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

Tomasz Trzcinski (tomasz.trzcinski@epfl.ch)
http://people.epfl.ch/tomasz.trzcinski

Vincent Lepetit (vincent.lepetit@epfl.ch)
http://cvlab.epfl.ch/~lepetit

Pascal Fua (pascal.fua@epfl.ch)
http://cvlab.epfl.ch/~fua

School of Computer and Communication Sciences
Swiss Federal Institute of Technology, Lausanne (EPFL)

Technical Report

June 7, 2011

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 very 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.

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

Contents

Chapter 1

Introduction

The problem of matching high-dimensional descriptors against large databases is pervasive in Computer Vision for applications such as image-retrieval, pose-estimation, and 3D-reconstruction. When there are millions of them, 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 SIFT or SURF. They rely on modified kd-trees [3, 4], multiple randomized kd-trees [5], hierarchical k-means (HKM) trees [6, 7], spill trees [8], vantage-point trees [9], or hashing functions [10]. A different approach to speeding up nearest-neighbor search is to binarize the real-valued descriptors using techniques such as Boosting [11], hashing [10, 12], PCA or LDA-based methods [13, 14], quantization [15] and Semantic or Spectral Hashing [16, 17]. 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 considerably more efficient but, nevertheless, remains too slow for large-scale applications. In favorable cases, the binary vectors can be used as indices to directly access their nearest neighbors [17] 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 to the full, it is therefore necessary to perform ANN on them. However, relatively 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 truly adapted to binary vectors because they involve computing a PCA decomposition. Others can be used by treating them as vectors of floating-point zeros and ones. However, even if the search-accuracy can be retained, this negates the advantage of binary vectors over real-valued ones, namely their compactness and the fact that the Hamming distance can be computed much faster than the Euclidean one. Finally, there are algorithms such as vantage-point trees and HKM that can be easily modified to only deal with binary vectors and use the Hamming distance as a similarity measure. However, as we will 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, that is, the points lying at the boundary of a Voronoi diagram, encompass a surprisingly large proportion of the space. In other words, the Voronoi diagram has very thick boundaries. This breaks the assumption made by many ANN algorithms that points can be neatly and unambiguously clustered with their closest neighbors. This phenomenon is essentially different from the well-known *curse of dimensionality* which becomes apparent only for the high dimensional data. In the case of binary spaces, the thick boundaries of the Voronoi diagram influence the search regardless of the data dimensionality, as we explain in details in Section 3.

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. This 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 [18]. 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.

In the remainder of the paper, we first discuss related work. We then show that obvious extensions of ANN methods to binary vectors either perform poorly or loose the advantages that accrue from binarization and we explain why. Finally, we evaluate the performance of our randomized ANN schemes and compare them against the state of the art.

Chapter 2

Related Work

We first briefly review the state-of-the-art in creating binary descriptors from images data and then discuss the main existing approaches to performing Approximate Nearest Neighbor search.

2.0.1 From Images to Binary Descriptors

Binary vectors can be computed directly from the images [19], or sound signals [20]. Alternatively, they can be derived from real-valued vectors using a hashing scheme such as Locality Sensitive Hashing (LSH) [10], Semantic or Spectral Hashing (SH) [16, 17], or LDAHash [14]. They all aim at generating compact binary representations that preserve the similarity between the original real-valued vectors. LSH is typically used to generate several randomized keys that index buckets in which linear search is performed. To avoid linear search, which can be costly if the bucket populations are large, Semantic and Spectral Hashing were designed to create binary vectors that can be used as table indices to directly access their nearest neighbors. This works remarkably well on GIST image descriptors [17] but is practical only when the typical Hamming distances between neighbors remain very small, that is, 0 or 1. However, this does not necessarily generalize. As we can see in Fig. 2.1, applying Spectral Hashing to SIFT descriptors computed on keypoints extracted from the image database of Section 3.1 yields much larger average Hamming distances between nearest neighbors. In practice, LDAHash [14], which relies on supervised training, produces average distances between nearest neighbors that are smaller but still too large to be used as indices.

In short, even when using sophisticated binary representations, quickly querying large databases still requires ANN effective methods.

