by DCT [1]
3:  $\mathbf{X}_{sq} \leftarrow$ Squared $(\mathbf{X})$          ▷ sums of $4^{\ell'}$ squares, $\ell' = \ell, \dots, L$
4:  $C \leftarrow$ Sample$(k, \mathbf{X})$                            ▷ initial cluster centers
5:  while not $converged$ do
6:      $C_{sq} \leftarrow$ Squared $(C)$                         ▷ recalculate $C_{sq}$
7:      for all $\mathbf{x}$ in $\mathbf{X}$ do                      ▷ assign points to centroids
8:          $\alpha[\mathbf{x}] \leftarrow$ SetLabel $(\mathbf{x}, C, \mathbf{x}_{sq}, C_{sq}, L)$      ▷ Algorithm 6
9:      $converged \leftarrow$ Recalculate $(\mathbf{X}, C, \alpha[\mathbf{X}])$      ▷ Algorithm 2
10: return $\alpha[\mathbf{X}]$
```

Algorithm 5 presents our k-means algorithm that leverages stepwise distance calculations over DCT-transformed data. Algorithm 6 shows the subroutine that assigns a point to a centroid, where level ℓ indicates the size of the square area of DCT features forming the *known* part in calculations. Starting from one known feature ($\ell = 0$), in each iteration it discards candidate centroids with a lower bound exceeding the lowest upper bound (Lines 7–8), expands the calculation for each candidate centroid (Line 11), and maintains the centroid of lowest upper bound (Lines 12–14). The iterations terminate when, at Line 5, we reach the last level, where bounds are exact distances, hence we select the centroid of lowest upper bound, or only one candidate centroid remains, hence we select it.

Algorithm 6 SetLabel $(\mathbf{x}, C, \mathbf{x}_{sq}, C_{sq}, L)$

```
 1: $\ell \leftarrow 0$;  $LB(C) \leftarrow \mathbf{0}$;  $UB_{\min} \leftarrow \infty$
 2: for all $\{\mathbf{c} \in C\}$ do
 3:     $d[\mathbf{c}] \leftarrow \mathbf{x}_{sq}[0] + \mathbf{c}_{sq}[0]$          ▷ coarsest distance estimate
 4:     $mask[\mathbf{c}] \leftarrow 1$                         ▷ candidate centroid c for x
 5: while $\ell \le L \wedge \sum_{\mathbf{c}} mask[\mathbf{c}] > 1$ do                 ▷ point undecided
 6:     for all $\{\mathbf{c} \in C | mask[\mathbf{c}] = 1\}$ do               ▷ active candidates
 7:         if $UB_{\min} < LB(\mathbf{c})$ then
 8:             $mask[\mathbf{c}] \leftarrow 0$       ▷ prune centroid at previous level
 9:         else                                   ▷ move on to this level
10:             $\mathcal{A} \leftarrow (\mathbf{x}, \mathbf{c}, \mathbf{x}_{sq}, \mathbf{c}_{sq}, \ell, d[\mathbf{c}])$
11:             $LB(\mathbf{c}), UB(\mathbf{c}), d[\mathbf{c}] \leftarrow$ DistToLevel $(\mathcal{A})$     ▷ Alg. 7
12:             if $UB(\mathbf{c}) < UB_{\min}$ then
13:                 $\alpha \leftarrow \mathbf{c}$
14:                 $UB_{\min} \leftarrow UB(\mathbf{c})$      ▷ keep lowest UB across c
15:     $\ell \leftarrow \ell + 1$
16: return $\alpha$
```

Algorithm 7 DistToLevel $(\mathbf{x}, \mathbf{c}, \mathbf{x}_{sq}, \mathbf{c}_{sq}, \ell, d)$

```
1: $d \leftarrow d - 2 \cdot \mathbf{x}[\ell] \cdot \mathbf{c}[\ell]$          ▷ known distance with all squares
2: $margin \leftarrow 2 \cdot \sqrt{\mathbf{x}_{sq}[\ell+1] \cdot \mathbf{c}_{sq}[\ell+1]}$
3: $LB \leftarrow d - margin$                                  ▷ lower bound
4: $UB \leftarrow d + margin$                                  ▷ upper bound
5: return $LB, UB, d$
```

Algorithm 7 updates the distance for a given pair in each level of the stepwise distance computation; it updates the *known* distance term to the current level ℓ and adds the Cauchy-Schwarz bounds for the unknown *margin*, using pre-computed sums of squares of the $\frac{4^{L+1}-4^{\ell+1}}{3}$ features from level $\ell + 1$ to level L. Lower and upper bounds differ in one sign, as in Equation 8.

