

A DYNAMIC CONTENT-BASED INDEXING METHOD FOR MULTIMEDIA DATABASES: *HIERARCHICAL CELLULAR TREE*

Serkan Kiranyaz and Moncef Gabbouj

Institute of Signal Processing, Tampere University of Technology, Tampere, Finland
serkan@cs.tut.fi, moncef.gabbouj@tut.fi

Abstract—this paper presents a novel indexing technique, Hierarchical Cellular Tree, which is designed to bring an effective solution especially for indexing on large-scale multimedia databases. A pre-emptive cell search mechanism is introduced in order to prevent the corruption of large multimedia item collections due to the limited discrimination obtained from the visual and aural descriptors. In addition to this, the similar items are focused within appropriate cellular structures, which will be the subject to mitosis operations when the dissimilarity emerges as a result of irrelevant item insertions. Mitosis operations ensure to keep the cells in a focused and compact form and yet the cells can grow into any dimension as long as the compactness prevails. The proposed indexing scheme is then optimized for a novel query method, the Progressive Query, in order to maximize the retrieval efficiency for the user point of view. Experimental results show that the speed of the retrievals is significantly improved.

Keywords—indexing; content-based; retrieval; multimedia.

I. INTRODUCTION

For three decades the researchers proposed several indexing techniques that are formed mostly in a hierarchical tree structure that is used to cluster (or partition) the feature space. Several *Spatial Access Methods* (SAMs) are proposed such as KD-Trees [1], R-tree [2], SS-tree [3], Hybrid-Tree [4], etc. Especially for content-based indexing and retrieval in large-scale multimedia databases, SAMs have several drawbacks and significant weaknesses. By definition an SAM-based indexing scheme partitions and works over a single feature space. However a multimedia database can have several feature types (visual, aural, etc.), each of which might also have multiple feature subsets. Furthermore, SAMs assume that query operation time and complexity are only related to accessing a disk page (I/O access time) containing the feature vector. This is obviously not a trivial assumption for multimedia databases and consequently, no attempt in the design of SAMs has been done to reduce the similarity distance computations (CPU time). In order to provide a more general approach to similarity indexing for multimedia databases, several static and dynamic Metric Access Methods (MAMs) are proposed such as VP-tree [8], MVP-tree [7], GNAT [11] whereas the dynamic ones, M-Tree [6] and other M-Tree variants such as M+-Tree [5]. The generality of MAMs comes from the fact that any MAM employs the indexing process by assuming only the availability of a similarity distance function, which satisfies three trivial rules: symmetry, non-negativity and triangular inequality

As a summary, the indexing structures so far addressed are all designed to speed up any QBE process by using some multidimensional index structure. However all of them have significant drawbacks and shortcomings for the indexing of large-scale multimedia databases. In order to overcome such problems and provide efficient solutions to the aforementioned shortcomings of the indexing algorithms especially for the multimedia databases, we develop a MAM-based, dynamic and self-

organized indexing scheme, the *Hierarchical Cellular Tree* (*HCT*). As its name implies, *HCT* has a hierarchic structure, which is formed into one or more levels. Each level is capable of holding one or more cells. The cell structure is nothing but an acronym of the node structure in M-tree. The reason we call a different name for it is because the cells further contain a tree structure, the Minimum Spanning Tree (MST), which refers to the database objects as its MST nodes. Among all the indexing structures available, M-tree shows the highest structural similarity to *HCT*, however there are several major differences in their design philosophies and objectives:

- M-tree is a generic MAM, designed to achieve a balanced tree with a low I/O cost in large data set. *HCT* is on the other hand designed for indexing multimedia databases where the content variation is seldom balanced and it is especially optimized for compactness (focused cells).
- M-tree works over the nodes with a maximum (fixed size) capacity M . *HCT* on the other hand has no limit for the cell size as long as the cell keeps a definite “compactness” measure.
- The split (mitosis) policies and objectives are completely different between M-tree and *HCT*.
- M-tree item insertion operation is based on sub-optimum cell search Most-Similar Nucleus (MS-Nucleus) whereas *HCT* is designed to perform an optimum search, so called Pre-emptive cell search.

