This ICCV paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

Web-scale image clustering revisited

Yannis Avrithis[†], Yannis Kalantidis[‡], Evangelos Anagnostopoulos[†], Ioannis Z. Emiris[†]
[†]University of Athens, [‡]Yahoo! Labs

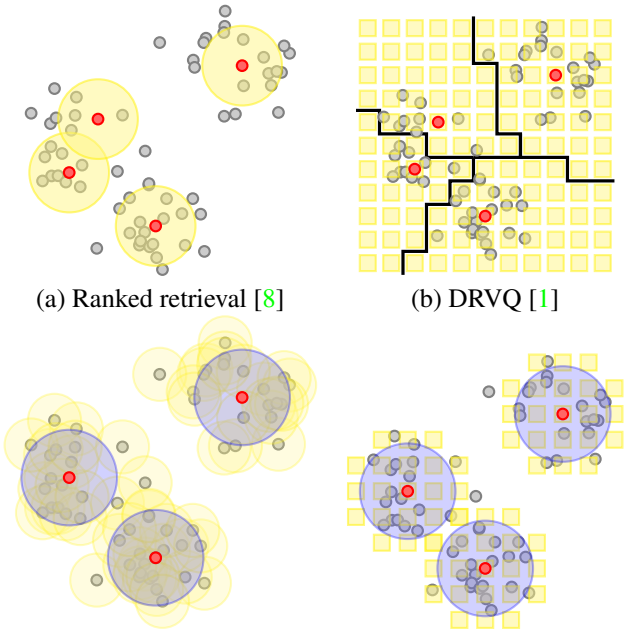
Abstract

Large scale duplicate detection, clustering and mining of documents or images has been conventionally treated with seed detection via hashing, followed by seed growing heuristics using fast search. Principled clustering methods, especially kernelized and spectral ones, have higher complexity and are difficult to scale above millions. Under the assumption of documents or images embedded in Euclidean space, we revisit recent advances in approximate k-means variants, and borrow their best ingredients to introduce a new one, inverted-quantized k-means *(IQ-means). Key underlying concepts are quantization of data points and multi-index based inverted search from centroids to cells. Its quantization is a form of hashing and analogous to seed detection, while its updates are analogous to seed growing, yet principled in the sense of distortion minimization. We further design a dynamic variant that is able to determine the number of clusters k in a single run at nearly zero additional cost. Combined with powerful deep learned representations, we achieve clustering of a 100 million image collection on a single machine in less than one hour.*

1. Introduction

NEARLY two decades ago [6], discovering duplicates among millions of web documents was the motivation behind one of the first locality sensitive hashing (LSH) schemes, later known as *MinHash* [7]. The same method was subsequently used to select seeds which, followed by efficient search and spatial verification, would lead to clustering and mining in collections of up to 10^5 images [10].

Many approaches followed, but problems have remained such as failing to discover infrequent documents, seed growing relying on heuristics, or more principled methods like medoid shift still being too costly to scale up [38]. Pairwise matching remains a problem that is inherently quadratic in the number of documents, and approximate nearest neighbor (ANN) search has been employed to help. Approximate k-means (AKM) is one such attempt [26], where each data point is assigned to the nearest centroid by ANN search. Binary k-means (BKM) [14] is another recent alternative where points and centroids are binarized and ANN search follows in Hamming space. But in this work we focus our attention on the *inverse* process.



(a) Ranked retrieval [8] (b) DRVQ [1]

(c) EGM [2] (d) This work: IQ-means

Figure 1. Different k-means variants. (●) Data points; (●) centroids; (●) search range; (●) estimated cluster extent, used to dynamically determine k.

Observing that data points remain fixed during k-means iterations, *ranked retrieval* [8] chooses to search for nearest data points using centroids as queries, as illustrated in Fig. 1a. This choice dispenses the need to rebuild an index at each iteration, and requires less queries because centroids are naturally fewer than data points. Points are examined more than once and not all points are assigned to centroids; it is observed however that distortion is not influenced much. If range queries were used, this method would be very similar to mean shift [9], except that centroid displacement is not independent here.

Dimensionality-recursive vector quantization (DRVQ) [1] relies on the same inverted centroid-to-data queries.

However, search is based on ideas extended from the inverted multi-index [4]. The entire search process for all centroids resembles a propagation on a two-dimensional grid, where each cell is visited only once and all cells are assigned to centroids, defining a discrete Voronoi diagram, as illustrated in Fig. 1b. Quantization on the grid is another form of hashing, but thanks to the orthogonal construction, it is now easy to visit cells by ascending distance.

As illustrated in Fig. 1c, *expanding Gaussian mixtures* (EGM) [2] uses a conventional data-to-centroid search, like AKM. What is special is that its probabilistic model allows an estimate of overlap between clusters and dynamic determination of the appropriate number of clusters, in contrast to Fig. 1a,b. This is a standing problem of k-means, contrary to other methods like medoid shift [31].

In this work we borrow the best ingredients from all the methods above and introduce a new k-means variant, called *inverted-quantized k-means* (IQ-means, or IQ-M), illustrated in Fig. 1d. Its unique properties are given below, summarizing our contributions:

1. We adopt the subspace quantization and multi-index search of DRVQ, yielding fast centroid-to-cell search, since actual data are in fact discarded. However, search cost is governed by the length of a single priority queue for all centroids, which is the bottleneck of DRVQ. Thus, we switch to independent queries per centroid, as in ranked retrieval. Even though cells may be visited more than once, the use of multiple independent queues offers a spectacular speed-up.
2. Although centroids are arbitrary vectors, they can still be quantized on the grid. We exploit this observation to obtain as a by-product the nearest centroids to each centroid during the same centroid-to-cell search process. We use this information to estimate pairwise cluster overlaps and purge centroids between iterations. This dynamically determines k, exactly as in EGM, and is referred to as *dynamic IQ-means* (D-IQ-M). This is unexpected since EGM assigns data points to multiple centroids as required by its probabilistic model.