4.3 Putting it all together

Both triangle-inequality-based checks and stepwise calculations prune distance calculations, the former by an one-off filtering, the latter by progressively disqualifying centroid candidates in each iteration. Here, we merge these two methods to craft an integrated solution, Marigold (Algorithm 8). Marigold performs stepwise distance calculations on DCT-transformed data using pre-computed information by DistToLevel (Algorithm 7), as Stepwise (Algorithm 5) does, using the same bounds as Triangle-based k-means (Algorithm 3) and updating them by UpdateBounds (Algorithm 4), and recalculates centroids by Recalculate (Algorithm 2).

Still, Marigold keeps track of the tightest among the bounds obtained by stepwise calculations and those obtained by the triangle inequality, as Algorithm 9 illustrates, hence it has stronger pruning capacity than Stepwise (Algorithm 5).

Algorithm 8 Marigold $(\mathbf{X}, k)$

```
 1: $L \leftarrow \log_4 N$                              ▷ levels for stepwise distances
 2: $\mathbf{X} \leftarrow$ Transform $(\mathbf{X})$                                  ▷ by DCT [1]
 3: $\mathbf{X}_{sq} \leftarrow$ Squared $(\mathbf{X})$          ▷ sums of $4^{\ell'}$ squares, $\ell' = \ell, \dots, L$
 4: $C \leftarrow$ Sample$(k, \mathbf{X})$                             ▷ initial cluster centers
 5: $\alpha[\mathbf{X}] = \emptyset$                                       ▷ null assignment
 6: $\ell_E(\mathbf{X}, C) = \ell_H(\mathbf{X}) = near[C] = \mathbf{0}$          ▷ initialize lower bounds
 7: $u_E(\mathbf{X}) = \infty$                                ▷ initialize upper bounds
 8: while not $converged$ do                                     ▷ main loop
 9:     $C_{sq} \leftarrow$ squared$(C)$                           ▷ recalculate $C_{sq}$
10:     for all $\mathbf{x}$ in $\mathbf{X}$ do                       ▷ assign points to centroids
11:         if $u_E(\mathbf{x}) > \max\{near[\alpha[\mathbf{x}]], \ell_H(\mathbf{x})\}$ then  ▷ Hamerly
12:             $\mathcal{A} \leftarrow (\mathbf{x}, C, \mathbf{x}_{sq}, C_{sq}, L, \alpha[\mathbf{x}], \ell_E(\mathbf{x}, C), u_E(\mathbf{x}))$
13:             SetLabelMG $(\mathcal{A})$                             ▷ Algorithm 9
14:     $converged, div[C] \leftarrow$ Recalculate $(\mathbf{X}, C, \alpha[\mathbf{X}])$    ▷ Alg. 2
15:     if not $converged$ then
16:         UpdateBounds $(\mathbf{X}, C, \ell_E, u_E, \ell_H, div, near)$      ▷ Alg. 4
17: return $\alpha[\mathbf{X}]$
```

Algorithm 9 SetLabelMG $(\mathbf{x}, C, \mathbf{x}_{sq}, C_{sq}, L, \alpha, \ell_E(C), u_E)$

```
 1: $\ell \leftarrow 0$
 2: for all $\{\mathbf{c} \in C\}$ do
 3:     $d[\mathbf{c}] \leftarrow \mathbf{x}_{sq}[0] + \mathbf{c}_{sq}[0]$          ▷ coarsest distance estimate
 4:     $mask[\mathbf{c}] \leftarrow 1$                         ▷ candidate centroid c for x
 5: while $\ell \le L \wedge \sum_{\mathbf{c}} mask[\mathbf{c}] > 1$ do                 ▷ point undecided
 6:     for all $\{\mathbf{c} \in C | mask[\mathbf{c}] = 1\}$ do               ▷ active candidates
 7:         if $u_E < \max\{\ell_E(\mathbf{c}), {}^{d(\alpha,\mathbf{c})}/_2\}$ then            ▷ Elkan check
 8:             $mask[\mathbf{c}] \leftarrow 0$       ▷ prune centroid at previous level
 9:         else                                   ▷ move on to this level
10:             $\mathcal{A} \leftarrow (\mathbf{x}, \mathbf{c}, \mathbf{x}_{sq}, \mathbf{c}_{sq}, \ell, d[\mathbf{c}])$
11:             $LB(\mathbf{c}), UB(\mathbf{c}), d[\mathbf{c}] \leftarrow$ DistToLevel $(\mathcal{A})$     ▷ Alg. 7
12:             $LB(\mathbf{c}) \leftarrow \sqrt{LB(\mathbf{c})}$;  $UB(\mathbf{c}) \leftarrow \sqrt{UB(\mathbf{c})}$
13:             if $LB(\mathbf{c}) > \ell_E(\mathbf{c})$ then
14:                 $\ell_E(\mathbf{c}) \leftarrow LB(\mathbf{c})$            ▷ keep highest LB per c
15:             if $UB(\mathbf{c}) < u_E$ then
16:                 $\alpha \leftarrow \mathbf{c}$
17:                 $u_E \leftarrow UB(\mathbf{c})$           ▷ keep lowest UB across c
18:     $\ell \leftarrow \ell + 1$
19: return $\alpha, \ell_E(C), u_E$
```

Algorithm 9 follows the pattern of SetLabel (Algorithm 6), yet introduces cardinal aspects of Triangle-based k-means (Algorithm 3) therein. In each level of stepwise calculations, it uses Elkan bounds to check, in Line 7, whether to retain each candidate centroid; thereby it injects the comparison of Lemma 2.2 using inter-centroid distances, into the bound comparison of Line 7, Algorithm 6. Line 12 of SetLabelMG obtains square roots of stepwise bounds to render them appropriate for triangle-inequality Euclidean-distance comparisons. Last, Lines 13–17 use stepwise bounds to *tighten, if possible, both* the Elkan lower bound for the examined centroid, used by Line 7 in the next iteration over centroids, and the Elkan lowest upper bound across centroids, used by Line 7 with the next centroid, maintaining the centroid of lowest upper bound as the running assignment. Upon reaching the last level, where bounds are exact distances, or if only one candidate centroid remains (Line 5), we settle on the running assignment. Line 19 returns, along with the assigned centroid, the (possibly exact) Elkan lower bounds of distances to each centroid and Elkan upper bound of the distance to the assigned centroid; subsequently, Algorithm 2 recalculates centroid positions and Algorithm 4 updates Elkan bounds based on centroid divergences and derives a Hamerly bound to be used in Line 11 of Algorithm 8.

