Noname manuscript No.
(will be inserted by the editor)

Balancing Search Space Partitions by Sparse Coding for Distributed Redundant Media Indexing and Retrieval

André Mourão · João Magalhães

Received: date / Accepted: date

Abstract Effective partitioning multimedia indexes is key for efficient kNN search. But existing algorithms are based on document similarity, without partition size or redundancy constraints. Our goal is to create an index partitioning algorithm that addresses the specific properties of a distributed system: load balancing across nodes, redundancy on node failure and efficient node usage under concurrent querying. We propose the representation of data with overcomplete codebooks. Each document is quantized into a small set of codewords and indexed on per-codeword partitions. Quantization algorithms are designed to fit data as best as possible, leading to a bias towards codewords that fit the principal directions of data in the original space. In this paper, we propose the balanced KSVD (B-KSVD) algorithm: it distributes data uniformly across codewords, according to the distribution in the original space.

The comprehensive experiments focused on measuring the effectiveness of partition size balancing and retrieval quality. Results show that B-KSVD better balances partition sizes (i.e., lower std. deviation on partition size distribution), compared to k-means and KSVD baselines. B-KSVD achieves 38% *1-recall* by inspecting only 1% of the full index, distributed over 10 partitions. k-means creates partitions with higher size variation and requires either larger codebooks or the inspection of larger portions of the index to achieve similar retrieval performance.

A Mourão and J Magalhães
NOVA LINCS
Faculdade de Ciências e Tecnologia
Universidade NOVA de Lisboa
Tel.: +351 212948300
Fax: +351 212954461
E-mail: a.mourao@campus.fct.unl.pt, jmag@fct.unl.pt

Keywords Search space partitioning · High-dimensional indexing · Dictionary design · Distributed search · Approximate nearest neighbor search

1 Introduction

There are two main index partitioning strategies [10]: (i) **horizontal** partitioning, or **sharding**, divides documents across nodes; and (ii) **vertical** or **term-based** partitioning, divides document features across multiple nodes.

Similarity-based partitioning simplifies the selection of retrieval resources, as documents relevant to a query are concentrated across a few shards [24]. In addition to dealing with high-dimensional document descriptors and its unknown structure, partitioning algorithms can take advantage of the distributed system properties, such as parallel processing, redundancy (i.e., index documents on more than one node) and ability to deploy additional nodes on demand (e.g., cloud-based systems).

Existing multimedia document distribution algorithms do not explore these properties; document allocation policies are either random, e.g., [31], or based on existing partitions on single node algorithms, e.g., [7]. Figure 1 illustrates the document-to-partitions and query-to-partitions assignment process for existing index partitioning techniques and the proposed technique. Colored partitions show which partitions were selected for inspection and the "contribution to the candidate set" bars are the expected contribution to the nearest neighbor candidate set. Figure 1 (a) shows a single assignment technique, where each document is assigned to a single partition (e.g. [7]). Figure 1 (b) shows a random assignment technique, where documents are assigned to partitions by node load, and queries are assigned to

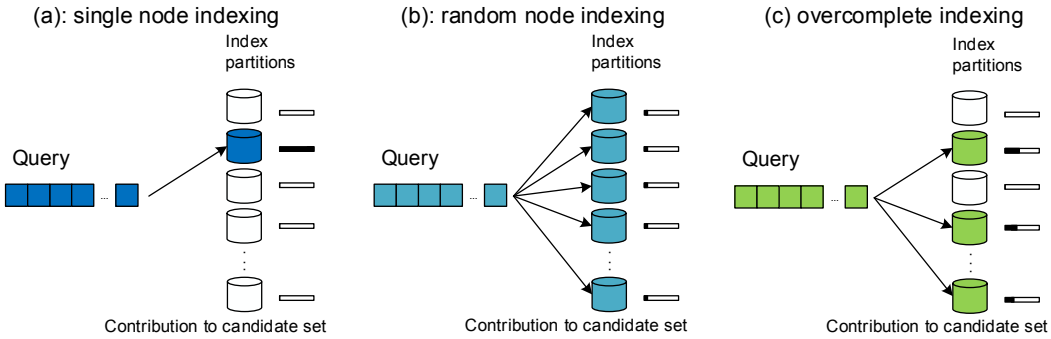

Fig. 1 Examples of the querying process of multiple document partitioning policies: (a) single node indexing, (b) random indexing, (c) redundant overcomplete indexing

all partitions. This technique is applied by some Map-Reduce systems (e.g., [31]). Figure 1 (c) shows the desired behavior of our proposed technique: select a small subset of partitions, each with a different, complementary set of nearest neighbors. This is a special type of partitioning: it distributes documents across partitions (similar to horizontal partitioning), but each document is placed on more than one partition, according to their similarity (similar to term-based vertical partitioning on text indexes).

Efficient nearest neighbor search requires effective, similarity-based search space partitioning techniques. But most similarity-based partitioning techniques inherently generate unevenly sized partitions. On single nodes, this is not a big problem, as the computational cost of accessing partitions is uniform. This unbalance becomes a problem when partitions are distributed across multiple nodes: *how to balance between load balancing and similarity-based indexing?*

Distributed systems increase the probability of nodes and network failures. Overcomplete partitioning enables redundant indexing where each document is indexed on multiple partitions, each with a different set of neighbors. This contrasts with traditional distributed systems redundancy which is achieved by node duplication.

To sum up, the goal of this paper is to study the impact of overcomplete data representations on the partition size balancing and retrieval performance of distributed indexes. Codebooks for overcomplete data representations are composed of a large number of codewords, each one corresponding to a partition of the search space. Indexing is achieved by encoding each media descriptor as a linear combination of just a few codewords.

The main contribution of this paper is the balanced KSVD algorithm (B-KSVD) which balances the allocation of data across an adjustable number of codewords. It was designed to address the following challenges:

- even distribution of data across codewords to achieve better load-balancing when allocating data to nodes;
- documents should be assigned to partitions with documents that are also close in the original space;
- redundant indexing: as document vectors are encoded with multiple codewords, each one corresponding to a space partition, data can be stored redundantly across multiple nodes.

The proposed approach offers several advantages. An overcomplete balanced index means that concurrent queries will be answered by different subsets of nodes, reducing the bottleneck of having all nodes answering all queries. (e.g. Figure 1 (b)). Furthermore, these properties mean that distributed indexes can still operate with adequate performance when a node fails. In other words, failure to inspect a partition (e.g., as a consequence of a node failure) will result in a slight performance decrease, instead of no results returned.

This paper is organized as follows: Section 2 details the related work in distributed search and space partitioning. Section 3 formalizes overcomplete redundant partitioning and details the proposed solutions. Section 4 describes the experiments and sections 5 and 6 contain the discussion and conclusions, respectively.

2 Related work

One of the works that goes towards our partitioning goals is by Ji et al [21]. They tested *horizontal* and *vertical* index partitioning, based on a Vocabulary Tree model quantization. Their work showed that vertical partitioning offers the best temporal performance on a distributed setting, without an increase in load imbalance. The bulk of distributed multimodal retrieval comes from combining Map Reduce [14] with single node algorithms or from distributing the feature spaces across nodes [30]. When applied to multimedia indexing [41, 29, 31], Map Reduce can be used to partition

indexes horizontally. As Map Reduce requires Map and Reduce nodes to be data agnostic, indexes must either: a) query all nodes for all queries [31], which does not meet our efficiency goal, or b) have all nodes accessing the full index [29], which is limited by the time it takes to fetch the relevant index subset. Moise et al [29] experiments also show that the overhead behind the Map and Reduce operations is considerable (e.g. copying data to Hadoop Distributed File Systems), as it is only optimized for massive batches of queries. Distributed tree-based systems have also been studied for horizontal index partitioning for multimedia indexing [7, 2], but the effectiveness of sub-tree based index partitioning is reduced when the dimensionality of the vectors to index increases [39], meaning that more nodes need to be queried.

Effective partitioning of the search space is a key part of approximate nearest neighbour algorithms. It enables faster search by inspecting the subset of the index where there is an higher density of nearest neighbours. Recent algorithms in this area rely on Hamming embeddings or on codebooks learned from data.