Another contribution is to revisit web-scale image clustering. To scale up, we choose a global image representation, in particular state of the art deep-learned features [20], which have been recently applied to visual search [5] and even shown to outperform local feature-based representations in low dimensions [28]. We achieve clustering of a 100 million image collection in less than one hour on a single machine, while estimating the number of clusters at the same time. Given that IQ-M time complexity depends on grid size rather than number of data points, this result opens the way to truly web-scale image clustering, given enough resources. We provide our implementation online[1].

[1] http://github.com/iavr/iqm

The remaining text is organized as follows. Section 2 discusses related work, while sections 3, 4 present our algorithms IQ-M and D-IQ-M, respectively. Experiments follow in section 5 and conclusions are drawn in section 6.

2. Related work

The interest in large scale image duplicate detection, clustering and mining is now stronger than ever. Leveraging the billions of images available in community photo collections, many recent approaches utilize their metadata, *e.g.* tags or geotags for clustering [19, 27, 11, 3]. Kennedy *et al.* [19] were among the first to extract textual and geographical patterns from metadata and use them in visual clustering. Most methods follow a two-stage approach, first by geographic location and then by visual similarity. However, clustering within each geographic cell remains quadratic: even if fast retrieval is employed, a query per image is still necessary. Other approaches find iconic images [22, 38, 32], *e.g.* for scene summarization [32] or 3D reconstruction [22]. With few exceptions [3], most methods focus on landmarks and points of interest [11, 23].

Following earlier work on discovering object categories from image segmentations [29], Chum and Matas [10] introduce an approach for web-scale clustering using visual information alone. Starting from local features, they use minHash collisions to find seed images which they grow via retrieval and expansion. Although faster than most of the aforementioned approaches due to hashing, it is limited by the memory cost of local descriptor indexing as well as the pairwise nature of geometric matching. It also focuses on popular scenes. Although our quantization is another form of hashing, we rather use a principled way of updating seeds as in k-means. Besides, our multi-index approximate search strategy is arguably the fastest possible, once images are represented in a Euclidean space.

Most aforementioned approaches use local features and descriptors, that hold the state-of-the-art in image retrieval but incur significant space and time cost [34]. We choose to sacrifice their matching quality for scalability and represent each image by a single global representation. Although several methods exist for aggregating local descriptors [17], we rather choose deep learned features, which are shown to be superior in low dimensions [5],[28]. We only assume a Euclidean space representation, so our method is generic and can apply well beyond image clustering.

Our approach is an approximation of k-means. Density based approaches like mean shift [9] and medoid shift [31] are typically inefficient for large datasets. Quick shift [35] is a fast approximate variant, but still requires nearest neighbor search with each point as a query. Medoid-based methods as well as kernelized methods like kernel k-means [30] and spectral clustering methods [36] are more generic in applying to non-Euclidean spaces but they are largely imprac-

tical above a few million points. There are several methods for estimating k even dynamically like component annihilation [12], DP-Means [21] and EGM [2]; here we choose to integrate our approach with EGM, which comes as a natural extension at nearly zero cost.

Parallelism has widely been utilized for large scale clustering [24, 15], while algorithms for distributed systems exist for many popular clustering algorithms like Parallel k-means [39] or parallel DB-SCAN [25]. We are however interested in large scale clustering without the need of a distributed grid. We show that we are able to provide an efficient k-means approximation that can cluster 100M images in less than an hour on a single machine, while a distributed implementation of standard k-means on the same dataset using 300 machines on the grid takes over one day.

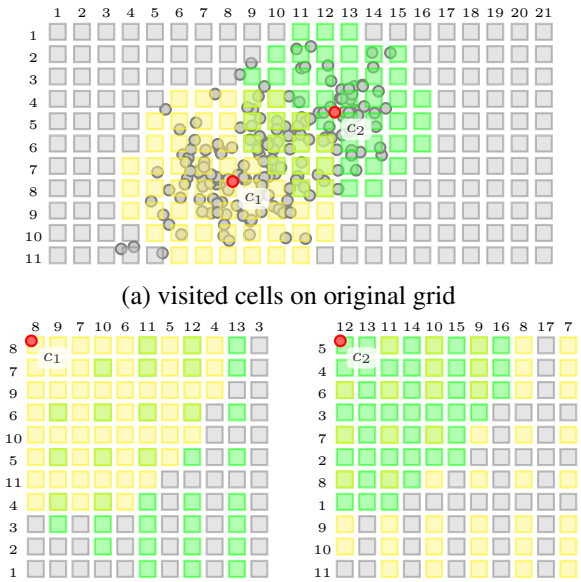
3. Inverted-quantized k-means (IQ-means)

Representation. We are given a dataset X of n points in $\mathbb{R}^d$, and the problem is to find k cluster centroids minimizing distortion as in k-means. IQ-M assumes the same representation and codebook building as in multi-indexing [4]. In particular, assuming d is even, $\mathbb{R}^d$ is expressed as the Cartesian product of two orthogonal subspaces, $S^1 \times S^2$, of $d/2$ dimensions each. Although this decomposition is subject to optimization [13], which we do apply in our experiments, we assume here the simplest decomposition whereby each vector x is written as a tuple (x^1, x^2) consisting of two subvectors $x^1, x^2 \in \mathbb{R}^{d/2}$.