While Algorithm 3 derives Elkan and Hamerly bounds from exact distances, MARIGOLD (Algorithm 8) extracts then from stepwise bounds. The earlier MARIGOLD abandons a point-to-centroid distance calculation, the looser the $\ell_E(\mathbf{x}, \mathbf{c})$ bound it derives. Still, the next iteration tightens such a bound, if needed. Eventually, MARIGOLD performs fewer distance calculations than both TRIANGLE-BASED k-MEANS and STEPWISE, at the cost of more inequality checks than STEPWISE (Line 11, Algorithm 8 and Line 13, Algorithm 9) and more square-root operations than both (Line 12, Algorithm 9).

In short, MARIGOLD prunes distance calculations effectively, incorporating Hamerly and Elkan bounds into bounds maintained by stepwise distance calculations. We emphasize that MARIGOLD provides the same results as Lloyd's algorithm.

5 EXPERIMENTAL RESULTS

We conduct an experimental study of k-means methods on data from real-world nanoARPES experiments. To enhance the reproducibility of our results, we provide an implementation-independent measure, the number of feature distance calculations. We measure runtime in two environments: by Python 3.10 implementation on a Windows 10, Intel Core i7-1165G7 machine @2.80GHz with 64GB RAM, and by C++ implementation with -O3 flag on an Ubuntu 20.04.4, Intel Core i7-10610U machine @1.8GHz with 48GB RAM. We report runtimes including pre-computations, while we run with five different initializations and report averages.

Methods. We juxtapose the following methods for k-means:

- LLOYD [30], the classical iterative k-means algorithm.
- ELKAN [16], which uses Elkan bounds only as in Section 4.1.
- HAMERLY [18], using the Hamerly bound only as in Section 4.1.
- TRIANGLE-BASED k-MEANS, as presented in Section 4.1.
- KMEANS-G* [20], a state-of-the-art algorithm that exploits geometric primitives to perform k-means in high dimensions.
- STEPWISE, performing stepwise calculations as in Section 4.2.
- MARIGOLD, which integrates stepwise calculations with the bounds of ELKAN and HAMERLY, as in Section 4.3.

Data sets. We run k-means on four data sets obtained from two nanoARPES studies and four image datasets from other fields. To illustrate the need to run k-means in high dimensions we juxtapose the results by clustering in reduced dimensionality and those obtained in high dimensions to the physical ground truth [10, 13].

- **gr_flake** is a map of 49×39 = 1911 nanoARPES images, each with resolution 256×256, on a graphene sample. Clustering with k = 30 separates areas in a way very similar to a systematic fit of the spectra. Figure 2(a) shows the original result of a fitted momentum (Figure 3(b) in [13]) whereas panels (b) and (c) show clustering at DCT dimensionalities 8 × 8 and 64 × 64. Using a higher DCT dimensionality gives a more accurate result.
- **misfit** comprises three maps of 12×14 = 168 nanoARPES images, each with resolution 1024×1024. **misfit_VB**, **misfit_Se3d**, and **misfit_Bi5d**, represent the same sample position observed at different energies (Figure S1 in [10] with a reversed color scale for **misfit_Se3d**). Figure 2(d)-(f) shows the photoemission intensity in a manually selected characteristic region of interest for the three data sets, while images (g)-(i) visualize k-means results with k = 16 performed on entire spectra, confirming that high-dimensional k-means is appropriate for such data analysis.
