Term Project

Dynamic vp-tree indexing for n-nearest neighbor search given pairwise distances

Ada Wai-chee Fu, Polly Mei-shuen Chan, Yin-Ling Cheung, Yiu Sang Moon
Department of Computer Science and Engineering,
The Chinese University of Hong Kong, Hong Kong; e-mail: adafu@cse.cuhk.edu.hk
Edited by T.Ozsu¨ and S. Christodoulakis. Received June 9, 1998 / Accepted January 31, 2000

Table of *contents*

01

Introduction

Abstract

In certain multimedia applications, representing domain objects as feature vectors isn't feasible. Instead, reliance on pairwise distances between data objects is the only viable input for content-based retrieval. To overcome this limitation, an approach maps each object to a k-dimensional point, aiming to maintain distances among these points. Although spatial access index methods like R-trees and KD-trees can enable quick searching based on these k-dimensional points, this approach unavoidably incurs information loss due to limited distance preservation. As an alternative, the investigation focuses on a distance-based indexing method, specifically exploring the application of the vantage point tree (vp-tree) method to address these challenges.

Technical report *key points*

- Investigating vp-trees for distance-based indexing in multimedia applications.
- vp-tree's n-nearest neighbor search shows scalability and efficiency over R∗-tree and M-tree.
- Proposed efficient update algorithms for vp-trees.
- Exploring vantage point selection's impact on reducing distance computations.

02

Why VP Tree is chosen?

Why VP Tree is chosen?

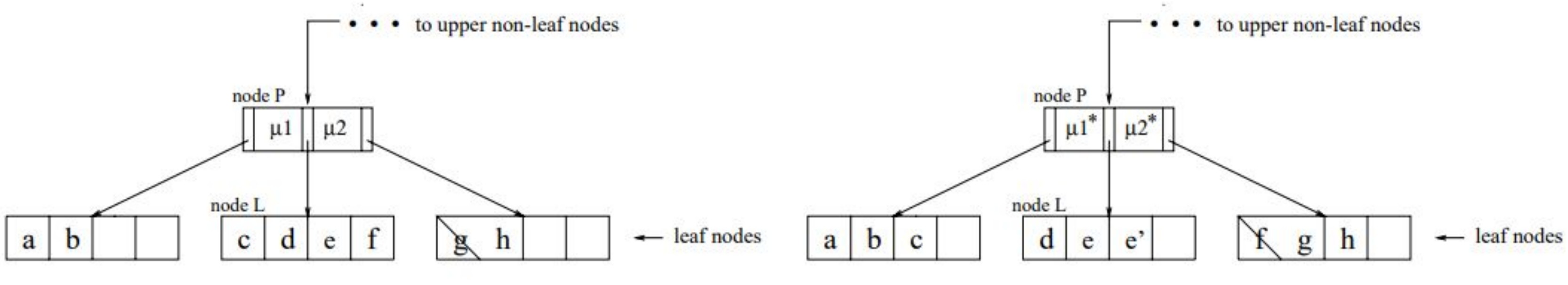
- There is no need to infer multidimensional points for domain objects before an index can be built. Instead, we build an index directly based on the distances given. This avoids pre-processing steps.
- The updates on the VP-tree are relatively easier than that of using **Feature Vectors.** For the later, after a certain amount of data objects are inserted or deleted, the mapping for the data points will no longer be as distance preserving as before. There will be a point where these distances need to be calculated once again for all the objects. By comparison, the updates for VP-tree are much more straightforward
- A distance-based indexing method such as the VP-tree is flexible: it’s not only applicable to multimedia objects given pairwise distances, but is also able to index objects that are represented as feature vectors of a fixed number of dimensions (the case when feature extraction functions are available).

03

Operations on VP Tree

Insertion

Case-1 : If any sibling leaf node of L is not full, redistribute all objects under P among the leaf nodes

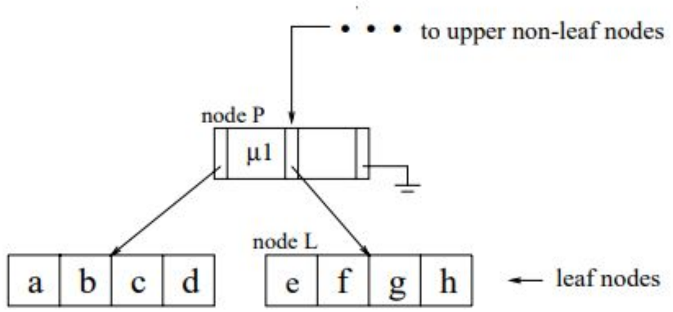


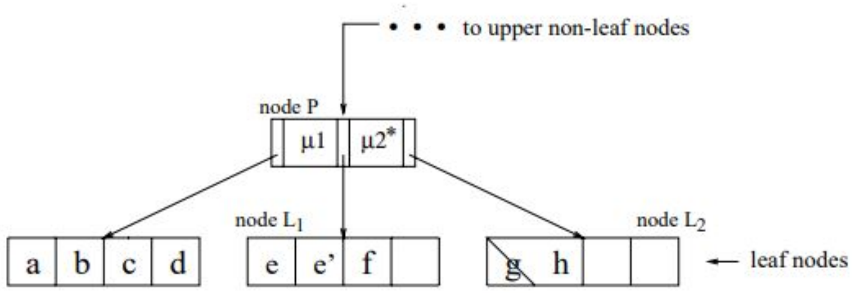
(a) A new object e' needs to be inserted into L and sibling leaf nodes are not full.

