

# In-Picture Search Algorithm for Content-Based Image Retrieval

Punpiti Piamsa-nga

*Department of Computer  
Engineering,  
Kasetsart University,  
Bangkok, 10900, Thailand  
pp@ku.ac.th*

Nikitas A. Alexandridis,  
Sanan Srakaew,  
George C. Blankenship Jr.

*Department of Electrical and  
Computer Engineering,  
George Washington University,  
Washington DC, 20052, USA  
{alexan, srakaew, blankeng}  
@seas.gwu.edu*

S. R. Subramanya

*Computer Science Department,  
University of Missouri-Rolla,  
Rolla, MO 65409, USA  
subra@umr.edu*

## Abstract

*Researchers are currently more interested in searching for fragments that are similar to a query, than a total data item that is similar to a query; the search interest is for “contains”, not “is”. This paper presents an  $O(\log n)$  algorithm, called the “generalized virtual node (GVN)” algorithm; the GVN algorithm is a search algorithm for data fragments that have similar contents to that of a query. An example of the use of the GVN algorithm is in the search of an image database for a certain picture object regardless to what their picture backgrounds are. Each image is transformed into characteristic features and these features are stored in a hierarchical multidimensional structure, called a “k-tree.” The k-tree is exploited to build a unified retrieval model for any types of multimedia data. The experimental results of this “in-picture” search algorithm on an image database demonstrate a search quality is qualitatively and quantitatively acceptable, with a retrieval time is faster than other algorithms, such as brute-force and Partial Matching.*

## 1. Introduction

Image (multimedia) data query can be classified into two different approaches: “a-whole-picture (a-whole-object) search,” and “in-picture (in-object) search.” Each approach generates a different type of query result. “A-whole-picture” or “thumbnail-based” search approach searches for data that is globally similar to the query input; on the other hand, an “in-object” search approach searches for a large piece of data contains a fragment that is similar to the query. An example of a-whole-picture search is to find a picture in a database using the picture or its thumbnail image as a query. An example of “in-picture” search is to find a picture that contains parts that are similar to the query, where the query is a part of an image regardless what the backgrounds are. Most of the

recent work in the field of multimedia retrieval emphasizes the “a-whole-object search” approach [2]; only a few researchers are working on “in-object search” approach [4][5].

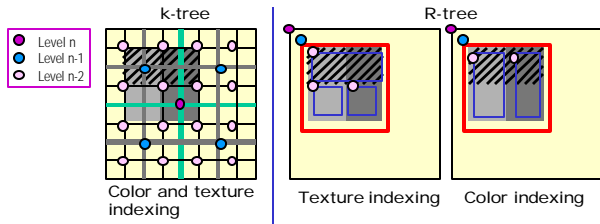
In this paper, a generalized “in-object” search algorithm, called the “Generalized Virtual-Node (GVN)” algorithm, is proposed. It uses a universal model that is able to represent the characteristic features of any multimedia datatype. The performance measurements of the algorithms have been generated using an image database. Using the proposed search algorithm, the accuracy is comparable to and the retrieval time is quite shorter than other algorithms, such as the “Partial-Matching (PM)” algorithm [4].

## 2. The k-tree index structure

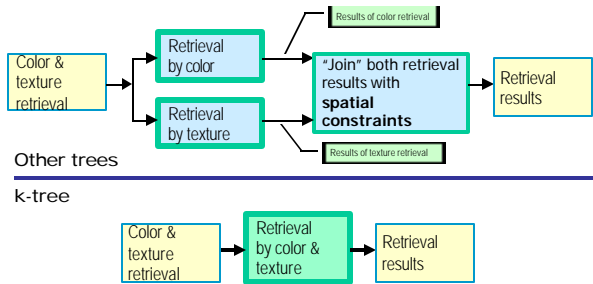
To create an image database, characteristic features (contents) of audio are extracted from the data and used as index key. A k-tree is a directed graph; each node has  $2^k$  incoming edges and one outgoing edge with a balanced structure. The use of a k-tree provides four main benefits [1].