- **MNIST** [14] comprises 28×28 = 784-size handwritten digits, in a training set of 60 000 images and a test set of 10 000 images. We upscale the test set to size 128×128 = 16384 and 256×256 = 65536.
- **Deep Globe Land Cover (DGLC)** contains three collections of RGB satellite images of land cover types, 1606 in **DGLC Train**, 171 in **DGLC Validation** and 172 in **DGLC Test**. We converted the images to grayscale and normalized resolutions to 256×256 in the first and 1024×1024 in the other collections.

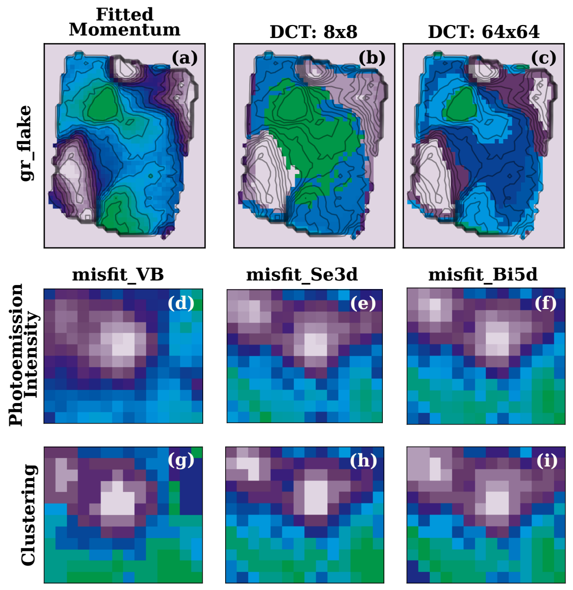
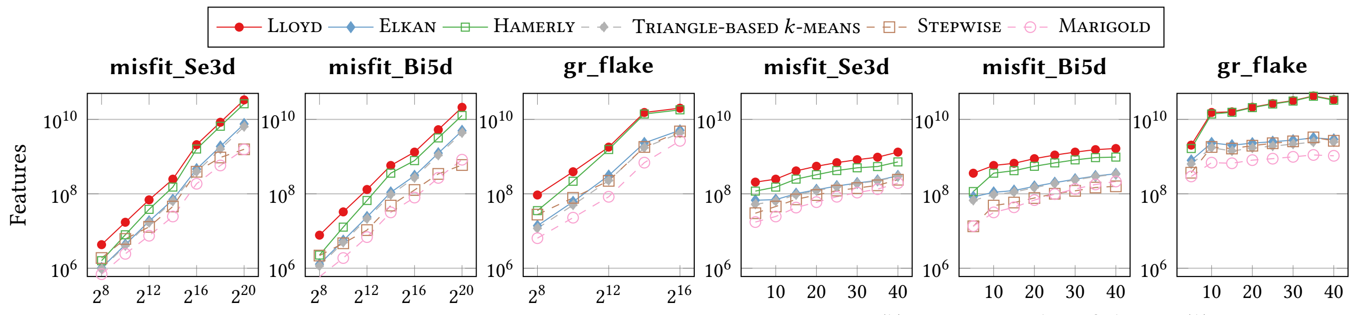


Figure 2: Clustering results on gr_flake and misfit data and respective ground truth. gr_flake: (a) fit result shown by color map and contour lines [13]; k-means result as color map, with the contours of (a) replicated, on (b) 8×8 and (c) 64×64 DCT features with k = 30. misfit: (d)-(f) photoemission intensity in specific region of interest [10]; (g)-(i) clustering with k = 16. In k-means results, colors are ordered as in the ground truth.

5.1 Distance calculations

We first report results on the number of feature distance calculation operations each method performs vs. the number of DCT features representing the data and that of clusters k.

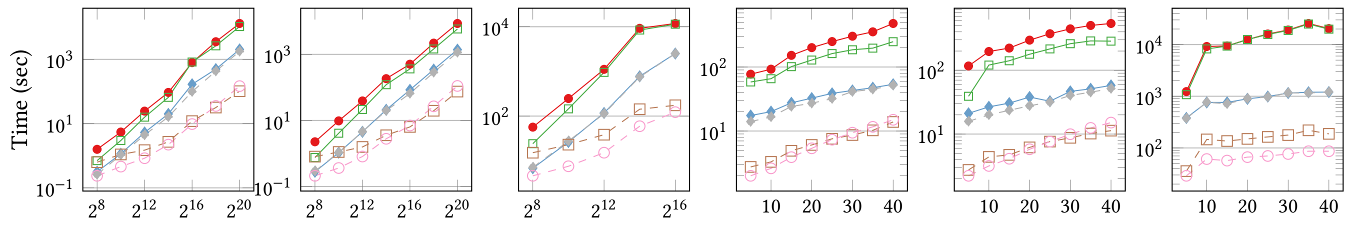
Varying DCT features. Figure 3(a) shows how the number of features entering distance calculations varies vs. the number of DCT features representing the data with k = 10 clusters. All methods show linear growth, with a gap between LLOYD and HAMERLY, on the one hand, and other methods, on the other hand; this result reconfirms that HAMERLY falters [18] on dimensionality above 50.