(b) All objects under P have been redistributed, e' gets inserted and boundary distances have been updated

Fig. 13a,b. Redistribution among leaf nodes

Case-2 : If L has a parent node P and P has room for one more child, split the leaf node L



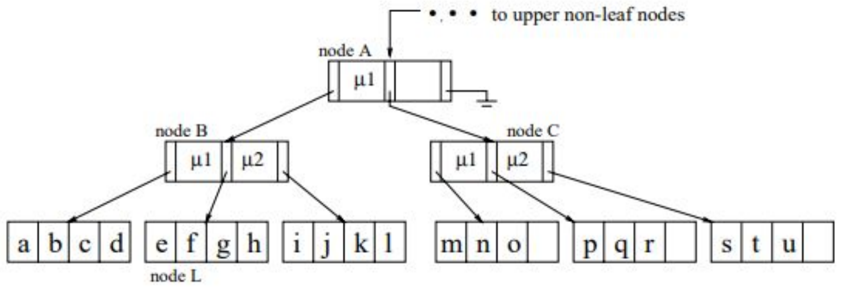


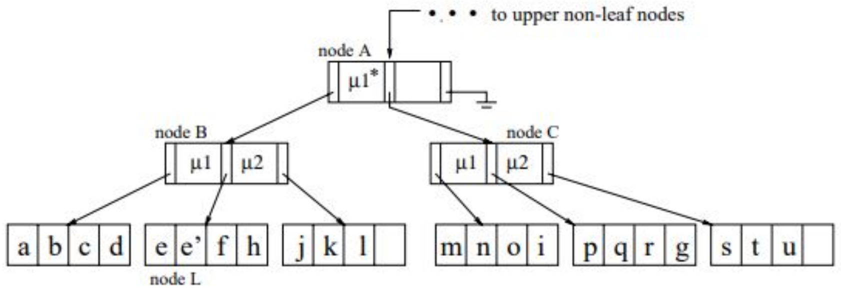
(a) e' needs to be inserted into L and all siblings are full, but the parent node P has room for one more child.

(b) L has been split into nodes L1 and L2, and e' gets inserted.

Fig. 14a,b. Splitting of leaf node

Case-3 : Else, suppose we can find a nearest ancestor A of L that is not full





(a) e' needs to be inserted into L, the entire B subtree is full, since the sibling subtree C has room, we choose to redistribute objects among B and C.

(b) Assume objects g and i are the farthest with respect to A's vantage point. After redistribution they have been moved to the C subtree and e' gets inserted.

Fig. 15a,b. Redistribution among subtrees

Deletion

Case-1: If L has a parent node P and P does not underflow.

1. If the total spare room of L’s siblings can hold all of the objects in L, redistribute the objects under P among F – 1 nodes, i.e., L is to be merged with its siblings.
2. Else, when the total spare room is not enough to hold all of the objects in L, redistribute the objects under P among F nodes.

Case-2: Locate a nearest ancestor A of L that does not underflow. Let B be the immediate child node of A, and B is also the ancestor of L.

1. if the (k+1)-st subtree has enough room to hold all the objects in B, we move the objects in B to the (k+1)-th subtree and delete B.
2. else if (k-1)-st subtree can hold , then do the same as done in   above for (k-1)-th subtree.
3. if the total spare room of the (k+1)-st and the (k-1)-th subtrees can hold all the objects in B.Move mid objects from B to the (k-1)-th subtree and the rest to the (k+1)-st subtree, and delete B.

