Balanced Search Space Partitioning for Distributed Media Redundant Indexing

André Mourão
NOVA LINCS
Faculdade de Ciências e Tecnologia
Universidade NOVA de Lisboa 2825-516 Caparica
a.mourao@campus.fct.unl.pt

João Magalhães
NOVA LINCS
Faculdade de Ciências e Tecnologia
Universidade NOVA de Lisboa 2825-516 Caparica
jmag@fct.unl.pt

ABSTRACT

This paper addresses the problem of balanced, redundant indexing of media information. Our goal is to partition and distribute the search index, taking advantage of the distributed systems properties: balanced load across nodes, redundancy on node down and efficient node usage under concurrent querying. We follow an information compression approach to solve this problem and propose to represent data with overcomplete codebooks, where each document is represented by only a few codewords and an indexing node is responsible for several codewords. Quantization algorithms are designed to fit the original data as best as possible, leading to bias towards codewords that fit the principal directions of data. In this paper, we propose the balanced KSVD (B-KSVD) algorithm, that distributes the allocation of data across a balanced number of codewords, according to the global distribution of data. Indexing experiments showed that B-KSVD can achieve 38% *1-recall* by inspecting only 1% of the full index, distributed over 10 partitions. Traditional methods based on k-means need to either use larger codebooks or to inspect a larger portion of the index to achieve the same retrieval performance.

KEYWORDS

Search space partitioning; Distributed search; High-dimensional indexing; Dictionary design; Clustering; Approximate nearest neighbor search

1 INTRODUCTION

The goals of index partitioning algorithms are to distribute documents across nodes based on document similarity, to facilitate the efficient selection of retrieval resources, such that documents relevant to a query are concentrated across a few shards [22]. There are two main index partitioning strategies [9]:

- **horizontal** partition or **sharding**: divide documents across nodes;
- **vertical** or **term-based** partition: divide document features across multiple nodes.

In addition to dealing with high-dimensional data and its unknown underlying structure, these algorithms have the opportunity to take advantage of the characteristics of new distributed systems (e.g. cloud environments), parallel processing, hardware redundancy (i.e. index documents on more than one node), and ability to deploy additional nodes on-demand. Existing multimedia document distribution do not explore these characteristics, as document allocation policies are either random, e.g. [29], or based on existing partitions of single node algorithms, e.g. [6]. One of the works that goes towards our partitioning goals is by Ji et al. [20]. They tested *global* and *local* indexing partitioning techniques (*horizontal* and *vertical* partition respectively), based on Vocabulary Tree model quantization, and showed that vertical partitioning offers the best temporal performance on a distributed setting, without an increase in load imbalance.

Our goal in this paper is to study the impact of overcomplete data representations on the load 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. The overcomplete property of the codebook, means that the number of partitions/codewords is much higher than the original data dimensionality. Indexing is achieved by encoding each media vector as a linear combination of just a few codewords.

The main contribution of this paper is the balanced-KSVD algorithm (B-KSVD) that distributes the allocation of data across a balanced number of codewords. B-KSVD is a distributed indexing algorithm that computes a codebook with an overcomplete set of codewords that addresses a number of challenges. The first one, is the even distribution of data across codewords to achieve better load-balancing when allocating data to nodes. The second, is that documents should be assigned to partitions with documents that are also close in the original space. And third, because a vector is encoded with multiple codewords, each one corresponding to a space partition, data will be stored redundantly across multiple nodes (each node has the capacity to serve multiple codewords).

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 good 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 performance decrease, instead of no results returned.

This paper is organized as follows: Section 2 details the related work in distributed media search and space partitioning. Section 3 details the formalization of overcomplete redundant partitioning and proposed solutions. Section 4 describes the experiments and section 5, we discuss our conclusions.

2 RELATED WORK

The bulk of distributed multimodal retrieval comes from combining Map Reduce [13] with single node algorithms or from distributing the feature spaces across nodes [28]. When applied to CBMI [27, 29, 38], Map Reduce can be used to partition indexes horizontally.

