VP-tree: Content-Based Image Indexing

© Il'ya Markov ♣

Saint-Petersburg State University
ilya.markov@gmail.com

Abstract

Recently, a lot of indexing methods were developed for multidimensional spaces and for image feature vector spaces in particular. This paper introduces an implementation of a *vantage point tree* method. The goal is to define dependencies between method's parameters and index efficiency criteria in the context of the content-based image retrieval task.

1 Introduction

An image is just a two-dimensional array of pixels and there are various ways of building up high-level abstractions from it; for example, color distributions based on histograms. This process is usually referred to as a *feature extraction* process. Each scalar piece of such high-level abstractions is called a *feature*. The set of features that comprises coherent high-level interpretations forms a *feature class*. Example feature classes are color, texture and shape [4].

Images in the database go through the feature extraction process, so every image is represented as a K-dimensional feature vector, where K is the number of features used to represent an image. The basic assumption is that two image feature vectors are similar according to a distance metric if and only if the corresponding images are similar too. So the image retrieval problem may be considered as a multidimensional nearest-neighbor search problem.

2 Related works

Recently, a lot of indexing methods were developed for multidimensional spaces and for image feature vector spaces in particular. R*-tree [1], k-d-tree [2], K-D-B-tree [8], SR-tree [7], SS-tree [9], VP-tree [4, 10], M-tree [5] and MVP-tree [3] are among them.

♣ I would like to thank my scientific adviser Boris Novikov and team leader Natalia Vassilieva for their help and support.
This work was partially supported by RFBR (grant 07-07-00268a).
Proceedings of the Spring Young Researcher's Colloquium On Database and Information Systems SYRCoDIS, Moscow, Russia, 2007

VP-tree was firstly introduced in [10]. Basic index building and search algorithms and their improvements were proposed. Improved algorithms store additional information about partition's boundaries and distances between each node and its ancestors to speed up the search. Also they form buckets by collapsing subtrees near leaf level into a flat structure in order to save space. Characteristics of the VP-tree are compared with those of the k-d-tree and the image retrieval task is taken up as an example. The results show that even simple implementation of the VP-tree is not worse than the k-d-tree.

[4] is all about content-based image retrieval task and the VP-tree is considered to be one of the solutions for the image indexing problem in that area. Additive and multiplicative optimistic adjustment mechanisms of the search threshold parameter are proposed. Also N-ary tree is taken as a data structure, instead of binary tree.

[6] proposes n-nearest neighbor search algorithm, which is shown by experiments to scale up well with the size of the dataset and the desired number of nearest neighbors. Experiments also show that the searching in the VP-tree is more efficient than that for the R*-tree [1] and M-tree [5]. In the same work solutions for the update problem are proposed.

3 Problem definition

As mentioned in [4], partitioning methods are generally based on absolute coordinate values of the vector space. For example, a partition in a K-dimensional hypercube is characterized by K pairs of coordinate values, each of which specifies the covering interval in the respective dimension. This type of partitioning structure is useful for queries based on absolute coordinates (such as range queries), but not so useful for nearest-neighbor search because the search structure in general doesn't maintain the distance information between points within a partition and partition's boundaries. As it turns out, this information is critical in pruning the search space for multi-dimensional nearest-neighbor search. VP-tree splits the search space by using relative distances between vantage point and its children, therefore, it is easy to calculate the distance between point and boundaries of partition, which it belongs to.

Moreover, VP-tree employs bounding spheres and therefore precision of space partitioning should be better than of those based on bounding rectangles,

because with increase of space dimension sphere volume decreases as compared to volume of cube in which it can be inscribed.

In our case the problem of image database indexing naturally appeared during the content-based image retrieval system prototype development. In accordance with previous statements the VP-tree was chosen as an index structure for that task.

While there are some works on this structure, the problem of the dependencies between index efficiency criteria and its parameters (arity, for instance) has not been investigated yet. We implement the VP-tree (based on [10] and [4]) and mark out a number of building algorithms and VP-tree parameters. Also some index efficiency criteria are considered. Our main goal is to define dependencies between them and give recommendations for tuning the VP-tree structure.