Case 3. Either L is the root node or all ancestors of L underflow

1. If the root node has at least two child nodes, we merge two child nodes of the root with the deletion of the given data point.
2. Original root node is deleted and the merged node becomes the new root node.
3. If the root node is a leaf node, it means that L is the root node and the deletion is carried out in the root node, we simply delete the data object from L.

```
begin
  average := $\lfloor(\mathrm{Num(k)} + \mathrm{Num(k+1)}) \div 2\rfloor$;
  if Num(k) > Num(k+1) then
      Let S be the set of objects stored in the k-th subtree plus the new object;
      Order the objects in S with respect to their distances from $A$'s vantage point v;
      Let w be the number of data objects that will be moved from k-th subtree to (k+1)-th subtree;
      w := Num(k) − average;

      Divide S into 2 subsets, $SS_1$ and $SS_2$ in order, where
         $SS_1 = \{S_1, S_2, ..., S_{\mathrm{Num(k)}-w}\}$ and
         $SS_2 = \{S_{\mathrm{Num(k)}-w+1}, S_{\mathrm{Num(k)}-w+2}, \ldots, S_{\mathrm{Num(k)}}\}$;

      for all $s_i \in SS_2$, delete $s_i$ from the k-th subtree;
      $A\uparrow.mu_k := (\max\{d(v, s_j) \mid \forall s_j \in SS_1\} + \min\{d(v, s_j) \mid \forall s_j \in SS_2\}) \div 2$;
      for all $s_i \in SS_2$, reinsert $s_i$ to the (k+1)-st subtree;
  else
      Let S be the set of objects stored in the (k+1)-st subtree plus the new object;
      Order the objects in S with respect to their distances from $A$'s vantage point v;
      Let w be the number of data objects that will be moved from (k+1)-st subtree to k-th subtree;
      w := Num(k+1) − average;

      Divide S into 2 subsets, $SS_1$ and $SS_2$ in order, where
         $SS_1 = \{S_1, S_2, ..., S_w\}$ and $SS_2 = \{S_{w+1}, S_{w+2}, \ldots, S_{\mathrm{Num(k+1)}}\}$;

      for all $s_i \in SS_1$, delete $s_i$ from the (k+1)-st subtree;
      $A\uparrow.mu_k := (\max\{d(v, s_j) \mid \forall s_j \in SS_1\} + \min\{d(v, s_j) \mid \forall s_j \in SS_2\}) \div 2$;
      for all $s_i \in SS_1$, reinsert $s_i$ to the k-th subtree;
  endif
end
```

VANTAGE POINT SELECTION

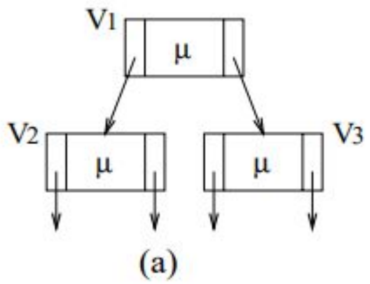
- Single vantage point.
- Reuse Vantage point.

Motivation

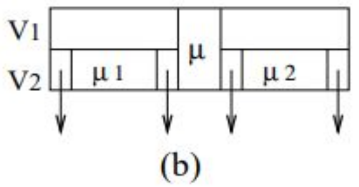
- In high dimension, distance calculation become expensive.
- Minimize distance computation to aim at efficient query processing.
- Despite being red, Mars is actually a cold place, full of iron oxide dust
- To make more efficient than MVP-Tree.

MVP-TREE

- Two VP for a SIngle node
- The first vantage point divides the space into two parts.
- The second vantage point divides each of these partitions into two.
- Fanout of a node in a binary mvp-tree is four.



Binary vp-tree



Binary mvp-tree

A single vantage point per level

- A single vantage point per level.
- At the root level, the first vantage point is selected using a specified method.
- For subsequent levels, each vantage point is chosen to be one of the farthest points.
- This sequential selection ensures vantage points for each level are strategically chosen to be distant from each other.

- The reason for selecting vantage points that are far apart is to ensure a relatively effective partitioning of the dataset.
- Effective partitioning contributes to better organization and search efficiency within the vp-tree.
- Since the number of non-leaf levels is assumed to be small, this method minimizes the number of distance computations during searches.
- This design choice allows for a higher fanout at non-leaf nodes, leading to a smaller tree size and potentially enhanced querying performance.

Reuse of vantage points

- Drawback of single VP-Point:
    - The primary drawback of using a single vantage point per level is the potential deviation from the original partitioning strategy of the vp-tree.
    - In the original method, each chosen vantage point should suit its associated region to a certain extent.

- Balancing Partitioning and Distance Computations:
    - The second method aims to strike a balance between achieving a favorable dataset partitioning and minimizing distance computations during queries.

- Different Vantage Points at the Same Level:
    - Unlike the previous approach, nodes at the same level may have different vantage points.
    - However, not every vantage point is unique; some may be reused.

- Pre-selection of Vantage Points (p):
    - Before constructing the vp-tree, a maximum number
    - *p* is fixed to determine the total number of vantage points to be used.
    - The selection process involves choosing the first vantage point based on the algorithm and ensuring that all *p* points are the farthest from each other.

- Construction of the vp-tree:
    - The vp-tree is then constructed using the pre-selected set of vantage points.
    - These pre-selected points act as the 'candidate vantage points' described .

- Managing Storage and Fanout:
    - The number of distance computations at non-leaf nodes is bounded by *p*.
    - If the number *p* is manageable in terms of memory, the vantage points can be stored outside the vp-tree, similar to the previous approach.
    - This reduces the storage size of the tree and increases the fanout of non-leaf nodes, particularly for high-dimensional feature vectors.

- Choosing an Appropriate *p*:
    - The optimal value for *p* is determined by trial and error for a given dataset.
    - It is recommended to keep *p* small to store all vantage points outside the tree and reduce the time required for selection.

Performance Evaluation

- Reusing vantage points exhibited the most favorable outcomes among the evaluated methods.
- The mvp-tree approach outperformed the 'single' method, implying that relying on only one vantage point for all nodes at the same level had negative implications for partitioning.
- The 'single' method's use of a sole vantage point for nodes at the same level led to suboptimal partitioning, resulting in increased multiple-path searching and more leaf accesses.