As Map Reduce requires Map and Reduce nodes to be data agnostic, indexes must either query all nodes for all queries [29], which does not meet our efficiency goal, or have all nodes have access to the full index [27], which is limited by the time it takes to fetch the relevant index subset. Moise et al. [27] 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 CBMI Aly et al. [2], Batko et al. [6], but the effectiveness of sub-tree based index partitioning is reduced when the dimensionality of the vectors to index increases [36], 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 search space in a data independent way, according to a set of randomly generated hyperplanes. Each hash bit represents an hyperplane in the original feature space that divides it in two, assigning a value of zero or one related to the side in which the document is. Hash codes are generated by concatenating multiple of the values of these functions. The search space is partitioned horizontally, according to the document's hash: documents with similar hash code that have high probability of being similar in the original space are stored on the same buckets. LSH spawn multiple techniques [11, 12, 32] that explore alternatives to hyperplane partitioning by applying other families of functions with different structures (e.g. grid).

Other works have focused on creating better hash functions, by exploring the structure of the data in original space. By leveraging on the distribution of documents in the original search space, data dependent hash functions [16, 24, 35, 37] can create better partitions for similarity search. Grauman and Fergus [15] authored a review of data dependent hash techniques.

Regression and codebook design. Sparse hashes are generated in a very high dimensional overcomplete 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 hashes with non-zero values. Lewicki and Sejnowski [23] 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.

Multiple techniques were developed to generate high dimensional sparse hashes. These techniques differ by the type of regulation applied to the hashes: l_0 penalty (e.g. OMP [31]), l_1 penalty (e.g. Lasso [34]), l_2 penalty (e.g. Ridge [17]) or a combination of the l_1 and l_2 penalties (e.g. Elasticnet [39]). 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, Ridge limits the coefficient magnitude by applying the l_2 penalty Elasticnet's penalty is a mixture of Lasso's l_1 penalties with Ridges 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 (or dictionary) as the basis of the transformation into the new space. Codebook computation algorithms such as K-SVD, select codewords that minimize reconstruction error. K-SVD [1] alternatively updates a codebook and the coefficients. Stochastic gradient descent techniques (e.g. [30]) update each example per iteration, to minimize reconstruction error. Cherian et al. [10] presented an index based on hashes created using l_1 regression and the Newton Descent for codebook learning. Borges et al. [8] presented an indexes based on sparse hashes created using l_0 regression and a codebook learned through K-SVD.

Quantization through clustering. Clustering techniques are one of the most used space partitioning techniques, with applications that range from image retrieval [26] to image indexing [18]. The search space is partitioned by generating a set of centroids, and vectors are assigned to the closest centroid according to a metric (e.g. euclidean distance). k-means, a popular clustering technique, aims to find the set of centroids that minimizes sum of squares within-cluster distances. Lloyd [25] 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 analysing the distribution of the seed centroids and the training data distribution. Fuzzy c-means clustering/soft clustering [7] techniques extends the assigning of documents to multiple clusters, by keeping membership information for documents to clusters (e.g. ratio of the distance to the centroids). Clustering techniques such as DBSCAN [14], do not set the number of centroids as a parameter, focusing instead on the cluster density and points per cluster.

Clustering techniques are behind some of the best performing nearest neighbour search algorithms. Jégou et al. [18] proposed an index that divides the space into a set of Voronoi cells through k-means based vector quantization. Further works improve candidate distance computation [19], descriptor quantization [21] and more effective centroid evaluation [5]. Tavenard et al. [33] proposed a technique for balancing k-means cluster size, by shifting cluster boundaries into parallel boundaries. They experiments showed less variability in the number of candidates retrieved per query.

3 SPACE PARTITIONING CODEBOOKS

On the previous section, we described how indexing partitioning techniques are applied to distributed search. Figure 1 illustrates the assignment of queries to partitions for existing index partitioning techniques and for our proposed technique. Figure 1 (a) shows a single assignment technique, where each document is assigned to a single partition (e.g. [6]). Figure 1 (b) shows a random assignment technique, where documents are assigned to a single partition randomly, and queries are assigned to all partitions. This technique is applied on some Map-Reduce systems (e.g. [29]). Figure 1 (c) shows our proposal: a similarity-based, multiple assignment technique. Under this partitioning paradigm, each document is assigned to multiple partitions, based on similarity in the original feature space. Inspecting multiple partitions will result on a incremental increases in retrieval performance. Conversely, node failures will also result

