Thick Boundaries in Binary Space and Their Influence on Nearest-Neighbor Search

Tomasz Trzcinski, Vincent Lepetit and Pascal Fua[a]

[a]{firstname.lastname}@epfl.ch
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Computer Vision Laboratory
CH-1015 Lausanne, Switzerland
tel. +41216936862

Abstract

Binary descriptors allow faster similarity computation than real-valued ones while requiring much less storage. As a result, many algorithms have recently been proposed to binarize floating-point descriptors so that they can be searched for quickly. Unfortunately, even if the similarity between vectors can be computed fast, exhaustive linear search remains impractical for truly large databases and Approximate Nearest Neighbor (ANN) search is still required. It is therefore surprising that relatively little attention has been paid to the efficiency of ANN algorithms on binary vectors and this is the focus of this paper.

We first show that binary-space Voronoi diagrams have thick boundaries, meaning that there are many points that lie at the same distance from two random points. This violates the implicit assumption made by most ANN algorithms that points can be neatly assigned to clusters centered around a set of cluster centers. As a result, state-of-the-art algorithms that can operate on binary vectors exhibit much lower performance than those that work with floating point ones.

The above analysis is the first contribution of the paper. The second one is two effective ways to overcome this limitation, by appropriately randomizing either a tree-based algorithm or hashing-based one. In both cases, we show that we obtain precision/recall curves that are similar to those than can be obtained using floating point number calculation, but at much reduced computational cost.

Keywords: Approximate Nearest Neighbor Search, Binary vectors, Locality Sensitive Hashing, Hierarchical k-means

1. Introduction

The problem of matching high-dimensional descriptors against large databases is pervasive in Computer Vision, *e.g.* in image-retrieval, or pose-estimation. When there are millions of such descriptors, linear search becomes prohibitively expensive, even after dimensionality reduction [1, 2] and no generic, exact, and more efficient algorithm is known.

Approximate Nearest Neighbor (ANN) search constitutes one effective approach to overcoming this limitation and there are many algorithms that can handle real-valued descriptors such as the Scale Invariant Feature Transform (SIFT) [3] or Speeded Up Robust Feature (SURF) [4] descriptors. These algorithms rely on modified kd-trees [5, 6], multiple randomized kd-trees [7], hierarchical k-means (HKM) trees [8, 9], spill trees [10], vantage-point trees [11], or hashing functions [12]. A different approach to speeding up nearest-neighbor search is to binarize the real-valued descriptors using techniques such as Boosting [13], hashing [12, 14], Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA) based methods [15, 16], quantization [17] and Semantic or Spectral Hashing [18, 19]. Because the similarity between the resulting binary vectors can be evaluated using the Hamming distance, which can be computed much faster than the Euclidean one on modern CPUs, linear search is more efficient but remains too slow for large-scale applications. In favorable cases, the binary vectors can be used as indices to directly access their nearest neighbors [19] which provides sub-linear complexity of the search. Unfortunately, this stops being possible when the typical Hamming distance between nearest neighbors is larger than a few units.

To get the best of both worlds under general conditions and to exploit the potential of binary descriptors, ANN search is necessary. Little attention has been paid to the performance of ANN algorithms on binary, as opposed to real-valued, vectors. Some of the algorithms discussed above such as Spectral Hashing are not adapted to binary vectors because they involve a PCA decomposition. Other methods can be used by treating binary descriptors as vectors of zeros and ones encoded as floating-point numbers. Even with the same search-accuracy, this encoding negates the advantages of binary vectors over real-valued ones: their compactness and the fact that the Hamming distance can be computed faster than the Euclidean one. Finally, there are algorithms such as vantage-point trees and HKM that can be modified to only deal with binary vectors and use the Hamming distance as a similarity

measure. However, as we show, their accuracy is much lower.

The first contribution of this paper is to show that this performance loss can be traced to the fact that in Hamming spaces, unlike in Euclidean ones, the number of points that lie at the same distance from two random points, *i.e.* the points lying at the boundary of a Voronoi diagram, encompass a large proportion of the space. In other words, the Voronoi diagram has thick boundaries. This breaks the assumption made by many ANN algorithms that points can be unambiguously clustered with their closest neighbors. This phenomenon is different from the well-known *curse of dimensionality* which becomes apparent only for the high dimensional data [20]. In the case of binary spaces, the thick boundaries of the Voronoi diagram influence the search regardless of the data dimensionality.

From this understanding comes the second contribution of the paper, an effective way to overcome the above mentioned problems inherent to Hamming spaces by creating multiple randomized data structures. Randomization produces structures that are independent from each other and therefore complementary. It solves the thick boundary problem and yields results similar to those obtained by converting the vectors to floating point values, but at a fraction of the computational cost. We instantiate this idea in two different ways, the first inspired by HKM trees and the second by the Locality-Sensitive Hashing scheme originally proposed for integer vectors [21]. In the first case, we replace the cluster centroids computed at each level of the tree by randomly chosen points and create multiple trees in this manner. In the second, we introduce an improved mechanism for selecting random subsets of coordinates used to index the vectors.

2. Related Work

Local descriptors are high-dimensional vectors describing local regions extracted from images, and used in many applications of Computer Vision. In many of those applications, we need to match keypoints extracted from images against a large database as we do in our experiments: The distances between descriptors of keypoints corresponding to the same 3D point should be small. While the first descriptors were real-valued vectors, a recent trend focuses on *binary* descriptors, as they are more compact and distances between them can be evaluated efficiently. They can be computed directly from the images [22], or from real-valued descriptors as in Locality Sensitive Hashing [12], Semantic or Spectral Hashing (SH) [18, 19], or LDAHash [16].