(a) Varying number of DCT features

(b) Varying number of clusters (k)

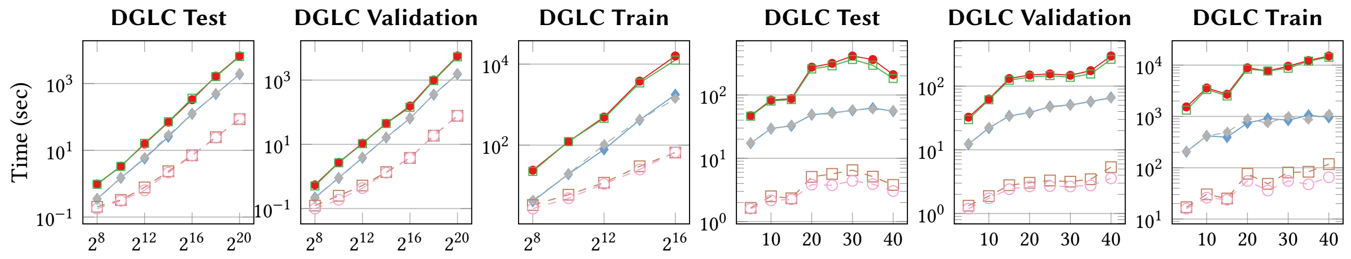
Figure 3: Features used vs. DCT features with $k = 10$ and vs. clusters k with 2^{14} DCT features.



(a) Varying number of DCT features

(b) Varying number of clusters (k)

Figure 4: Total time vs. DCT features with $k = 10$ and vs. clusters k with 2^{14} DCT features.



(a) Varying number of DCT features

(b) Varying number of clusters (k)

Figure 5: Total time vs. DCT features with $k = 10$ and vs. clusters k with 2^{14} DCT features, DGLC data.

Triangle-based k-means does well on lower dimensionality, paralleled by Elkan. As dimensionality grows, Stepwise and Marigold become the best performers. Marigold performs best in lower dimensions, but Stepwise covers the gap as dimensionality grows, as its bounds compensate for the missing pruning capacity.

Varying clusters. Figure 3(b) shows how the number of features used in distance calculations varies vs. k with $128 \times 128 = 2^{14}$ DCT features. The pattern starts out matching Figure 3(a), but gaps among methods are accentuated as k grows. The overhead of centroid-to-centroid distance calculations grows with k; Stepwise outperforms methods subject to that overhead, especially on misfit, where that overhead is more significant as part of the total cost due to fewer data points. Marigold performs the best overall.

5.2 Runtime

Having established the pattern of implementation-independent feature distance calculations, we examine wall-clock time, which is affected by operations we have hitherto ignored, including inequality checks, table look-ups, and comparisons.

Varying DCT features. Figure 4(a) shows how the total runtime varies vs. the number of DCT features used to represent the data with $k = 10$, with Python implementations. Performance follows the previously observed pattern, with two cardinal differences. First, gaps are now starker than before. Second, beyond the lowest dimensionality, the performance of Stepwise is consistently better than that of triangle-inequality-based methods. Even if Stepwise uses more features for point-to-centroid distance calculations, it has a smaller overhead of comparisons and bound updates; bound calculations involve calculating inter-centroid distances, used in both Elkan and Hamerly checks. Hamerly comes very close to Lloyd in terms of wall-clock time, as it is hard to compensate the overhead of keeping track of the nearest non-assigned centroid to each point, and nearest other centroid to each centroid. For the same reason, methods using Elkan bounds, which involve inter-centroid disances, fare worse then Stepwise. Marigold, benefiting from both forms of bounds, performs best.