We also assume there are two sub-codebooks U^1, U^2 trained independently on projections of sample data on S^1, S^2 respectively. Each U^ℓ contains s sub-codewords, partitioning S^ℓ into s disjoint subsets for $\ell = 1, 2$. Then, codebook $U = U^1 \times U^2$ contains $s \times s$ codewords and partitions $\mathbb{R}^d$ into $s \times s$ cells. We thus refer to each codeword $u \in U$ as a *cell*, while U can be seen as a discrete two dimensional *grid*. Given sub-codewords $u_i^1 \in U^1, u_j^2 \in U^2$ with $i, j \in [s] = \{1, \dots, s\}$, we represent cell $(u_i^1, u_j^2) \in U$ by the multi-index notation u_α with α being the integer tuple $(i, j) \in I = [s] \times [s]$. Every point x can be quantized to a cell $q(x) = (q^1(x^1), q^2(x^2))$, where $q^\ell(x^\ell) = \arg\min_{u^\ell \in U^\ell} \|x^\ell - u^\ell\|$ is the nearest sub-codeword of U^ℓ to projection x^ℓ of x on subspace S^ℓ for $\ell = 1, 2$.

Update step. Next, similarly to DRVQ [1], all points of X are quantized on the grid and a discrete two-dimensional distribution p of points over cells is constructed. In particular, for each cell u_α, probability $p_\alpha = |X_\alpha|/n$ measures the empirical frequency of points falling into u_α, where $X_\alpha = \{x \in X : q(x) = u_\alpha\}$. Further, the mean $\mu_\alpha = \frac{1}{|X_\alpha|} \sum_{x \in X_\alpha} x$ of all points in X_α is kept for each cell u_α. At this point, dataset X may be discarded. An arbitrary initial set C of k centroids is assumed.

As in all k-means variants, the algorithm then alternates



(a) visited cells on original grid

(b) search block of c_1 (c) search block of c_2

Figure 2. Centroid-to-cell search during assignment, for two centroids c_1, c_2. (●) Data points; (●) centroids; (□) cells V_1 visited by c_1; (□) cells V_2 visited by c_2; (□) other cells.

between an assignment and an update step, where the latter is simply given by weighted average

$$c_m \leftarrow \frac{1}{P_m} \sum_{\alpha \in A_m} p_\alpha \mu_\alpha, \tag{1}$$

for all centroids $c_m \in C$. Here, $P_m = \sum_{\alpha \in A_m} p_\alpha$ is the proportion of points assigned to centroid c_m and $A_m = \{\alpha \in I : a(u_\alpha) = m\}$ contains the indices of all cells assigned to c_m during the assignment step, where

$$a(u) = \arg\min_{c_m \in C} \|u - c_m\| \tag{2}$$

is the index to the nearest centroid c to cell $u \in U$. In other words, cells c_α with their sample mean μ_α and probability p_α have completely replaced the original data. Still, assignment (2) of cells to centroids is the bottleneck.

Assignment step. Here is where fast search is required. Although assignment rule (2) implies a cell-to-centroid search, we follow the inverse process as explained in section 1. As in ranked retrieval [8], this process takes the form of a set of individual queries for nearest cells, one for each centroid. Search follows a multi-indexing approach in this work, in particular using the multi-sequence algorithm [4].

Fig. 2a illustrates part of the grid, with two centroids c_1, c_2 and the set of nearest cells to each centroid, say V_1, V_2, in different colors. Recall that rows and columns of the grid correspond to sub-codewords in U^1, U^2. These

Algorithm 1: Centroid-to-cell multi-sequence search

```
function SEARCH(C, U, f)
    for u_α ∈ U do
        a[α] ← 0; dist[α] ← ∞;                          ▷ cell assignments
        visit[α] ← FALSE; cen[α] ← 0                    ▷ cen: section 4
    for c_m ∈ C do
        (k^1, d^1) ← NN_w(c_m^1, U^1)                   ▷ nearest..
        (k^2, d^2) ← NN_w(c_m^2, U^2)                   ▷ ..sub-codewords
        f.INIT(m, (k_1^1, k_1^2)); V ← ∅                ▷ initialize list
        Q.INIT(); Q.PUSH((0, 0), 0)                     ▷ initialize queue
        while ¬Q.EMPTY() do
            ((i, j), d) ← Q.EXTRACT-MIN()
            α ← (k_i^1, k_j^2)                          ▷ current cell index
            V ← V ∪ α; visit[α] ← TRUE                  ▷ record cell
            if f(m, α, d) then break                    ▷ terminate?
            if i = 0 ∨ visit[k_{i-1}^1, k_{j+1}^2] then
                Q.PUSH((i, j + 1), d_i^1 + d_{j+1}^2)   ▷ right
            if j = 0 ∨ visit[k_{i+1}^1, k_{j-1}^2] then
                Q.PUSH((i + 1, j), d_{i+1}^1 + d_j^2)   ▷ below
        for α ∈ V do visit[α] ← FALSE                   ▷ reset
    return a                                            ▷ cell assignments
function f.INIT(m, α)
    n ← 0                                               ▷ number of points visited
function f(m, α, d)
    if d < dist[α] then a[α] ← m; dist[α] ← d           ▷ re-assign
    n ← n + |X_α|; return n ≥ T                         ▷ target reached?
```

are just shown by their indices in Fig. 2a. Due to the independent search processes, a number of cells, shown in color overlay, belong to both V_1, V_2 and will be visited twice, triggering a comparison to determine which of c_1, c_2 is nearest. To understand the search process, Fig. 2b,c illustrate what search looks like with c_1, c_2 as queries respectively.

