Random Binary Search Trees for Approximate Nearest Neighbour Search in Binary Space

Michał Komorowski(✉) and Tomasz Trzciński

Institute of Computer Science, Warsaw University of Technology,
Nowowiejska 15/19, 00-665 Warsaw, Poland
michalkomorowski1984@gmail.com, t.trzcinski@ii.pw.edu.pl

Abstract. Approximate nearest neighbour (ANN) search is one of the most important problems in computer science fields such as data mining or computer vision. In this paper, we focus on ANN for high-dimensional binary vectors and we propose a simple yet powerful search method that uses Random Binary Search Trees (RBST). We apply our method to a dataset of 1.25M binary local feature descriptors obtained from a real-life image-based localisation system provided by Google as a part of Project Tango [7]. An extensive evaluation of our method against the state-of-the-art variations of Locality Sensitive Hashing (LSH), namely Uniform LSH and Multi-probe LSH, shows the superiority of our method in terms of retrieval precision with performance boost of over 20%.

Keywords: Approximate nearest neighbour search · Binary vectors · Random Binary Search Trees · Locality sensitive hashing

1 Introduction

The goal of nearest neighbour search is to find vectors from a database that lie close to a query vector. This is a common use case in disciplines such as computer vision [17] or data mining [15]. However, often finding the exact nearest neighbour is costly while retrieving approximate neighbours is sufficient. Therefore several successful solutions in the area of Approximate Nearest Neighbour Search (ANN) have been proposed and among them the two most prominent ones are hierarchical structure (tree) based methods [2,5] and hashing based methods [6,20].

One of the typical computer vision tasks where ANN search is used due to prohibitive amounts of data points is image-based localisation [4,13]. ANN search is typically used in this context to find similarities between local feature descriptors extracted from different images. The majority of works on ANN focus on descriptors that are vectors of real numbers [5,10,12,16]. However, extraction of real-valued descriptors is time consuming so they are often substituted with binary descriptors when real-time performance is required. At the same time methods suitable for real-valued descriptors do not seem to work equally well when applied to binary ones [18].

In this paper, we propose ANN search method that uses Random Binary Search Trees (RBST) to find similar vectors within a database of binary vectors. As a use case of our method we take image-based localisation problem and we evaluate our method on a real world dataset of over 1 million binary local feature descriptors obtained within the frames of Google Project Tango [7] collaboration. Our ANN search method outperforms the state of the art in terms of retrieval accuracy, while providing similar recall and memory consumption.

Several other types of trees have been proposed in the literature for indexing of binary descriptors e.g.: k-means trees, kd-trees, or vantage-points trees [8]. However, their application to binary descriptors leads to severe performance drops, as indicated in [18]. Therefore we compare our proposed Random Binary Search Trees method with Local Sensitivity Hashing method [6] and its further modifications: Uniform LSH [18] and Multi-probe LSH [11].

2 Random Binary Search Trees

In this section, we propose a simple yet powerful ANN method for indexing and searching a database of binary descriptors. We draw the inspiration for the method from standard Binary Search Trees (BST) [3]. These structures are well designed for speeding up search process and building up on their success, we propose a modified version of them, called Random Binary Search Trees. Our proposed RBST differ from standard Binary Search Trees in the following aspects.

Firstly, all paths from the root node to the leafs in our RBST have the same length. Secondly, the leafs of our RBST are used to store binary descriptors. The most important difference, however, is the fact that the nodes of our trees store a bit mask. It specifies which bit of a binary descriptor needs to be checked in order to decide if a given descriptor should be assigned to the left or to the right branch of a given node during indexing and search. Thanks to this setup RBST are extremely fast as no distances must be calculated in order to create and search them - the fast binary operation AND is used instead.

In the indexing stage, we use one or more Random Binary Search Trees to store the information about binary descriptors from our database. Each descriptor from the database traverses the tree from the root towards the leaves. While traversing the tree, the descriptor is assigned to left or right branch based on the output of the binary AND operation on the descriptor and the bit mask of the node. In the querying stage, we use those constructed trees to search for candidate nearest neighbours by traversing the trees with a query descriptor and retrieving candidates per each tree. The final set of candidates is returned as a union of candidates across the trees. In the last stage of search, candidate descriptors are sorted based on their Hamming distance to the query descriptor. We then retrieve N descriptors with the smallest distance.