Varying clusters. Figure 4(b) shows the runtime, per iteration and total, vs. the number of clusters k while using $128 \times 128 = 2^{14}$ DCT features. The results corroborate our previous observations. Moreover, the runtime of Hamerly approaches and even exceeds that of Lloyd as the number of clusters grows; that is reasonable, as more

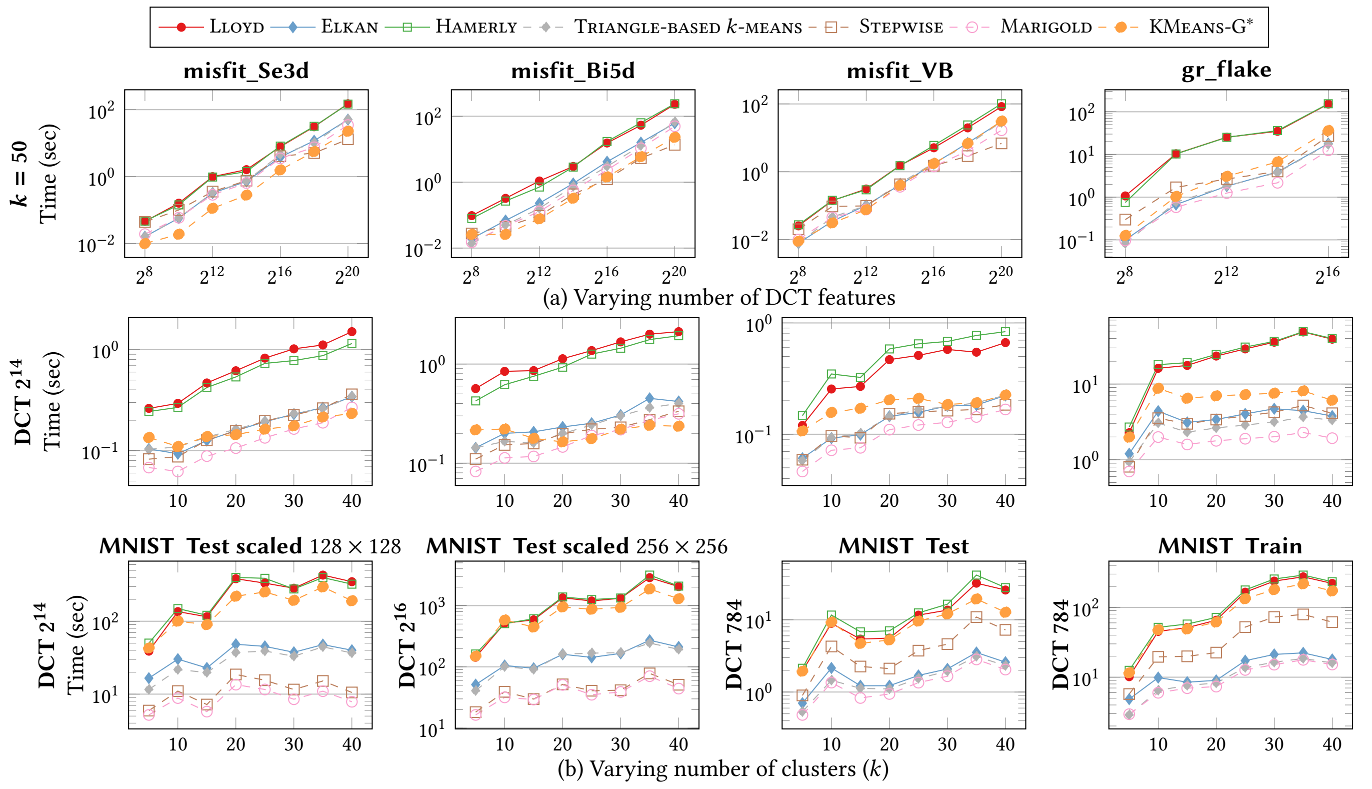


Figure 6: Total time vs. DCT features with $k = 50$ and vs. clusters k with 2^{16}, 2^{14}, or 784 DCT features, C++.

clusters imply greater bound maintenance overhead. We occasionally observe runtime non-increasing with k, as a higher k may be a better fit to the data. Still, Stepwise and Marigold steadily present an advantage over the next best-performing method. Among those two, Marigold is preferable when the ratio of data points per cluster is high, while Stepwise gains ground as that ratio falls.

DGLC data. Figure 5 presents runtime vs. the number of DCT features representing the data with $k = 10$ and vs. the number of clusters k using $128 \times 128 = 2^{14}$ DCT features on the DGLC data with Python implementations. The pattern matches that in Figure 4, while now the benefit of stepwise distance calculations in Marigold and Stepwise is accentuated as k grows.

Comparing implementations. Figure 6 shows runtime with the C++ implementation, adding two more data sets in the comparison, on two regimes: varying DCT features with $k = 50$ clusters and varying clusters with 2^{14} DCT features. Now Stepwise and Marigold do not always have an advantage over Elkan and Triangle-based k-means on lower DCT dimensions, as overhead computations are faster in C++; calculating stepwise bounds in the hope of eschewing some distance calculations is an overkill in lower dimensions. However, their advantage shines as dimensionality grows and is highlighted with increasing clusters. Remarkably, Stepwise and KMeans-G* gain ground vs. Marigold as the number of clusters grows, particularly on the misfit datasets, which comprise 168 data points each, as opposed to 1911 data points in gr_flake. KMeans-G* performs well on data sets of very few data points, such as misfit, but misses out on data sets with many data points, such as MNIST. These results indicate that geometry-based bounds are less helpful in large high-dimensional data sets. The advantage of Marigold is most pronounced on the gr_flake and MNIST data, which have more data points, reconfirming that Marigold manages high ratios of data points per cluster well.

