Exact Nearest Neighbour Search within Constrained Neighbourhood Using the Forest of Vp-Tree-Like Structures

E Myasnikov

1Geoinformatics and Information Security department, Samara National Research University, 34 Moskovskoye shosse, Samara 443086, Russia

E-mail: mevg@geosamara.ru

Abstract. In this paper, we address the problem of fast nearest neighbour search. Unfortunately, well-known indexing data structures, such as vp-trees perform poorly on some datasets and do not provide significant acceleration compared to the brute force approach. In the paper, we consider an alternative solution, which can be applied if we are not interested in some fraction of distant nearest neighbours. This solution is based on building the forest of vp-tree-like structures and guarantees the exact nearest neighbour search in the epsilon-neighbourhood of the query point.

1. Introduction

Fast search for similar fragments of images is an important element in the construction of systems for detecting unauthorized changes in images. Usually, to solve such a problem, some feature information is extracted from the image fragments, and then a search is carried out using one of the fast nearest neighbor methods. The most commonly used techniques to speed up searches are space partitioning data structures and hashing methods.

The most well-known space partitioning data structures are kd-trees [1], ball-trees [2], vp-trees [3], etc. While kd-trees are mostly effective in low-dimensional spaces, vp-trees demonstrated excellent results in high dimensional spaces in a number of papers [5, 6]. It is worth noting that search methods in such trees are based on the metric properties of feature spaces (more often, Euclidean) and allow to find exact neighbors.

Hashing methods are based on the idea of mapping image data to some numeric (binary) codes so that the search can be carried out by efficient code comparison. The most well-known hashing technique is Locality Sensitive Hashing (LSH) [7], which is based on random projections. This technique performs approximated nearest neighbor search and often uses several hash tables to improve its accuracy. While this technique was proposed for Euclidean distances, later it was extended to p-norm, Mahalanobis, and kernel-based measures [8, 9].

Apart from random projections, the next group of methods derives hyperplanes for projections by analyzing data. Such techniques use Principal Component analysis [10], Local Discriminant analysis [11], or other methods to find better projections and reduce the description length.

Another interesting research direction was devoted to performing hashing in reduced spaces obtained using nonlinear dimensionality reduction techniques such as Laplacian Eigenmaps [13], t-SNE [13], and some others. While nonlinear techniques potentially better snap the underlying data structure, their long runtime limits their use in the considered problem.
In this paper, to address the considered problem, we propose an alternative solution, which can be applied if we are not interested in some fraction of distant nearest neighbours. This solution is based on building the forest of vp-tree-like structures and guarantees the exact nearest neighbour search in the epsilon-neighborhood of the query point. Using known image datasets, we compare the proposed approach with two alternatives, namely, base vp-trees and the brute force approach.

The work has the following structure. Section 2 describes the baseline and the proposed technique. Section 3 describes the datasets and presents the results of the experimental study. The work ends with a conclusion and a list of references.

2. Methods

2.1. Base algorithm

Vp-tree [3, 4] is a tree-like indexing structure, which splits multidimensional space using hyperspheres. The construction of vp-tree starts with a selection of a vantage point \( v \) from the set of points \( S \). The standard approach assumes that the selection is carried out in such a way as to maximize the variance of distances to the remaining points of the set \( S \). To speed up computations, the selection is done from a small subset of points and the variance is estimated using another random subset of \( S \).

After that, all the remaining points are split into two disjoint subsets \( S_1 \) and \( S_2 \) of the same cardinality according to the distances from the selected vantage point \( v \):

\[
S_1 = \{ p \in S \mid d(p, v) \leq d_m \},
\]

\[
S_2 = \{ p \in S \mid d(p, v) > d_m \}.
\]

Here \( d_m \) is the median value of the distances \( d(p, v), p \in S \).

Next, the root of the binary tree is created corresponding to the vantage point \( v \), and the process is repeated recursively for the subsets \( S_1 \) and \( S_2 \). After the creation of the corresponding nodes, these nodes become the siblings of the root.