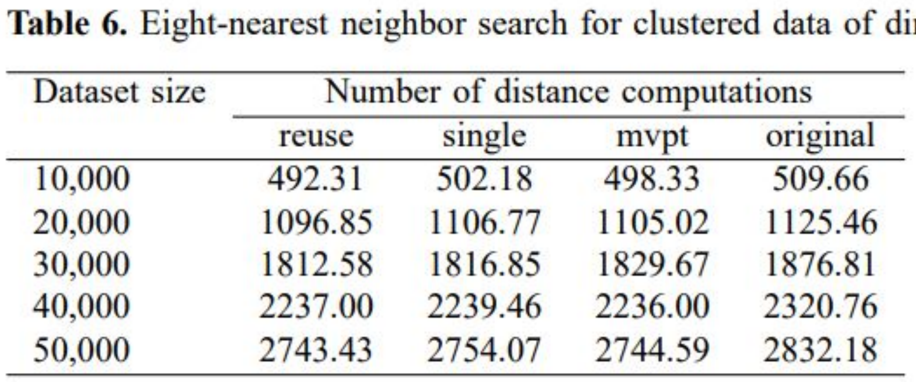


Table 6. Eight-nearest neighbor search for clustered data of dimension 30

Dataset size	Number of distance computations			
	reuse	single	mvpt	original
10,000	492.31	502.18	498.33	509.66
20,000	1096.85	1106.77	1105.02	1125.46
30,000	1812.58	1816.85	1829.67	1876.81
40,000	2237.00	2239.46	2236.00	2320.76
50,000	2743.43	2754.07	2744.59	2832.18

- Page Access Comparison:
- Optimal Method for Page Access:
- Comparison with mvp-tree:
- Advantage in Page Access:

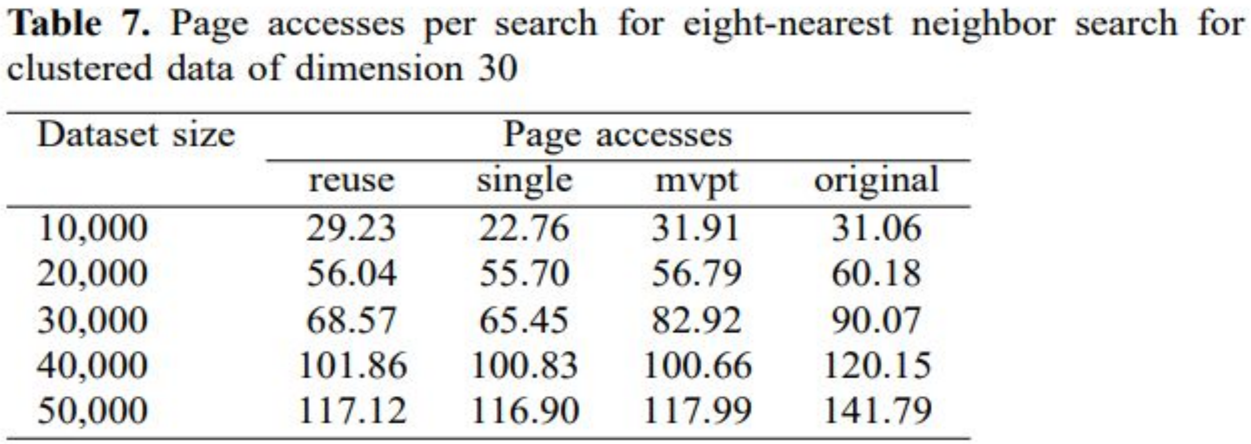


Table 7. Page accesses per search for eight-nearest neighbor search for clustered data of dimension 30

Dataset size	Page accesses			
	reuse	single	mvpt	original
10,000	29.23	22.76	31.91	31.06
20,000	56.04	55.70	56.79	60.18
30,000	68.57	65.45	82.92	90.07
40,000	101.86	100.83	100.66	120.15
50,000	117.12	116.90	117.99	141.79

V-P TREES

N-nearest Neighbour Search

INTRODUCTION:

- Multimedia data retrieval rarely yields exact matches, prompting the preference for nearest neighbor queries.
- Users typically seek a selection of similar objects to a given query, necessitating the identification of n-nearest neighbors (usually more than one).
- An algorithm for n-nearest neighbor search in vp-trees is described, tailored for efficient retrieval of similar objects.
- The algorithm's concept parallels that of n-nearest neighbor searches in other index trees such as R-trees or M-trees, underlining similarities in approach and objective.