2.0.2 Approximate Nearest Neighbor Search

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. For example, Cover Trees [21] can perform an exact search with sublinear complexity, but only if the data lies in a subspace.

Many efficient approximate algorithms have therefore been proposed for large-scale search. According to a recent comparative study [22], the best ones for querying large databases are the randomized kd-trees [5] and hierarchical k-means tree algorithm [6, 7]. In Section 3, we will test them on databases of binary vectors and show that their performance suffers severely in Hamming spaces.

The randomized kd-trees [5] are a recent modification of the original kd-trees [23], 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 quickly looses its effectiveness as dimensionality increases, in part because quantization errors can prevent the finding of vectors that are close to the splitting boundaries [24]. 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. In the implementation of [5], a priority queue is maintained across all

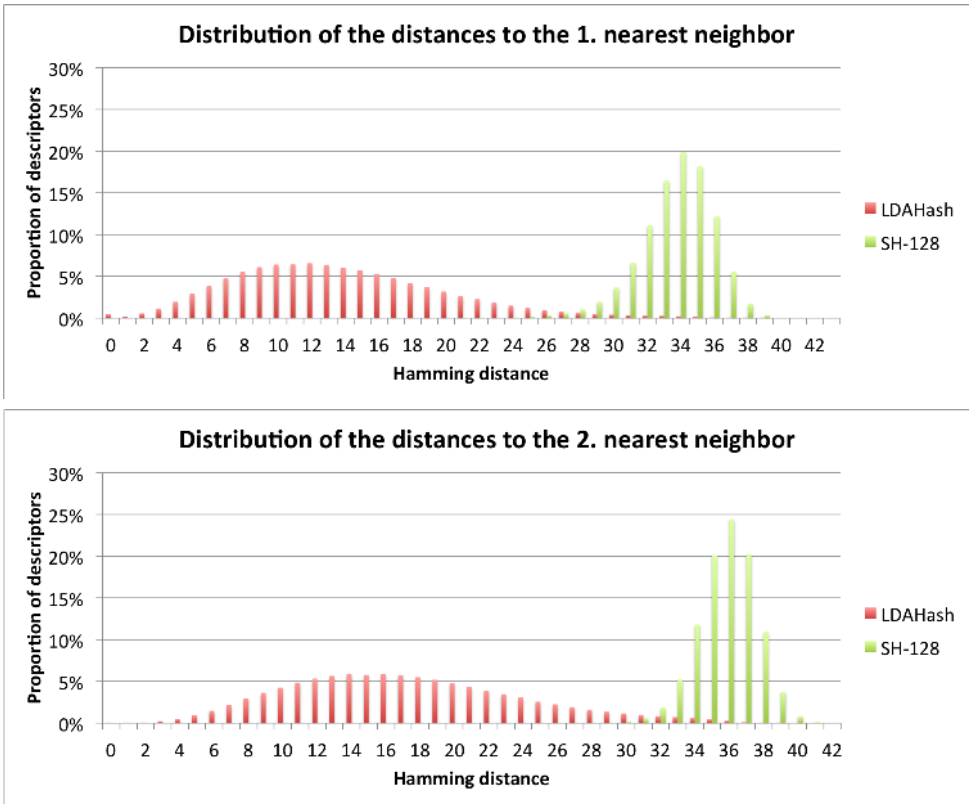


Figure 2.1: Comparison of the distributions of distances from the descriptor to its first and second nearest neighbors in the Hamming spaces generated using LDAHash and SH on our 500k database. The average Hamming distance between descriptors is larger than 1 for both LDAHash and SH-generated 128-bit 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.

randomized trees to order the search by increasing distance to each bin boundary. The trade-off between accuracy and speed is controlled by choosing the number of nodes to be visited. Unfortunately, as we will see in Section 3, this is a very brittle technique when applied to binary vectors because a query vector can easily be moved to the wrong branch if only one of its bit is flipped, for example due to noise.

The balanced box-decomposition (BBD) tree method [4] extends the kd-tree algorithm by constraining the ratio between the longest and shortest side of the data cells resulting from space partitions. Even though this improves results for some datasets, our initial experiments showed that it performs worse on binary vectors than a well designed k-means implementation [22].