SH was designed to create binary vectors that can be used as table indices to directly access their nearest neighbors. However, as shown in Fig. 1, applying it to SIFT descriptors yields too large average Hamming distances between nearest neighbors to be practical. LDAHash [16] produces average distances that are smaller but still too large.

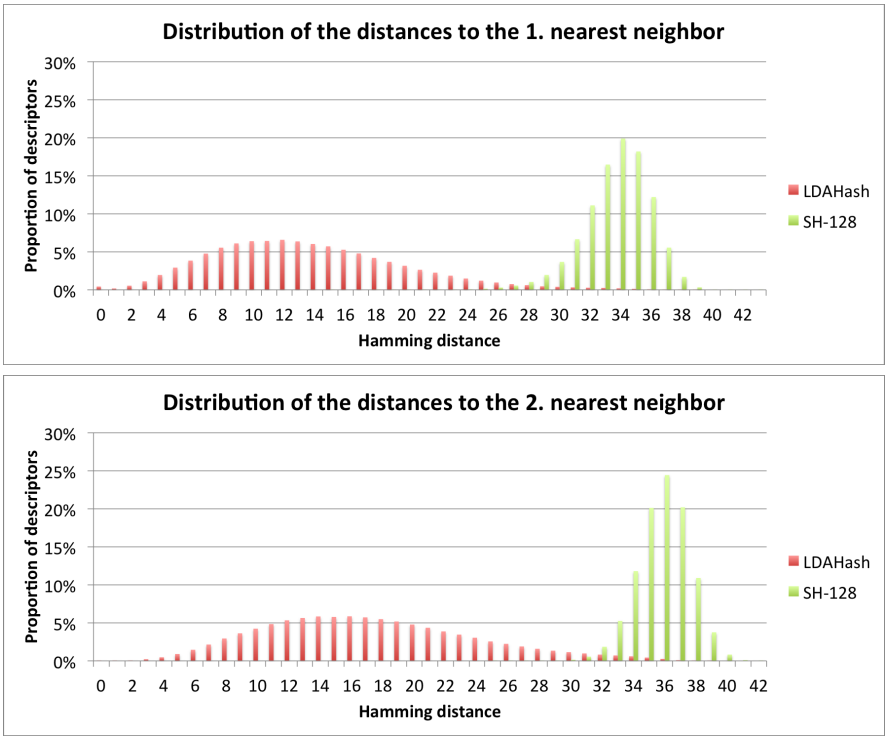


Figure 1: Comparison of the distributions of distances from the descriptor to its first and second nearest neighbors in the Hamming spaces generated using LDAHash and SH on our 500k database. The average Hamming distance between descriptors is larger than 1 for both LDAHash and SH-generated 128-bit descriptors. However, because the distances are spread more widely for LDAHash vectors, all ANN algorithms tend to perform better on those, which is not all that surprising since SH was designed for a different purpose.

In short, even when using sophisticated binary descriptors, quickly querying large databases still requires effective ANN methods. Even though Nearest Neighbor search has been widely discussed in the literature, no known generic algorithm is both exact and more efficient than brute force search.

Many efficient approximate algorithms have been proposed for large-scale search. According to a recent comparative study [24], the best ones for querying large databases are the randomized kd-trees [7] and hierarchical k-means tree algorithm [8, 9].

The randomized kd-trees [7] are a recent modification of the original kd-trees [25], which involved building a tree by recursively splitting in half along the dimension in which it exhibits the greatest variance. This performs well in low-dimensional spaces but looses its effectiveness as dimensionality increases [26]. To prevent this, *sets* of randomized kd-trees can be built by recursively splitting along dimensions randomly chosen among the first D dimensions of greatest variance. Combining several trees with different splits mitigates the effects of quantization errors. Unfortunately, as we show in Section 3, this is a brittle technique when applied to binary vectors because a query vector can be moved to the wrong branch if only one of its bit is flipped, *e.g.* due to noise.

The hierarchical k-means tree [8, 9] represents another successful alternative to brute force search. It recursively uses the k-means algorithm to split the data into k clusters. At run-time, a query vector follows the branch that corresponds to the closest centroid and back-tracking can be invoked to explore several leaves. Hierarchical k-means rely on means of vectors, which is problematic when dealing with binary vectors as we also see in Section 3.

Vantage-point trees [11] avoid the need to compute means by recursively picking a single vector among the data that reaches a node and splitting the others into those that are closer and those that are further. As we will show in Sec. 3, this method also performs poorly on binary vectors.

In short, state-of-the-art ANN techniques work well on real-valued vectors but not on binary ones. Furthermore, there is little work in connection to the latter. In [27], an Additive Binary Tree (ABT) is associated to each binary vector. Each one of its nodes contains the frequency of 1's in a subpart of the vector and this structure is used to stop the computation of the distance between two vectors early when the match is not promising. This approach, however, is still linear in the size of the database, and the speed gain is not clear compared to the full computation of the Hamming distance on modern hardware. In [23], the database is represented by a 256-ary tree in which each node corresponds to one byte of the vector. The parts of the tree that contain only one vector are pruned and replaced by a single leaf. This approach is sensitive to noise as changing a single bit may change how the branches are explored. In [28], vectors are represented by a number of

Table 1: Precisions when looking for the first and second nearest neighbors for different methods and comparable query times, approximately 0.2 ms per query, for a dataset of 500k descriptors. The performances of state-of-the-art methods drop when they are applied on the binary 128-vectors obtained by running LDAHash [16] on SIFT vectors. This is especially noticeable for the HKM algorithm when the centroids are forced to be binary vectors. The Parc-trees and Uniform LSH are two methods we introduce in Section 4 to avoid this loss of accuracy.