1-Nearest Neighbour:

- The algorithm in reference [10] focuses on single-nearest-neighbor search utilizing a threshold value σ for distance estimation.
- It operates by searching within the range of d(v, q) ± σ, where v is a vantage point, to locate the nearest neighbor.
- The vp-tree's node structure partitions subsets based on distances, influencing the exploration path for nearest neighbor search.
- Efficient exploration is determined by whether the hypersphere falls within one subset or spans multiple subsets.
- The choice of σ impacts exploration efficiency, affecting the likelihood of finding the nearest neighbor.
- Notably, the algorithm's implementation doesn't specify a predefined limit on the number of search trials with increasing σ values.

n-Nearest Neighbours:

- The previous algorithm can be generalized for n-nearest neighbour search.
- The algorithm uses a procedure called Search, which operates recursively and dynamically adjusts the threshold value σ during traversal.
- σ is adapted based on the distance between the query object and the current nth-nearest candidate, ensuring the n-nearest neighbors are within a maximum distance of σ from the query point.
- This approach employs a depth-first search strategy and dynamically determines σ using the distances of encountered data objects at leaf nodes during the search

Performance Evaluation

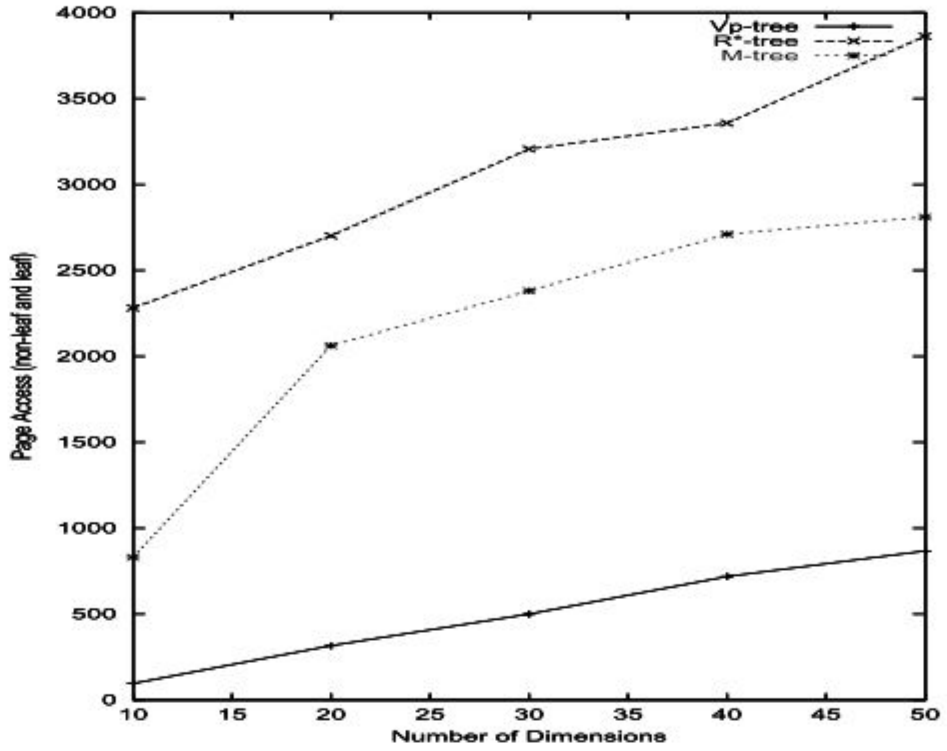
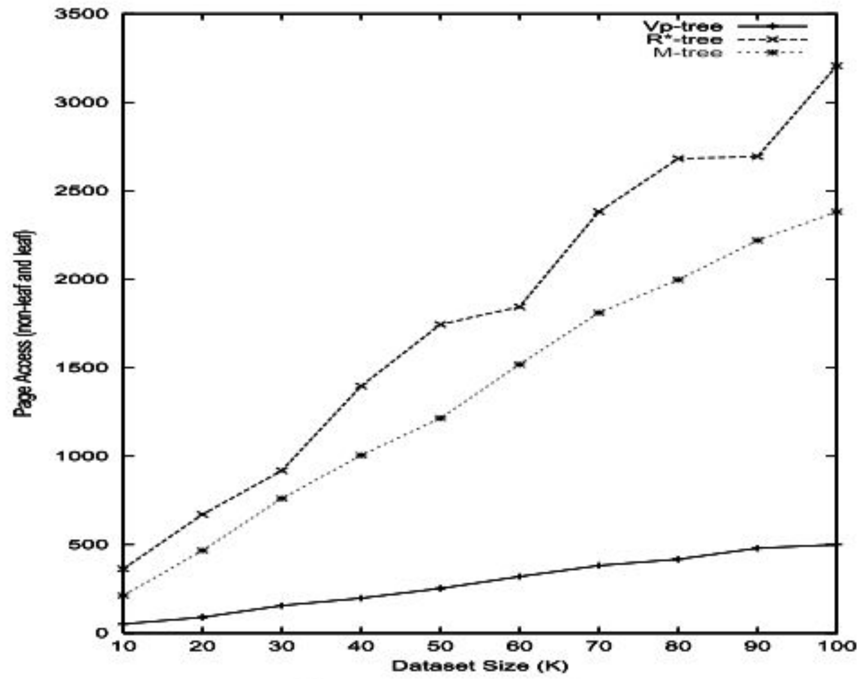
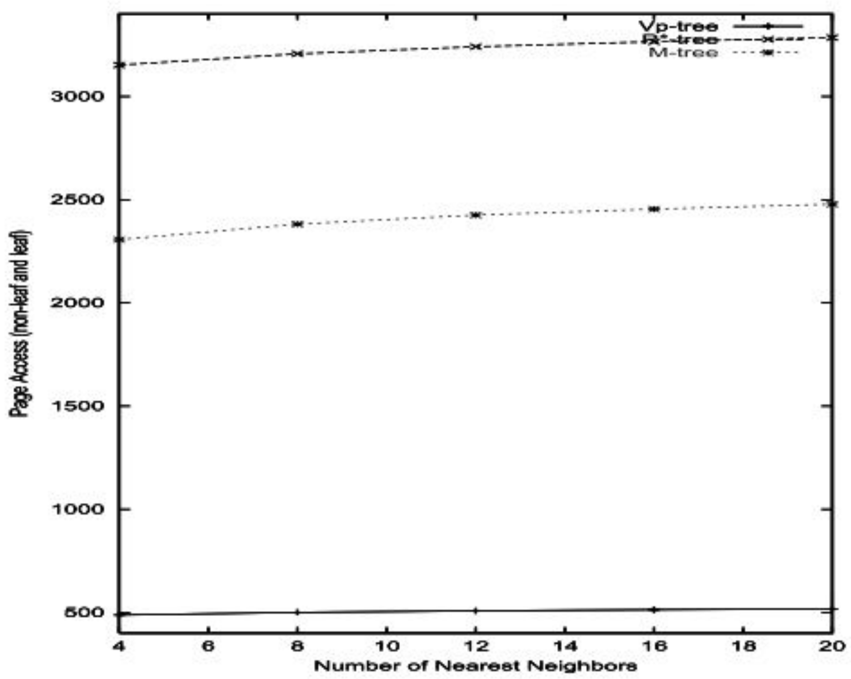