6 CONCLUSIONS

The need to cluster high-dimensional data by k-means arises often in the natural sciences. Observing that the core k-means operation is a search for a point's nearest centroid, we brought the know-how of nearest-neighbor search in high dimensions into k-means. We devised Marigold, an efficient algorithm for high-dimensional k-means that produces the same result as Lloyd's algorithm. Marigold integrates triangle-inequality-based pruning with a powerful stepwise distance calculation method that progressively refines the data-representation granularity while maintaining tight upper and lower distance bounds to facilitate pruning centroid candidates. Our results demonstrate that Marigold reduces the volume of distance calculations and, consequently, runtime by approximately one order of magnitude, while its Stepwise component is also competitive when the data points per cluster ratio is low. Our results pave the way to real-time k-means in critical applications such as nanoARPES in condensed-matter physics.

ACKNOWLEDGMENTS

This research work was supported by VILLUM FONDEN (grant VIL40558) and partially by the Danish Council for Independent Research (grant DFF-1051-00062B). We also thank Bart Olsthoorn and Alexander Balatsky for fruitful discussions on this topic.

REFERENCES

[1] Nasir Ahmed. 1991. How I came up with the discrete cosine transform. *Digital Signal Processing* 1, 1 (1991), 4–5.

[2] Nir Ailon, Ragesh Jaiswal, and Claire Monteleoni. 2009. Streaming k-means approximation. *NeurIPS* 22 (2009).

[3] Harry C. Andrews and William K. Pratt. 1968. Fourier transform coding of images. In *Proc. Hawaii Int. Conf. System Sciences*. 677–679.

[4] Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, and Sergei Vassilvitskii. 2012. Scalable k-means++. *PVLDB* 5, 7 (2012), 622–633.

[5] Luca Becchetti, Marc Bury, Vincent Cohen-Addad, Fabrizio Grandoni, and Chris Schwiegelshohn. 2019. Oblivious dimension reduction for k-means: beyond subspaces and the Johnson-Lindenstrauss lemma. In *STOC*. 1039–1050.

[6] Shai Ben-David, Dávid Pál, and Hans Ulrich Simon. 2007. Stability of k-means clustering. In *COLT*. 20–34.

[7] Christos Boutsidis, Anastasios Zouzias, and Petros Drineas. 2010. Random projections for k-means clustering. *NeurIPS* 23 (2010).

[8] Christos Boutsidis, Anastasios Zouzias, Michael W Mahoney, and Petros Drineas. 2014. Randomized dimensionality reduction for k-means clustering. *TIT* 61, 2 (2014), 1045–1062.

[9] Andrei Broder, Lluis Garcia-Pueyo, Vanja Josifovski, Sergei Vassilvitskii, and Srihari Venkatesan. 2014. Scalable k-means by ranked retrieval. In *WSDM*. 233–242.

[10] Alla Chikina et al. 2022. One-dimensional electronic states in a natural misfit structure. *Physical Review Materials* 6, 9 (2022), L092001.

[11] Renato J. Cintra and Fábio M. Bayer. 2011. A DCT approximation for image compression. *IEEE Signal Processing Letters* 18, 10 (2011), 579–582.

[12] Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. 2015. Dimensionality reduction for k-means clustering and low rank approximation. In *STOC*. 163–172.

[13] Davide Curcio et al. 2020. Accessing the Spectral Function in a Current-Carrying Device. *Physical Review Letters* 125, 23 (2020), 236–403.

[14] Li Deng. 2012. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* 29, 6 (2012), 141–142.

[15] Kamil Dimililer. 2022. DCT-based medical image compression using machine learning. *Signal, Image and Video Processing* 16, 1 (2022), 55–62.

[16] Charles Elkan. 2003. Using the triangle inequality to accelerate k-means. In *ICML*. 147–153.

[17] Edward W. Forgy. 1965. Cluster analysis of multivariate data : efficiency versus interpretability of classifications. *Biometrics* 21, 3 (1965), 768–769.

[18] Greg Hamerly. 2010. Making k-means even faster. In *SDM*. 130–140.

[19] Philip Hofmann. 2021. Accessing the spectral function of in operando devices by angle-resolved photoemission spectroscopy. *AVS Quantum Science* 3, 2 (2021), 021101.

[20] Hassan Ismkhan and Mohammad Izadi. 2022. K-means-G*: Accelerating k-means clustering algorithm utilizing primitive geometric concepts. *Information Sciences* 618 (2022), 298–316.

[21] Ruoming Jin, Anjan Goswami, and Gagan Agrawal. 2006. Fast and exact out-of-core and distributed k-means clustering. *Knowledge and Information Systems* 10, 1 (2006), 17–40.

[22] Panagiotis Karras and Nikos Mamoulis. 2008. Hierarchical synopses with optimal error guarantees. *ACM Trans. Database Syst.* 33, 3 (2008), 18:1–18:53.