For each query c_i, the w nearest sub-codewords are found in U^1, U^2, and ordered by ascending distance to c_i, for $i = 1, 2$. A $w \times w$ *search block* is thus determined for c_i. For $w = 11$, the two 11×11 search blocks of c_1, c_2 are shown in Fig. 2b,c, illustrating row/column selection and ordering. Row/column numbers refer to the numbers of Fig. 2a, but are re-arranged such that centroid c_i and its nearest cells appear on the top-left corner of the block. For instance, top-left cells $(8, 8)$ and $(5, 12)$ of the two blocks are indeed where c_1, c_2 are placed on the grid of Fig. 2a. Observe however that due to re-arrangement, the nearest cells to c_2 are no longer contiguous in the block of c_1 and vice versa. They rather appear interlaced, and in higher dimensions they would appear randomly shuffled.

Search. The search process is outlined in Algorithm 1. For each centroid c, the w nearest sub-codewords are given by a list of ascending (squared) distances d^ℓ and indices k^ℓ for $\ell = 1, 2$, specifying a search block. Nearest cells in the block are visited by ascending (squared) distance d to c using a priority queue Q, as in the multi-sequence algorithm [4]: a cell to the right is visited if the one above right is visited, and a cell below is visited if the one below left is visited. There are substantial differences, though.

First, a function f determines the action to be taken at each visited cell. Alternative functions are discussed in section 4, but here f merely updates the current assignment a and lowest distance *dist* found for each cell u_α. Second, f also controls search termination. Alternatives are again discussed in section 4, but here f counts the total number of underlying points in visited cells, and terminates when this reaches a target number T. Finally, property *visit* is global over the entire grid, indirectly accessed via indices k^ℓ and reset after each block is searched, with the help of an additional list V of visited cells. This implies that space $w \times w$ and its initialization is no longer necessary [4]; the algorithm is linear in the number of visited cells.

Algorithm 2: Centroid-to-centroid search function f

```
function f.INIT(m, α)
    cen[α] ← m                                          ▷ centroid per cell
    N_m ← ∅                                             ▷ (neighbors,distances) of centroid c_m
    n ← 0                                               ▷ number of points visited
function f(m, α, d)
    if d < dist[α] then a[α] ← m; dist[α] ← d           ▷ re-assign
    if cen[α] ≠ 0 then N_m ← N_m ∪ (cen[α], d)
    n ← n + |X_α|; return n ≥ T
```

4. Dynamic IQ-means

While IQ-means searches from centroids to cells at each iteration, its dynamic version also searches from centroids to centroids, and keeps track of the nearest neighboring centroids of each centroid, while both queries and indexed points are constantly updated. Similarly to EGM [2], it then uses this neighborhood information to compute cluster overlaps and purge clusters between iterations in an attempt to automatically determine k.

Search. The most interesting aspect of this centroid-to-centroid search process is that it relies on the same indexing structure; in fact, even though centroids are constantly updated, it is a mere by-product of centroid-to-cell search, so it comes at negligible cost. All that is needed is to keep some additional information per cell and change the definition of function f in Algorithm 1. The key observation is that although centroids are arbitrary vectors, they can still be quantized on the grid, just like data points.

The additional property *cen* holds up to one centroid index per cell and is initialized to zero by Algorithm 1. As shown in Algorithm 2, each centroid c_m is subsequently

quantized to cell u_α just before search and its index m is recorded in $cen[\alpha]$. This operation comes at no cost, since the w nearest sub-codewords to each centroid are readily available from NN_w in Algorithm 1 and we can just take the first ones, $\alpha = (k_1^1, k_1^2)$.

A list N_m of nearest centroid indices and distances is also maintained for each centroid c_m, and is emptied just before search. Then, for each cell α visited, a nonzero $cen[\alpha]$ means that another centroid is found and is inserted in N_m along with distance d. List N_m can be constrained to hold up to a fixed number of neighbors; no particular ordering is needed because cells, hence neighboring centroids, are always found by ascending distance to c_m.

Purging. Once neighboring centroids are found, cluster overlaps may be estimated. Following EGM [2], we model the distribution of points assigned to cluster c_m by an isotropic normal density $\mathcal{N}(x|c_m, \sigma_m)$, where σ_m is simply the standard deviation of points assigned to cluster m, estimated only from cell information by

$$\sigma_m^2 \leftarrow \frac{1}{P_m} \sum_{\alpha \in A_m} p_\alpha \|\mu_\alpha - c_m\|^2. \tag{3}$$

Then, the same purging algorithm as in EGM applies, roughly iterating over all clusters m in descending order of population P_m, and purging clusters that overlap too much with the collection of all clusters that have been kept so far. Given the normal cluster densities, pairwise overlaps are computed in closed form at the cost of one vector operation per pair. This algorithm is quadratic in k.

5. Experiments

In this section we evaluate the proposed approaches on large scale clustering and compare against relevant state-of-the-art methods. We first present the datasets and features used, as well as implementation details and evaluation protocol. We then report results on three publicly available datasets, including a dataset of 100 million images.