Fig. 9. Comparison with M-tree and R^*-tree on synthetic clustered data, dataset size=30,000, #nearest neighbors=8

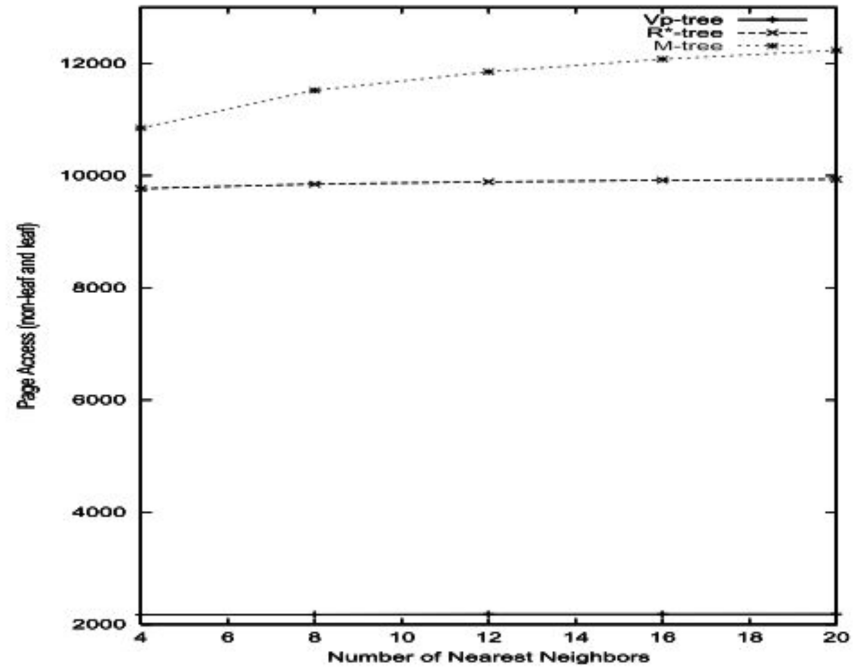


(a) #nearest neighbors=8

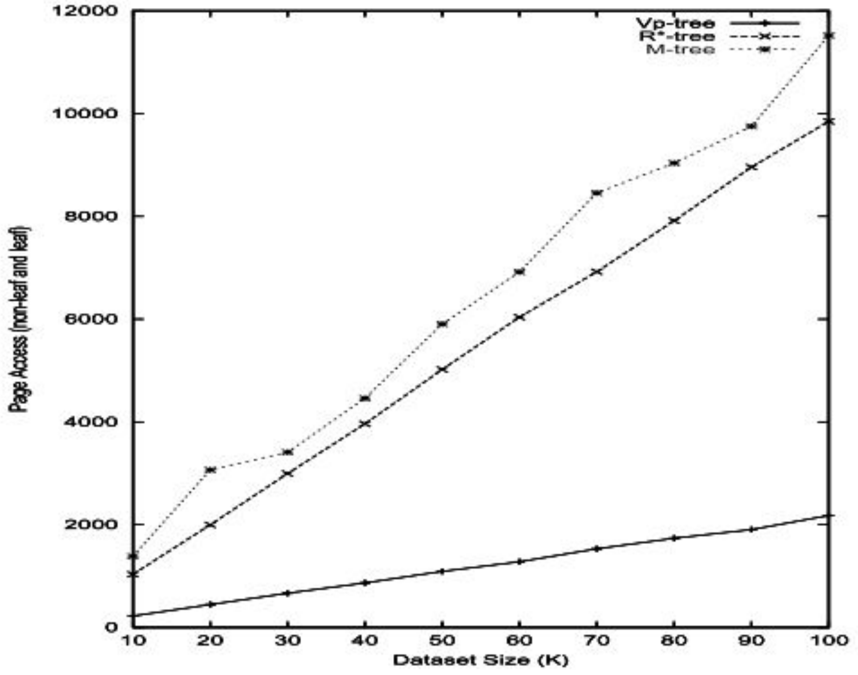


(b) dataset size=30,000

Fig. 8a,b. Comparison with the M-tree and R^*-tree on synthetic clustered data, dimension=30