4 Basics

Given a metric space (S,d) and its finite subset $S_D \subset S$ representing a database. Since any metric's range may be normalized to the interval [0, 1] without affecting the nearest-neighbor relation, we can consider only those metrics without loss of generality [10].

Let us discuss binary partitioning case for simplicity. Assume some data point v from S_D is chosen as the *vantage point*. For any other point $p \in S_D - \{v\}$ distances from v are computed and the median μ is chosen among them. Then the whole data set S_D is divided into two parts in accordance with the vantage point and the median: $S_{\le}$ is a set of points which distances from v are equal or less than μ and $S_{>}$ is a set of points which distances from v are greater than μ.

Suppose we need to find nearest neighbors for some point q with their distances from q being less than a specific threshold σ. It turns out that if $d(v,q) \le \mu - \sigma$ we need to explore only the $S_{\le}$ subset and if $d(v,q) > \mu + \sigma$ the only $S_{>}$ subset has to be explored.

This observation is based on the triangle inequality: if $d(v,q) \le \mu - \sigma$ then for each $p \in S_{>}$ the following is true: $d(q,p) \ge |\, d(v,p) - d(v,q) \,| > |\, \mu - (\mu - \sigma) \,| = \sigma$, i.e. $d(q,p) > \sigma$. It means that the subset can be excluded from a search.

And if $d(v,q) > \mu + \sigma$ then for each $p \in S_{\le}$ we have $d(q,p) \ge |\, d(v,q) - d(v,p) \,| > |\, (\mu + \sigma) - \mu \,| = \sigma$. So $d(q,p) > \sigma$ and we can exclude the $S_{\le}$ subset. Preceding is shown in Figure 1.

So we can effectively prune one half of the search space if the following conditions are met: the power of the $S_{>}$ subset approximately equals the power of the $S_{\le}$ subset and $d(v,q)$ does not meet the following two-sided inequality: $\mu - \sigma < d(v,q) \le \mu + \sigma$. Thus, the main task is to build approximately balanced tree with the quantity of points falling within the two concentric circles being as little as possible. [10] shows that in order to meet these requirements we need to choose vantage points from the *corners* of the space with the maximum second moment of their distances from the other points.

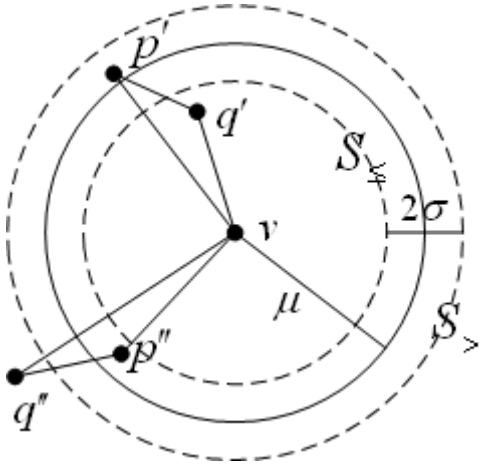


Figure 1: Data set partitioning and nearest-neighbor search. All points from the $S_{>}$ subset (such as p') are too far from the data point q' (further than σ). And points from $S_{\le}$ (such as p'') are too far from q''.

5 Algorithms and parameters

5.1 Building the index

We use simple algorithms based on [10] to find vantage points and build VP-tree. In accordance with [4], N-ary tree was chosen as a data structure. It means that we need to find N-1 border instead of one median to split search space into N partitions.

It is necessary to mention that the index structure implementation is not the main problem of our image retrieval system prototype, therefore, database tables instead of disk blocks were chosen as an index data storage for simplicity.

```
function BuildVP_TreeLevel(S)
    if S = ∅ return ∅
    node.vp := GetVP(S)
    node.borders := GetBorders(S, vp, N)
    for each sequential lborder, uborder from borders
        BuildVP_TreeLevel(
          {s ∈ S - {vp} | lborder < d(p,s) ≤ uborder} )
    return node

function GetVP(S)
    P := random sample of S
    for each p ∈ P
        D := random sample of S
        spread = GetSecondMoment(D, p)
        if (spread > best_spread)
            best_spread := spread
            vp := p
```