First, the structure of the k-tree is feature independent. Therefore, the positions of the nodes in the tree are always the same, no matters what features are. Figure 1 shows the comparison between using k-tree and R-tree structures as indices by using two different features. Obviously, since R-tree index structures depend on contents of the data, their index structures of different features are not the same. The problem of inconsistent index structure occurs when a multiple-feature query comes. If the indices of different features or different datatypes are processed individually, the database JOIN operation is needed to merge results from each individual index and filter ones that do not comply the temporal or spatial constraints. Compared to other feature-dependent index structure (illustrated in Figure 2), using the k-tree approach to search every feature altogether takes shorter

computing time than using feature-dependent structure to search on many indices individually, merge all results, and filter them with spatial constraints.



**Figure 1: Comparisons between  $k$ -tree and other index tree structures**



**Figure 2: Comparison of multiple feature processing on  $k$ -tree and other index trees**

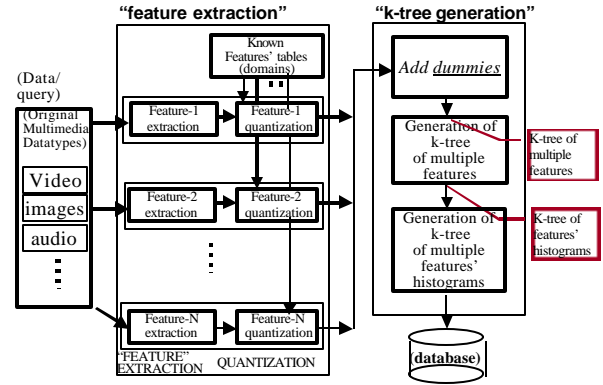
Second, since a  $k$ -tree is a hierarchical data structure, multiresolution processing can be exploited into this structure. In the  $k$ -tree, the lower amounts of features used to describe global information are stored at the higher level of the tree. On the other hand, the higher details of features and local information are stored at the lower level of the tree. Therefore, users' requirements can be adapted between time and accuracy by selecting appropriate level of the tree that is optimal for their requirement.

Third, the complexity of data structure affects only the degree  $k$  of the tree. Consequently, an algorithm for a particular datatype or feature can be reused for another datatype or another feature.

Forth, the  $k$ -tree-based feature index for a feature can be used for many types of queries [1].

### 3. The generalized indexing/retrieval model

The  $k$ -tree structure is used to retain location information and a *histogram* is used to store the characteristics of each portion the data that corresponds to a part of the tree. This generalized model is depicted in Figure 3. First, either general mathematical models, or special methods extract the feature of interest. Second, the domain of datatype is reduced into a set and each item in the database is also mapped to the set. Third, virtual data values are added to data items, if necessary, to create such that each item will generate a balanced  $k$ -tree. A  $k$ -tree is built using histogram values for each feature.



**Figure 3: Generalized indexing/retrieving model**

### 4. In-picture search algorithm

The basic idea of the algorithm is to exploit multiresolution processing in order to eliminate unnecessary branches in the  $k$ -tree structure, where each branch has information of image in the corresponding area. Histogram-based features are used to represent each desired feature [2]. The "histogram intersection" distance function [3] is used to determine which branches of tree the more should be more fruitful during a search. The Algorithm 1 has the details of the limited version of the Virtual-node (VN) algorithm, where the each dimension of data or query must have same power-of-two size; data size and query size do not necessarily have to be equal. The Algorithm 2 is the generalized version of Algorithm 1. The Algorithm 2 does not need to restrict the sizes of data and query to be a power of two. Figure 4 shows the examples of virtual nodes for one-dimensional data.