Precision for	**first position**		**second position**	
	SIFT descriptors	binary descriptors	SIFT descriptors	binary descriptors
hierarchical k-means with real-valued centroids	0.94	0.82	0.93	0.79
kd-trees	0.98	0.78	0.97	0.75
hierarchical k-means with binary centroids	-	0.32	-	0.29
vantage-point trees	0.35	0.17	0.31	0.15
parc-trees	0.94	0.91	0.91	0.92
Original LSH for binary vectors [21]	-	0.92	-	0.95
Uniform LSH	-	0.93	-	0.96

random permutations of bits. For each permutation, the vectors are sorted in a lexicographic order and when the query comes, the binary search is find the closest vectors. Although this method provides a sub-linear complexity, the memory required to store the sorted lists is a multiple of the dataset size. Moreover, it is reported to provide identical performance to the original LSH [21] which is much more memory efficient.

3. Thick Borders and Performance Loss

In this section, we first demonstrate that state-of-the-art ANN algorithms directly applied to binary vectors perform worse than when applied to floating-point ones. We then show that Voronoi diagrams have thick boundaries in binary spaces, which is what causes this performance drop.

3.1. ANN on Binary Vectors

To perform the comparison, we collected many images of Venice from the Flickr [1] database and created a first dataset containing 500k feature points and their SIFT descriptors. We then binarized these using the publicly available implementation of LDAHash into 128-bits vectors [16] whose length was shown to provide a good compromise. We used exhaustive linear-search to find the closest neighbors of each descriptor and we use this information as a ground-truth.

Table 1 summarizes our precision results for the first and second positions. The first simply is the the percentage of correct nearest neighbor that are retrieved. The second is computed by retrieving two nearest neighbors and checking whether both, only one, or none are the correct first two nearest neighbors of the query. The average proportion of correct matches divided by 2 is then taken to be the precision at the second position.

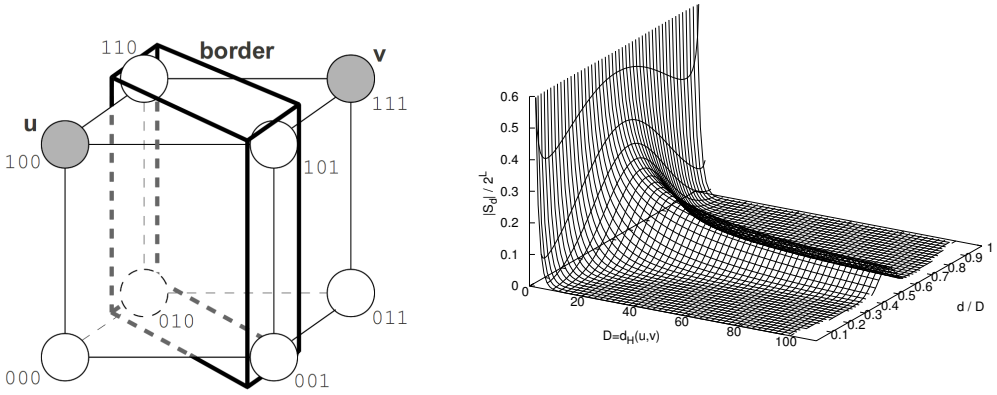
The results for the kd-trees and HKM algorithms were obtained using the publicly available code of the FLANN library [24], which automatically optimizes the algorithms parameters. We used our own implementation of the vantage-point trees.

The kd-trees and vantage-point trees can work on binary vectors without any modification since they do not involve averaging. By contrast, HKM involve computing centroids. We therefore tested two different versions of the algorithm, either rounding the coordinates of the centroids so that they remain binary vectors or using the floating-point coordinates.

As a baseline, we plot in the first column the results for matching the SIFT floating-point vectors. In the second column, we plot the systematically worse equivalent results using binary vectors. The degradation is noticeable for kd-trees and HKM, even if we treat binary vectors as floating-point ones. The performance drop is even more noticeable for the vantage-point trees.

As a sanity check, even though Spectral Hashing [19] is not truly designed to produce vectors that can be searched by ANN but rather used as indices to directly access their nearest neighbors, we ran the same series of tests on the vectors it produces. The ANN precision rates are globally lower but we observed the same behavior.

[1] `http://www.flickr.com`



(a) (b)

Figure 2: Thick borders of Voronoi diagrams in binary space. (a) A significant proportion of the space is equidistant from two arbitrary points. In this example, four vectors are equidistant to the vectors $\mathtt{u}$ and $\mathtt{v}$ which accounts for half of the population. This makes the HKM algorithm fail on binary vectors. (b) Proportion of the binary vectors $\mathtt{w}$ that belong to the sets S_d defined in Eq. (1) as a function of the distance $D = d_H(\mathtt{u}, \mathtt{v})$ and d. It is maximal for $d = D/2$, which corresponds to the set of vectors equidistant to $\mathtt{u}$ and $\mathtt{v}$, and remains large even for large values of D. This phenomenon differs from the *curse of dimensionality* as it affects the data regardless of its dimensionality.

3.2. Interpretation

That kd-trees perform poorly on binary vectors is not that surprising since the splits are performed one dimension at a time and binary vectors can take only two values per dimension. Hence this method is sensitive to flip noise.

To understand the performance drops for the HKM and the vantage-point tree, one must consider that the topology of the Hamming space is different than the Euclidean one. This is because of the discrete nature of the binary spaces where many vectors are *equidistant* to two random points.