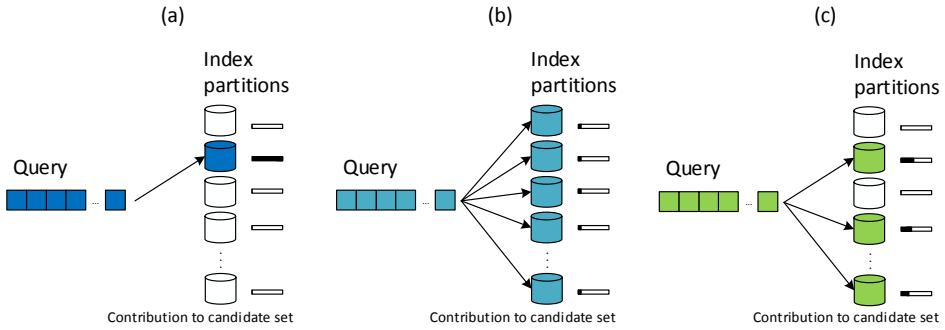

Figure 1: Examples of the querying process of multiple document partitioning policies: (a) single node indexing, (b) random indexing, (c) redundant overcomplete indexing

on incremental losses in retrieval performance, instead of returning no results (Figure 1 (a)). To create representations that fit this partitioning paradigm, partition methods must have the following properties:

- fixing **overcompleteness** of the codewords as a parameter, i.e. sparsity factor;
- give partition **membership information** (e.g. distance to centroid, reconstruction weight) to allow candidate selection inside partitions;
- **partitions** should group **similar documents** in the original space;
- generate **evenly sized** partitions.

Formally, consider the original vector $y \in \mathbb{R}^n$, a codeword $x \in \mathbb{R}^k$ and a sparsity coefficient s.

$$y \in \mathbb{R}^n \rightarrow x \in \mathbb{R}^k$$
$$\text{where } \|x\|_0 = s \text{ and } s \ll n \ll k. \tag{1}$$

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. S For a set of vectors $y_a, y_b, y_c \in Y \in \mathbb{R}^{m,n}$ and corresponding codewords $x_a, x_b, x_c \in X \in \mathbb{R}^{m,k}$, our goal is to generate codewords that respect the following property:

$$\|y_a - y_b\|_2 < \|y_a - y_c\|_2 \rightarrow \|x_a + x_b\|_0 < \|x_a + x_c\|_0 \tag{2}$$

In other words, vectors that are close in the original space have non-zero coefficients on similar positions in the codeword space than vectors that are further apart. These codewords are the basis to generate a set of partitions $P, p \in P \subset Y$. Our balancing goal is to minimize the differences on partition sizes: $|p_a| - |p_b|, \forall p_a, p_b \in P$.

After studding the properties of the space partitioning in the literature, we arrived at two families of methods that have the potential to meet the desired properties: sparse hashes and clustering. Sparse coding techniques are designed to generate overcomplete representations of the search space: our reasoning is that codebook atoms 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, we'll detail further how we applied these families of methods.

3.1 Codebooks by Sparse Coding

Sparse codewords can be computed as sparse high-dimensional hashes. Sparse hashes offer a number of advantages over binary hashes for search space partitioning: sparse coding techniques are designed to be overcomplete, offer real-valued membership (e.g. representative hash values) and offer control over the sparsity of the solution and thus, redundancy. The steps for generating sparse hashes are:

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

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

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

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. Eq. 3 generates an hash with the desired properties, using a previously computed dictionary. Techniques for dictionary computation include K-SVD [1] and Stochastic Gradient Descent techniques. On this paper, we'll focus on K-SVD, and on an adaptation that takes number of documents per bucket into account.

3.1.1 *KSVD and OMP.*

Dictionary computation requires solving the following optimization problem:

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

where $D \in \mathbb{R}^{n \times k}$ is a dictionary, learned from the data, $Y \in \mathbb{R}^{m \times n}$ is the the original doc. vector, $X \in \mathbb{R}^{m \times k}$ is the sparse hash and s is

the sparsity coefficient. Solving for both D and X is computationally hard. KSVD alternatively optimizes the solution for D and X. KSVD updates each dictionary atom iteratively (represented by i), while fixing other atoms $j_{[0,k]} \neq i$.