#### Algorithm 1: The Virtual-Node (VN) in-picture search algorithm

**Case A) If query's tree aligns within the  $k$ -tree structure of data:**

- A.1 Find the distances between feature in root of the query tree and nodes of the data at level  $L_{i-1}$  – nodes with solid-line link in Figure 4 – of the stored item. If distances are equal to the distance between the query and their parents (node of the data at level  $L_i$ ), the query could be found within those child nodes.
- A.2 Repeat *Case A* recursively on this child node. If there is no distance at level  $L_{i-1}$  close to the distance to the parent, the query is "not aligned". Follow *Case B* below.

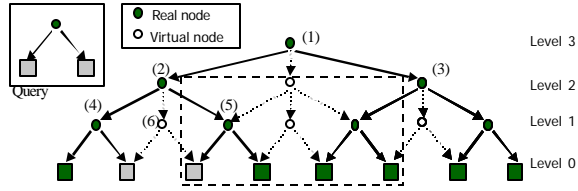
**Case B) If the query data falls in between two or more nodes:**

- B.1 If no node in  $k$ -tree (darker nodes in Figure 4 or rectangular areas in Figure 5) can be a candidate, virtual nodes (white nodes) between two nodes have to be generated from the parts of their child nodes.
- B.2 Repeat the whole algorithm into a new tree; use the whole algorithm within the dashed box in Figure 4.

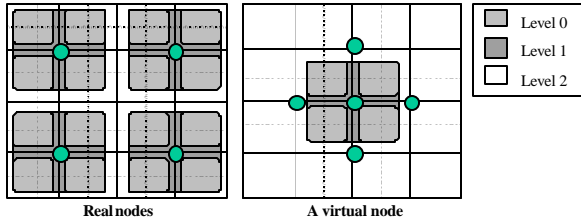
**Case C) If height of query is equal to a node height:**

C.1 Use histogram distance function to calculate the distance then

C.2 Return the distance and location.



**Figure 4: A virtual-node structure for one-dimensional data ( $k=1$ )**



**Figure 5: A virtual-node structure for images ( $k=2$ ) (shows only one of five nodes, which each has its center at the circles)**

**Algorithm 2: The Generalized Virtual-Node (GVN) in-picture search algorithm**

*ExtendedQuery* = AddDummies (*Query*)

*FeatureOfExtendedQuery* =  
FeatureExtraction(*ExtendedQuery*)

**VirtualNodeComparison** (*FeatureOfExtendedQuery*,  
*FeatureOfExtendedData*, *ROOT*, *distance1*,  
*TentativeLocation1*)

**IF** (*distance1* < *threshold1*) **THEN BEGIN**

Find "QueryRepresentative," the largest node in the  $k$ -tree of *FeatureOfQuery*, where no parts of dummies are included.

**VirtualNodeComparison** (*QueryRepresentative*,  
*FeatureOfExtendedData*,  
*TentativeLocation1*, *distance2*,  
*TentativeLocation2*)

**IF** (*distance2* < *threshold2*) **THEN BEGIN**

Find the final distance by calculating the distance between the query and area of data where the beginning of the area is at *TentativeLocation2*.