5.1. Experimental setup

Datasets. We experiment on three publicly available datasets. *SIFT1M* [16] consists of 1M 128-dimensional SIFT vectors, and a learning set of 100K vectors. *Paris* [37] contains 500K images from Flickr and Panoramio, crawled by geographic bounding box query around Paris city center. The ground truth consists of 79 landmark clusters covering 94K dataset images. *Yahoo Flickr Creative Commons 100M (YFCC100M)* [33] contains a subset of 100 million public Flickr images with a creative commons license.

Features and codebooks. For Paris and YFCC, we use convolutional neural network (CNN) features to globally represent images. In particular, we use the AlexNet architecture [20] as a pre-trained model provided by Caffe deep learning framework [18]. We use the output of the last fully connected layer (fc7) as a 4096-dimensional feature vector for each image. By learning a covariance matrix from the entire dataset, we further reduce to 128 dimensions, which not only speeds up the search process, but also does not harm performance [5]. For IQ-means, we permute the dimensions to balance the variance between the two subspaces before multi-indexing [13]. For IQ-means on SIFT1M, we use the separate learning set for off-line learning of the sub-codebooks, while on Paris and YFCC we use a 10M-vector random subset of YFCC.

Compared methods. For the smaller SIFT1M and Paris datasets we compare the proposed IQ-means (IQ-M) and dynamic IQ-means (dynamic IQ-M or D-IQ-M) methods against the fastest approaches from the related work that can also scale to large datasets: Ranked Retrieval (RR) [8] and Approximate k-means (AKM) [26]. DRVQ [1] was found to be faster than these methods but of significantly lower quality, so it is not included in the comparison. Binary k-means (BKM) [14] is only slightly faster than AKM, so it is also not included. As all methods are approximations of k-means, we further report the upper bounds given by k-means. For the large YFCC100M dataset, no related method can run on a single machine due to space and time requirements[2]. As a baseline, we apply k-means on the non-empty multi-index cell centroid vectors, which is referred to as *cell-k-means* or CKM. This can be seen as an approximation of IQ-means, where although actual points are discarded as in IQ-means, cells are not weighted. Given all 100M vectors as input, we also compare to a distributed implementation of k-means, referred to as DKM, on 300 machines on the grid using Spark[3]. Again, this experiment provides an upper bound on performance.

Implementation. We implement the offline learning process and clustering interface in Matlab, using the Yael library[4] for exact nearest neighbor search, assignment and k-means clustering. Subspace search of centroids to sub-codewords is also using Yael, while all remaining IQ-means iteration, as outlined in Algorithm 1, is implemented in C++, interfaced through a single MEX call. For any other method that requires ANN search, *i.e.* ranked retrieval [8] (RR) and Approximate k-means (AKM) [26] we use the FLANN library[5]. Observe that RR's own search algorithm WAND is particularly targeted to documents and does not apply to Euclidean spaces. Unless otherwise stated, all experiments are performed on a single machine.

[2]The 128-dimensional visual feature vectors alone require 52GB of space. One could of course use *e.g.* PQ-encoding yielding also fast search, but again this would just be an alternative to our implementation of RR.

[3]http://spark.apache.org/

[4]https://gforge.inria.fr/projects/yael/

[5]http://www.cs.ubc.ca/research/flann/

Evaluation protocol. We report *clustering time* (total or per iteration) and *average distortion* on SIFT1M and Paris with varying number of centroids k and data points n. Time does not include off-line learning of sub-codebooks for IQ-means; unless otherwise stated, total clustering time does include encoding as explained in Table 1. Average distortion is the squared Euclidean distance of each point to the nearest centroid, averaged over the dataset. Given the ground truth labels of Paris, we also adopt the measures of *precision* (or purity) and *recall* [37]. YFCC100M has no associated ground truth, so in order to report more than just clustering time, we also present precision on a public set of noisy labels extracted through image classification [33]. We measure the *average precision* over all clusters, where precision is defined as the percentage of the most popular class in the cluster, *i.e.* the class present in the cluster most times. In all algorithms, centroids C are initialized as k random vectors from the dataset X. We run each experiment five times and report mean measurements.

5.2. Results

Tuning. We first evaluate the effect of the main parameters of IQ-means on its performance, as measured by average distortion and running time. These are the sub-codebook size or grid size s, which determines how fine the space partition is, the size w of the search block and the search target T; the latter two determine the accuracy of search from centroids to cells. The finer the grid is, the higher the quality of data representation, but the more cells need to be visited; and the more accurate search is, the longer it takes. For convenience, we set $T = (n/k)t$ where t is a normalized target parameter with respect to the average cluster population under uniform distribution.

Table 1 presents results on SIFT1M for varying s and t, which confirm our expectations. It appears that $s = 512$ and $t = 5$ are reasonable trade-offs. We choose those settings for the remaining experiments on SIFT1M and Paris, which are of comparable size. On the other hand, we choose $s = 8$K for the larger YFCC, so that the total number of cells $s^2 = 64$M is comparable to $n = 100$M. We set the search block size $w = 16$ on SIFT1M and Paris, and $w = 512$ on YFCC. Increasing w further would only make search slower without improving distortion. This is particularly important considering that sub-codeword search is the most time-consuming part of Algorithm 1.