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

where $\|...\|_F$ is the Frobenius norm. Sparsity is enforced by using only the atoms with non-zero coefficients: I is the set of all index with non-zero coefficients that use atom i for reconstruction.

By fixing j atoms, the value for atom Di 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 with OMP.*

KSVD enforces the creation of sparse representations that group similar vectors in the original space on non-zero coefficients. When generating multiple codewords, KSVD will inherently create unbalanced representations, as the dictionary atoms are biased towards the principal directions of the data on the original space. As our goal is to minimize the differences between the number of elements per partition P_j and the mean number of documents per partition $\bar{P}$, we adapted KSVD to reduce the magnitude of dictionary atoms assigned with more documents. The KSVD alternate optimization process is similar to Eq.5; Our adaptation is applied to Eq.6 E decomposition; after the rank-1 approximation, we multiply the G matrix by the penalization factor B:

$$\begin{aligned} &\text{balanced } \hat{E}_i = B \times G \sum^{1} V^T \\ &B = \frac{1}{\left(\left|P_{j\in[0,k]}\right| + r\right)^e} \end{aligned} \tag{7}$$

where $\left|P_{j\in[0,k]}\right|$ contains the number of documents assigned to partitions j, computed using the previous iteration of the dictionary. 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 generated dictionary atoms, creating non-orthogonal balanced representations. The regularization parameters r and s control the magnitude of this distortion.

3.1.3 *Random dictionary with OMP.*

We can also measure the impact of dictionary learning on the computation of sparse hashes, by testing OMP with a dictionary generated from the Gaussian distribution with zero mean and unit std. deviation.

$$D \in \mathbb{R}^{n\times k} \subset \mathcal{N}(0,1) \tag{8}$$

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

3.2 Codebooks by Soft Quantization

Our quantization process can be seen as a type of soft clustering, where the cluster membership is controlled by a fixed s sparsity factor. Our focus is to measure how well these clusters can represent neighbour data in a balanced way, and how using multiple clusters affects these process in an high dimensional feature space. Our clusteing process is the following:

- find the centroids
- project the documents to s, redundant clusters
- search 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 hash values:

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

$$x_{i\in[0,k]} = \begin{cases} \|c_i - y\|_2, \text{ for } c_i \in c \\ 0, \text{ otherwise} \end{cases} \tag{9}$$

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, does not meet our desired properties.

Fuzzy c-means clustering [7] techniques extends the assigning of documents to clusters, by keeping membership information to multiple clusters (e.g. ratio of the distance to the centroids). It optimizes the intra-cluster objective function of k-means, combined with membership information, which allows documents to be in more than one clusters. In our preliminary experiments, fuzzy c-means produced very unbalanced clusters: all documents were assigned to only 20 clusters, irregardless of the total generated centroids (512, 1024, 2048, 4096, 8192). Due to this extreme balance, we did not pursue further experiments using fuzzy c-means.

3.2.1 *k-means centroids.*

k-means is one the most most widely applied clustering functions in nearest neighbour search. It tries to find the set of centroids $C \in \mathbb{R}^{n\times k}$ that minimizes the distances of the points to the centroids of their clusters. k-means tends to produce similarly sized clusters, which is a desirable property for our balanced partition goal (i.e. higher focus on balancing cluster size for better partitioning). The k-means clustering process minimizes the following expression:

$$\arg\min_C \sum_{c_i \in C} \sum_{y_j \in P_i} \left\|c_i - y_j\right\|_2 \tag{10}$$

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:

$$C \in \mathbb{R}^{n\times k} \subset Y \tag{11}$$

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 we can create over-complete codebooks that generate sparse, high dimensional codewords. To measure how well the proposed methods meet our partitioning and retrieval goals, we'll evaluate them from three prospectives:

- **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 whether the partitions capture the original space nearest neighbours;

Dataset: We tested the index partitioning methods on the Billion Vectors dataset [18, 19]. 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 allows us to measure the partitioning impact of multiple dimensionalities and feature distributions.

Metrics: In addition to load balancing quality metrics, which are 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 neighbours retrieved.