This affects the vantage-point trees because many vectors may lie on the splitting sphere: If the dimensionality of the binary space is L and the sphere radius is D, the proportion of uniformly distributed vectors that lie on the sphere boundary is $\frac{1}{2^L}\binom{L}{D}$. For example, for $L = 16$ and $D = 8$, this

represents 20% of the binary space, an enormous fraction. This is problematic because the algorithm depends on the assumption that the splits separate the data well.

The same thing happens with the HKM trees, especially when one binarizes the centroids. As explained below, the boundaries of the Voronoi diagram defined by such binarized centroids contain a significant proportion of the binary space. This is detrimental to the algorithm because points in those thick boundaries can be arbitrarily assigned to one or the other cluster and can fall down the wrong branch of the tree at run-time.

Let us consider two L-dimensional binary centroids $\mathtt{u}$ and $\mathtt{v}$. We would like to evaluate the number of vectors around the boundary defined by $\mathtt{u}$ and $\mathtt{v}$, that is, around the hyperplane made of $\mathtt{w}$ vectors equidistant from $\mathtt{u}$ and $\mathtt{v}$. To this end, let us consider the cardinality of the sets S_d defined by:

$$S_d = \{\mathtt{w} \text{ such that } d_H(\mathtt{v},\mathtt{w}) = d_H(\mathtt{u},\mathtt{w}) + D - 2d\}\ , \tag{1}$$

where $d_H(\cdot,\cdot)$ is the Hamming distance, and D the Hamming distance between $\mathtt{u}$ and $\mathtt{v}$. The S_d family spans the Hamming space, with $\mathtt{u} \in S_0$, $\mathtt{v} \in S_D$, and $S_{D/2}$ the set of vectors vectors equidistant from $\mathtt{u}$ and $\mathtt{v}$.

$\mathtt{u}$ and $\mathtt{v}$ have $L-D$ bits in common and D bits that are different. Let us first consider the case when D is even. For a vector $\mathtt{w}$ to belong to S_d, d bits among the D different bits must be changed between $\mathtt{u}$ and $\mathtt{w}$. In addition any number n of bits among the $L-D$ common bits can also be flipped between $\mathtt{u}$ and $\mathtt{w}$: We then have $d_H(\mathtt{u},\mathtt{w}) = n + d$ and $d_H(\mathtt{v},\mathtt{w}) = n + D - d$, and $\mathtt{w}$ indeed belongs to S_d.

The number of possible such $\mathtt{w}$ vectors is therefore $2^{L-D}\binom{D}{d}$ and their proportion of the full space is $\frac{2^{L-D}}{2^L}\binom{D}{d} = \frac{1}{2^D}\binom{D}{d}$. Remarkably this expression does not depend on the dimension L of the space but only on D and d. We plot its values in Fig. 2(b).

This expression reaches its maximum for $d = D/2$, that is, for the set of vectors that lie at equal distance from vectors $\mathtt{u}$ and $\mathtt{v}$. Using Stirling's approximation [30], this expression for $d = D/2$ can be approximated by $\sqrt{\frac{2}{\pi D}}$ when D increases, and therefore slowly decreases towards 0 (see Fig. 2(b)). For example, when $D = 2$, 50% of the space lies at equal distance from the 2 centroids! For $D = 64$, 10% of the space is still equidistant from the centroids. As a result, the borders of the Voronoi diagram defined by $\mathtt{u}$ and $\mathtt{v}$ contain a significant proportion of the binary space which leads to a severe performance drop of the HKM algorithm.

When D is odd, no vector is equidistant from `u` and `v`. However, we can derive a similar expression for the number of points for which the distances to `u` and `v` differ by 1. The number of such points remains large. The borders of the Voronoi diagram defined by the centroids in the HKM algorithm therefore contain a significant proportion of the binary space.

4. Randomized Data Partitioning

In this section, we address the above-mentioned shortcoming of state-of-the-art ANN search algorithms in Hamming spaces, and describe two simple yet effective ANN search methods that work by randomizing data partitioning.

4.1. Parc-Trees

This first algorithm relies on multiple trees. Like the nodes of a hierarchical k-means (HKM) [8, 9] tree, the parc-tree nodes split the data into k parts by storing k vectors we call *centroids*, and associating the data with the closest centroid. Each non-terminal node has k children, corresponding to the different parts. By contrast with HKM, we do not optimize on the centroids but randomly select them among the data vectors that reach the node, except those which have previously been used. The recursion stops when the number of data vectors is less than k. Because of the randomization, the trees are independent from each other.

At run-time, a query vector recursively follows the branch associated to the closest node vector until it reaches a leaf, as in HKM. In HKM however, the leaves have to store all the data that reach them, and a linear search over this data is required to find the vector closest to the query vector as the candidate nearest neighbor. In parc-trees, the centroids belong to the dataset and when the query vector reaches a leaf, we already computed its distances to the centroids of the nodes it visited. Hence, a candidate nearest neighbor is chosen to be the closest vector among those in the leaf and the centroids of the visited nodes.

The query operation is repeated over all T trees and the best match is retained. This allows the parc-trees to mitigate the quantization error introduced by the thick Voronoi boundary: We can find the correct nearest neighbor even if it is present in only one visited node among all the trees.