To evaluate dynamic IQ-means, Fig. 3 shows how the final estimated number of clusters k' after termination depends on the original one k. While k' is nearly linear in k for IQ-means—some clusters are still lost due to quantization—there is a saturation effect with increasing value of overlap threshold τ that controls purging [2]. It is thus possible, given an unknown dataset, to begin clustering with an overestimation of k and let the algorithm purge clusters as needed; subsequent iterations then become increasingly faster. As a fair trade-off, we choose $\tau = 0.6$ in subsequent experiments.

	s (for $t = 5$)				t (for $s = 512$)		
	128	256	512	1024	1	2	5
encode (s)	4.570	8.380	16.44	33.70	16.44	16.44	16.44
search (s)	3.153	4.366	7.760	12.78	6.418	7.557	7.760
distortion	4.816	4.545	4.403	4.343	4.425	4.412	4.403

Table 1. Encode/search times (sec) and average distortion ($\times 10^4$) for 20 iterations on SIFT1M for $k = 10^4$ and varying values of grid size s and normalized search target t. Encoding includes quantization of points on the grid and the inversion process to compute cell population p_α and means μ_α (1).

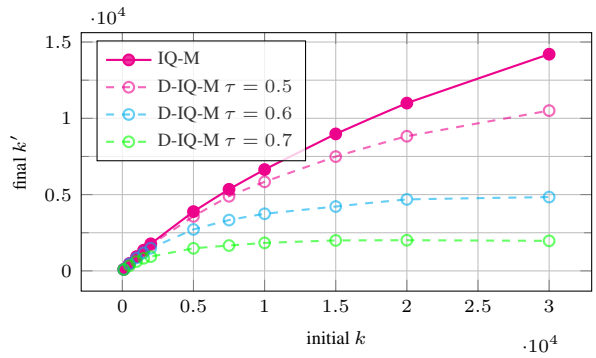
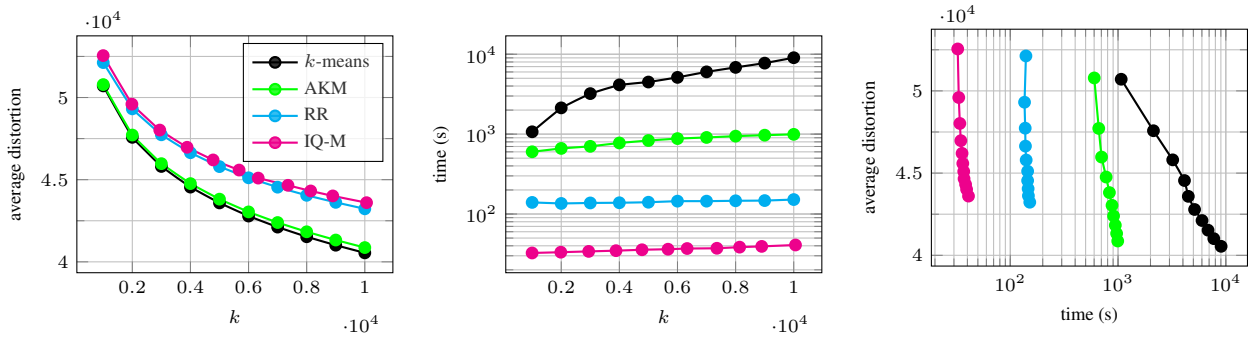


Figure 3. Final k' versus initial k number of centroids on SIFT1M for varying overlap threshold τ.

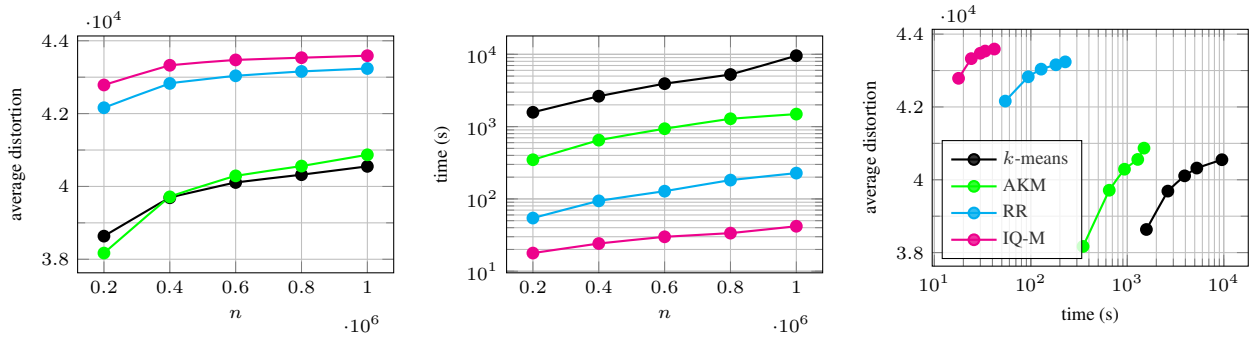
Comparisons. We then evaluate the performance of IQ-means against competing methods under varying number of clusters k. Fig. 4a,4b measure average distortion and running time for k up to 10^4 on SIFT1M. The quality of k-means and AKM is close, indicating that most points are correctly assigned to centroids; the quality of RR and IQ-means is also close, indicating the loss of accuracy due to unassigned points. There is also a clear ordering of running times: AKM is faster than k-means by approximate search, RR is even faster by inverted search, and IQ-means is the fastest by more efficient inverted search. Fig. 4c shows distortion versus time: the more the approximation the higher the distortion, but at spectacular gain in speed.

Fig. 5 shows a different experiment for SIFT1M. We now fix k and vary n by clustering subsets of increasing size from the original dataset. Otherwise distortion and time measurements remain the same as in Fig. 4. Now both distortion and time are increasing with n for all methods. Again, as shown in Fig. 5a, distortion is similar for k-means and AKM, higher for RR and slightly higher for IQ-means. Observe that in IQ-means data points are quantized on a fixed grid and the algorithm operates on cell distributions alone, regardless of n. The increase of time with n in Fig. 5b is in fact only due to the encoding of data points, which is linear in n. The gain in speed varies up to more than two



(a) average distortion *vs.* k (b) time *vs.* k (c) average distortion *vs.* time

Figure 4. Average distortion and total time for 20 iterations on SIFT1M for varying number of clusters k. Time for IQ-means includes encoding of data points that is constant in k, but not codebook learning, which is performed on a different dataset.