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 balance between similarity and even balancing. We set the sparsity coefficient to $s = 10$ for all algorithms, and varied the codeword size k and thus, number of partitions, (512, 1024, 2048, 4096, 8192).

4.1 Balanced partitioning

We defined balanced partition as the minimization of the differences in the number of documents per partition p (i.e. standard deviation of the partition size distribution). For distributed retrieval, balanced partitions minimize the differences in the expected load on the nodes with the matching partitions (i.e. non-zero coefficients), at indexing and query time, as illustrated in Figure 1 (c). Thus, the goal of this experiment is to measure how the selected techniques distribute the documents across partitions, for multiple numbers of partitions and feature types. To create the partitions, documents were assigned to the partitions with corresponding non-zero codeword positions, for each partition method, feature type and number of partitions. This experiment shows the resulting partition sizes.

Figure 2 shows the the behaviour of the partitioning algorithms for the GIST and SIFT features (different columns) and number of partitions (different rows). For readability, each chart is divided in two: the smaller chart shows the occupation of the top 20 partitions, where the variation in scale of number of documents is higher. The larger chart shows the variation for the remaining partitions (20 to number of partitions k). The X-axis represents the partitions, sorted in descending order of number of indexed documents (i.e. partitions with more documents are to the left). The Y-axis represents the

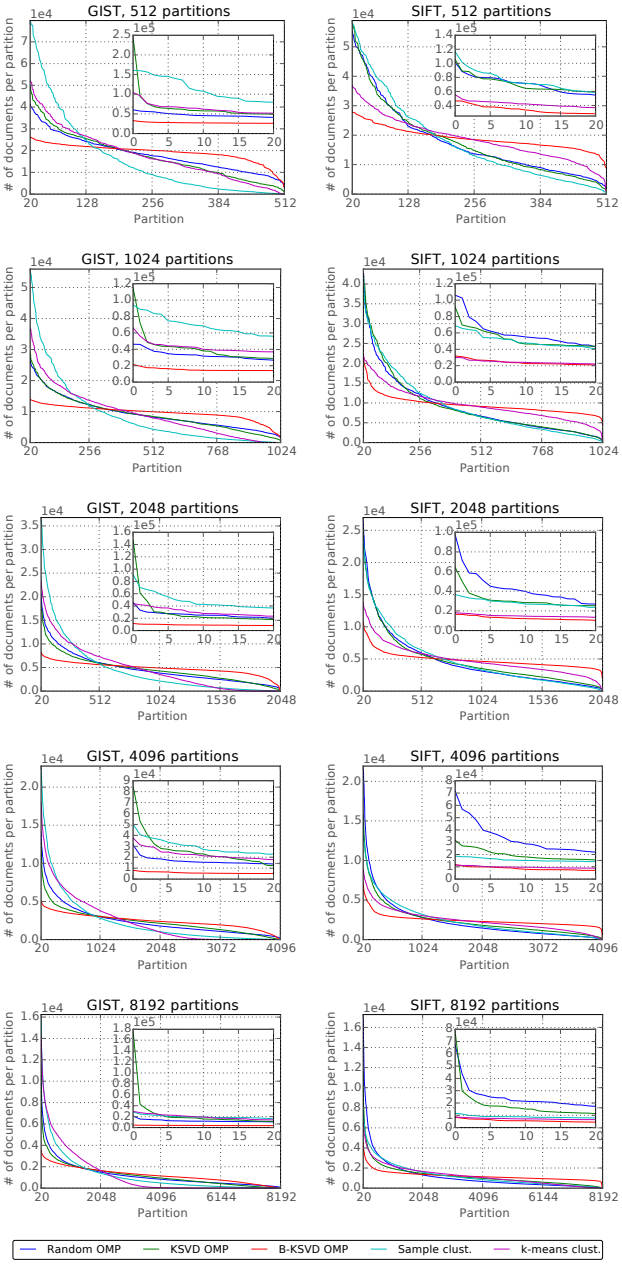

Figure 2: Sorted bucket distribution for multiple feature types and number of partitions. The smaller, inner chart shows the top 20 partition sizes, and the larger chart shows the remaining partition sizes.