Hamming embeddings. Binary hash techniques such as LSH (Locality Sensitive Hashing) [3], partition the search space in a data independent way, according to a set of randomly generated hyperplanes. Each binary hash bit represents an hyperplane in the original feature space that divides it in two, assigning a value of zero or one, according to the document position. Binary hashes are generated by concatenating the output of multiple hyperplane functions. The search space is partitioned horizontally, according to the document hash: documents with similar hash codes have a high-probability of being similar, so they are stored on the same buckets. LSH spawn multiple techniques [13, 12, 34] that explore different families of hyperplane partitioning functions (e.g. grid).

Other works have focused on creating better hash functions, by exploring the structure of the data in the original space. Leveraged by the distribution of documents in the original search space, data dependent hash functions can create better partitions for similarity search [17, 38, 40, 26]. Grauman and Fergus [16] provide a comprehensive review of data dependent hash techniques.

Regression and codebook design. Sparse hashes are generated in a very high dimensional space. Documents with more non-zero coefficients on the same hash positions have higher degrees of similarity than documents with no common non-zero coefficients. Effective partitioning is achieved by having only a very small subset of non-zero hash coefficients. Lewicki and Sejnowski [25] show that the transformation of dense feature representations into a sparse high-dimensional representations achieves a high degree of compression, while preserving locality structure on the non-null coefficients.

In the area of sparse coding, multiple techniques were developed to generate high dimensional sparse hashes. These techniques differ on the type of regulation applied to the hashes: l_0 penalty (e.g. OMP (Orthogonal Matching Pursuit) [33]), l_1 penalty (e.g. Lasso [37]), l_2 penalty with coefficient thresholding (e.g. Ridge [18]) or a combination of the l_1 and l_2 penalties (e.g. Elasticnet [42]). OMP controls sparsity by greedily selecting the most correlated coefficient at each iteration with the current residual (l_0 pseudo-norm penalty). Lasso does sparse selection by applying the l_1 penalty, Elasticnet's penalty is a mixture of l_1 penalties with l_2 penalties, having both the sparsity properties of l_1 penalty and the limited coefficient magnitude of the l_2 penalty.

Regression techniques use a codebook/dictionary as the basis of the transformation into the new space. Codebook computation algorithms such as KSVD [1], estimate the codewords that minimize the reconstruction error. Stochastic gradient descent techniques (e.g. [32]) update each example per iteration, to minimize reconstruction error. Cherian et al [11] presented an index based on hashes created using l_1 regression and the Newton Descent for codebook learning. Borges et al [9] presented an index based on sparse hashes created using l_0 regression and a codebook learned through KSVD. However, none of these works address the index sharding problem.

Quantization through clustering. Clustering techniques are one of the most used space partitioning techniques, with applications on image indexing and retrieval [28, 19]. The search space is partitioned by generating a set of centroids; vectors are assigned to the closest centroid according to a metric (e.g. euclidean distance). k-means, a popular clustering technique, aims at finding the set of centroids that minimizes sum of squares within-cluster distances. Lloyd [27] proposed a local search solution that is still widely applied today. On the original formulations, the initial seed centroids are selected randomly from the training data, which may greatly increase the convergence time. k-means++ [4] is a centroid selection technique that estimates a good set of seed centroids, by analyzing the distribution of the seed centroids and the training data distribution. Fuzzy c-means clustering/soft clustering [8] techniques extends the assignment of documents to multiple clusters, by keeping track of document-to-cluster membership information. Clustering techniques such as DBSCAN [15], do not set the number of cen-

troids as a parameter, focusing instead on cluster density and points per cluster.

Clustering techniques are behind some of the best performing nearest neighbour search algorithms. Jégou et al [19] proposed IVFADC, an index that divides the space into a set of Voronoi cells through k-means based vector quantization. Further works improve candidate distance computation [20] and descriptor quantization [22]. Tavenard et al [35] proposed a technique for balancing k-means cluster size, by shifting cluster boundaries into parallel boundaries. Their experiments showed less variability in the number of candidates retrieved per query. Babenko and Lempitsky [5] created Inverted Multi-Index (IMI), which generalizes IVFADC by using product codebooks for individual cell construction. [6] relaxed orthogonality constraints of IMI (Non-Orthogonal IMI), to better fit data distribution. Their experiments show that such partitioning still results on a large number of empty cells (60%-80% for IMI and 40%-50% for NO-IMI). The large number of empty partitions of NO-IMI shows how existing algorithms are still far from our goal of well balanced index partitions, even after the relaxation of orthogonality constraints.

Consistent distribution. Consistent assignment of requests (e.g. documents or queries) to nodes (i.e. same request will always be assigned to the same node) as been studied for load balancing on static content servers or caches, [23, 36]. Consistent hashing [23] is based on the creation of a distribution space, based on numeric angle value: nodes and assigned random, uniformly distributed angle values; documents are assigned consistent angle values (similar requests always have the same values), designed to be uniformly distributed across angle value space. Document to node assignment is based on the distance between document and node value. Rendezvous or Highest Random Weight hashing [36] is based on generating node indexing priorities based on hashing the concatenation of the document and the node identifiers. As these techniques are designed for static asset caching, they are optimized for exact hash matches (e.g. SHA256, MD5 hash functions) and do not give similarity guarantees on approximate nearest neighbour search. Load-balancing is also dependent on the quality of the distribution of the hashes generated by the selected hash function.

3 Space Partitioning Codebooks

Current research on the distribution of multimedia indexes is focused on similarity-based partitioning or pure load balancing. Balancing partition sizes while preserving similarity may appear contradictory: if the data on a given space is not uniformly distributed, *how can we guarantee a fair partitioning of space in both the densely and sparsely populated regions?*

We propose incorporating partition size balancing and redundancy at codebook level. An index composed of balanced, redundant partitions enables a flexible distributed retrieval process. The sequential inspection of multiple partitions leads to incremental retrieval performance increases. Conversely, not inspecting a partition (e.g., node failure), should result in a small retrieval performance loss, instead of no results returned. Another important factor is how does a distributed retrieval system deals with parallel query streams. On use-cases with more query streams, having more uniformly sized partitions is beneficial: queries will be distributed across more partitions, and uniform partition sizes guarantee that the expected partition inspection time is more uniform. Thus, partitioning methods should have the following properties:

- generate **partitions** that group **documents** that are similar in the original space;
- generate **evenly sized** partitions.
- setting a fixed codeword **over completeness**, i.e. sparsity factor;
- compute partition **membership magnitude** (e.g. distance to centroid, reconstruction weight) to allow candidate selection inside partitions;

Formally, consider the original vector $y \in \mathbb{R}^n$, a sparse vector $x \in \mathbb{R}^k$ and a sparsity coefficient s. We represent a set of m, n-dimensional vectors in the original space as $Y \in \mathbb{R}^{m,n}$ and the corresponding, k-dimensional sparse vectors as $X \in \mathbb{R}^{m,k}$ Our goal is to find a function f such that:

$$f(y) = x, \text{where } \|x\|_0 = s \text{ and } s \ll n \ll k. \tag{1}$$

$\|...\|_0$ is the l_0 pseudo-norm: the sparse vector x must have exactly s non-zero values/coefficients. Forcing sparsity to be equal to the sparsity factor s, instead of the general constraint of smaller or equal, ensures that each document will be placed exactly on s partitions. For a set of vectors $y_a, y_b, y_c \in \mathbb{R}^n$ and corresponding sparse vectors $f(y_a) = x_a, f(y_b) = x_b, f(y_c) = x_c \in \mathbb{R}^k$, our goal is to generate sparse vectors with the following property:

$$\begin{aligned} &\text{if } \|y_a - y_b\|_2 < \|y_a - y_c\|_2 \text{ then} \\ &\quad \|x_a + x_b\|_0 < \|x_a + x_c\|_0 \end{aligned} \tag{2}$$

In the above expression, Equation 2, vectors that are close in the original space have more non-zero coefficients on similar positions in the sparse vector space than vectors that are farther apart. Sparse vectors are the basis to generate a set of partitions P, $p \in P \subset Y$.

Each sparse vector position corresponds to a partition, and the value at that vector position quantifies the "membership magnitude" to the partition. Our balancing goal is to minimize the differences in partition sizes.

After studying the properties of the space partitioning in the literature, we arrive at two families of methods that have the potential to meet the desired properties: sparse coding and clustering. Sparse coding techniques are designed to generate overcomplete representations of the search space: our reasoning is that codebook codewords can act as the basis of the partitions. For clustering techniques, centroids and distance to centroids act as codebook and codewords respectively, using soft clustering for redundant partitioning. The following sections detail how we applied these families of methods.