Along with the indexing techniques addressed so far, certain query techniques have to be used to speed up a QBE process within indexed databases. The most common query techniques are *Range* and *kNN Queries*. Unfortunately both of the query techniques may not provide efficient retrieval scheme in the user point of view due to their parameter dependency. In order to eliminate that and provide a user-friendly query scheme, recently a novel retrieval method, the *Progressive Query* (*PQ*), has been proposed [10]. When the database has an indexing structure, *PQ* can replace *kNN* and *range* queries whenever a *query path* (*QP*) over which *PQ* proceeds, can be formed. Instead of relying on some unknown parameters such as k or ϵ , *PQ* provides periodic (with a user-defined time period) query results along with the query process and lets the user browse around the queries obtained and stops the ongoing query in case the results obtained so far are satisfactory and hence no further time should unnecessarily be wasted. Therefore, the proposed (*HCT*) indexing technique has been designed to work in harmony with *PQ* in order to evaluate the retrieval performance in the end.

The rest of this paper is organized as follows: in Section II we introduce the generic *HCT* design. Section III is devoted for *PQ* operation over *HCT* indexing. Section IV reports some experimental results. Finally Section V concludes the paper.

II. *HCT* OVERVIEW

HCT is a dynamic, cell-based and hierarchically structured indexing method, which is purposefully designed for *PQ* operations and advanced browsing capabilities within large-scale multimedia databases. It is mainly a hierarchical clustering method where the items

are partitioned depending on their relative distances and stored within cells on the basis of their similarity proximity. The similarity distance function implementation is a *black-box* for the *HCT* and it is a self-organized tree, which is implemented by genetic programming principles. This basically means that the operations are not externally controlled, instead each operation such as item insertion, removal, mitosis, etc. are carried out according to some internal rules within a certain level and their outcomes may uncontrollably initiate some other operations on the other levels.

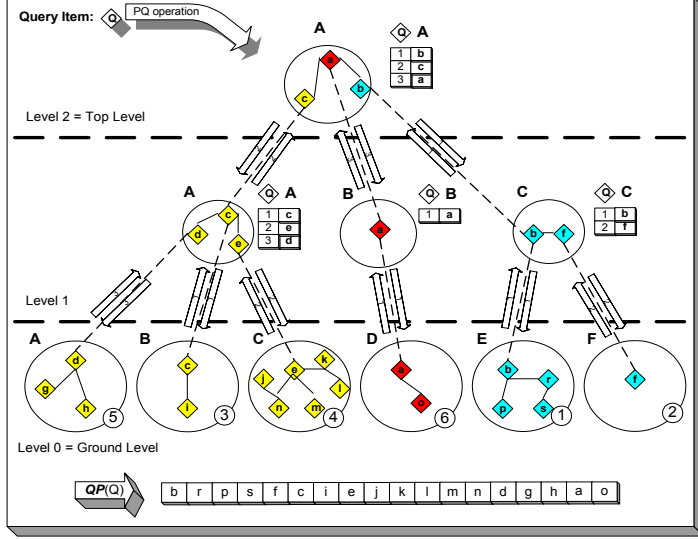


Figure 1: A Sample 3-levels HCT body.

A. Cell Structure

A cell is the basic container structure, in which the similar database items are stored. The ground level cells contain the entire database items. Each cell further carries a MST where the items are spanned via its nodes. This internal MST is used to keep the minimum (dis-)similarity distance of each individual item to the rest of the items in the cell. So this scheme resembles MVP-tree structure; instead of using some (pre-fixed) number of items, all of the cell items are now used as the vantage points for any (other) cell item. In *HCT*, the cell size is kept flexible, which means there is no fixed cell size that cannot be exceeded. However there is a maturity concept for the cells in order to prevent a mitosis operation before the cell reaches a certain level of maturity. Otherwise we cannot obtain healthy information whether or not the cell is ready for mitosis since there is simply not enough statistical data that are gathered from the cell items and its MST. Therefore, using a similar argument for the organic cells, a maturity cell size (i.e. $N_M \geq 5$) is set for all the cells in *HCT* body (level independent).