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

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. Max, Mean 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**		
Algorithm	Features	Max	Med	σ	Max	Med	σ	Max	Med	σ	Max	Med	σ	Max	Med	σ
Random OMP	GIST	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
Random OMP	SIFT	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 (1×10^3 and 10×10^3 candidates per partition, respectively)

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

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. 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 std. deviation (σ), larger partition (Max), and median (Med) partition size ($k/2$).

KSVD learns a dictionary with the most prevalent directions of the data in the original space. Combined with OMP greedy atom selection, KSVD sparse representations are highly biased towards principal directions, which is clear on the top 20 charts. B-KSVD's bias managed to counteract KSVD's greediness and generated the most balanced solutions (σ columns on Table 1) The effect is more clear at the edge partitions (i.e. the ones with more documents and the ones with fewer documents): 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

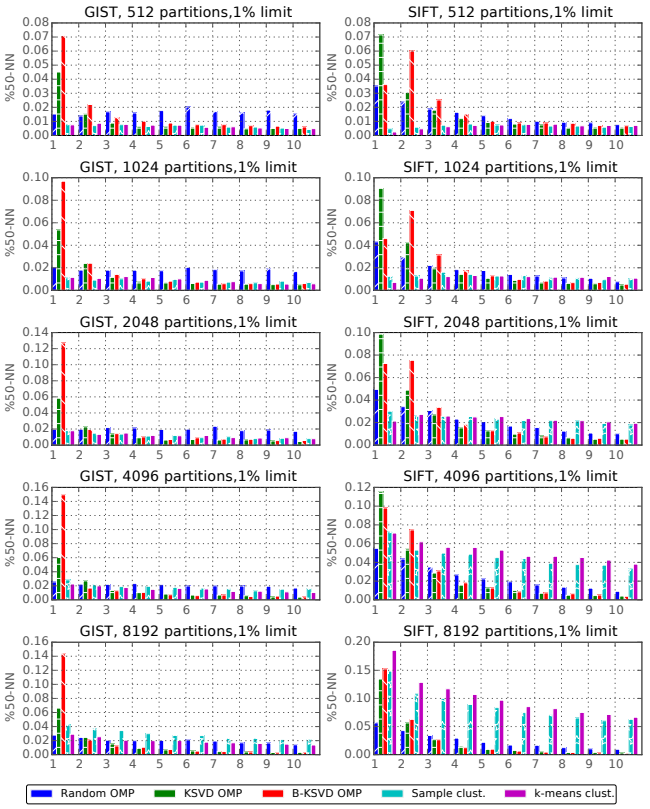

Figure 3: Inter-node partition %50-NN for individual partitions, for multiple feature types and number of partitions

consistently closer to the expected value (mean) that other methods, meaning the decrease in 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.

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

On this experiment, we also measured the impact of the codebook computation, versus random and sampling techniques. Random dictionary OMP 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. partitions with over 1/8 of the total 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 your dictionary to the data has a large impact on balancing partitions.

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

These experiment 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 codewords

On this section, we'll measure the retrieval impact of searching on over-complete partitions. An advantage of real-valued codewords over binary indexes is that codeword values represent document-partition membership. By having a measure of membership of the documents and queries to partitions, one can prioritize candidate selection at two levels:

Inter-partition search: Table 2 shows the aggregated results for the search process. From each partition, we selected 0.1% and 1% of total index size, for a combined limit of 1% and 10%, respectively.

The advantages of KSVD based methods are clear on the limited search conditions (e.g. inspecting 1% of the index). When using smaller search limits, the reconstruction coefficient represents similarity in the original space better than distance to cluster centroids. For larger limits (10%) and more partitions, k-means and sample clustering methods are able to retrieve a larger set of candidates. Examining Table 2 and other experiments omitted due to space constraints, we concluded that 1-recall results follow the same pattern as %50NN. This means that both method families are able to index the first nearest neighbour at higher rates than the remaining 49 nearest neighbours.

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%.

B-KSVD offers the best results on the first partition (i.e. higher membership) for GIST partitions (14% of 50 nearest neighbours, examining, 1000 documents, i.e. 0.1% of the index) The number of nearest neighbours decreases for lower membership partitions. The impact of the remaining partitioning methods is in the order of 2% of the 50 nearest neighbours, which is still an impressive value for 0.1% partition 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 8192 partitions experiments. For larger numbers of partitions, clustering-based solutions offer better results. We reckon that the smaller partitions will mean that the remaining candidates will have an higher probability of being the nearest neighbours.