3.1 Codebooks by Sparse Coding

Sparse vectors can be high dimensional sparse hashes, generated using a codebook representative of the original space. Sparse hashes offer some advantages over binary hashes for search space partitioning: sparse coding techniques are designed to be overcomplete, real-valued membership (i.e., representative values on the non-null dimensions of the sparse hash) and control over the sparsity of the solution and thus, redundancy. Another advantage is that these techniques work on non-uniform or unbalanced feature spaces; codebooks are learned using feature-specific training data, meaning that they will always follow document distribution. The effectiveness of distribution is not limited to feature orthogonality: KSVD will index document on redundant non-orthogonal partitions, maintaining similarity-based indexing guarantees. The sparse hash generation steps are:

- compute the dictionary/codebook D from training data;
- use D to create an hash with s non-zero coefficients and assign them to the corresponding partitions;
- for search, inspect the s partitions corresponding to non-zero coefficients.

The process for the generation of sparse hashes that follow Equation 1 goals, is to solve the following optimization problem:

$$\arg\min_x \|Dx - y\|_2, \text{ subject to } \|x\|_0 = s, \tag{3}$$

where $D \in \mathbb{R}^{n \times k}$ is a dictionary, learned from the data, $y \in \mathbb{R}^n$ is the the original vector, $x \in \mathbb{R}^k$ is the sparse hash and s is the sparsity coefficient. Note that we set the sparsity constraint x to be equal to s, instead of the usual smaller or equal requirement. This was to ensure that generated hashes respect our redundancy goals (fixed number of partitions per document). Equation 3 generates a hash with the desired properties, using a previously computed dictionary. Techniques for dictionary computation include K-SVD [1] and Stochastic Gradient Descent techniques.

We chose to combine OMP [33] with a K-SVD based codebook for two reasons. In our previous work [9], we found that OMP offers a good trade-off between reconstruction error and hash computation time when compared to other l_0 or l_1 based regularization techniques. The second reason is its greedy nature: on each iteration, OMP chooses the codeword that better minimizes the reconstruction error of the hash computed in the previous iteration. It does not change the coefficients from the previous iteration. OMP's greediness is tied with the retrieval process, where candidate selection starts at the partition with the highest reconstruction coefficient. OMP guarantees that choosing more codewords for reconstruction does not alter the previously computed coefficients, which is a desirable property for a system that can dynamically change how many partitions to inspect.

3.1.1 KSVD and OMP

Equation 3 shows how to generate a sparse hash x for a vector y, based on an (existing) dictionary D. Thus, one must first generate the dictionary D that adequately represents documents in the original space. Computing the dictionary requires solving the following optimization problem: find the dictionary D and set of sparse hashes X that minimizes the reconstruction error against a set of vectors Y:

$$\begin{aligned} &\arg\min_{D,X} \|DX - Y\|_2, \\ &\text{subject to } \|x\|_0 = s, \text{for } x \in X \end{aligned} \tag{4}$$

where $Y \in \mathbb{R}^{m \times n}$ are the original document vectors, $D \in \mathbb{R}^{n \times k}$ is a dictionary, to be learned from the data, $X \in \mathbb{R}^{k \times m}$ are the sparse hashes (one per column), x is a sparse hash vector (column of X) and s is the sparsity coefficient.

Solving for both D and X is NP-hard. KSVD alternatively optimizes the solution for D and X. KSVD updates each dictionary codeword iteratively (represented by i), while fixing other codewords $j_{[0,k]} \neq i$. By decomposing Equation 4 into KSVD iterative process, we arrive at the following formulation, for the iteration where

the codeword i is fixed:

$$\begin{aligned} &\arg\min_{D_i,(x_i)_I} \|D_i(x_i)_I + (E_i)_I - Y\|_F^2 \\ &E_i = \sum_{j_{[0,k]} \neq i} \|D_j x_j - Y\|_F^2 \end{aligned} \tag{5}$$

where $\|...\|_F$ is the Frobenius norm, and E_i is the reconstruction error for the (fixed) codewords, $j_{[0,k]} \neq i$. Sparsity is enforced by using only the dimensions with non-zero coefficients: I is the set of all index with non-zero coefficients that use atom i for reconstruction.

By fixing j codewords, the value for codeword D_i can be computed by finding a rank-1 matrix approximation of E_i, $\hat{E}_i$, and factorizing the result into D_i and x_i.

$$\hat{E}_i = G \sum^1 V^T \tag{6}$$

This decomposition will yield D_i as the first column of G and x_i as the first column of $V \times \sum^1$.

3.1.2 Balanced KSVD

KSVD enforces the creation of sparse representations that group similar vectors in the original space on codewords with non-zero coefficients. When generating multiple sparse hashes, KSVD will inherently create unbalanced representations, as the dictionary codewords are biased towards the principal directions of the data on the original space. Babenko and Lempitsky [6] already showed how relaxing orthogonality constraints reduces the number of non-empty partitions. Our goal is to take it further, and minimize the standard deviation σ in the distribution of documents across partitions:

$$\begin{aligned} &\arg\min_{D,X} \|DX - Y\|_2 + \sigma(\|X^T\|_0), \\ &\quad \text{subject to } \|x\|_0 = s, \text{for } x \in X \end{aligned} \tag{7}$$

The transposed matrix X^T combined with the l_0 pseudo-norm as $\|X^T\|_0$, represents a vector with k elements, containing the number of documents per codeword. It contrasts with the sparsity constraint $\|x\|_0 \leq s, \text{for } x \in X$, which represents the number of codewords per document. Thus, $\sigma(\|X^T\|_0)$ is the standard deviation in the number of documents per partition.

To achieve this goal, B-KSVD reduces the magnitude of dictionary codewords that have more documents assigned to. Its alternate optimization process is similar to Equation 5; balancing is applied to Equation 6's E decomposition; after the rank-1 approximation, we multiply the G matrix by the penalty factor B:

$$\begin{aligned} &\text{balanced } \hat{E}_i = B \times G \sum^1 V^T \\ &B = \frac{1}{\left(\|X_{-1}^T\|_0 + r\right)^e} \end{aligned} \tag{8}$$

where X_{-1} represent the hashes computed using the previous iteration of the dictionary. Therefore, $\|X_{-1}^T\|_0$ is a k-dimensional vector, containing the number of documents assigned to partitions on the previous iteration. This formulation matches the number of documents per codeword formulation of $\|X^T\|_0$, stated on Equation 7. e is the parameter to control the magnitude of the penalty and r is a regularization factor to avoid division by zero for partitions with zero documents. This penalty distorts the estimation of the dictionary codewords, creating non-orthogonal balanced representations. The regularization parameters r and s control the magnitude of this distortion, balancing between similarity-based indexing and balanced partitions.

3.1.3 Random dictionary

We can also measure the impact of dictionary learning on the computation of sparse hashes, by using a random dictionary. OMP will compute sparse hashes using random codewords, generated from the Gaussian distribution with zero mean and unit standard deviation.

$$D \in \mathbb{R}^{n \times k} \subset \mathcal{N}(0, 1) \tag{9}$$

Random dictionaries show how OMP clusters data without prior search space information from dictionary computation.

3.2 Codebooks by Soft Quantization

An alternative interpretation of the sparse vector computation process follows soft clustering, where the cluster membership is controlled by a fixed s sparsity factor. Its focus is to measure how well these clusters can represent neighboring data in a balanced way, and how using multiple clusters affects the sparse vector computation process in a high dimensional feature space. The clustering process is the following:

- find the centroids
- project the documents to the closest s clusters
- search in the matching cluster posting lists

Consider a set of cluster centroids $C \in \mathbb{R}^{n \times k}$. Our clustering process finds the set of closest centroids $c \in \mathbb{R}^{n \times s} \subset C$, and assigns the Euclidean distances to those centroids as the sparse vector values:

$$\arg\min_c(\|c_i - y\|_2), \text{ for } c_{i \in [0,k]} \in C, \text{ with } |c| = s$$

$$x_{i \in [0,k]} = \begin{cases} \|c_i - y\|_2, \text{ for } c_i \in c \\ 0, \text{ otherwise} \end{cases} \quad (10)$$

To find the set of centroids that best represent the feature space, we have selected three techniques, random sampling, k-means and fuzzy c-means. Alternative clustering techniques such as DBSCAN do not allow setting the number of clusters and thus, do not meet our desired properties.