*Distance* = *distance2*

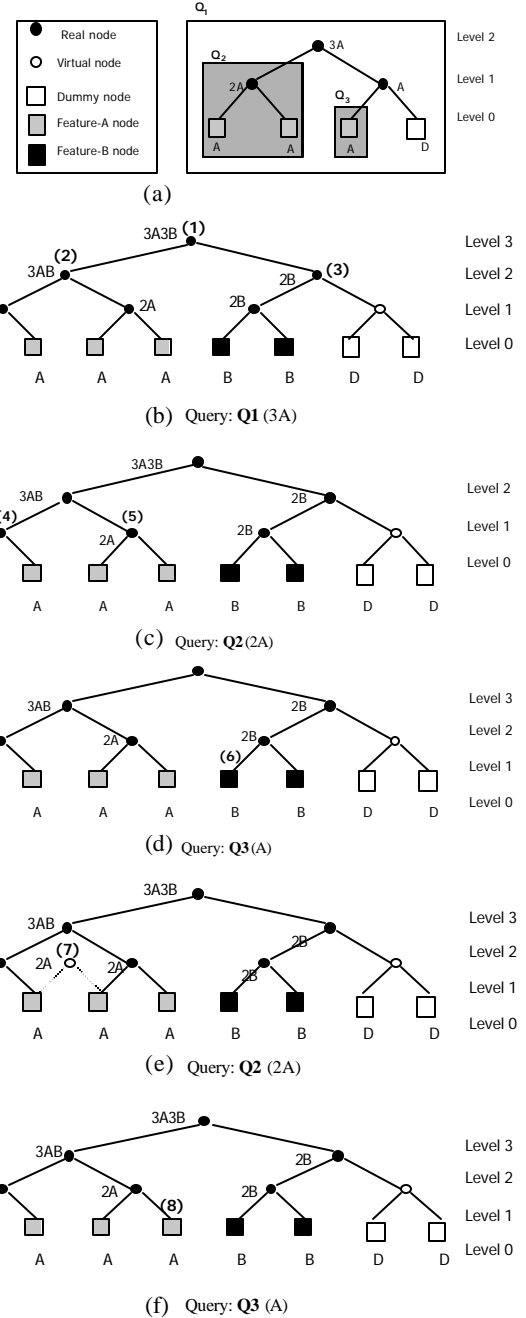
*Location* = *TentativeLocation2*

**RETURN**

**END**

**END**

An example of a GVN algorithm search using one-dimensional data is illustrated in Figure 6; however, the main concept of the algorithm is still the same when it is applied on images.



**Figure 6: Illustration of searching using the GVN algorithm**

Suppose a query is a sequence of three consecutive nodes of feature, which has only two values (A and B) in its domain. A dummy node (D) is appended to a query's feature. The query's feature with a dummy is used to construct a binary tree ( $k=1$ ) of histogram. Similarly, a 6-node-long feature of a search's target is appended by two dummy nodes and then a binary tree of the searched target is constructed and shown in Figure 6 (b). All comparisons use the histogram intersection function [3] as a distance function.

The search begins at (1) by comparing the root of the query (**Q1**) and the root of the searched target. If the comparison result at (1) shows that the target is located at nodes under (1), then **Q1** continues compare with nodes (2) and (3). The comparison at (2) shows that the target should be under (2).

To determine more accurate result, the query's subtree, that does not have a dummy, is used as the key of the search. In Figure 6 (a), the search continues by using the query **Q2**, which is the largest subtree of the query that does not have a dummy node. The search continues at (4) and (5). It is noticeable that (5) is possibly a result. Then the second largest tree (**Q3**) is used to compare at the consecutive location to (5). The comparison at (6) does not give a good result; thus, a virtual-node in Figure 6 (e) is generated and **Q2** is used as a search key again. A possible result is found at (7) and **Q3** is used to compare again with the node next to results of (7). Finally, the position of the target is found.

## 5. Time complexity analysis

Let  $k$  be a number of spatio-temporal dimensions of the data;  $S_d$  be the average size of data in the database; and  $S_q$  be the size of query. (The size of the data is the number of leaves in a query or data record tree.) For the brute-force, the time complexity is  $(\sqrt[k]{S_d} - \sqrt[k]{S_q} + 1)^k$ . For the PM, the time complexity is  $(S_d/S_q) + 3^k$ . For the GVN, the time complexity is  $3^k \cdot \log_2(S_d/S_q)$  plus time of generating the virtual node, which is  $(3^k - 2^k) \cdot \log_2(S_d/S_q)$ . The brute-force algorithm has linear time complexity; it grows with  $S_d$ . The new algorithm has time complexity  $O(\log(S_d/S_q))$ . However, as  $k$  grows, the constant term may have a more significant effect on the processing time.

## 6. Experiments

The experiments have been performed for searching image data using the GVN, PM, and brute-force algorithms. The experiments were concerned primarily with two issues: the search results and retrieval times.