5 CONCLUSION

On this paper, we proposed balanced over-complete partitioning representations for distributed retrieval. We formalized the requirements to create overcomplete representations, to redundant document indexing, where partitions contain overlapping subsets of data. Parallel processing and redundancy are achieved searching the overcomplete partitions. We proposed representations based on sparse hashing 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.

We showed that computing codebooks that penalise larger partitions, creates more balanced partitions, and a corresponding positive retrieval impact. B-KSVD achieves 38% *1-recall* by inspecting only 1% of the full index, distributed over 10 partitions. Clustering methods based on k-means performed better with more partitions with higher search limits. In addition, combining these techniques with effective single node retrieval techniques that can use the cluster membership value as an heuristic for search, can improve flexibility and performance of large scale distributed indexes.

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 and by the project NOVA LINCS Ref. UID/CEC/04516/2013.

REFERENCES

[1] M. Aharon, M. Elad, and A. Bruckstein. 2006. K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation. *IEEE Trans. on Signal Processing* 54, 11 (2006), 4311–4322.

[2] Mohamed Aly, Mario Munich, and Pietro Perona. 2011. Distributed Kd-Trees for Retrieval from Very Large Image Collections. In *Proc. of BMVC*.

[3] Alexandr Andoni and Piotr Indyk. 2008. Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. *ACM Commun.* 51, 1 (2008), 117–122.

[4] David Arthur and Sergei Vassilvitskii. 2007. k-means++: The advantages of careful seeding. In *Proc. of ACM-SIAM SODA*. 1027–1035.

[5] Artem Babenko and Victor Lempitsky. 2016. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *Proc. of IEEE CVPR*.

[6] Michal Batko, Fabrizio Falchi, Claudio Lucchese, David Novak, Raffaele Perego, Fausto Rabitti, Jan Sedmidubsky, and Pavel Zezula. 2010. Building a web-scale image similarity search system. *Multimed Tool Appl* 47, 3 (2010), 599–629.

[7] James C. Bezdek. 1981. *Pattern Recognition with Fuzzy Objective Function Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA.

[8] Pedro Borges, André Mourão, and J. Magalhães. 2015. High-Dimensional Indexing by Sparse Approximation. In *ACM ICMR'15*. ACM.

[9] Stefan Büttcher, Charles LA Clarke, and Gordon V Cormack. 2010. *Information retrieval: Implementing and evaluating search engines*. Mit Press.

[10] A. Cherian, S. Sra, V. Morellas, and N. Papanikolopoulos. 2014. Efficient Nearest Neighbors via Robust Sparse Hashing. *IEEE TIP* 23, 8 (2014), 3646–3655.

[11] O. Chum, J. Philbin, and A. Zisserman. 2008. Near Duplicate Image Detection: min-Hash and tf-idf Weighting. In *Proc. of BMVC*.

[12] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. of SCG*. ACM, 253–262.

[13] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (2008), 107–113.

[14] Martin Ester, Hans-Peter Kriegel, Jrg Sander, and Xiaowei Xu. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In *Proc. of KDD*. 226–231.

[15] Kristen Grauman and Rob Fergus. 2013. *Machine Learning for Computer Vision*. Springer Berlin Heidelberg, Chapter Learning Binary Hash Codes for Large-Scale Image Search, 49–87. DOI: http://dx.doi.org/10.1007/978-3-642-28661-2_3

[16] G. E. Hinton and R. R. Salakhutdinov. 2006. Reducing the Dimensionality of Data with Neural Networks. *Science* 313, 5786 (2006), 504–507. DOI: http://dx.doi.org/10.1126/science.1127647

[17] A. E. Hoerl and R. W. Kennard. 2004. *Ridge Regression*. John Wiley and Sons, Inc. DOI: http://dx.doi.org/10.1002/0471667196.ess2280

[18] Hervé Jégou, M. Douze, and C. Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Trans. on PAMI* 33, 1 (2011), 117–128. DOI: http://dx.doi.org/10.1109/TPAMI.2010.57