Fuzzy c-means clustering [8] techniques extend the assignment of documents to clusters and keep membership information to multiple clusters (e.g., ratio of the distance to the centroids). As with k-means, these techniques minimize the sum of document to centroid distances, taking into account membership and cluster fuzziness information (degree of overlap between clusters). In our preliminary experiments, fuzzy c-means produced very unbalanced clusters: all documents were assigned to only 20 clusters, regardless of the total generated centroids (512, 1024, 2048, 4096, 8192). Due to this extreme unbalance, we did not pursue further experiments using fuzzy c-means.

3.2.1 k-means centroids

k-means is one the most widely applied clustering functions in nearest neighbor search. It estimates a set of centroids $C \in \mathbb{R}^{n \times k}$ that minimizes the distances between the points to the centroids of their clusters. The k-means clustering process minimizes the following expression:

$$\arg\min_C \sum_{c_i \in C} \sum_{y_j \in P_i} \|c_i - y_j\|_2 \quad (11)$$

where $C \in \mathbb{R}^{n \times k}$ is the set of cluster centroids, $P_i, i \in [0, k]$ is the set of documents $y_j \in P$ that are assigned to centroid C_i. The k-means initialization requires the selection of a set of points as the initial centroids. We selected k-means++ [4] centroid initialization, as it selects points that give a good representation of the search space and lead to faster convergence, on a large set of experiments and datasets.

3.2.2 Random centroids

We tested a random sampling technique that selects a random set of points C from the training data Y:

$$C \in \mathbb{R}^{n \times k} \subset Y \quad (12)$$

The sampling process makes no assumptions regarding distribution. We expect that the algorithm will select more points in denser regions of the training data space. As with the random dictionary with OMP regression, this technique is a baseline to measure the impact of centroid selection for the creation of evenly balanced partitions.

4 Experiments

We've described how to create over-complete codebooks that generate sparse, high dimensional hashes. To measure how well the proposed methods meet the stated partitioning and retrieval goals, we evaluated them from three perspectives:

- **Balanced partitioning**: measure how the tested methods manage to balance the size of the partitions;
- **Inter-partition retrieval**: measure the cumulative impact of searching on more than one partition;
- **Intra-partition retrieval**: measure how well the partitions capture the original space nearest neighbours;

Dataset: We tested the index partitioning methods on the Billion Vectors dataset [19, 20]. It contains 1 million descriptors from two feature types: GIST (960 dimensions) and SIFT features (128 dimensions). The datasets were split into a training, validation and test subsets[1]. We extracted 1000 queries per feature type from the test set. Having two types of features allow us to measure the partitioning impact of feature type on the partitioning process.

Metrics: In addition to load balancing quality metrics, which are the number of documents per partition p and standard deviation σ of partition size versus the mean, we evaluated the following retrieval quality metrics, averaged over 1000 queries:

- 1-recall@r: average rate of queries for which the 1-nearest neighbor was returned. r changes with the number of candidates inspected.
- %kNN: average percentage of true k nearest neighbors retrieved.

[1] `http://corpus-texmex.irisa.fr/`

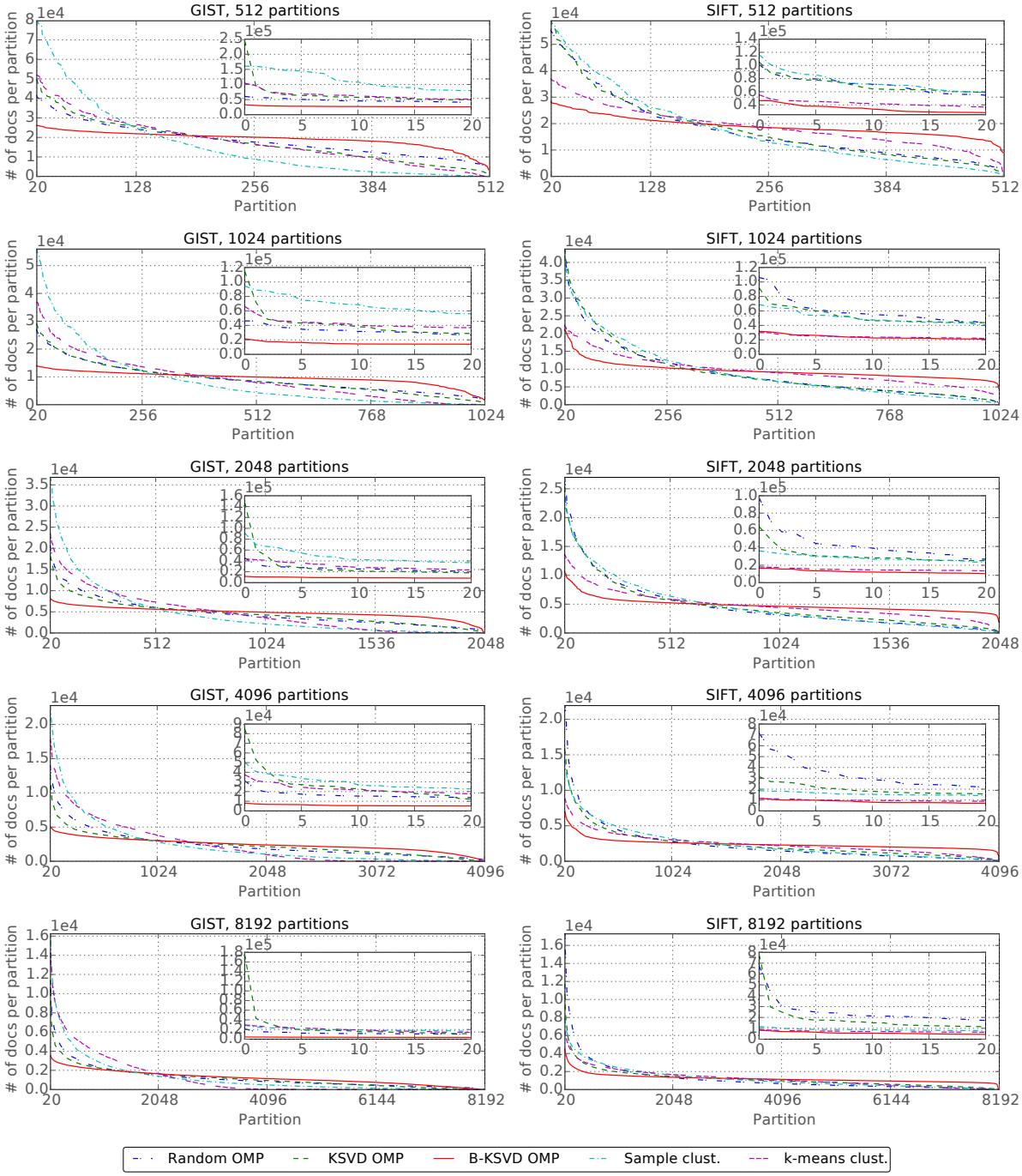

Fig. 2 Sorted partition size distribution for multiple feature types and number of partitions. The smaller, inner chart shows the top 20 partition sizes, and the larger chart shows the remaining partition sizes.

Parameters: Based on preliminary experiments, we found that setting the exponent of the penalty to $c = 2$ and regularization factor to $r = 0.001$ offered the best trade-off between similarity and even balancing. We set the sparsity coefficient to $s = 10$ for all algorithms and varied the codebook size k and thus, the number of partitions (512, 1024, 2048, 4096, 8192).