Our Random Binary Search Trees algorithm is controlled by four parameters: N equals to number of approximate nearest neighbours retrieved with default $N = 10$, N_{tree} defines the number of RBST to be created, D specifies the maximum depth of a tree and N_{test} defines how many dimensions of a binary descriptor can be checked in a single tree. Although each node can check only one dimension, this parameter allows us to randomly subsample the space of binary dimensions across different binary trees and increases robustness of our method.

The randomness of our RBST stems from the fact that bits masks for nodes are selected randomly from a given set of bits. A similar idea is used in [9]. However, the trees proposed in [9] were not used to index binary descriptors, but to classify keypoints. A related method can also be found in [4] where trees are generated in supervised way using a stability metric. We evaluated application of this approach to our RBST, however, in our experiments trees proposed in [4] were up to 3 orders of magnitude slower than our Random Binary Search Trees. Our proposed RBST may also look similar to Randomised Binary Search Trees [14]. However, there are few differences. In comparison to our RBST data structure proposed in [14] associates a priority with every inserted key, use rotations to balance a tree and does not store list of keys (in our case descriptors) in leafs.

2.1 Bits Selection Metrics and Hash Codes

We also used the following bit metrics to weight the probability of a given bit to be selected for a mask in the nodes: Shannon entropy of a bit, its conditional entropy and its empirical stability. We define the empirical stability metric as a number of descriptors representing the same 3D point with equal value of a given bit to a total number of descriptors of the same 3D point. After calculating those bit metrics, we used their distribution as bias in the selection of a bit mask for each node. Bits with the higher values of bit metrics are used more often to generate RBST. In order to limit memory consumption, we also used hash codes of binary descriptors, instead of the raw vectors. For hashing the descriptors we used Semi-Supervised Hashing method [19] with various hash code lengths (32, 64, 128, 256 bits). Although in some cases bit metrics or hash codes increased the performance of our method, the improvement was rather negligible and, therefore, in the remainder of this paper, we rely on random bit selection for the node masks.

3 Evaluation

In this section, we evaluate the accuracy and efficiency of our RBST method and compare it with the state of the art. To increase robustness of our evaluation, we run our experiments 10 times, each time on a different subset of 100K descriptors extracted from dataset of 1.26M 512-dimensional binary FREAK descriptors [1]. This dataset was obtained from Google Project Tango [7] collaboration and was generated using state-of-the-art 3D reconstruction methods. As evaluation metrics, we use Precision@N defined as number of correctly retrieved nearest neighbours within the first N descriptors retrieved. Similarly, we compute Recall@N defined as the ratio of retrieved nearest neighbours describing the same 3D point within N returned descriptors versus all descriptors describing given 3D point. We also measure querying time and average the results over 10 runs. All experiments are run using a server with 32 GB of RAM and Intel(R) Xeon(R) 2.60 GHz CPU.

3.1 Initial Experiments

To validate our method and verify the appropriate range of parameters, we first run an initial set of experiments with the following set of parameters: $N_{tree} = \{1, 3, 6, 9, 12\}$, $D = \{20, 30, 40, 50\}$ and $N_{test} = \{64, 128, 256, 512\}$. Based on obtained results, we defined a default set of parameters to be evaluated against the state of the art in the next sections, as they give a good balance between the precision, the recall and the average query time: $D = \{30, 40, 50\}$ and $N_{test}= 256$.

We also discovered the following trends. Firstly, the higher value of N_{tree} the higher Precision@10, at the expense of the average query time. The dependence between N_{tree} and the average query time, assuming other parameters remain unchanged, is quasi-linear. Secondly, the lower value of D, the higher average query time. Shallower trees have leafs with higher number of descriptors and since the last step of search includes sorting candidate vectors, the more candidates we retrieve, the longer the sorting. The highest precision can be obtained for the trees with the highest depth, for which majority of leafs contain not more than a few nodes but at the cost of the recall. As to N_{test}, the higher value of this parameter the smaller average query time (even 2 times or more), because descriptors are spread across higher number of leafs. This, in turn, results from a fact that more bits are taken into account while generating trees.