The influence of parameters T and k on the obtained precision and computational time can be seen in Fig. 3. The performance increases with the

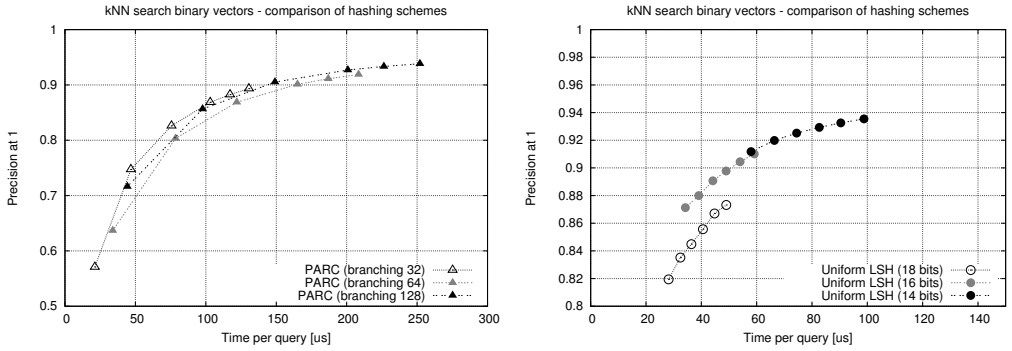


Figure 3: Comparison of precision for first position with different parameters for the parc-trees and Uniform LSH on the 500k binary vectors dataset. For PARC trees (left) we plot the results for $T = 8, 16, 24, 32, 36, 40$ trees and different branching factors. For Uniform LSH (right) we plot the results for $30, 35, 40, 45, 50, 55, 60$ hashing tables with various key lengths.

number of trees T, until it saturates, linearly with T. Increasing the branching factor k also improves the performance. The average tree depth then decreases, but the computation time still increases: A tree of depth d contains $k + k^2 + k^3 + \ldots k^d \approx k^d$ vectors, so a tree of S data vectors is approximatively of depth $\log S / \log k$. The number of distance computations required when dropping a query vector into the tree is therefore $k \log S / \log k$, which increases with k sublinearly.

Most of the operations involved by this approach are Hamming distance computations. They amount to an `xor` operation followed by a `popcnt` instruction present on modern CPUs, and are much faster to evaluate than the Euclidean distance between floating-point vectors. Moreover, the T trees can be simultaneously queried on a multi-core machine, which means we incur only a limited penalty for using several trees.

4.2. Uniform LSH

We also developed an ANN method inspired by the Hashing-based method of [21], which involves converting integer vectors into binary ones and randomly selecting and concatenating bits from them to generate multiple hashing keys. A query vector is then matched against the vectors in the buckets corresponding to its keys values by linear search. The smaller the lengths of the keys, the greater the size of the buckets becomes, which yields higher precision at the cost of increased computational time. This simple scheme

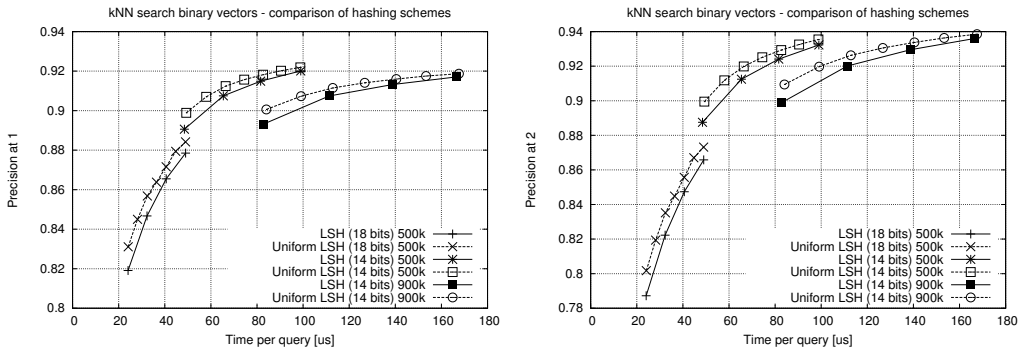


Figure 4: Comparison of precision for first (left) and second (right) positions for the original LSH and Uniform LSH with hashing keys optimized as explained in Section 4.2. We varied the number of keys, their lengths and the sizes of the datasets. While the improvement is limited, this demonstrates that the hashing keys can be optimized in Hamming spaces.

performs well, as our experiments show. As for parc-trees, most of the operations are Hamming distance evaluations, which can be performed efficiently, and searches, which can be parallelized.

However, the random selection of coordinates may lead to unnecessary overhead, as some coordinates may be selected more frequently than others, whereas some of them may not be picked at all. This problem can be solved by increasing the number of keys, but to run fast, it is desirable to use as few keys as possible.

To resolve this dilemma, we optimize the keys so that the bits selected to generate the keys are distributed more uniformly. This way, the keys generate more various partitioning of the database. More formally, we can define the keys K_i as sets containing the selected bits coordinates: $K_i = \{c_{ij} \in [1; L] \mid j \in [1; n]\}$ where L is the dimensionality of the binary vectors, n is the number of bits in a key. We also define N_k as the number of times a given coordinate is used in a key: $N_k = |\{c_{ij} = k \mid i \in [1; m] \text{ and } j \in [1; n]\}|$, where m is the number of keys. Then we optimize the keys to minimize

$$\min_{\{K_i\}} \sum_k (N_k - N)^2 \tag{2}$$

with $N = n \cdot m/L$, the ideal number of times a coordinate should be picked. To do this, we use a simple greedy algorithm that generates the keys one by one, by randomly selecting the bits among those which were used less

often for the previous keys. We call this modification Uniform LSH as the distribution of the bits used in the keys is optimized to be more uniform and hence partition the dataset better.

We experimented with different numbers of keys and numbers of bits per key used. The results of those experiments are shown in Fig. 3. Computation times increase linearly with the number of keys, but the precision of the search also increases. Similarly, the lower the number of bits per key used, the bigger the data partitions and hence the longer the search time. However, since we perform a linear search within the selected data partition, shorter keys (and bigger data partitions) lead to performance improvement.

Overall, as it can be seen in Fig. 4, the resulting Uniform LSH algorithm improves performances over those of the original LSH.