(a) average distortion *vs.* n (b) time *vs.* n (c) average distortion *vs.* time

Figure 5. Average distortion and total time for 20 iterations on SIFT1M for $k = 10^4$ and varying number of data points n. Time for IQ-means includes encoding of data points that is linear in n, but not codebook learning.

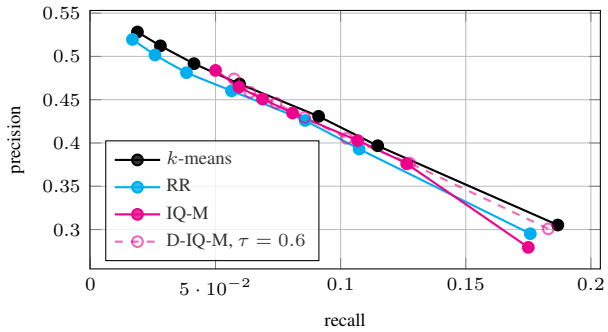


Figure 6. Precision *vs.* recall for varying k on Paris.

	CKM	DKM	D-IQ-M
k/k'	100000	100000	85742
time (s)	13068.1	7920.0	**140.6**
precision	0.474	**0.616**	0.550

Table 2. Time per iteration and average precision for cell-k-means, dynamic IQ-means and distributed k-means on YFCC100M with initial $k = 10^5$. For DKM, we use Spark on 300 machines.

	IQ-M			D-IQ-M		
k/k'	100K	150K	200K	86K	120K	152K
time (s)	212.6	271.1	325.8	140.6	249.6	277.2

Table 3. Time per iteration and k/k' for IQ-means and dynamic IQ-means on YFCC100M.

(nearly three) orders of magnitude compared to k-means, while the loss in distortion is reasonable.

Given the existing ground truth of Paris dataset, Fig. 6 further evaluates IQ-means and dynamic IQ-means against other methods on a precision-recall diagram. Due to quantization, it appears that our methods do not reach the upper-left extreme of high precision and low recall, which can be improved with a finer grid at higher cost. Otherwise, all methods are comparable regardless of their cost. Of course, none of these methods is anywhere near in performance to more expensive dedicated methods like iconoid shift [38].

One could just take into account that we are using a global feature reduced to 128 dimensions per image. What is important is that in terms of classification, all approximations are equivalent to k-means in practice.

Large scale experiments. To demonstrate the scalability of IQ-means, we perform clustering on the YFCC100M dataset. We fix the grid size to $s = 8192$, leaving 13M non-empty cells for the 100 million vectors. Following the tuning experiments, we use overlap threshold $\tau = 0.6$ for dynamic IQ-means and end up with $k = 85742$ after 20

Figure 7. Mining example: subsets of similar clusters for (a) Paris and (b) Paris+YFCC100M. Images in red outline are from the Paris ground truth.

iterations.

We report timings and average precision in Table 2 for $k = 10^5$ clusters. We observe that the proposed approach is two orders of magnitude faster than CKM. Both approaches discard the initial data points but dynamic IQ-means further weights each cell with its point statistics and gives more consistent clusters in terms of label precision. Distributed k-means on 300 machines takes 2.2 hours per iteration on average, *i.e.* one order of magnitude slower than dynamic IQ-means. In terms of precision, D-IQ-M performs better than CKM and while the upper bound of DKM is high, the latter requires far more time and resources. In Table 3 we further present timings under varying k/k' for IQ-means and dynamic IQ-means.

To visualize the dynamic IQ-means result we show in Fig. 7 a subset of a sample Paris cluster when clustering is performed either on the Paris dataset alone or Paris along with the entire YFCC100M dataset. We handpick the cluster to depict approximately the same images from one of the annotated landmarks of the Paris dataset. There are 2511 and 7382 images respectively in this cluster. Annotated ground truth images are depicted in a red outline, while the rest is a random sample in Fig. 7a and the images closest to the annotated samples in Fig. 7b.

6. Discussion

By quantizing data points on a grid of two subspaces and applying inverted search from centroids to cells using multi-indexing, we have achieved an extremely fast variant of k-means that can be used in any application where input data lie on a Euclidean space. We have also achieved dynamic estimation of the number of clusters in a single run at nearly zero cost. Data points are extremely compressed (*e.g.* 26 bits per point on YFCC100M), which is a significant space improvement compared to all known methods. By using global deep learned image representation, we have applied this method to clustering 10^8 images on a single machine in less than one hour. Although the result cannot be compared to dedicated more costly mining methods, it is shown to be on par with other k-means variants that are orders of magnitude slower. In fact, the assignment step, conventionally the bottleneck of k-means, turns out to be faster than the update step in IQ-means, which leaves small margin for further improvement.

Acknowledgements. Y. Avrithis and I. Z. Emiris are partially supported by the European Social Fund and Greek National Fund through Research Funding Program ARISTEIA, project "ESPRESSO" (70/3/11893). We thank Clayton Mellina and the Flickr Vision Team for their help with CNN features and the distributed k-means experiment.