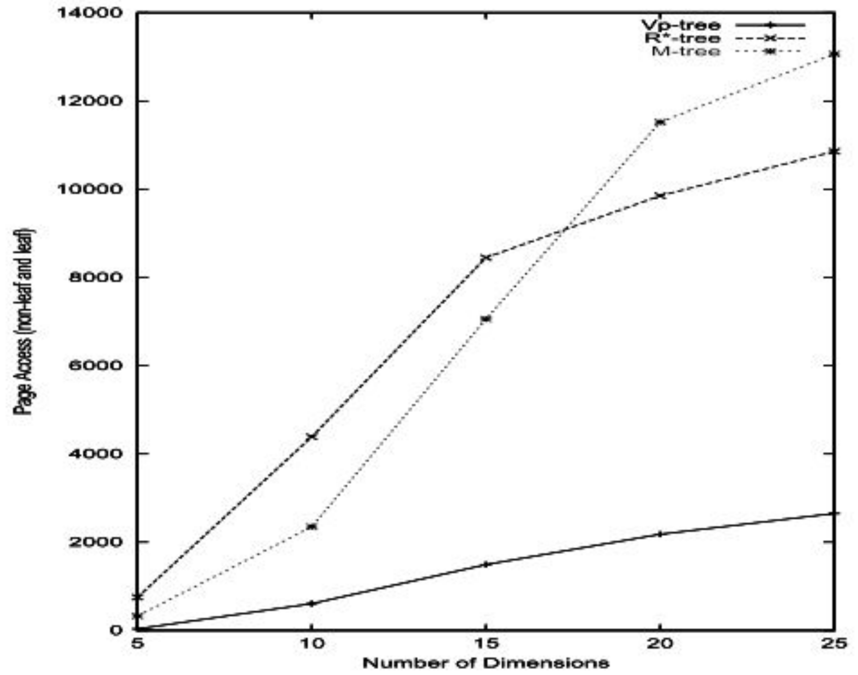


(a) #nearest neighbors=8

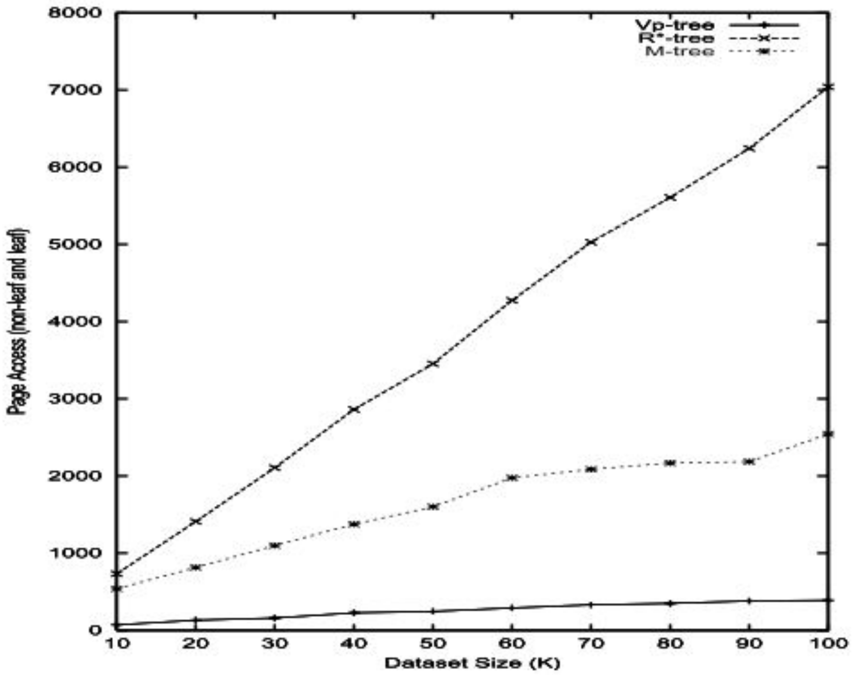


(b) dataset size = 30,000

Fig. 10a,b. Comparison with M-tree and R^*-tree on synthetic uniform data, dimension=20