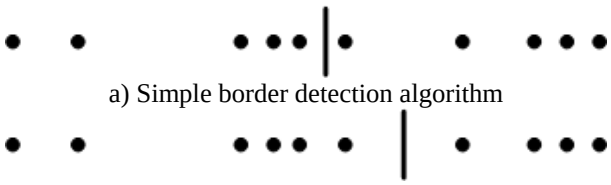
```
    return vp

function GetBorders(S, vp, N)
    P := random sample of S
    sort P in accordance with distances from vp
    for i from 1 to N – 1
        p_low := P[i * |P| / N]
        p_up := P[i * |P| / N + 1]
        border = (d(vp, p_up) – d(vp, p_low)) / 2
        borders.add(border)
    return borders
```

5.2 Parameters

It is necessary to discuss GetBorders function in more detail. This simple algorithm works as follows: selected sample set is sorted by the distances from each $p \in P$ to the vantage point in ascending order. Sorted set is divided into N parts. Arithmetic mean of the distances of the two boundary points is treated as a border between those parts.

Such implementation is too straightforward and can violate the following restriction from the previous section: the quantity of points falling within the two concentric circles must be as little as possible. The restriction will be violated if the margins between boundary points and borders are too small (smaller than σ). Thus, the DDR (distance delta rate) parameter is used to increase the margins. In that case the border can be set not only between boundary points of the two sets, but between points with the maximum distance from each other within bounds determined by the DDR. The example is shown in Figure 2.



a) Simple border detection algorithm

b) Improved algorithm: border was moved to the right by one point (it is allowed by the current DDR value). If we move it to the left by 3 points margins will increase but the tree will be too unbalanced

Figure 2: Border detection between two sets

DDR parameter is a trade-off between balancing and margins. So it has an influence on both conditions of the tree building process mentioned in the previous section and must be chosen carefully.

There are three other parameters that take part in the process: the CRVP (capacity rate for vantage point), the CRSM (capacity rate for second moment) and the CRB (capacity rate for borders). Each of them defines a ratio of the sample set size in the vantage point search, the second moment calculation and the border detection, accordingly, to the whole database size. Here is a trade-off between precision of that objects detection and the duration of the tree building process. The bigger the sample sets are – the more precise vantage points and borders we have, but the more time it takes to build a tree (more points are involved in the process while each point needs some calculations). The smaller the sample sets are – the less time the computations take and the more inaccurate vantage points and borders are.

The fifth important parameter is a tree arity (AR).

5.3 Searching the tree

To perform a search on the VP-tree we generalize the searching algorithm from [10] for the N-ary tree. The following function returns all nearest neighbors for a query point q within a threshold σ. Vp parameter is a root of a VP-tree at first.

```
function GetNN(vp, q, σ, result)
    dist := d(vp, q)
    if (dist <= σ) and (vp != q) result.add(vp)
    if vp is a leaf return
    lborder := max{b ∈ vp.borders | b < dist - σ}
    uborder := min{b ∈ vp.borders | b ≥ dist + σ}
    for each new_vp between lborder and uborder
        GetNN(new_vp, q, σ, result)
```

6 Theoretical basics of experiments and hypotheses

The goal of the experiments is to define the index efficiency criteria, which are further discussed, and to determine the dependencies between these criteria and the parameters mentioned in the previous section. We plan to define such dependencies from each of the parameters (leaving other ones unchanged during each experiment) and from some parameters combinations.

6.1 Searching time

We consider the searching time to be the most important criterion, because the main purpose of an index is to speed up a fetching process. This value can be compared with the time of the whole database scan, since we need to look through the whole database to find the nearest neighbors for some point without using any index. More formally, the following ratio can be calculated to define the criterion: $t_s = \frac{t_{ind}}{t_{rs}}$, where t_{ind} is a nearest-neighbors searching time with an index utilization and t_{rs} is a row scan time.

Moreover, this criterion can be computed in terms of distance metric $d(.)$ evaluation operations. Obviously, to find nearest neighbors for some point without using an index, one needs to do as many metric evaluation operations as many images there are in the database. So we can denote this type of criterion as t_d and equate it to the following: $\frac{\Sigma_{ind}}{\Sigma_{rs}} = \frac{\Sigma_{ind}}{|S_D|}$, where Σ_{ind} is a quantity of the distance metric evaluation operations during a nearest-neighbor search with an index utilization.

t_s and t_d are not necessarily proportional, because the second representation does not take the sorting process duration into account, while we have to sort points to get the final result.