5. Results

In this section, we compare parc-trees and uniform LSH against kd-trees and HKM trees. The results were obtained for the 500k Venice dataset of 128-bit binary descriptors from Section 3.1. To see if the results hold also for bigger datasets, we created two more datasets containing 900k and 1.5M binary descriptors which were generated by binarizing the SIFT descriptors extracted from more Flickr images of Venice. We draw the plots by setting the parameters of all algorithms so that the query time is approximately the same. To produce the different points in the plots, the number of hashing tables of LSH varied from 30 to 60. The test datasets are larger than the ones used to evaluate the recent FLANN library [24] which mostly contained only 100k vectors. Furthermore, datasets of comparable sizes are frequently used for real-life applications, such as image-based 3D reconstruction. The results presented here are the average over three runs. The computation times were evaluated on a computer with two Intel Xeon E5620 2.4 GHz CPUs and 48 Gb RAM.

Fig. 5 presents the comparison of different ANN search algorithms applied to binary descriptors. LSH outperforms all tree-based methods. Out of those, parc-trees remain the best. The speed-up of LSH and parc-trees over the other algorithms is especially visible for higher precision levels. For instance, for 500k dataset KD-trees needs approximately $300\mu s$ to reach the precision at second position equal to 0.85, whereas it takes parc-trees and Uniform LSH less than $100\mu s$ and $50\mu s$, respectively. As the dataset size grows, it takes more time to find the nearest neighbors, but the relative ordering of the

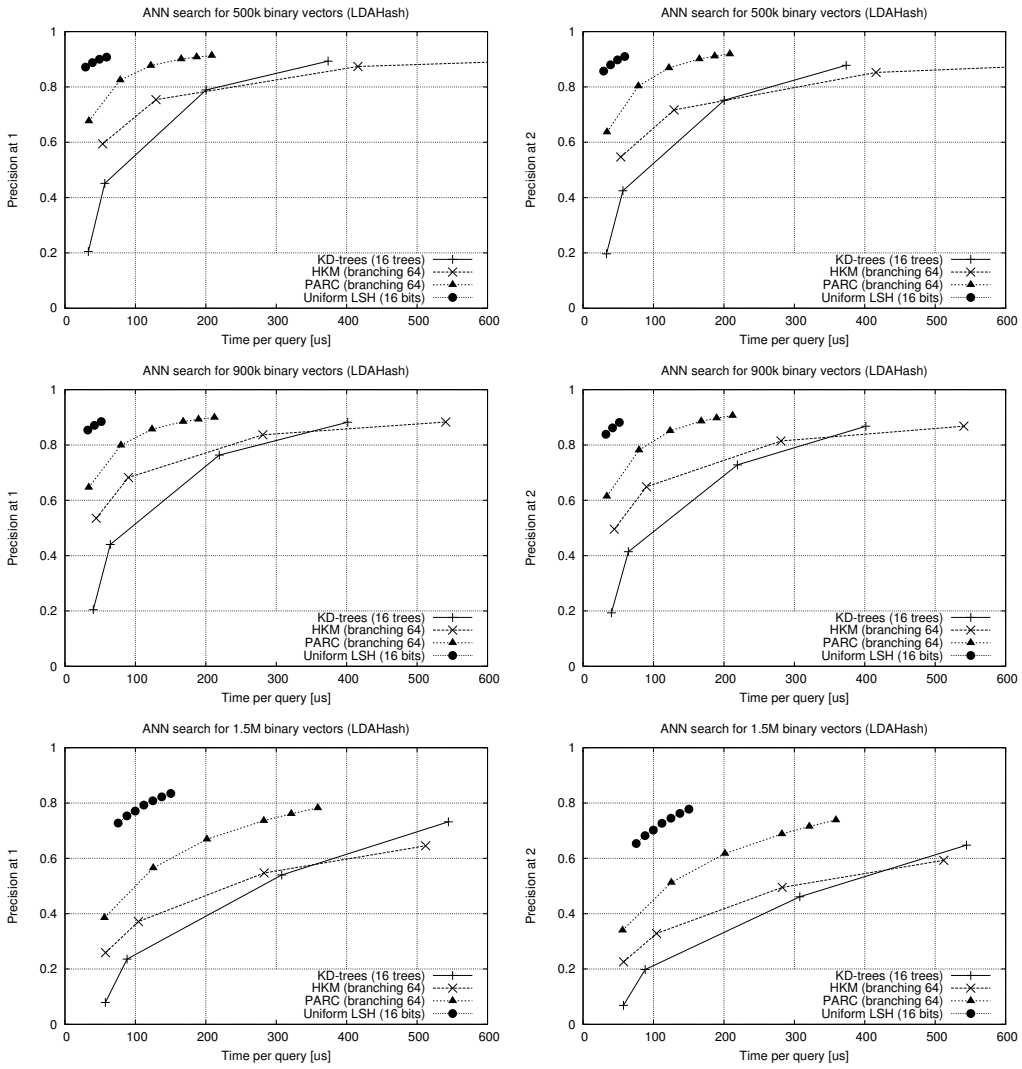


Figure 5: Comparison of precision for first and second positions for different ANN search algorithms on 500k, 900k and 1.5M binary vector datasets. Uniform LSH outperforms the parc-trees, which themselves outperform all the other state-of-the-art methods for all configurations.

performances remains the same for all the methods: LSH performs the best, followed by the parc-trees. For the 1.5M dataset, LSH achieves a precision of 0.7 at first position about an order of magnitude faster than KD-trees and HKM.