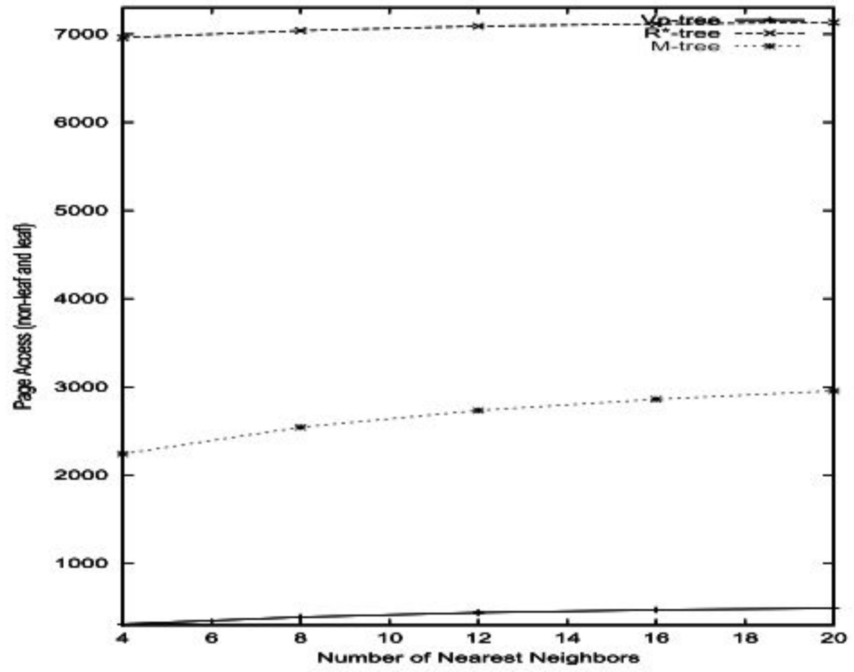


(a) synthetic uniform data, dataset size=30,000

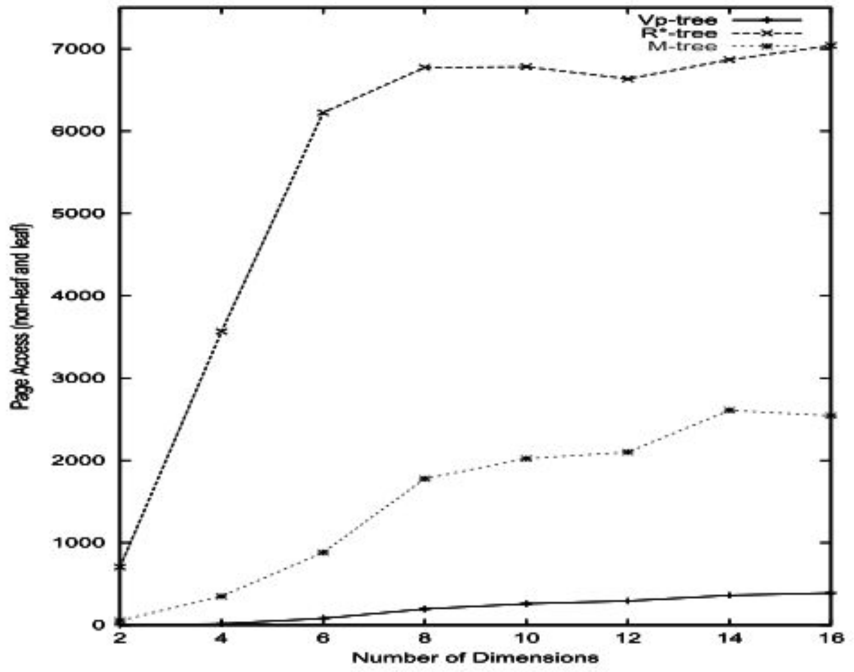


(b) real data, number of dimensions = 16

Fig. 11a,b. Comparison with M-tree and R^*-tree, dataset size=30,000, #nearest neighbors=8



(a) dimension=16



(b) #nearest neighbor=8

Fig. 12a,b. Comparison with M-tree and R^*-tree on real data, dataset size=30,000

04

Conclusion

CONCLUSIONS:

- Distance-based indexing structures like the vp-tree offer advantages over feature vector inference approaches for content-based retrieval in multimedia data.
- Vp-trees eliminate preprocessing steps, handle updating efficiently, and are applicable to multi-dimensional data representations.
- Experimentation demonstrates the scalability and superior performance of the vp-tree in n-nearest neighbor search compared to R∗-tree and M-tree across various data types.
- Efficient vantage point selection methods contribute to reducing computational complexity in vp-tree implementations.
- Overall, the vp-tree emerges as a robust and scalable solution for content-based retrieval, showcasing promising potential for multimedia data applications.

05

Future Works/ Suggestions

CONCLUSIONS:

- **Vantage Point Selection Methods:**
  - Investigating alternative methods for selecting vantage points, aiming to address the criticism of asymmetric regions in vp-trees.
  - Considering vantage points positioned farther away from the search space to achieve more symmetric boundaries.
- **Efficient Vantage Point Location:**
  - Exploring efficient methods, such as the heuristic in reference [24], to locate vantage points that are distant from the search space's center.
  - Utilizing standard deviation as a metric to identify vantage points with fewer data points around the spherical boundary created by the median.
- **Improvements to vp-tree:**
  - Considering enhancements inspired by advancements in R∗-tree and X-tree structures to improve the performance of vp-trees.
  - Exploring ideas such as introducing redundancy or incorporating flavors of linear search into vp-tree structures.

Thank you!