Two kinds of experiments can be conducted to compute these values: single-neighbor and multiple-neighbors searches with a query-point q. The first one looks up for the most relevant point p, i.e. the point with the minimum $d(q, p)$ value, while the second one has to find nearest neighbors with their distances from q being less than a specific threshold σ. In the first case, system with an index has to calculate only one subset of the node q to find the most relevant point, that one with the lowest upper border, whereas it may need to look into other sub-trees to get more points in the second case.

6.2 Index degradation

As the database size grows new points are added to the index, so the tree gets more and more unbalanced. This process affects the criteria, mentioned above, and can completely reduce the index efficiency right up to the need of its full rebuilding. We plan to hold some experiments to define this affection, especially that cases with complete degradation.

The first type of experiments implies adding arbitrary points to the database and computing the above criteria to define the rules of their changing. For the second reason, i.e. defining the worst cases, we plan to add quite similar points to force the tree to become unbalanced rapidly. All types of experiments imply adding a great amount of points to affect the searching criteria.

6.3 Index building process duration

Based on section 5.3, the tree building process duration can be quite an important criterion in some cases. It depends on four parameters, mentioned in section 4: CRVP, CRSM, CRB and AR. From the algorithms of that section we can derive the following formula for the mean tree building time: $t_{mean} = k \cdot \Sigma_{vp} \cdot (CRVP \cdot CRSM \cdot |S|^2 + CRB \cdot |S|)$, where k is some coefficient and Σ_{vp} is a vantage points quantity (which strongly depends on the AR parameter). This formula arises from the following reasoning: we need $k \cdot (CRVP \cdot |S|) \cdot (CRSM \cdot |S|)$ time to process a vantage point sample set and to calculate a second moment for each point, based on another sample set, while searching for a vantage point. And additional $k \cdot (CRB \cdot |S|)$ quantity of time is needed to find point's borders. We plan to check this formula during the experiments.

6.4 Image feature vector space dimension

There are different image features we intend to use in our system, so the dependencies between the dimension of the image feature vectors and index efficiency criteria, mentioned above, are very important to us and we plan to hold some experiments to define them.

As we said before, the VP-tree is based on bounding spheres. And the main advantage of a sphere is that its volume is less than the volume of a cube, in which the sphere is inscribed and this ratio decreases with the increase of the searching space dimension.

7 Conclusion

We have implemented the VP-tree index structure as an alternate solution for the nearest-neighbor search problem in the context of the content-based image retrieval task. Different criteria were proposed to estimate index efficiency. We plan to hold a number of experiments to define those criteria and to determine the dependencies between them and algorithms' parameters, stated in the work.

References

[1] N. Beckmann, H. P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *ACM SIGMOD Record*, volume 19(2), pages 322–331, 1990.

[2] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. In *Communications of ACM*, volume 18(9), pages 509–517, 1975.

[3] T. Bozkaya and M. Ozsoyoglu. Distance-based Indexing for High-Dimensional Metric Spaces. In *ACM SIGMOD Record*, volume 26, pages 357–368, 1997.

[4] T. C. Chiueh. Content-based image indexing. In *Proceedings of the 20th VLDB Conference*, pages 582–593, 1994.

[5] P. Ciaccia, M. Patella, P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In *Proceedings of the 23rd International Conference on VLDB*, Athens, Greece, pages 426-435, 1997.

[6] A. W.-C. Fu, P. M.-S. Chan, Y.-L. Cheung, Y. S. Moon. Dynamic vp-tree indexing for n-nearest neighbor search given pair-wise distances. VLDB Journal, volume 9(2), pages 154-173, 2000.

[7] N. Katayama, S. Satoh. The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, Tucson, Arizona.

[8] J. T. Robinson. The K-D-B-tree: A Search Structure for Large Multidimensional Dynamic Indexes. In *Proceedings of the ACM SIGMOD*, Ann Arbor, USA, pages 10-18, 1981.

[9] D. A. White, R. Jain. Similarity Indexing With the SS-tree. In *Proceedings of the 12th International Conference on Data Engineering*, New Orleans, USA, pages 516-523, 1996.

[10] P. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, Orlando, pages 311-322, 1992.