The hierarchical k-means tree [6, 7] represents another successful alternative to brute force search. Instead of building a tree by recursively splitting in half the data that reach each node, it uses the k-means algorithm to split it into k clusters. When a vector is dropped down the tree, it follows the branch that corresponds to the closest centroid and back-tracking can be invoked to explore several leaves. Since finding the closest centroid can itself be slow, randomized kd-trees have been used to do it [25]. Hierarchical k-means rely on means of vectors, which is problematic when dealing with binary vectors as we will also see in Section 3. A very recent extension [26] to the k-means approach involves precomputing a nearest-neighbor graph and exploring it using a hill-climbing technique. In essence, it amounts to replacing the tree by the neighborhood graph. This makes the search more reliable as it cannot get stuck in a wrong branch. However, storing this graph involves large amounts of memory, which negates in part the rationale for using binary vectors, and does not deliver as large gains in speed over earlier techniques than the ones we report here.

Vantage-point trees [9] 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. However, this performs poorly on binary vectors because the number of vectors sitting at the border between the two groups is much larger than when using floating point ones, as discussed in Section 3. We believe the same phenomenon explains the difficulties we encountered with the BBD algorithm [4].

In short, state-of-the-art ANN techniques work well on real-valued vectors but not on binary ones. Furthermore, by contrast to the vast number of methods that have been designed for the former, there is very 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 sub-part of the vector while the length of the sub-part decreases as the node depth increases. 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 [20], 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. Even if breadth-first search were used, this approach is very sensitive to noise as changing a single bit may completely change how the branches are explored. In [28], a number of random permutations of bits is chosen to represent the vectors. For each permutation, the vectors are sorted in a lexicographic order and when the query comes, the binary search is performed to determine the vectors with the lowest Hamming distance. Although this method provides a sub-linear complexity of the ANN search, the memory required to store all the sorted lists is a multiple of the dataset size. Moreover, it is reported to provide identical performance to the original LSH [18] which is much more memory efficient.

Chapter 3

Thick Borders and Performance Loss

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

3.1 ANN on Binary Vectors

To perform the proposed comparison, we collected many images of Venice from the Flickr [1] database and created a first dataset containing 500k feature points and their associated SIFT descriptors. We then binarized these using the publicly available implementations of LDAHash into 128-bits vectors [14] whose length was shown to provide a good compromise between accuracy of mapping between the original and Hamming space and the length of the codeword. We used exhaustive linear-search to find the closest neighbors of each descriptor, which we then treat as the ground-truth against which we compare the output of the various approximate algorithms.

LDAHash serves as a representative of the class of techniques that produce binary vectors using a hashing scheme [10, 16, 17] whose performance compares well to theirs [14]. Furthermore LDAHash maximizes the information gain per bit, and produces uniformly distributed and uncorrelated strings of bits. This, in turn, poses significant challenge for the nearest-neighbor search algorithms as no value has any predictive information about any other value.

Table 3.1 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 hierarchical k-means algorithms were obtained using the publicly available code of the FLANN library [22], which automatically optimizes the algorithms parameters. We used our own implementation of the vantage-point trees.

The kd-trees and vantage-point trees can be made to work on binary vectors without any modification since they do not involve averaging. By contrast, the hierarchical k-means 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 allowing their coordinates to be floating-point numbers.

To give 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. More specifically, the degradation is very noticeable for the kd-trees and the hierarchical k-means, 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 [17] is not truly designed to produce vectors that can be searched by ANN but rather used as indices to directly access their nearest neighbors,

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

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 [18]	-	0.92	-	0.95
Uniform LSH	-	0.93	-	0.96

Table 3.1: 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 performance of state-of-the-art methods drop when they are applied on the binary 128-vectors obtained by running LDAHash [14] 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.

we ran the same series of test on the vectors it produces. The ANN precision rates are globally lower but the observed the same overall behavior.

3.2 Interpretation

That kd-trees perform poorly on binary vectors is not all that surprising when one considers that the splits are performed one dimension at a time and that binary vectors can take only two values per dimension. This makes this method very sensitive to flip noise, which is to be expected in binarized vectors.

To understand the performance drops for the hierarchical k-means and the vantage-point tree, one must consider that the topology of a space of binary vectors is quite different from that of real valued ones. This is in part because of the discrete nature of the binary spaces where many vectors are *equidistant* to two random points.