Cell nucleus is the item, which represents the owner cell on the higher level(s). Since during the top-down cell search for an item insertion, these nucleus items are used to decide the cell into which the item should be inserted, it is essential to promote the best item for this representation on any instant. When there is only one item in the cell, it is obviously the nucleus item of that cell. Otherwise the nucleus item is assigned by using the cell MST as the item having the maximum number of branches (connections to other items). This heuristics makes sense since it is the unique item to which majority of the items has the closest proximity to it (according to the MST optimality on the minimal branch weights). Contrary to static nucleus assignment of the some other MAM-based indexing schemes such as M-tree, the cell nucleus is dynamically verified and if necessary updated for *HCT* whenever an

operation is performed over the cell in order to maintain the best representation of the (dynamically changing) cell and there is no computational cost for this so far since it can be extracted directly from the MST (branch) data.

Once a cell reaches maturity (a pre-requisite for the compactness feature calculation) then a regularization function, f , can be expressed using the following statistical cell parameters:

$$CF_C = f(\mu_C, \sigma_C, r_C, \max(w_C), N_C) \geq 0 \quad (1)$$

where μ_C and σ_C is the mean and standard deviation of the MST branch weights, w_C , of the cell C . r_C is the covering radius, that is the distance from the nucleus within which all the cell items lie and $N_C > N_M$ is the number of items in the cell C . Accordingly a regularization function should then be implemented to minimize the compactness feature, CF_C . In the limit, the highest compactness can be achieved when $CF_C = 0$ which means that all the cell items are identical.

B. Level Structure

HCT body is hierarchically partitioned among one or more levels, as one sample example shown in Figure 1. In this example there are three levels that are used to index 18 items. Apart from the top level, each level contains various numbers of cells that are created by mitosis operations occurred in that level. The top level contains single cell and when this cell splits, then a new top level is created above this level. As mentioned earlier, the nucleus item of each cell in a particular level is represented on the higher level. Each level is responsible for taking logs about the operations performed in it, such as number of mitosis operations, the statistics about the compactness feature of the cells, etc. Note that each level dynamically tries to maximize the compactness of their cells although this is not a straightforward process to do since the incoming items may not show a similarity to the items present in the cells and therefore, such dissimilar item insertions will cause a temporary degradation on the overall (average) compactness of the level. So each level, while analyzing the effects of the (recent) incoming items on the overall level compactness, should employ necessary management steps to provide a trend of improving compactness in due time (i.e. with the future insertions). Within a period of time (i.e. during a number of insertions or after some number of mitosis occurs), each level updates its compactness threshold according to the compactness feature statistics of the mature cells, into which an item inserted. Therefore, $CThr_L$ value for a level L can be estimated as follows:

$$CThr_L = \frac{k_0}{P} \sum_{\substack{C \in S_P \\ |N_C| > N_M}} CF_C = k_0 \mu_{CF_C} \quad \forall C \in S_P \quad (2)$$

where S_P is the set of mature cells, upon which P insertions are recently performed and $0 < k_0 \leq 1$ is the compactness enhancement rate, which determines how much enhancement will be targeted for the next P insertions beginning from the moment of the latest $CThr_L$ setting. If $k_0 = 0$ then the cells will split each time they reach the maturity and in this case *HCT* split policy will be identical to M-tree.

C. HCT Operations

Item insertion is a level-based operation and can be implemented per item basis. Let $nextItem$ be the item to be inserted into a target level indicated by a number, $levelNo$. Let $d(\cdot)$ be the similarity distance function, O be the object to be inserted, O_N^i and $r(O_N^i)$ be the nucleus object and its covering radius for the i^{th} cell, C_i , respectively.

Let d_{\min} be the distance to the closest nucleus item (in the upper level).