To verify our approach, we applied it to Aerial Triangulation. We ex-

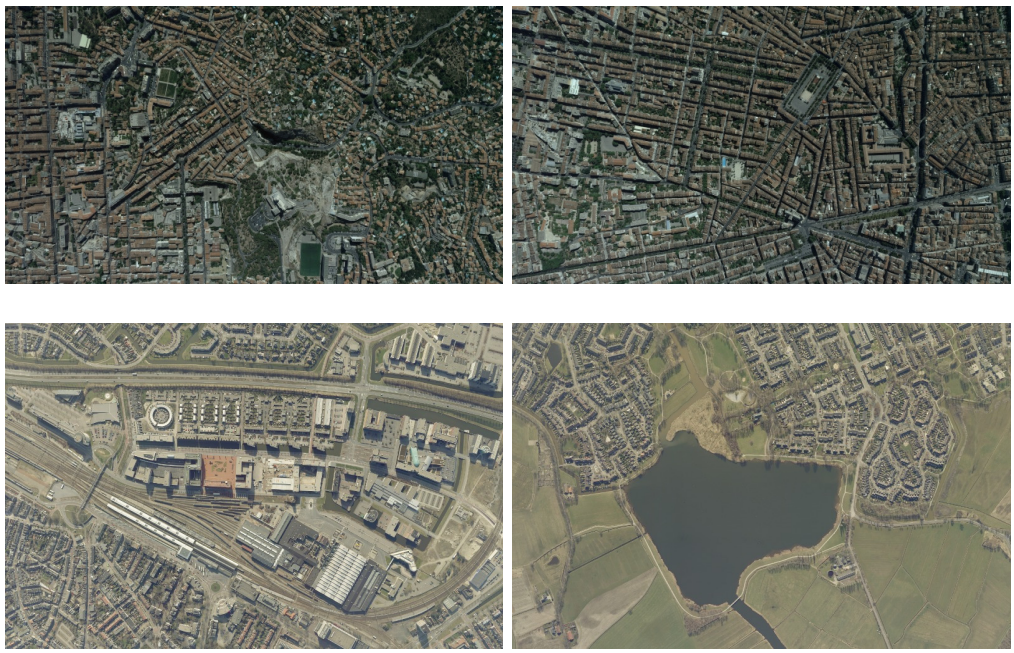


Figure 6: **Top:** Two out of the 25 13824×7680 pixels images from the Marseilles dataset. **Bottom:** Two out of the 68 11500×7500 pixels images from the Zwolle dataset.

tracted feature points from aerial images, and match them using the same binary descriptors as before. Matched points correspond to the same 3D points, and we use these matches to jointly optimize the 3D points and the camera parameters by bundle block adjustment [29].

We tested our binary search strategy on two datasets of large aerial images. The first dataset contains 25 high resolution (13824×7680) images of Marseilles [2], two of which are shown in Fig. 6. The second dataset consists of 68 11500×7500 aerial images of the Dutch city of Zwolle.

Each image contains approximately 400k binary keypoints which makes exhaustive feature matching, even on binary vectors, excessively slow. Our approach reduces the matching time by a factor 20 over linear search with a 95% accuracy, which is consistent with the results reported in Section 5. The

[2]From "Benchmarking of Image Matching Approaches for DSM computation" http://eurosdrbenchmarkofimagematching.ign.fr/

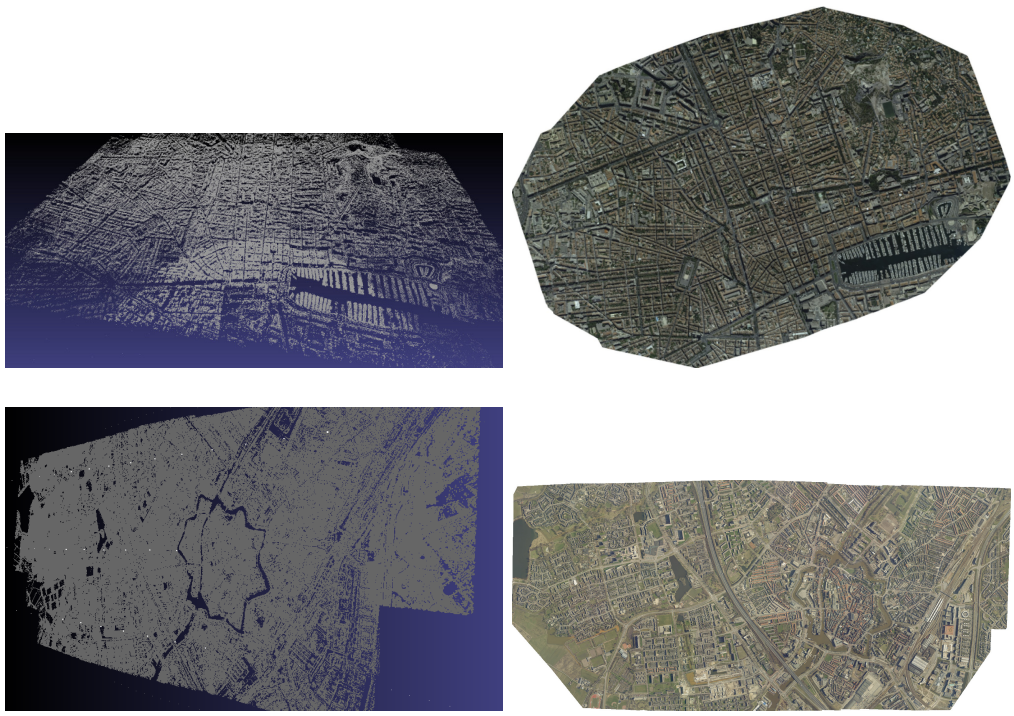


Figure 7: **Top left:** The aerial triangulation of 1.1M 3D points and **top right:** the generated ortho-image for the Marseilles dataset. **Bottom left:** The aerial triangulation of 2.1M 3D points and **bottom right:** the ortho-image made of 68 individual images (right) for the Zwolle dataset.

final aerial triangulations and the combined ortho-images for the Marseilles and Zwolle datasets are shown in Fig. 7.