This severely 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 can separate the data well.

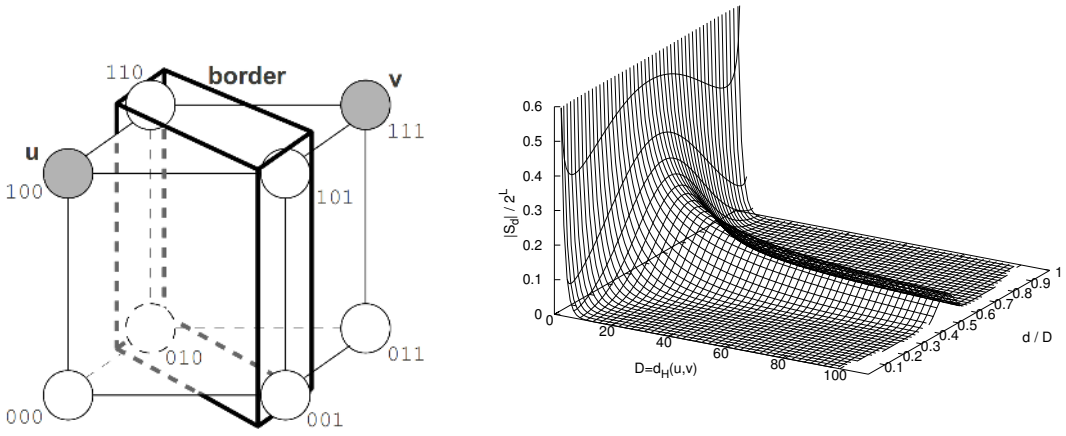
The same thing happens with the hierarchical k-means trees, especially when one binarizes the centroids. As we explain below, the boundaries of the Voronoi diagram defined by such binarized centroids contain a significant proportion of the binary space. This is detrimental to the algorithm because points in those thick boundaries can be arbitrarily assigned to one or the other cluster and, therefore, easily 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:

$$S_d = \{\mathtt{w} \text{ such that } d_H(\mathtt{v},\mathtt{w}) = d_H(\mathtt{u},\mathtt{w}) + D - 2d\}\,, \tag{3.1}$$

where $d_H(\cdot,\cdot)$ is the Hamming distance. 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 $\mathtt{v}$.

Let us denote the Hamming distance between $\mathtt{u}$ and $\mathtt{v}$ by D, $D = d_H(\mathtt{u},\mathtt{v})$. That means that $\mathtt{u}$ 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



(a) (b)

Figure 3.1: 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 u and v which accounts for half of the population. This makes the HKM algorithm fail on binary vectors. (b) Proportion of the binary vectors w that belong to the sets S_d defined in Eq. (3.1) 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 u and v, and remains large even for large values of D.

flipped between u and w: We then have $d_H(\mathtt{u}, \mathtt{w}) = n + d$ and $d_H(\mathtt{v}, \mathtt{w}) = n + D - d$, and w indeed belongs to S_d.

The number of possible such 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. 3.1(b).

This expression reaches its maximum for $d = D/2$, that is, for the set of vectors that lie at equal distance from the two vectors u and v. Using Stirling's approximation [29], it is easy to show that this expression for $d = D/2$ can be approximated by $\sqrt{\frac{2}{\pi D}}$ when D increases, and therefore decreases towards 0 but very slowly as shown in Fig. 3.1(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 exactly equidistant from the centroids. As a result, the borders of the Voronoi diagram defined by u ans v contain a significant proportion of the binary space which leads to 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.

Chapter 4

Randomized Data Partitioning

In this section, we address the above-mentioned shortcoming of state-of-the-art ANN search algorithms in Hamming spaces. We describe two simple yet effective ANN search methods that both work by randomizing data partitioning. We first introduce a tree-based algorithm that relies on a forest of decision trees with randomized centroids, which we will refer to as *parc-trees*. We then analyze the LSH method originally proposed for integer values mapped to binary strings [18] and show how it can be improved for the needs of similarity search in binary spaces, yielding a technique we will refer to as *Uniform LSH*.