To execute a query by the nearest neighbor, we traverse the created vp-tree starting from its root. We visit first those nodes where the nearest neighbor points are most likely to appear and update current estimates of the distances to near points already found. Using the properties of metric, in particular, triangle inequality, we ignore the nodes of the tree, which are more distant than the current distance estimation, and, therefore, cannot contain the nearest neighbor points. Such an approach allows skipping a significant amount of data and significantly speeding up the nearest neighbor search.

2.2. Proposed approach

Unfortunately, the described above search procedure can be time-consuming for some datasets, as we still need to visit many nodes during tree traversal. When the distance \( d_q = d(v, q) \) to the query point \( q \) is close enough to the splitting (median) value \( d_m \), we have to visit both siblings of the current node. To avoid such a situation, we propose an alternative approach to the construction of an indexing structure.

The proposed approach assumes that we can specify some value \( \varepsilon \) of the neighborhood, within which the nearest neighbors of interest may be located. That is the proposed approach guarantees that we found exact nearest neighbors if the distance to the query point is less than the \( \varepsilon \) value.

The data structure in the proposed approach is a forest of one or more vp-trees, each constructed using the following algorithm:

- **Input**: the set \( S \) of points, neighborhood value \( \varepsilon \), maximum bucket size \( b \).
- **Algorithm**:
  1. If the set \( S \) contains less than \( b \) points, then create a node with the bucket of size \( |S| \) and exit.
  2. Otherwise, select the vantage point \( v \in S \) as in the base approach. Search for the median value \( d_m \) of the distances \( d(v, p), p \in S \) from the vantage point \( v \).
3. Split all the points \( p \in S \) of the current set \( S \) into three subsets according to the distances to the vantage point \( v \):

\[
S_1 = \{ p \in S \mid d(p,v) < d_m - \epsilon \},
\]

\[
S_2 = \{ p \in S \mid d(p,v) > d_m + \epsilon \},
\]

\[
S_3 = \{ p \in S \mid d_m - \epsilon \leq d(p,v) \leq d_m + \epsilon \}.
\]

Create the current node of the constructed tree and repeat steps 1-3 recursively for the subsets \( S_1 \) and \( S_2 \). Accumulate the points from the subsets \( S_1 \) over all created nodes into the subset \( S_3 \).

4. If the subset \( S_3 \) is equal to the subset \( S \), then create a node with the bucket of size \(|S|\) and exit.

5. Otherwise, set \( S = S_3 \), and proceed to step 1 to create the next tree.

To execute a query by the nearest neighbor, we traverse all the trees of the created forest in the following way:

Input: query point \( q \), forest \( F = \{ T_i \} \).

1. For each tree \( T_i \) of the created forest \( F \), go down from the root of the tree to the only terminal node \( t_i \), passing to the left subtree if \( d(q,v) < d_m \), and to the right subtree, otherwise.

2. Search for the nearest neighbors in the baskets corresponding to the found nodes \( t_i \) using the exhaustive search.

As it can be seen, to search the nearest neighbors, the proposed approach takes \( O(H_f) \) operations (here \( H_f \) is the height of a tree) to locate a basket in a tree and \( b \) operations to search in a basket, except the last basket. In the worst case when \( \epsilon \) is big enough, all the points are located in the \( \epsilon \)-neighborhood, it is impossible to construct any trees, and the method is equivalent to the brute force approach. When \( \epsilon \) is zero, the set \( S_3 \) is empty and the only constructed tree is equivalent to the standard vp-tree. In intermediate cases, the efficiency will depend on method parameters and data distribution.

3. Experiments

The study of the proposed technique was carried out using an ordinary laptop based on Intel Core i3-6100U CPU @ 2.3 GHz. To evaluate the algorithms, we used two datasets. The first dataset contains special features for finding duplicate fragments on digital images. The features are based on key points extraction with the subsequent calculation of features, based on the extracted information. The features were calculated for a set of digital images in a sliding window. Due to the limited scope, we refer to the previous paper [14] for more details. Dimensionality of the feature space is 256.