4.1 Balanced partitioning

The goal of this experiment is to measure how the selected techniques distribute documents across partitions, for multiple numbers of partitions and feature types. Documents were assigned to the partitions with corresponding non-zero codewords/centroids, for each

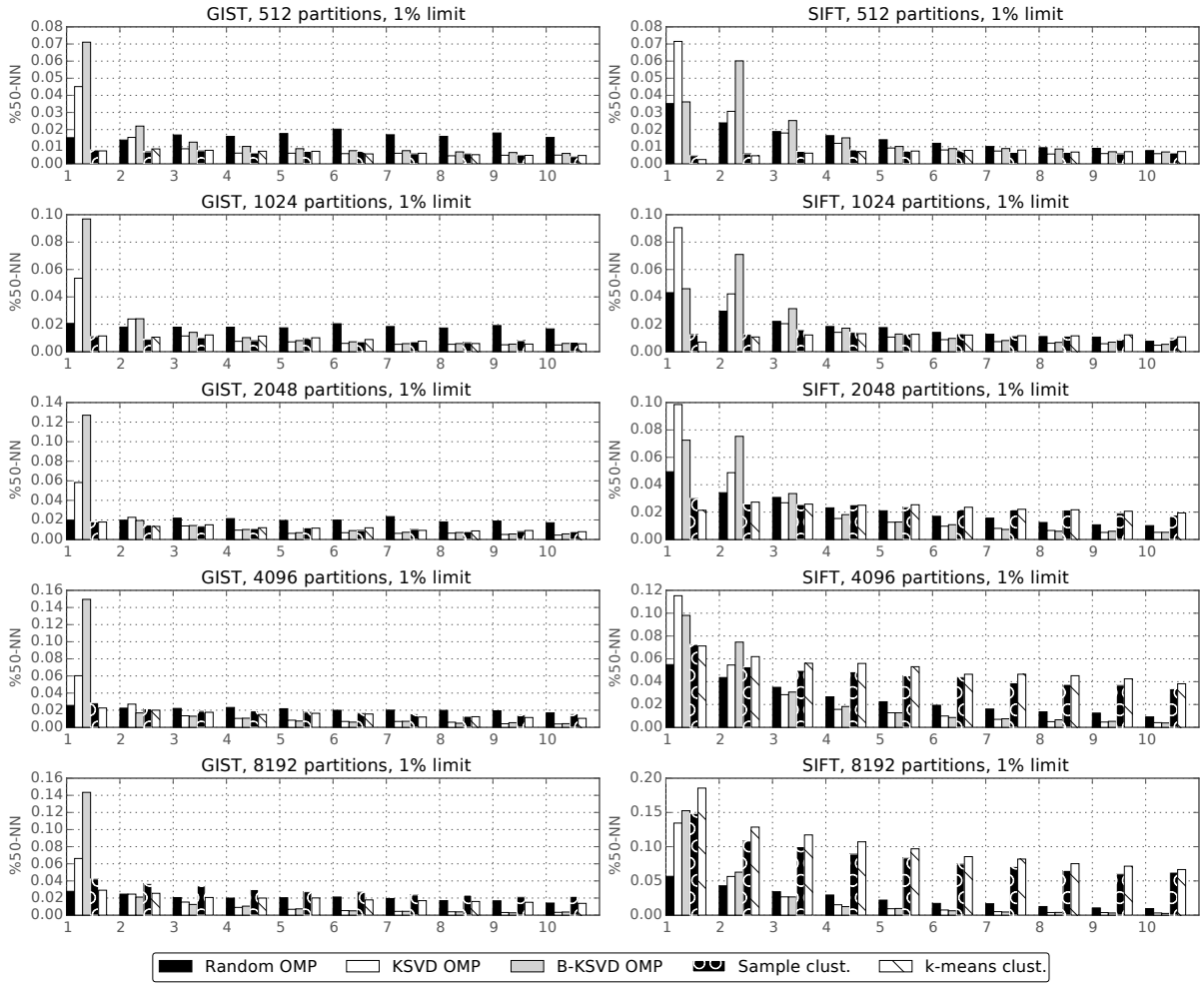

Fig. 3 Inter-node partition: %50-NN of individual partitions, for multiple feature types and number of partitions

partition method, feature type and the number of partitions.

Figure 2 shows the behaviour of the partitioning algorithms for the GIST and SIFT features (left and right side charts respectively) and the number of partitions (different rows). For readability, each chart is divided into two parts: the smaller chart shows the occupation of the top 20 partitions, where the variation in scale of the number of documents is higher. The larger chart shows the variation for the remaining partitions (20 to k). The X-axis represents the partitions, sorted in descending order of the number of indexed documents (i.e., partitions with more documents are to the left). The Y-axis represents the number of documents on that partition. Note that, as the goal is to show the relative differences between partitioning methods, the Y-axis scale is different across charts. It is also important to note that the sum of the sizes of the partitions is the same for all partitioning methods (index size $m \times s$).

Table 1 shows the detailed standard deviation (σ), larger partition (Max), and median (Med) partition size ($k/2$). KSVD learns a dictionary with the principal directions of the data in the original space. Combined with OMP greedy codeword selection, KSVD sparse representations are highly biased towards principal directions, which is clear on the top 20 charts. B-KSVD managed to counteract KSVD's greediness and generated the most balanced solutions (σ columns on Table 1). This effect is clearer at the partitions with the larger and smaller partitions: on the top 20 positions, B-KSVD is less affected than KSVD, by the most popular directions of the data; the occupation of the partition at median value is also consistently closer to the expected value (Mean) than other methods, meaning the decrease in the number of documents is much slower and gradual than the other retrieval methods tested. B-KSVD is also the most stable solution, offering the best balancing properties for all partition sizes and feature types.

Table 1 Partition balancing results: Max is the size of the largest partition (bold values: lowest is best), Med is the size of the median partitions, partition size / 2 (bold values: closest to mean is best), and σ is the standard deviation of the partition sizes (bold values: lowest is best). The Mean value is the same for all methods for a set partition size, as all methods produce solutions with similar sparsity s. Mean, Max, Med and σ values are on base 10^3.

Partitions:	512			1024			2048			4096			8192		
Mean:	**19.5**			**9.8**			**4.9**			**2.4**			**1.2**		
GIST															
Algorithm	Max	Med	σ	Max	Med	σ	Max	Med	σ	Max	Med	σ	Max	Med	σ
Random OMP	60.4	17.6	9.7	46.2	8.2	6.0	45.6	3.8	3.9	31.4	1.8	2.2	21.2	0.8	1.3
KSVD OMP	245.9	16.7	16.4	115.1	8.4	7.6	148.0	4.3	4.8	85.1	2.1	2.6	176.2	0.9	2.4
B-KSVD OMP	**33.0**	**20.1**	**4.2**	**21.5**	**9.9**	**2.4**	**11.5**	**4.9**	**1.3**	**8.1**	**2.4**	**0.9**	**4.5**	**1.1**	**0.7**
Sample clust.	160.9	8.9	27.0	94.3	4.3	13.7	91.4	2.1	7.6	50.2	1.0	3.9	28.8	0.5	2.1
k-means clust.	104.4	16.2	14.9	66.2	8.0	8.9	43.8	3.5	5.3	37.6	1.0	3.4	29.6	0.0	2.3
SIFT															
Algorithm	Max	Med	σ	Max	Med	σ	Max	Med	σ	Max	Med	σ	Max	Med	σ
Random OMP	101.4	13.7	15.9	106.0	6.7	10.3	97.6	3.1	6.0	71.5	1.4	3.6	67.6	0.7	2.1
KSVD OMP	105.4	14.9	16.0	91.9	6.5	9.9	64.5	3.5	4.8	31.6	1.8	2.4	78.2	1.0	1.5
B-KSVD OMP	**46.4**	**18.5**	**4.8**	**31.8**	**9.1**	**3.0**	**16.5**	**4.6**	**1.4**	11.8	**2.3**	**0.8**	**8.3**	**1.1**	**0.5**
Sample clust.	116.6	13.0	18.2	68.8	6.5	9.7	36.6	3.3	4.8	18.7	1.7	2.4	11.6	0.8	1.2
k-means clust.	55.3	18.4	8.4	30.0	9.0	4.4	18.1	4.4	2.4	**11.6**	2.2	1.5	8.6	1.0	1.0

Table 2 Intra-node, cumulative retrieval results for 1% and 10% global search limits ($\mathbf{1 \times 10^3}$ and $\mathbf{10 \times 10^3}$ candidates per partition, respectively)