4.1 Parc-Trees

The algorithm relies on a forest of trees. Each one operates on the same principle as a hierarchical k-means (HKM) [6, 7] tree: At each node, the data is split into k clusters according to its Hamming distance to cluster centers. However, unlike in the original HKM algorithm, these centers are randomly selected and their locations *not* optimized. Instead, we use several trees that are independent from each other because the centers are selected randomly.

Parc-trees are controlled by two parameters, T the number of trees and k the branching factor. Each node contains k vectors we will refer to as *centroids* by analogy to the standard HKM algorithm. They are randomly chosen among the data vectors that reach a node when dropped into the tree according to the same rules as the HKM algorithm, except those which have previously been used. The recursion stops when the number of data vector is less than k. At run-time, when a query vector is dropped down the tree, it recursively follows the branch associated to the closest centroid until it reaches a leaf. The closest vector among those in the visited nodes becomes a candidate nearest neighbor.

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.

Note that most of the operations involved by this approach are Hamming distances 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 without having to write any special-purpose code, which means we incur only a limited penalty for using several trees.

4.2 Uniform LSH

We also developed an ANN method directly inspired by the Hashing-based method of [18], which involves converting integer vectors into binary ones and projecting them on a randomly chosen set of coordinates 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 number of coordinates used, the greater the size of the buckets becomes, which yields higher precision at the cost of increased computational time. This simple scheme performs very well, as

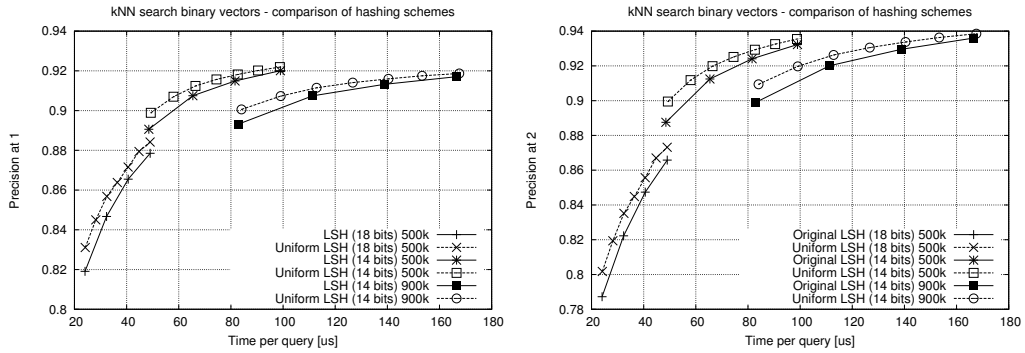


Figure 4.1: Comparison of precision for first and second positions for original LSH for binary vectors and Uniform LSH with hashing keys optimized as explained in Section 4.2. We varied the key lengths and the sizes of the datasets. While the improvement is limited, this demonstrates that the hashing keys can be easily optimized in Hamming spaces thanks to the discrete nature of the binary vectors.

our experiments show. As for parc-trees, most of the required operations are Hamming distance evaluations, which can be performed very 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 exploit the discrete nature of binary vector coordinates and the simple relation between the keys and the binary vectors to optimize the keys so that the picked coordinates 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 subsets of 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, and N_k 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 with a simple greedy algorithm

$$\min_{\{K_i\}} \sum_k (N_k - N)^2 \tag{4.1}$$

with $N = n \cdot m/L$, the ideal number of times a coordinate should be picked. We call this modification Uniform LSH. As it can be seen in Fig. 4.1, this improves performances.

Chapter 5

Results

In this section, we compare parc-trees and uniform LSH against kd-trees, hierarchical k-means trees, and LSH. The results were obtained for the 500k Venice dataset of 128-bit binary descriptors from Section 3.1 and two more datasets of sizes 900k and 1.5M created from a greater number of images. We draw the plots by setting the parameters of all algorithms so that the query time is approximately the same. Note that the datasets we use are significantly larger than the ones used to evaluate the very recent FLANN library [22] which was mostly done on the basis of only 100k vectors. Furthermore, datasets of comparable sizes are frequently used for real-life applications, such as large-scale image-based 3D reconstruction or aerial triangulation. All the computation times were evaluated on a computer with 2 Intel Xeon E5620 2.4 GHz CPUs and 48 Gb RAM.