[23] Shrikant Kashyap and Panagiotis Karras. 2011. Scalable kNN search on vertically stored time series. In *KDD*. 1334–1342.

[24] Younsik Kim et al. 2021. Deep learning-based statistical noise reduction for multidimensional spectral data. *Review of Scientific Instruments* 92, 7 (2021), 073901.

[25] Juris Klonovs, Mohammad Ahsanul Haque, Volker Krüger, Kamal Nasrollahi, Karen Andersen-Ranberg, Thomas B. Moeslund, and Erika Geraldina Spaich. 2016. *Distributed Computing and Monitoring Technologies for Older Patients*. Springer.

[26] Pavel V. Kolesnichenko, Qianhui Zhang, Changxi Zheng, Michael S. Fuhrer, and Jeffrey A. Davis. 2021. Multidimensional analysis of excitonic spectra of monolayers of tungsten disulphide: toward computer-aided identification of structural and environmental perturbations of 2D materials. *Machine Learning: Science and Technology* 2, 2 (2021), 025021.

[27] Amit Kumar, Yogish Sabharwal, and Sandeep Sen. 2004. A simple linear-time $(1+\varepsilon)$-approximation algorithm for k-means clustering in any dimensions. In *FOCS*. 454–462.

[28] Ju-Hong Lee, Deok-Hwan Kim, and Chin-Wan Chung. 1999. Multi-dimensional Selectivity Estimation Using Compressed Histogram Information. In *SIGMOD*. 205–214.

[29] Sabina Leonelli. 2020. Scientific Research and Big Data. In *The Stanford Encyclopedia of Philosophy* (Summer 2020 ed.). Metaphysics Research Lab, Stanford University.

[30] Stuart P. Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* 28, 2 (1982), 129–136.

[31] Konstantin Makarychev, Yury Makarychev, and Ilya Razenshteyn. 2019. Performance of Johnson-Lindenstrauss transform for k-means and k-medians clustering. In *STOC*. 1027–1038.

[32] Norsyela Muhammad Noor Mathivanan, Nor Azura Md Ghani, and Roziah Mohd Janor. 2019. A comparative study on dimensionality reduction between principal component analysis and k-means clustering. *IJEECS* 16, 2 (2019), 752–758.

[33] Arshad M. Mehar, Kenan Matawie, and Anthony Maeder. 2013. Determining an optimal value of k in k-means clustering. In *BIBM*. 51–55.

[34] Charles N. Melton et al. 2020. K-means-driven Gaussian Process data collection for angle-resolved photoemission spectroscopy. *Machine Learning: Science and Technology* 1, 4 (2020), 045015.

[35] Dan Pelleg and Andrew Moore. 1999. Accelerating exact k-means algorithms with geometric reasoning. In *KDD*. 277–281.

[36] Han Peng et al. 2020. Super resolution convolutional neural network for feature extraction in spectroscopic data. *Review of Scientific Instruments* 91, 3 (2020).

[37] Earl W. Plummer and William Eberhardt. 1982. Angle-resolved photoemission as a tool for the study of surfaces. *Advances in Chemical Physics* 49 (1982), 533.

[38] Eli Rotenberg and Aaron Bostwick. 2014. microARPES and nanoARPES at diffraction-limited light sources: opportunities and performance gains. *Journal of Synchrotron Radiation* 21, 5 (2014), 1048.

[39] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65.

[40] David Sculley. 2010. Web-scale k-means clustering. In *TheWebConf*. 1177–1178.

[41] Zekai Sen. 2016. *Spatial Modeling Principles in Earth Sciences*. Springer, Chapter 2.8.1, 57–59.

[42] Michela Testolina and Touradj Ebrahimi. 2021. Review of subjective quality assessment methodologies and standards for compressed images evaluation. In *Applications of Digital Image Processing XLIV*, Vol. 11842. SPIE, 302–315.

[43] Robert Tibshirani, Guenther Walther, and Trevor J. Hastie. 2000. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 63 (2000).

[44] Aurora Torrente and Juan Romo. 2021. Initializing k-means clustering by bootstrap and data depth. *Journal of Classification* 38, 2 (2021), 232–256.

[45] Sergei Vassilvitskii and David Arthur. 2006. k-means++: The advantages of careful seeding. In *SODA*. 1027–1035.

[46] Sheng Wang, Yuan Sun, and Zhifeng Bao. 2020. On the Efficiency of k-Means Clustering: Evaluation, Optimization, and Algorithm Selection. *PVLDB* 14, 2 (2020), 163–175.

[47] Ian H. Witten, Eibe Frank, and Mark A. Hall. 2011. *Data Mining: Practical Machine Learning Tools and Techniques* (3rd ed.). Morgan Kaufmann Publishers Inc., USA.

[48] Yan Zhu, Jian Yu, and Caiyan Jia. 2009. Initializing k-means clustering using affinity propagation. In *HIS*, Vol. 1. 338–343.