Partitions:	512		1024		2048		4096		8192	
GIST: 1% limit										
Algorithm	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall
Random OMP	**0.13**	0.20	0.14	0.22	0.16	0.26	0.16	0.27	0.16	0.28
KSVD OMP	0.10	0.15	0.11	0.19	0.12	0.21	0.13	0.20	0.13	0.21
B-KSVD OMP	**0.13**	**0.23**	**0.16**	**0.26**	**0.18**	**0.27**	**0.20**	**0.29**	**0.19**	**0.30**
Sample clust.	0.02	0.03	0.03	0.05	0.05	0.09	0.09	0.15	0.16	0.25
k-means clust.	0.02	0.03	0.03	0.05	0.04	0.06	0.06	0.10	0.09	0.13
SIFT: 1% limit										
Algorithm	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall
Random OMP	0.14	0.20	0.16	0.23	0.19	0.27	0.21	0.36	0.21	0.30
KSVD OMP	0.16	**0.24**	**0.19**	**0.33**	0.21	0.33	0.23	0.37	0.23	0.37
B-KSVD OMP	**0.17**	**0.24**	**0.19**	0.27	**0.22**	**0.34**	0.23	**0.38**	0.25	0.40
Sample clust.	0.03	0.05	0.07	0.11	0.13	0.19	**0.25**	0.37	0.44	**0.59**
k-means clust.	0.03	0.05	0.06	0.10	0.11	0.16	**0.25**	0.35	**0.46**	**0.59**
GIST: 10% limit										
Algorithm	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall
Random OMP	0.53	0.62	**0.49**	**0.60**	0.42	0.56	0.34	0.44	0.28	0.41
KSVD OMP	0.41	0.50	0.38	0.51	0.33	0.47	0.29	0.39	0.26	0.35
B-KSVD OMP	**0.57**	**0.69**	0.47	**0.60**	0.36	0.50	0.28	0.39	0.22	0.33
Sample clust.	0.21	0.28	0.37	0.46	0.54	0.65	0.68	0.78	0.72	0.82
k-means clust.	0.21	0.27	0.44	0.53	**0.66**	**0.74**	**0.76**	**0.85**	**0.80**	**0.89**
SIFT: 10% limit										
Algorithm	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall	%50NN	1-recall
Random OMP	0.63	0.75	0.61	0.72	0.56	0.67	0.50	0.66	0.43	0.54
KSVD OMP	0.65	0.77	0.62	0.75	0.56	0.69	0.50	0.62	0.41	0.52
B-KSVD OMP	**0.67**	**0.78**	0.63	0.74	0.60	0.69	0.51	0.64	0.44	0.58
Sample clust.	0.43	0.51	0.69	0.79	0.89	0.95	**0.95**	0.98	0.92	0.98
k-means clust.	0.46	0.56	**0.84**	**0.90**	**0.96**	**0.99**	**0.95**	**0.99**	**0.93**	**0.99**

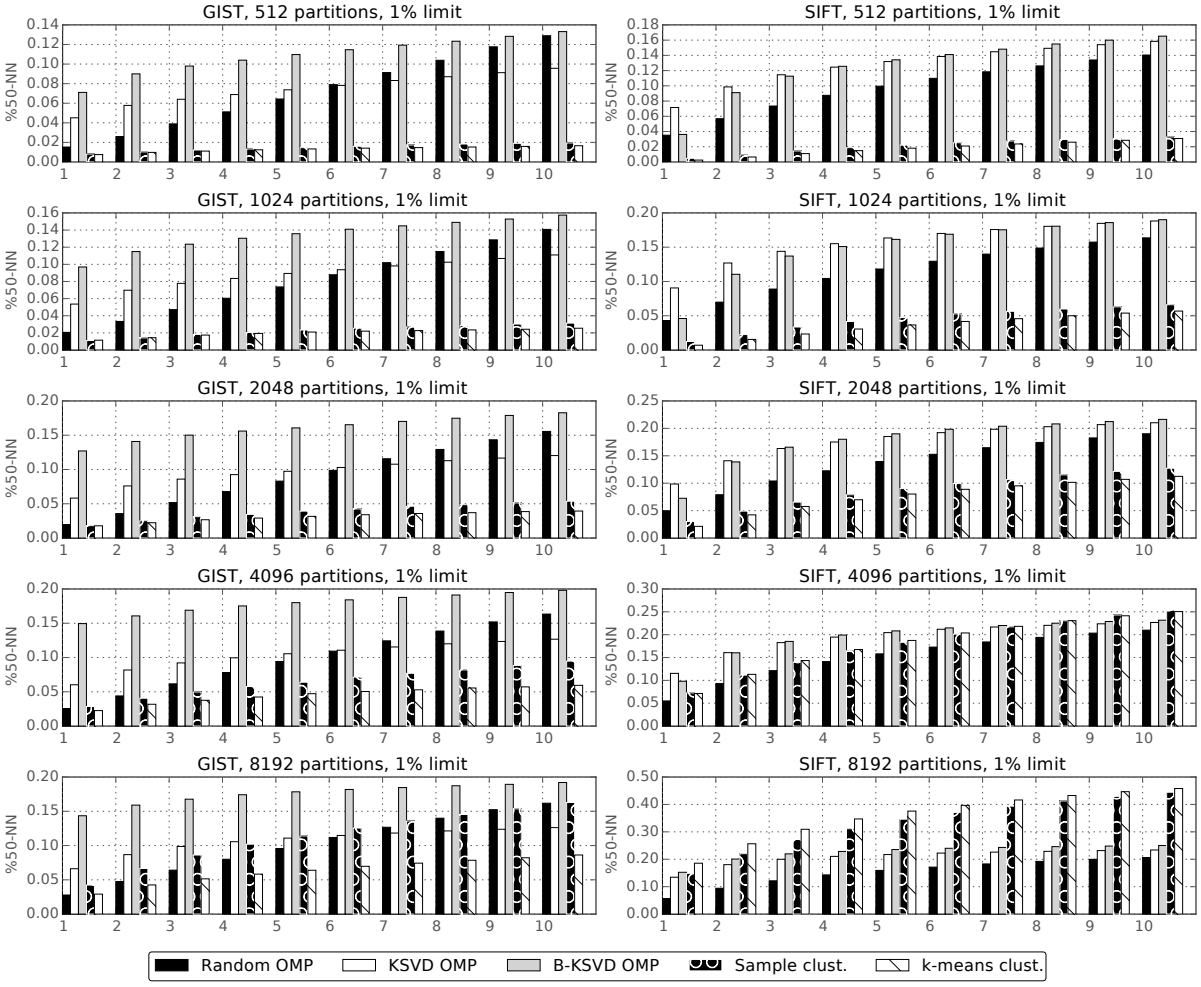

Fig. 4 Cumulative inter-node partition: global %50-NN, for multiple feature types and number of partitions

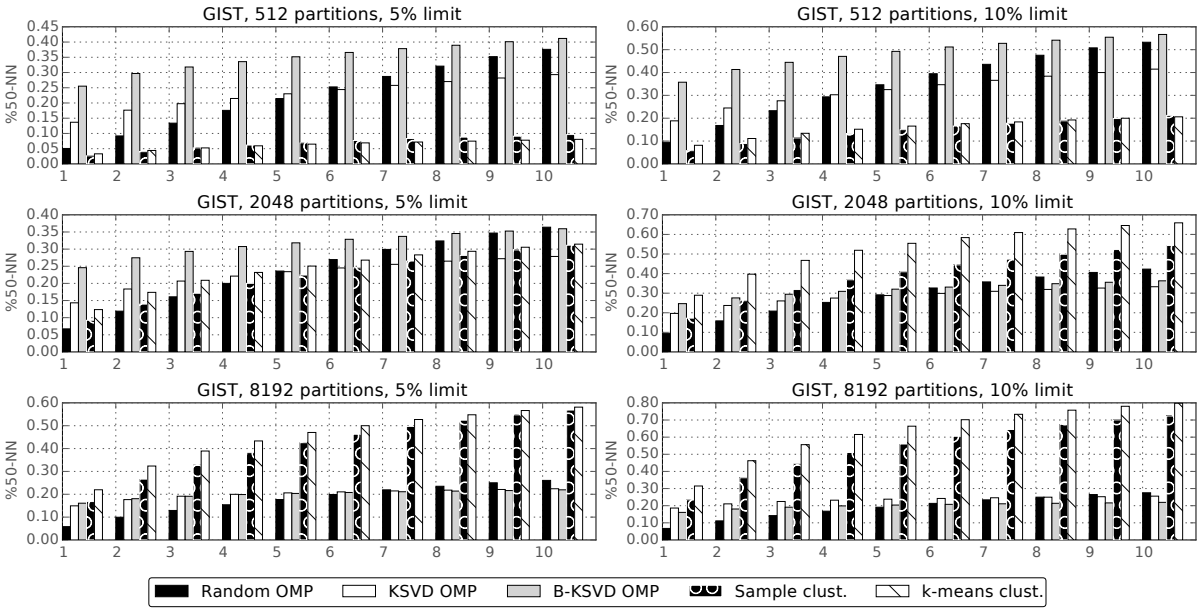

Fig. 5 Cumulative inter-node partition: global %50-NN, for GIST features and multiple limits