Fig. 5.1 presents the comparison of different ANN search algorithms applied to binary descriptors. LSH outperforms all of the tree-based methods. Out of those, however, 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.

To confirm that our approach is practical, we applied it to Aerial Triangulation of sets of large images. Aerial Triangulation refers to the refinement of camera external parameters of image sets based on the image content and is well understood photogrammetry problem. We use the same binary descriptors as before to match points across images. These matches form tracks in successive images, each of which correspond to a 3D location. We used these tracks to jointly optimize the 3D points structure and the camera parameters by bundle block adjustment.

We tested our binary search strategy on two datasets of large aerial images. The first dataset contains 25 very high re'solution (13824×7680) images of Marseilles [1], two of which are shown in Fig. 5.2. 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 final aerial triangulations and the combined ortho-images for the Marseilles and Zwolle datasets are shown in Figs. 5.3 and 5.4 respectively.

To summarize, the obtained results confirm that some popular ANN search methods developed for real-valued vectors fail when applied to binary ones. By contract, LSH and parc-trees perform much better by relying on multiple randomized partitioning of the databases.

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

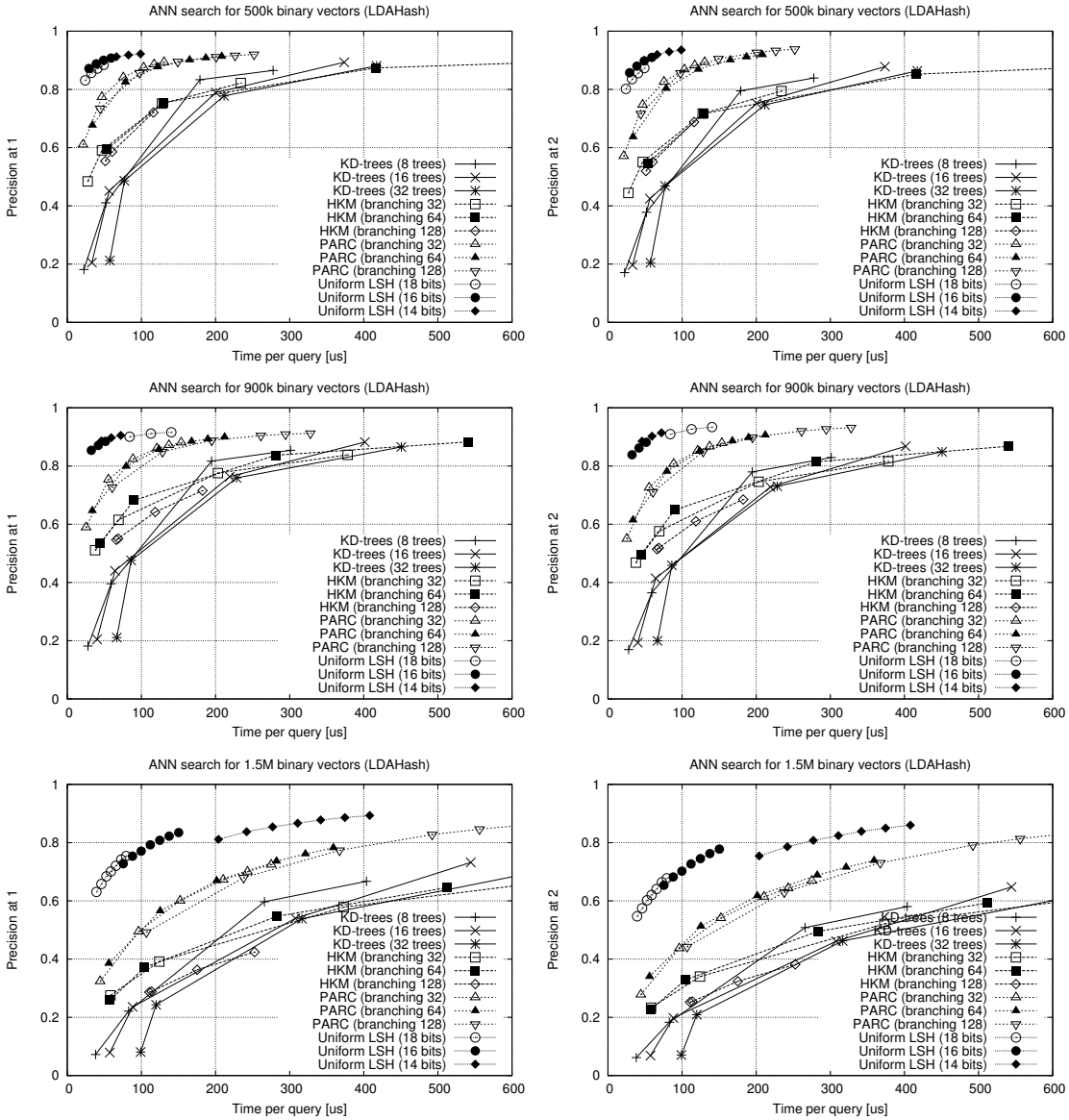


Figure 5.1: Comparison of precision at first and second positions for different ANN search algorithms and different parameter values on 500k, 900k and 1.5M binary vector datasets. Uniform LSH outperforms the parc-trees, which themselves outperform all the other state-of-the-art methods for all configurations.

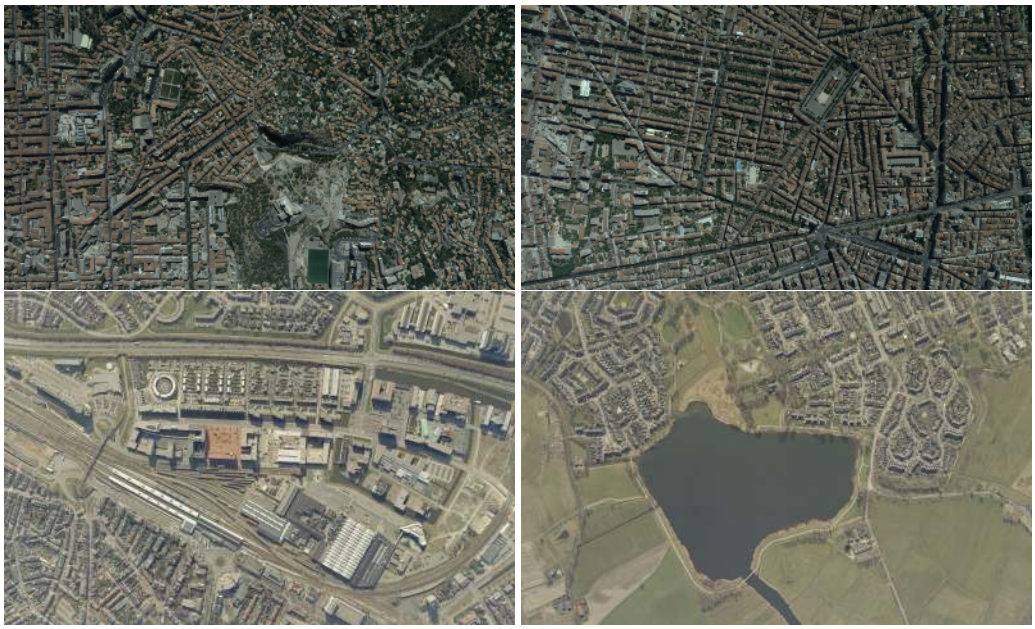


Figure 5.2: **Top:** Two out of the 25 1382r4×7680 pixels images from the Marseilles dataset. **Bottom:** Two out of the 68 11500×7500 pixels images from the Zwolle dataset.