### 6.1 Experimental platform

The image database consists of over 30,000 photographic pictures, where the database size is over one gigabyte. Picture sizes in the database are ranged from  $320 \times 200$  to  $1024 \times 1024$  pixels. The index of this database is generated by using color and texture histograms. The histograms are constructed by counting the number of pixels of each color or texture in a particular area of an image clip. The GVN, PM, and brute-force algorithms are used as the search algorithms.

### 6.2 Measuring effectiveness of the search

To evaluate the effectiveness of the search, the results from each query are a list of picture identifications with their distances. Three metrics are used to measure the effectiveness of the retrieval: *retrieval time*, *recall*, and *precision*. Retrieval time is the period between when the input query comes and when the outputs are produced. Recall indicates the proportion of relevant data in the whole database that are retrieved by a query. Precision signifies the proportion of the retrieved data that are relevant to the query.

To measure the recall and precision, a  $64 \times 64$ -pixel picture in the database was randomly selected as a query. The selected picture was imposed onto each picture in the image database; the imposed position was randomly located and kept as a reference. All imposed pictures were used to generate a test database. The index of this database was generated using color and texture histogram features. (The  $k$ -tree is a quad tree ( $k=2$ ) where each node contains both color and texture histogram features in a corresponding area.) The GVN, brute-force, and PM algorithms were used as the search algorithms.

During the search, if a distance between a query and a part of an image was less than a specified *similarity threshold*  $T$  that part of image was counted as a *retrieved* result. If the positions of the retrieved results were within the length of query (64 pixels for this test case), the target were defined as a "*hit* ( $H$ ).". If they are not within 64 pixels from the imposed positions, the retrieved outputs are defined as a "*false alarm* ( $F$ ).". If an imposed clip is not retrieved, the target was defined as a "*miss* ( $M$ ).". In general, Recall and precision can be defined by:  $Recall = H/(H+M)$ , and  $Precision = H/(H+F)$ .

However, because the "hits" are determined from the distances that are lower than the threshold  $T$ , both the recall and precision varies to the values if the similarity threshold  $T$ . In this paper, the recall and precision are modified in order to eliminate effect of the similarity threshold by using *probabilities* of the "hits" instead of the *number* of the "hits." Therefore, we introduce a modified version of recall and precision to evaluate the similarity search.

The definitions of the modified recall and modified precision are shown as follows. Let  $d_i$  be a distance between the query and retrieved data  $i$ .

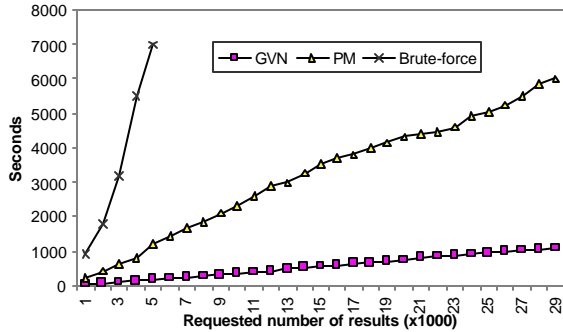
$$\text{Modified Recall} = \frac{\sum_{i=0}^H (1-d_i)}{\sum_{i=0}^H (1-d_i) + M}$$

$$\text{Modified Precision} = \frac{\sum_{i=0}^H (1-d_i)}{\sum_{i=0}^H (1-d_i) + F}$$

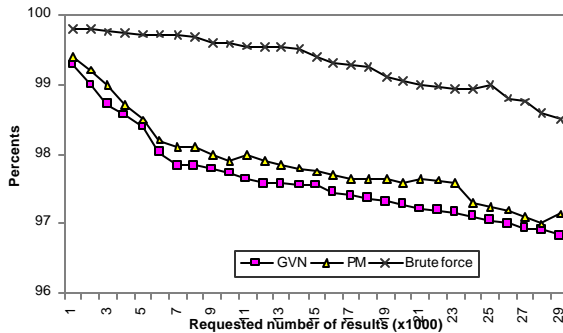
### 6.3 Quantitative results

The process to evaluate the effective of the search is iterated several times with different target image. All measured results are used to calculate the averages of