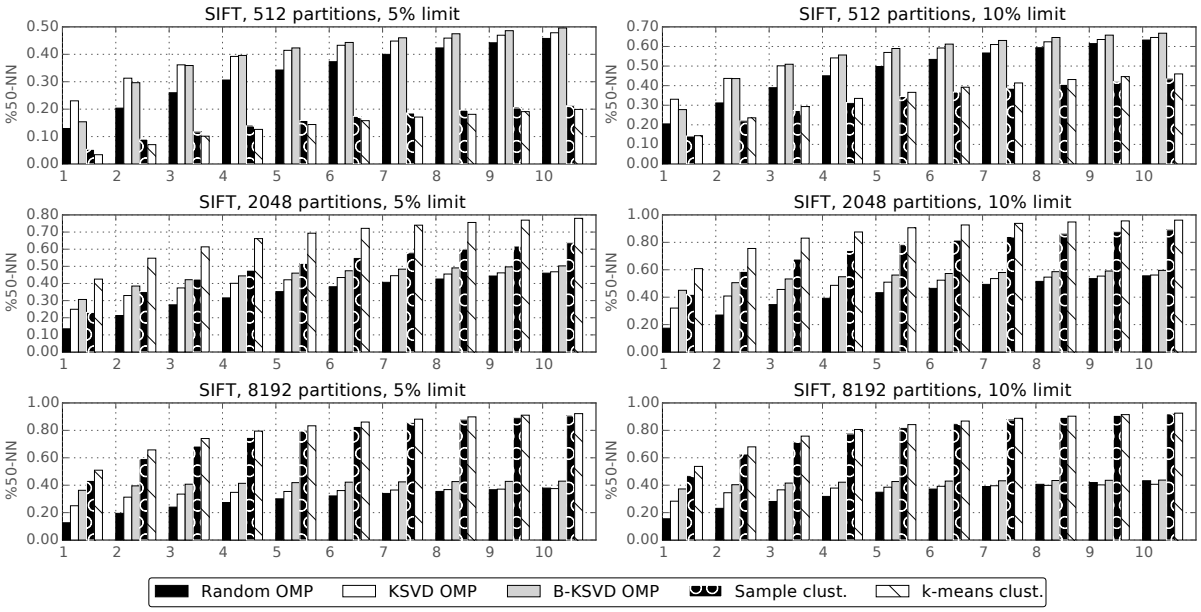

Fig. 6 Cumulative inter-node partition: global %50-NN, for SIFT features and multiple limits

k-means performance is greatly affected by feature type. For SIFT features, k-means partition size balancing is in line with B-KSVD for the top 20 positions, with a faster decay in the number of documents on the smaller partitions. For GIST features, the unbalanced distribution is more clear and appears earlier (top 20).

We also compared the impact in balancing partition sizes of the data-dependent approaches (k-means, KSVD and B-KSVD) versus random and sampling techniques. OMP with the random dictionary balancing varied greatly for the type of features used: for GIST, it is in line with k-means; for SIFT it has the most unbalanced distribution of all tested methods (e.g. some partitions have over 1/8 of the total number of indexed documents). Sample clustering also shows large unbalances, where larger partitions clustered most of the documents. The large balancing variations for these methods shows that adjusting codebooks to the data has a large impact on balancing partitions.

The impact of the number of partitions is also clearly visible. The tested partitioning methods are not designed to handle a higher number of partitions and generates a large number of very small or empty partitions (visible on the right side of X-axis of Figure 2 charts). The exception is B-KSVD, that managed to keep evenly sized partitions, regardless of the number of partitions.

These experiments showed how different partitioning methods distribute documents across partitions. B-KSVD countered the greedy nature of regular KSVD and offered the most uniform partitions. On the following sections, we'll show how it affects the retrieval performance.

4.2 Searching redundant partitions

Balancing partition sizes is only desirable if it does not degrade retrieval performance. In this section, we'll measure the retrieval impact of searching on over-complete partitions. An advantage of real-valued over binary hashes is that sparse hash values represent the document-partition membership likelihood. By having a measure of membership of the documents and queries to partitions, one can prioritize candidate selection at partition or global level.

Intra-partition search: On this experiment, we measure the retrieval performance of individual partitions, Figure 3. For each query, we selected 1000 candidates (i.e., 0.1% of total index size) for each corresponding partition, for a combined limit of 1%. The order of partition inspection is defined by the coefficients (OMP-based approaches) or the inverse of the distance to the centroid (centroid-based approaches).

B-KSVD offers the best results on the first partition (i.e., higher membership) for GIST partitions (14% of 50 nearest neighbors, examining, 1000 documents, i.e., 0.1% of the index). The number of nearest neighbors decreases for lower membership partitions. The impact of the remaining partitioning methods is in the order of 2% of the 50 nearest neighbors, for a 0.1% search limit.

For SIFT, the partition results show a different pattern. KSVD and B-KSVD also retrieve the most results on the top membership positions, for all but the experiments with 8192 partitions. For larger numbers of partitions, clustering-based solutions offer better results.

Inter-partition (global) search: Table 2 shows the aggregated results for the search process. From each partition, we selected 0.1% and 1% of total index size. With the (fixed) sparsity factor s set to 10, the combined limit is 1% and 10% of total index size, respectively.

The advantages of KSVD based methods are clear on the limited search conditions (inspecting 1% of the index). When using smaller search inspection limits, the reconstruction coefficient represents similarity in the original space better than the distance to cluster centroids. For larger search inspection limits (10%) and more partitions, clustering methods can retrieve a larger set of candidates. The same findings are also valid for 1-recall results. The main difference is that they are slightly higher that %50NN results. This means that both sparse coding and clustering methods can index the first nearest neighbor at higher rates than the remaining 49 nearest neighbors.

Incremental inter-partition search: Figures 4, 5 and 6 detail Table 2 results, by showing the incremental %50NN gains of inspecting all 10 partitions. Figure 4 shows the results for a 1% search limit; Figures 5 and 6 show what happens when increasing the search limit to 5% and 10% of index size. As with Figure 3 results, partitions are inspected by coefficient or inverse distance-to-centroid order.

These results show some interesting differences in retrieval effectiveness. For GIST features, B-KSVD achieves good %50NN when inspecting one partition with 1%-5% search limits. We can also see that there is only an average difference in %50NN between 5 and 10% when inspecting one versus all (10) partitions, regardless of partition size. For SIFT features, the behavior is similar, with higher %50NN gains when inspecting more partitions. KSVD behaves similarly, but with lower initial %50NN on most parameter configuration. Random OMP follows a trend more similar to clustering based approaches: the initial %50NN is lower, and the gains from inspecting more partitions are higher.

On k-means and other clustering-based techniques, having more partitions increases retrieval performance. This property is more evident when dealing with more partitions and using higher search limits.

As both codebook and clustering methods are based on greedy atom selection, adding the document to more partitions does not affect the results for previous partitions. Thus, we can see the expected retrieval performance of setting the value of the redundancy factor to between one and ten by looking at precision levels at those levels in Figures 4, 5 and 6.

5 Discussion

On the previous section, we showed how search space partitioning algorithms affect the balanced distribution of documents across partitions, and the corresponding impact on retrieval performance. This fulfills the initial goal of this paper: create balanced search space partitions for distributed indexing and retrieval. We argue that these experiments show that B-KSVD can work for the partitioning of a distributed search index better than the baseline methods.

A clear pattern across experiments is that sparse coding-based techniques have better precision with lower search limits and a smaller number of partitions, while centroid based approaches have better results when searching with higher limits and more centroids. This is due to centroid-based techniques being based on selecting points in the original space. Selecting more centroids increases the probability of getting better coverage of the search space. This property also results in having the nearest neighbors more spread out over more partitions, resulting in higher gains when inspecting more partitions. This is visible when dealing with SIFT features and a large number of partitions (e.g., 8192). We can also observe that centroid selection as a non-negligible impact on retrieval performance.