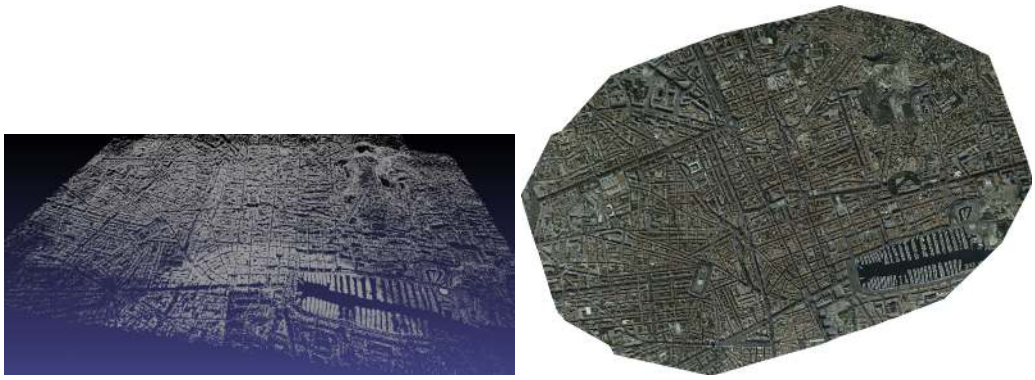


Figure 5.3: **Top:** The aerial triangulation of 1.1M 3D points and **bottom:** the generated ortho-image for the Marseilles dataset.

Figure 5.4: **Top:** The aerial triangulation of 2.1M 3D points and **bottom:** the ortho-image made of 68 individual images (right) for the Zwolle dataset.

Chapter 6

Conclusion

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

Future work will focus on improving the selection of hashing keys for Uniform LSH to accommodate the fact that, in some cases, specific bits in the binary vectors might be more significant than others.

Bibliography

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

[2] M. Brown, G. Hua, and S. Winder, "Discriminative Learning of Local Image Descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 43–57, 2010.

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

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

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

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

[7] D. Nister and H. Stewenius, "Scalable Recognition With a Vocabulary Tree," in *Conference on Computer Vision and Pattern Recognition*, 2006.

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

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

[10] A. Andoni and P. Indyk, "Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions," *Communications of the ACM*, vol. 51, no. 1, 2008.

[11] G. Shakhnarovich, "Learning Task-Specific Similarity," Ph.D. dissertation, Massachusetts Institute of Technology, 2005.

[12] B. Kulis and T. Darrell, "Learning to Hash With Binary Reconstructive Embeddings," in *Advances in Neural Information Processing Systems*, 2009, pp. 1042–1050.

[13] M. Raginsky and S. Lazebnik, "Locality-Sensitive Binary Codes from Shift-Invariant Kernels," in *Advances in Neural Information Processing Systems*, 2009.

[14] C. Strecha, A. Bronstein, M. Bronstein, and P. Fua, "LDAHash: Improved Matching With Smaller Descriptors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2011.

[15] Y. Gong and S. Lazebnik, "Iterative Quantization: a Procrustean Approach to Learning Binary Codes," in *Conference on Computer Vision and Pattern Recognition*, 2011.

[16] R. Salakhutdinov and G. Hinton, "Semantic Hashing," *International Journal of Approximate Reasoning*, vol. 50, no. 7, 2009.

[17] Y. Weiss, A. Torralba, and R. Fergus, "Spectral Hashing," *Advances in Neural Information Processing Systems*, vol. 21, pp. 1753–1760, 2009.

[18] A. Gionis, P. Indik, and R. Motwani, "Similarity Search in High Dimensions Via Hashing," in *International Conference on Very Large Databases*, 2004.

[19] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "BRIEF: Binary Robust Independent Elementary Features," in *European Conference on Computer Vision*, September 2010.

[20] M. Miller, M. Rodriguez, and I. Cox, "Audio Fingerprinting: Nearest Neighbor Search in High Dimensional Binary Spaces," *Journal of VLSI Signal Processing Systems*, vol. 41, no. 3, November 2005.

[21] A. Beygelzimer, S. Kakade, and J. Langford, "Cover Trees for Nearest Neighbor," in *International Conference on Machine Learning*, 2006.

[22] M. Muja and D. Lowe, "Fast Approximate Nearest Neighbors With Automatic Algorithm Configuration," in *International Conference on Computer Vision*, 2009.

[23] J. Friedman, J. Bentley, and R. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Transactions on Mathematical Software*, vol. 3, no. 3, September 1977.

[24] Y. Amit, D. Geman, and B. Jedynak, "Efficient Focusing and Face Detection," *Face Recognition: From Theory to Applications*, 1998.

[25] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman, "Object Retrieval With Large Vocabularies and Fast Spatial Matching," in *Conference on Computer Vision and Pattern Recognition*, 2007.

[26] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, "Fast Approximate Nearest-Neighbor Search with k-Nearest Neighbor Graph," 2011.

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

[28] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *STOC*, 2002, pp. 380–388.

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