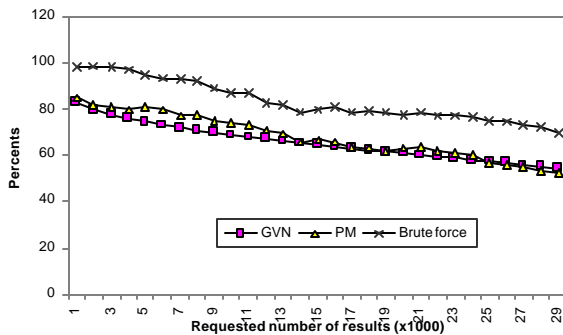
retrieval time, recall, and precision for all GVN, PM, and brute-force algorithms. The experiment results are presented in Figure 7 shows that retrieval time using the GVN algorithm is much faster than using the PM algorithm, while there are little differences for recall and precision.



(a) Retrieval time



(b) Recall



(c) Precision

Figure 7. Experimental results

### 6.4 Qualitative results

The qualitative results are also shown by the examples of the first 20 results in Figure 8. All examples do not have the same size but they are scaled just for being displayed. The square box in each example shows the position of picture that the content is similar to the query. Note that, by the benefit of the  $k$ -tree that the spatial information is embedded in the tree, all pictures have sky blue at the upper parts and dark green at the lower parts.

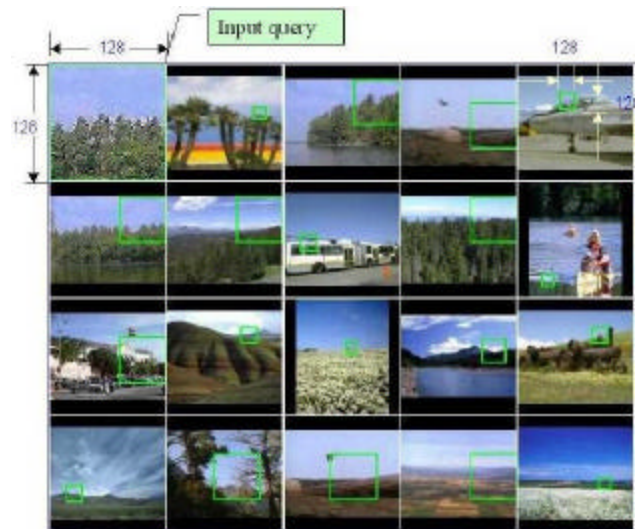


Figure 8. Querying results using color feature. (The picture at the top-left corner is the query. Decreasing accuracy of results is left-to-right, top-to-bottom)

## 7. Conclusion

A generalized algorithm to search parts of image or multimedia data is presented. This algorithm is extensible onto other types of multimedia data with less effort since it is developed on the unified multimedia retrieval model. The search results show that the proposed algorithm is faster and more accurate than other algorithms.

## 8. References

- [1] P. Piamsa-nga, "A unified histogram-based multiresolution approach for content-based multimedia retrieval" *D.Sc. Dissertation*, George Washington University, 1999.
- [2] J. R. Smith, "Integrated spatial and feature image systems: retrieval, analysis, and compression," *Ph.D. Thesis*, Columbia University, 1997.
- [3] M. J. Swain and D. H. Ballard, "Color Indexing," *International Journal of Computer Vision*, 7:1, 1991.
- [4] S. R. Subramanya, P. Piamsa-nga, N. A. Alexandridis, and A. Youssef, "A scheme for content-based image retrieval for unrestricted query formats," *International Conference on Imaging Science, Systems, and Technology (CISST'98)*, Las Vegas, Nevada, July 1998
- [5] A. Yoshitaka and T. Ichikawa, "A survey on content-based retrieval for multimedia database," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 11, No. 1, Jan.-Feb 1999. pp. 81-93.