3.2 Comparison with the State of the Art

In this section, we compare our RBST against the competitive approaches for ANN search in binary spaces. Figures 1 and 2 show the results of experiments. Following the evaluation protocol of [18] we plot Precision and Recall results obtained against average query times. We compare our method against 3 variants of Local Sensitive Hashing (LSH) algorithm, as they were shown to provide the best performances in [18]. We use our own implementation of those algorithms. The parameters of all the methods were optimised using grid search approach. In the case of RBST the evaluation was done for $N_{tree} = \{1, 3, 6, 9, 12\}$ trees. For the hashing methods, the number of hash tables used were equal to $\{1, 2, 4, 8, 16\}$. For LSH and Uniform LSH the hash length was 56 and for Multi-Probe 28. We report average memory consumption as memory required by the algorithms to build indexing structures for descriptors and not descriptors themselves.

Figure 1 shows that RBST provides better performance with respect to the state of the art hashing methods in terms of search precision, given equal query time. The performance boost is especially visible for lower average query times

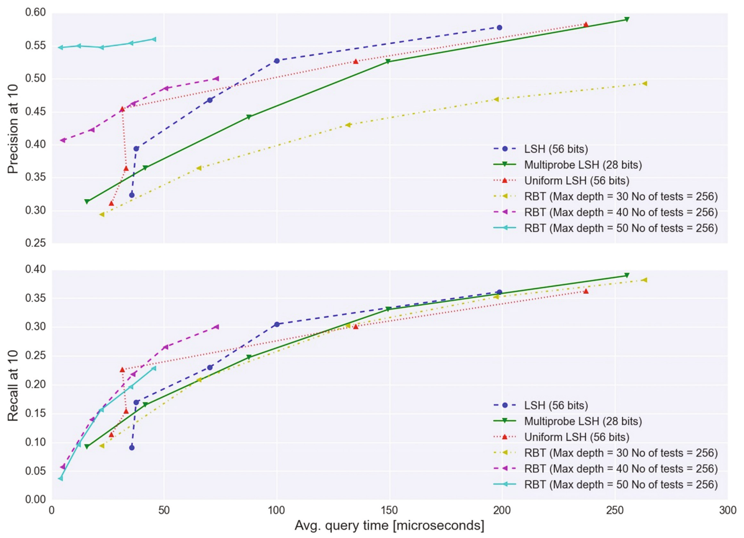

Fig. 1. Precision@10 and Recall@10 versus the average query time.

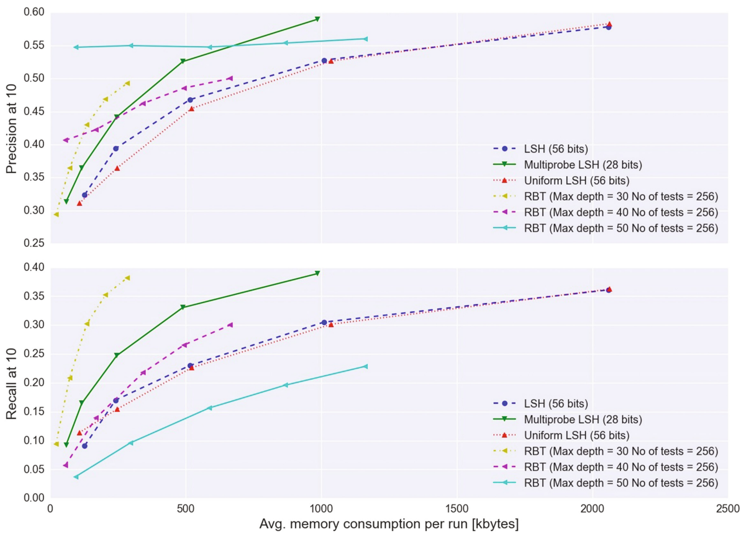

Fig. 2. Precision@10 and Recall@10 versus the average memory consumption.