6. Conclusion

We showed that Voronoi diagrams in Hamming spaces have thick borders, which reduces the precision of state-of-the-art ANN algorithms. We then proposed two techniques that rely on randomized data partitioning to overcome this problem and yield precisions that are comparable to those obtained using floating-point vectors at a fraction of the computational cost.

[1] K. Mikolajczyk, C. Schmid, A. Zisserman, Human Detection Based on a Probabilistic Assembly of Robust Part Detectors, European Conference on Computer Vision, 2004, 69–81.

[2] M. Brown, G. Hua, S. Winder, Discriminative Learning of Local Image Descriptors, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (1), 2011, 43–57.

[3] D. Lowe, Distinctive Image Features from Scale-Invariant Keypoints, International Journal of Computer Vision 20 (2), 2004, 91–110.

[4] H. Bay, T. Tuytelaars, L. Van Gool, SURF: Speeded Up Robust Features, European Conference on Computer Vision, 2006.

[5] J. Beis, D. Lowe, Shape Indexing Using Approximate Nearest-Neighbour Search in High-Dimensional Spaces, Conference on Computer Vision and Pattern Recognition, 1997, 1000–1006.

[6] S. Arya, D. Mount, N. Netanyahu, R. Silverman, A. Wu, An Optimal Algorithm for Approximate Nearest Neighbor Searching Fixed Dimensions, Journal of the ACM 45, 1998, 891–923.

[7] C. Silpa-Anan, R. Hartley, Optimised kd-Trees for Fast Image Descriptor Matching, Conference on Computer Vision and Pattern Recognition, 2008.

[8] K. Fukunaga, P. Narendra, A Branch and Bound Algorithm for Computing K-Nearest Neighbors, IEEE Transactions on Computing 24, 1975, 750–753.

[9] D. Nister, H. Stewenius, Scalable Recognition with a Vocabulary Tree, Conference on Computer Vision and Pattern Recognition, 2006.

[10] T. Liu, A. Moore, A. Gray, K. Yang, An Investigation of Practical Approximate Nearest Neighbor Algorithm, Advances in Neural Information Processing Systems, 2004.

[11] P. Yianilos, Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces, Fourth ACM-SIAM Symposium on Discrete Algorithms, 1993.

[12] A. Andoni, P. Indyk, Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions, Communications of the ACM 51 (1), 2008, 117–122.

[13] G. Shakhnarovich, Learning Task-Specific Similarity, Ph.D. thesis, Massachusetts Institute of Technology, 2005.

[14] B. Kulis, T. Darrell, Learning to Hash with Binary Reconstructive Embeddings, Advances in Neural Information Processing Systems, 2009,1042–1050.

[15] M. Raginsky, S. Lazebnik, Locality-Sensitive Binary Codes from Shift-Invariant Kernels, Advances in Neural Information Processing Systems, 2009, 1509–1517.

[16] C. Strecha, A. Bronstein, M. Bronstein, P. Fua, LDAHash: Improved Matching with Smaller Descriptors, IEEE Transactions on Pattern Analysis and Machine Intelligence 34, 2012.

[17] Y. Gong, S. Lazebnik, Iterative Quantization: A Procrustean Approach to Learning Binary Codes, Conference on Computer Vision and Pattern Recognition, 2011.

[18] R. Salakhutdinov, G. Hinton, Semantic Hashing, International Journal of Approximate Reasoning 50 (7), 2009, 969–978.

[19] Y. Weiss, A. Torralba, R. Fergus, Spectral Hashing, Advances in Neural Information Processing Systems 21, 2009, 1753–1760.

[20] P. Indyk, R. Motwani, Approximate nearest neighbors: Towards removing the curse of dimensionality, Proceedings of the 30th Symposium on Theory of Computing, 1998, 604–613.

[21] A. Gionis, P. Indik, R. Motwani, Similarity Search in High Dimensions via Hashing, International Conference on Very Large Databases, 1999.

[22] M. Calonder, V. Lepetit, C. Strecha, P. Fua, BRIEF: Binary Robust Independent Elementary Features, European Conference on Computer Vision, 2010.

[23] M. Miller, M. Rodriguez, I. Cox, Audio Fingerprinting: Nearest Neighbor Search in High Dimensional Binary Spaces, Journal of VLSI Signal Processing Systems 41 (3), 2005, 285–291.

[24] M. Muja, D. Lowe, Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, International Conference on Computer Vision, 2009.

[25] J. Friedman, J. Bentley, R. Finkel, An Algorithm for Finding Best Matches in Logarithmic Expected Time, ACM Transactions on Mathematical Software 3 (3), 1977, 209–226.

[26] Y. Amit, D. Geman, B. Jedynak, Efficient Focusing and Face Detection, Face Recognition: From Theory to Applications, 1998, 143–158.

[27] S.-H. Cha, S. Srihari, Nearest Neighbor Search Using Additive Binary Tree, Conference on Computer Vision and Pattern Recognition, 2000.

[28] M. Charikar, Similarity Estimation Techniques from Rounding Algorithms, Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, 2002, 380–388.

[29] R. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, Cambridge University Press, 2000.

[30] J. Stirling, *Methodus Differentialis [electronic Resource] : Sive Tractatus De Summatione Et Interpolatione Serierum Infinitarum.*, J. Whiston and B. White, 1764.