[19] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg. 2011. Searching in one billion vectors: re-rank with source coding. *ArXiv e-prints* (2011). arXiv:cs.IR/1102.3828

[20] Rongrong Ji, Ling-Yu Duan, Jie Chen, Lexing Xie, Hongxun Yao, and Wen Gao. 2013. Learning to Distribute Vocabulary Indexing for Scalable Visual Search. *IEEE Trans. on Multimedia* 15, 1 (2013), 153–166. DOI: http://dx.doi.org/10.1109/TMM.2012.2225035

[21] Y. Kalantidis and Y. Avrithis. 2014. Locally Optimized Product Quantization for Approximate Nearest Neighbor Search. In *Proc. of IEEE CVPR*. 2329–2336. DOI: http://dx.doi.org/10.1109/CVPR.2014.298

[22] Anagha Kulkarni and Jamie Callan. 2010. Document Allocation Policies for Selective Searching of Distributed Indexes. In *Proc. of ACM CIKM (CIKM '10)*. 449–458. DOI: http://dx.doi.org/10.1145/1871437.1871497

[23] Michael S. Lewicki and Terrence J. Sejnowski. 2000. Learning Overcomplete Representations. *Neural Comput.* 12, 2 (Feb. 2000), 337–365. DOI: http://dx.doi.org/10.1162/089976600300015826

[24] Zhen Li, Huazhong Ning, Liangliang Cao, Tong Zhan, Yihong Gong, and Thomas S. Huang. 2011. Learning to Search Efficiently in High Dimensions. In *Neural Information Processing Systems*.

[25] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE Trans. on Information Theory* 28, 2 (1982), 129–137.

[26] Joao Magalhaes and Stefan Rueger. 2007. High-dimensional Visual Vocabularies for Image Retrieval. In *ACM SIGIR'07*. ACM, New York, NY, USA, 815–816. DOI: http://dx.doi.org/10.1145/1277741.1277923

[27] Diana Moise, Denis Shestakov, Gylfi Gudmundsson, and Laurent Amsaleg. 2013. Indexing and Searching 100M Images with Map-reduce. In *ICMR'13*. 17–24. DOI: http://dx.doi.org/10.1145/2461466.2461470

[28] André Mourão and João Magalhães. 2015. Scalable Multimodal Search with Distributed Indexing by Sparse Hashing. In *ACM ICMR'15*. ACM, New York, NY, USA, 283–290. DOI: http://dx.doi.org/10.1145/2671188.2749310

[29] Marius Muja and David G. Lowe. 2014. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Trans. on PAMI* 36 (2014).

[30] Bruno A Olshausen and David J Field. 1996. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature* 381, 6583 (1996), 607–609.

[31] Y. Pati, R. Rezaiifar, and P. Krishnaprasad. 1993. Orthogonal Matching Pursuit : recursive function approximation with application to wavelet decomposition. In *Asilomar Conf. on Signals, Systems and Computer*.

[32] Maxim Raginsky and Svetlana Lazebnik. 2009. Locality-sensitive binary codes from shift-invariant kernels. In *NIPS*. 1509–1517.

[33] Romain Tavenard, Hervé Jégou, and Laurent Amsaleg. 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. http://hal.inria.fr/inria-00576886 QUAERO.

[34] Robert Tibshirani. 1994. Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society, Series B* 58 (1994), 267–288.

[35] A. Torralba, R. Fergus, and W.T. Freeman. 2008. 80 Million Tiny Images: A Large Data Set for Nonparametric Object and Scene Recognition. *IEEE Trans. on PAMI* 30, 11 (Nov 2008), 1958–1970. DOI: http://dx.doi.org/10.1109/TPAMI.2008.128

[36] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proc. of VLDB*. 194–205.

[37] Yair Weiss, Antonio Torralba, and Robert Fergus. 2008. Spectral Hashing. *NIPS* 9, 1 (2008), 6.

[38] Z. Yang, S. i. Kamata, and A. Ahrary. 2009. NIR: Content based image retrieval on cloud computing. In *Proc of ICIS*, Vol. 3. 556–559.

[39] Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the Elastic Net. *Journal of the Royal Statistical Society, Series B* 67 (2005), 301–320.