(<50 µs), where our proposed RBST algorithm leads to over 20% precision increase over the next best Uniform-LSH method. At the same time, our evaluation shows that the precision increase does not lead to any significant recall drops. If we consider both Precision and Recall our RBST achieve the best results for $D = 40$ and $N_{test} = 256$.

Figure 2 compares various methods in terms of memory consumption. Although particular results depend on the tested configuration, one can see that for $D = 40$ and $N_{test} = 256$ RBST performs au pair with the state-of-the-art methods, falling short only of the Multi-probe LSH, which is highly optimised for memory consumption. We can therefore conclude that our proposed RBST search method provides significant precision increase, while remaining competitive in terms of recall and memory consumption.

4 Summary

In this article, we proposed to use Random Binary Search Trees (RBST) algorithm to index and search binary descriptors. We tested a wide range of configurations and we compared them with Locality Sensitive Hashing (LSH) and its two variations. The experiments showed that, although RBST are a relatively simple data structure, they give better or equal results to the competing hashing algorithms.

Future work on ANN search with our trees includes improving the linear search stage after retrieving the initial set of candidate descriptors, as this part remains a bottleneck of the algorithm. Furthermore, application of a more complex bit metric that can measure dependencies between the bits could lead to the improved precision and search efficiency and should also remain within the scope of future work.

Acknowledgment. This research was supported by Google's Sponsor Research Agreement under the project "Efficient visual localisation on mobile devices". We thank Oskar Dylewski for the implementation of LSH, Uniform LSH and Multi-probe LSH.

References

1. Alahi, A., Ortiz, R., Vandergheynst, P.: FREAK: fast retina keypoint. In: CVPR (2012)
2. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Commun. ACM **18**(9), 509–517 (1975)
3. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press, Cambridge (2001)
4. Feng, Y., Fan, L., Wu, Y.: Fast localization in large-scale environments using supervised indexing of binary features. IEEE Trans. Image Process. **25**(1), 343–358 (2016)
5. Fukunaga, K., Narendra, P.M.: A branch and bound algorithm for computing k-nearest neighbors. IEEE Trans. Comput. **100**(7), 750–753 (1975)
6. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. VLDB **99**(6), 518–529 (1999)
7. Google Tango. https://get.google.com/tango/
8. Kumar, N., Zhang, L., Nayar, S.: What is a good nearest neighbors algorithm for finding similar patches in images? In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008. LNCS, vol. 5303, pp. 364–378. Springer, Heidelberg (2008). doi:10.1007/978-3-540-88688-4_27

9. Lepetit, V., Fua, P.: Keypoint recognition using randomized trees. IEEE Trans. Pattern Anal. Mach. Intell. **28**(9), 1465–1479 (2006)
10. Liu, T., Moore, A., Gray, A., Yang, K.: An investigation of practical approximate nearest neighbor algorithm. In: NIPS (2004)
11. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Multi-probe LSH: efficient indexing for high-dimensional similarity search. In: VLDB (2007)
12. Nister, D., Stewenius, H.: Scalable recognition with a vocabulary tree. In: CVPR (2006)
13. Sattler, T., Leibe, B., Kobbelt, L.: Fast image-based localization using direct 2d-to-3d matching. In: ICCV (2011)
14. Seidel, R., Cecilia, R.A.: Randomized search trees. Algorithmica **16**(4), 464–497 (1996)
15. Shakhnarovich, G., Viola, P.A., Darrell, T.: Fast pose estimation with parameter-sensitive hashing. In: ICCV (2003)
16. Silpa-Anan, C., Hartley, R.: Optimised kd-trees for fast image descriptor matching. In: CVPR (2008)
17. Torralba, A., Fergus, R., Weiss, Y.: Small codes and large image databases for recognition. In: CVPR (2008)
18. Trzcinski, T., Lepetit, V., Fua, P.: Thick boundaries in binary space and their influence on nearest-neighbor search. Pattern Recogn. Lett. **33**(16), 2173–2180 (2012)
19. Wang, J., Kumar, S., Chang, S.F.: Semi-supervised hashing for large-scale search. IEEE Trans. Pattern Anal. Mach. Intell. **34**(12), 2393–2406 (2012)
20. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS, vol. 21, pp. 1753–1760 (2009)