References

[1] Y. Avrithis. Quantize and conquer: A dimensionality-recursive solution to clustering, vector quantization, and image retrieval. In *ICCV*. 2013. 1, 3, 5

[2] Y. Avrithis and Y. Kalantidis. Approximate Gaussian mixtures for large scale vocabularies. In *ECCV*. 2012. 1, 2, 3, 4,

5, 6

[3] Y. Avrithis, Y. Kalantidis, G. Tolias, and E. Spyrou. Retrieving landmark and non-landmark images from community photo collections. In *ACM Multimedia*, 2010. 2

[4] A. Babenko and V. Lempitsky. The inverted multi-index. In *CVPR*, 2012. 2, 3, 4

[5] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky. Neural codes for image retrieval. In *ECCV*, 2014. 2, 5

[6] A. Broder. On the resemblance and containment of documents. In *Proceedings of the Compression and Complexity of Sequences*, page 21, 1997. 1

[7] A. Broder, M. Charikar, A. Frieze, and M. Mitzenmacher. Min-wise independent permutations. In *ACM Symposium on Theory of Computing*, pages 327–336, 1998. 1

[8] A. Broder, L. Garcia-Pueyo, V. Josifovski, S. Vassilvitskii, and S. Venkatesan. Scalable k-means by ranked retrieval. In *Web Search and Data Mining*, 2014. 1, 3, 5

[9] Y. Cheng. Mean shift, mode seeking, and clustering. *PAMI*, 17(8):790–799, 1995. 1, 2

[10] O. Chum and J. Matas. Large-scale discovery of spatially related images. *PAMI*, 32(2):371–377, Feb 2010. 1, 2

[11] D. Crandall, L. Backstrom, D. Huttenlocher, and J. Kleinberg. Mapping the world's photos. In *WWW*, 2009. 2

[12] M. Figueiredo and A. Jain. Unsupervised learning of finite mixture models. *PAMI*, 24(3):381–396, 2002. 3

[13] T. Ge, K. He, Q. Ke, and J. Sun. Optimized product quantization for approximate nearest neighbor search. In *CVPR*, 2013. 3, 5

[14] Y. Gong, M. Pawlowski, F. Yang, L. Brandy, L. Boundev, and R. Fergus. Web scale photo hash clustering on a single machine. In *CVPR*, 2015. 1, 5

[15] K. Heath, N. Gelfand, M. Ovsjanikov, M. Aanjaneya, and L. J. Guibas. Image webs: Computing and exploiting connectivity in image collections. In *CVPR*, 2010. 3

[16] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *PAMI*, 33(1), 2011. 5

[17] H. Jégou and A. Zisserman. Triangulation embedding and democratic aggregation for image search. In *CVPR*, 2014. 2

[18] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 5

[19] L. Kennedy, M. Naaman, S. Ahern, R. Nair, and T. Rattenbury. How flickr helps us make sense of the world: Context and content in community-contributed media collections. In *ACM Multimedia*, volume 3, pages 631–640, 2007. 2

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*. 2012. 2, 5

[21] B. Kulis and M. I. Jordan. Revisiting k-means: New algorithms via bayesian nonparametrics. In *ICML*, 2012. 3

[22] X. Li, C. Wu, C. Zach, S. Lazebnik, and J.-M. Frahm. Modeling and recognition of landmark image collections using iconic scene graphs. In *ECCV*, pages 427–440, 2008. 2

[23] Y. Li, D. J. Crandall, and D. P. Huttenlocher. Landmark classification in large-scale image collections. In *ICCV*, 2009. 2

[24] T. Liu, C. Rosenberg, and H. Rowley. Clustering billions of images with large scale nearest neighbor search. In *WACV*, 2007. 3

[25] M. A. Patwary, D. Palsetia, A. Agrawal, W.-k. Liao, F. Manne, and A. Choudhary. A new scalable parallel DBSCAN algorithm using the disjoint-set data structure. In *SC*, 2012. 3

[26] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, 2007. 1, 5

[27] T. Quack, B. Leibe, and L. Van Gool. World-scale mining of objects and events from community photo collections. In *CIVR*, pages 47–56, 2008. 2

[28] A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson. Visual instance retrieval with deep convolutional networks. Technical report, 2014. 2

[29] B. Russell, A. Efros, J. Sivic, W. Freeman, and A. Zisserman. Using multiple segmentations to discover objects and their extent in image collections. In *CVPR*, 2006. 2

[30] B. Schölkopf, A. Smola, and K. Muller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998. 2

[31] Y. A. Sheikh, E. A. Khan, and T. Kanade. Mode-seeking by medoidshifts. In *ICCV*, 2007. 2

[32] I. Simon, N. Snavely, and S. Seitz. Scene summarization for online image collections. In *ICCV*, 2007. 2

[33] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. YFCC100M: The new data in multimedia research. *Communications of the ACM*, to appear. 5, 6

[34] G. Tolias, Y. Avrithis, and H. Jégou. To aggregate or not to aggregate: Selective match kernels for image search. In *ICCV*, 2013. 2

[35] A. Vedaldi and S. Soatto. Quick shift and kernel methods for mode seeking. In *ECCV*, 2008. 2

[36] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416, 2007. 2

[37] T. Weyand, J. Hosang, and B. Leibe. An evaluation of two automatic landmark building discovery algorithms for city reconstruction. In *RMLE*. 2010. 5, 6

[38] T. Weyand and B. Leibe. Discovering favorite views of popular places with iconoid shift. In *ICCV*, 2011. 1, 2, 7

[39] W. Zhao, H. Ma, and Q. He. Parallel k-means clustering based on MapReduce. In *Cloud Computing*, pages 674–679. 2009. 3