Another dataset is the Corel Image Features Data Set [15]. This dataset contains features, calculated from the digital images of the Corel image collection (http://corel.digitalriver.com/). The Corel Image Features Data Set contains 68,040 instances.

The following features have been used in the experiments: color moments [17] calculated for each color component: mean, standard deviation, and skewness (dimensionality of the feature space is 9); texture features based on co-occurrence matrices [18] (second angular moment, contrast, inverse difference moment, and entropy were computed in four directions. Dimensionality of the feature space is 16).

The numerical results for the above dataset are shown in Tables 1-3. Each table contains a set of rows corresponding to different neighborhood sizes \( \epsilon \). A particular row shows the number of trees in the forest \( |F| \), the construction \( t_{\text{create}} \) and the search \( t_{\text{search}} \) time (in seconds), and the fraction of nearest neighbor distances above the neighborhood size. The latter indicates that the corresponding nearest neighbors reside outside the neighborhood and cannot be correctly located.

Figure 1 shows the dependency of the search time improvement over the base vp-tree algorithm on the size of the neighborhood (dot line). In addition, you can see the graph of the fraction \( s \) of nearest neighbor distances above the size \( \epsilon \) of the neighborhood (solid line).
**Table 1.** Experimental results for duplicate fragments dataset.

| Proposed approach | $\varepsilon$ | $|F|$ | $t_{\text{constr}}$, sec | $t_{\text{search}}$, sec. | $s$, % |
|-------------------|---------------|-------|--------------------------|---------------------------|------|
|                   | 0.01          | 108   | 2.5                      | 2.8                       | 6.1  |
|                   | 0.005         | 60    | 1.1                      | 0.87                      | 23.0 |
|                   | 0.004         | 39    | 0.82                     | 0.58                      | 26.7 |
|                   | 0.003         | 24    | 0.61                     | 0.39                      | 36.4 |
|                   | 0.002         | 14    | 0.47                     | 0.35                      | 45.1 |
|                   | 0.001         | 7     | 0.37                     | 0.14                      | 55.6 |
| Vp-tree           |               |       |                          |                           |      |
| Brute force       |               |       |                          |                           |      |

**Table 2.** Experimental results for Corel Image Features Data Set: color moments.

| Proposed approach | $\varepsilon$ | $|F|$ | $t_{\text{constr}}$, sec | $t_{\text{search}}$, sec. | $s$, % |
|-------------------|---------------|-------|--------------------------|---------------------------|------|
|                   | 0.5           | 302   | 6.7                      | 0.62                      | 0.69 |
|                   | 0.4           | 164   | 4.6                      | 0.39                      | 2.09 |
|                   | 0.3           | 81    | 2.9                      | 0.41                      | 6.29 |
|                   | 0.2           | 35    | 1.8                      | 0.11                      | 14.18|
|                   | 0.1           | 13    | 1.3                      | 0.078                     | 33.76|
| Vp-tree           |               |       |                          |                           |      |
| Brute force       |               |       |                          |                           |      |

**Table 3.** Experimental results for Corel Image Features Data Set: texture features.

| Proposed approach | $\varepsilon$ | $|F|$ | $t_{\text{constr}}$, sec | $t_{\text{search}}$, sec. | $s$, % |
|-------------------|---------------|-------|--------------------------|---------------------------|------|
|                   | 0.25          | 88    | 3.1                      | 0.29                      | 4.6  |
|                   | 0.2           | 52    | 2.3                      | 0.18                      | 7.4  |
|                   | 0.15          | 30    | 1.5                      | 0.14                      | 12.7 |
|                   | 0.1           | 17    | 1.1                      | 0.078                     | 24.7 |
|                   | 0.05          | 9     | 0.84                     | 0.046                     | 39.2 |
| Vp-tree           |               |       |                          |                           |      |
| Brute force       |               |       |                          |                           |      |
As it can be seen, the considered technique allows choosing a threshold between the neighborhood size $\varepsilon$ and search time $t_{\text{search}}$. It is worth noting that the considered technique should not be applied if distant nearest points cannot be skipped. Contrary, if we can lose 20-30% distance neighbors, we can expect a significant performance increase.