This contrasts with dictionary-based approaches, which transform the original feature space into a new space, based on the principal directions of the original space. Gains in performance decrease as one inspects the hash dimensions with a smaller reconstruction coefficients.

B-KSVD achieves good distribution and works well for low search limits (1% of total index size), and a small number of partitions (512 to 1024). This is an interesting property to answer concurrent queries on distributed indexes: having a more balanced distribution means that the probability of querying the same node for multiple queries decreases.

k-means works better for higher limits and number of partitions, due to being based on working in the original search sparse. It offers more predictable, linear performance increase, at the cost of by having larger partitions.

6 Conclusion

On this paper, we propose B-KSVD, a codebook learning technique for the creation of balanced, over-complete search partitions. We formalized the requirements to create overcomplete representations with redundant document indexing, where partitions contain overlapping subsets of data. We proposed representations based on sparse coding and clustering models, and an adaption

to the KSVD algorithm, balanced KSVD (B-KSVD), that distributes hash values across positions, according to the global distribution.

Experiments showed that computing codebooks that penalize larger partitions creates more balanced partitions, and has a positive retrieval impact.

Acknowledgements

This work has been partially funded by the CMU Portugal research project GoLocal Ref. CMUP-ERI/TIC/0033/2014, by the H2020 ICT project COGNITUS with the grant agreement No 687605, by the project NOVA LINCS Ref. UID/CEC/04516/2013 and by the Ph.D. scholarship grand SFRH/BD/95064/2013.

References

1. Aharon M, Elad M, Bruckstein A (2006) K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. IEEE Trans on Signal Processing 54(11):4311–4322
2. Aly M, Munich M, Perona P (2011) Distributed Kd-Trees for Retrieval from Very Large Image Collections. In: Proc. of BMVC
3. Andoni A, Indyk P (2008) Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. ACM Commun 51(1):117–122
4. Arthur D, Vassilvitskii S (2007) k-means++: The advantages of careful seeding. In: Proc. of ACM-SIAM SODA, pp 1027–1035
5. Babenko A, Lempitsky V (2015) The inverted multi-index. IEEE Tran on PAMI 37(6):1247–1260, DOI 10.1109/TPAMI.2014.2361319
6. Babenko A, Lempitsky V (2016) Efficient indexing of billion-scale datasets of deep descriptors. In: Proc. of IEEE CVPR
7. Batko M, Falchi F, Lucchese C, Novak D, Perego R, Rabitti F, Sedmidubsky J, Zezula P (2010) Building a web-scale image similarity search system. Multimed Tool Appl 47(3):599–629
8. Bezdek JC (1981) Pattern Recognition with Fuzzy Objective Function Algorithms. Kluwer Academic Publishers, Norwell, MA, USA
9. Borges P, Mourão A, Magalhães J (2015) High-dimensional indexing by sparse approximation. In: ACM ICMR'15, ACM
10. Büttcher S, Clarke CL, Cormack GV (2010) Information retrieval: Implementing and evaluating search engines. Mit Press
11. Cherian A, Sra S, Morellas V, Papanikolopoulos N (2014) Efficient nearest neighbors via robust sparse hashing. IEEE TIP 23(8):3646–3655
12. Chum O, Philbin J, Zisserman A (2008) Near duplicate image detection: min-hash and tf-idf weighting. In: Proc. of BMVC
13. Datar M, Immorlica N, Indyk P, Mirrokni VS (2004) Locality-sensitive hashing scheme based on p-stable distributions. In: Proc. of SCG, ACM, pp 253–262
14. Dean J, Ghemawat S (2008) Mapreduce: Simplified data processing on large clusters. Commun ACM 51(1):107–113
15. Ester M, Kriegel HP, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proc. of KDD, pp 226–231
16. Grauman K, Fergus R (2013) Machine Learning for Computer Vision, Springer Berlin Heidelberg, chap Learning Binary Hash Codes for Large-Scale Image Search, pp 49–87. DOI 10.1007/978-3-642-28661-2_3
17. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507, DOI 10.1126/science.1127647
18. Hoerl AE, Kennard RW (2004) Ridge Regression. John Wiley and Sons, Inc., DOI 10.1002/0471667196.ess2280
19. Jégou H, Douze M, Schmid C (2011) Product quantization for nearest neighbor search. IEEE Trans on PAMI 33(1):117–128, DOI 10.1109/TPAMI.2010.57
20. Jégou H, Tavenard R, Douze M, Amsaleg L (2011) Searching in one billion vectors: re-rank with source coding. ArXiv e-prints 1102.3828
21. Ji R, Duan LY, Chen J, Xie L, Yao H, Gao W (2013) Learning to distribute vocabulary indexing for scalable visual search. IEEE Trans on Multimedia 15(1):153–166, DOI 10.1109/TMM.2012.2225035
22. Kalantidis Y, Avrithis Y (2014) Locally optimized product quantization for approximate nearest neighbor search. In: Proc. of IEEE CVPR, pp 2329–2336, DOI 10.1109/CVPR.2014.298
23. Karger D, Lehman E, Leighton T, Panigrahy R, Levine M, Lewin D (1997) Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In: Proc. of ACM STOC, STOC '97, pp 654–663
24. Kulkarni A, Callan J (2010) Document allocation policies for selective searching of distributed indexes. In: Proc. of ACM CIKM, CIKM '10, pp 449–

458, DOI 10.1145/1871437.1871497
25. Lewicki MS, Sejnowski TJ (2000) Learning overcomplete representations. Neural Comput 12(2):337–365, DOI 10.1162/089976600300015826
26. Li Z, Ning H, Cao L, Zhan T, Gong Y, Huang TS (2011) Learning to search efficiently in high dimensions. In: Neural Information Processing Systems
27. Lloyd S (1982) Least squares quantization in pcm. IEEE Trans on Information Theory 28(2):129–137
28. Magalhaes J, Rueger S (2007) High-dimensional visual vocabularies for image retrieval. In: ACM SIGIR'07, ACM, New York, NY, USA, pp 815–816, DOI 10.1145/1277741.1277923, URL `http://doi.acm.org/10.1145/1277741.1277923`
29. Moise D, Shestakov D, Gudmundsson G, Amsaleg L (2013) Indexing and searching 100m images with map-reduce. In: ICMR'13, pp 17–24, DOI 10.1145/2461466.2461470
30. Mourão A, Magalhães Ja (2015) Scalable multimodal search with distributed indexing by sparse hashing. In: ACM ICMR'15, ACM, New York, NY, USA, pp 283–290, DOI 10.1145/2671188.2749310, URL `http://doi.acm.org/10.1145/2671188.2749310`
31. Muja M, Lowe DG (2014) Scalable nearest neighbor algorithms for high dimensional data. IEEE Trans on PAMI 36
32. Olshausen BA, Field DJ (1996) Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature 381(6583):607–609
33. Pati Y, Rezaiifar R, Krishnaprasad P (1993) Orthogonal Matching Pursuit : recursive function approximation with application to wavelet decomposition. In: Asilomar Conf. on Signals, Systems and Computer
34. Raginsky M, Lazebnik S (2009) Locality-sensitive binary codes from shift-invariant kernels. In: NIPS, pp 1509–1517
35. Tavenard R, Jégou H, Amsaleg L (2011) Balancing clusters to reduce response time variability in large scale image search. In: International Workshop on Content-Based Multimedia Indexing (CBMI 2011), Madrid, Spain, URL `http://hal.inria.fr/inria-00576886`, qUAERO
36. Thaler DG, Ravishankar CV (1998) Using name-based mappings to increase hit rates. Trans on Networking 6(1):1–14, DOI 10.1109/90.663936
37. Tibshirani R (1994) Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B 58:267–288
38. Torralba A, Fergus R, Freeman W (2008) 80 million tiny images: A large data set for nonparametric object and scene recognition. IEEE Trans on PAMI 30(11):1958–1970, DOI 10.1109/TPAMI.2008.128
39. Weber R, Schek HJ, Blott S (1998) A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proc. of VLDB, pp 194–205
40. Weiss Y, Torralba A, Fergus R (2008) Spectral hashing. NIPS 9(1):6
41. Yang Z, i Kamata S, Ahrary A (2009) Nir: Content based image retrieval on cloud computing. In: Proc of ICIS, vol 3, pp 556–559
42. Zou H, Hastie T (2005) Regularization and variable selection via the elastic net. Journal of the Royal Statistical Society, Series B 67:301–320