That was the case in the first experiment with more than 10 times faster search compared to the base vp-tree (see Fig 1.a). Let us note that the vp-tree provided only 1.57 times improvement over the brute force (follows from Table 1) for this dataset.

Figure 1. Dependency of the search time improvement over the base vp-tree $r = t_{\text{vp,search}}/t_{\text{search}}$ and the fraction $s$ of nearest neighbor distances above the threshold $\varepsilon$ on the size $\varepsilon$ of the neighborhood.

4. Conclusion

In this paper, we’ve proposed and studied the solution of the nearest neighbor problem, which can be applied if we are not interested in some fraction of distant nearest neighbors. This solution is based on building the forest of vp-tree-like structures and guarantee the exact nearest neighbour search in epsilon-neighborhood of the query point. While the proposed approach demonstrated only a slight improvement on the Corel Image Features Data Set, it significantly outperformed the base technique on the duplicate fragments dataset showing more than 10 times acceleration. The shortcoming of the approach is a possible loss of distant points outside the $\varepsilon$-neighborhood.

5. References

[1] Bentley J L 1975 Multidimensional binary search trees used for associative searching Communications of the ACM 18(9) 494-546
[2] Linde Y, Buzo A, Gray R M 1980 An algorithm for vector quantizer design IEEE Transactions on Communications 28(1) 84–94
[3] Uhlmann J 1991 Satisfying General Proximity/Similarity Queries with Metric Trees
Information Processing Letters 40(4)

[4] Yianilos 1993 Data structures and algorithms for nearest neighbor search in general metric spaces Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms pp 311–21

[5] Kumar N, Zhang L and Nayar S 2008 What Is a Good Nearest Neighbors Algorithm for Finding Similar Patches in Images? Lecture Notes in Computer Science 5303 364-74

[6] Myasnikov E 2021 Nearest neighbor search in hyperspectral data using binary space partitioning trees IEEE Xplore Digital Library, Proc. 11th Workshop on Hyperspectral Imaging and Signal Processing: Evolution in Remote Sensing (WHISPERS) pp 1-4 doi: 10.1109/WHISPERS52202.2021.9484041

[7] Gionis A, Indyk P and Motwani R 1999 Similarity search in high dimensions via hashing Proc. Int. Conf. Very Large Databases

[8] Datar M, Immorlica N, Indyk P and Mirrokni V 2004 Locality-sensitive hashing scheme based on p-stable distributions In Ann. Symp. Comput. Geometry

[9] Kulis B and Grauman K 2009 Kernelized locality-sensitive hashing for scalable image search In Proc. IEEE Int. Conf. Comp. Vis.

[10] Wang J, Kumar S and Chang S-F 2012 Semi-supervised hashing for large scale search IEEE Trans. Pattern Anal. Mach. Intell. 34(12) 2393–406

[11] Bronstein M and Fua P 2012 LDAHash: Improved matching with smaller descriptors IEEE Trans. Pattern Anal. Mach. Intell.

[12] Liu W, Wang J, Kumar S and Chang S-F 2011 Hashing with graphs In Proc. Int. Conf. Mach. Learn

[13] Shen F, Shen Ch, Shi Q, van den Hengel A and Tang Zh 2013 Inductive Hashing on Manifolds Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '13) 1562–69

[14] Kuznetsov A and Myasnikov E 2016 Copy-move detection algorithm efficiency increase using binary space partitioning trees CEUR Workshop Proceedings 1638 pp 373-8

[15] https://archive.ics.uci.edu/ml/datasets/Corel+ Image+Features

[16] Swain M and Ballard D 1991 Color indexing International Journal of Computer Vision 7(1)

[17] Stricker M and Orengo M 1995 Similarity of color images Proc. SPIE Conf. on Vis. Commun. and Image Proc.

[18] Haralick R, Shanmugam K and Dinstein I 1973 Texture features for image classification IEEE Trans. on Sys. Man. and Cyb. SMC-3(6)

Acknowledgments
The reported study was funded by RFBR according to the research project no. 20-37-70053.