Insert algorithm can therefore, be expressed as follows:

```

Insert (nextItem, levelNo)
➤ Let top level number: topLevelNo and the single cell in top level: cell-T
➤ If(levelNo > topLevelNo) then do:
  o Create a new top level: level-T with number topLevelNo+1
  o Create a new cell in level-T: cell-T
  o Append nextItem into cell-T.
  o Return.
➤ Let the Owner (target) cell in level levelNo: cell-O
➤ If(levelNo = topLevelNo) then do:
  o Assign cell-O = cell-T
➤ Else do:
  o Create a cell array for Pre-emptive cell search: ArrayCS[], put cell-T into it
  o Assign cell-O = PreEmptiveCellSearch (ArrayCS[], nextItem, topLevelNo)
➤ Append nextItem into cell-O.
➤ Check cell-O for Post-Processing:
  o If cell-O is split then do:
    ▪ Let item-O, item-N1 and item N2 be old nucleus item (parent) and new nucleus items (2 child)
    ▪ Remove( item-O, levelNo+1)
    ▪ Insert(item-N1, levelNo+1)
    ▪ Insert(item-N2, levelNo+1)
  o Else if nucleus item is changed within cell-O then do:
    ▪ Let item-O and item-N be old and new nucleus items.
    ▪ Remove( item-O, levelNo+1 )
    ▪ Insert( item-N, levelNo+1 )
➤ Return.

```

The *pre-emptive* cell search rationale can be expressed as follows: fetch all the nucleus items whose cells in the lower level may provide the closest object, $\Delta_i = d(O, O_N^i) - r(O_N^i) \leq d_{\min} \quad \forall C_i$, among all the nucleus objects that are in the owner cell C . Accordingly the *Pre-emptive* cell search algorithm, **PreemptiveCellSearch**, can be expressed as follows:

```

PreemptiveCellSearch (ArrayCS[], nextItem, LevelNo)
➤ By searching  $\forall O_N^i \mid O_N^i \in C_i \wedge \forall C_i \in \text{ArrayCS}$  find the most similar item, item-MS and  $d_{\min}$ .
➤ If (curLevelNo = levelNo + 1) then do:
  o Let the owner cell of item-MS: cell-MS in the (target) level (with level number: levelNo)
  o Return cell-MS
➤ Create an array for cell search: NewArrayCS[] =  $\emptyset$ 
➤ For  $\forall O_N^i \mid O_N^i \in C_i \wedge \forall C_i \in \text{ArrayCS}$ , do:
  o If (  $\Delta_i = d(O, O_N^i) - r(O_N^i) \leq d_{\min}$  ) then do:
    ▪ Find the owner cell of (nucleus) item  $O_N^i$  in the lower level:  $cell - C_N^i$ 
    ▪ Append  $cell - C_N^i$  into NewArrayCS[]
➤ End loop.
➤ PreemptiveCellSearch(NewArrayCS[], nextItem, curLevelNo-1 )

```

Due to space limitations, item(s) removal is not described in this article.

III. PQ OVER HCT

The *Progressive Query* (PQ) is a novel retrieval scheme, which presents *Progressive Sub-Query* (PSQs) retrieval results periodically to the user and allows user to interact with the ongoing query process. Due to the several advantages, HCT is designed to work in harmony with PQ. More information about PQ can be obtained in [10].

PQ operation over HCT is executed synchronously over two parallel processes: HCT tracer and a generic process for PSQ formation using the latest QP segment. HCT tracer is a recursive algorithm, which traces among the HCT levels in order to form a QP (segment) for the next PSQ update. When the time allocated for this operation is completed, this process is paused and the next PSQ retrieval result is formed and presented to the user. Then HCT tracer is re-activated for the next PSQ update and both processes stay alive unless the user stops PQ or entire PQ process is completed (when all the indexed database items are covered). The following **HCTtracer** algorithm implements HCT tracer operation, which basically extracts the next QP segment into a generic array, ArrayQP[] as shown in Figure 1. It is initially called with the top-level number and an item from the single cell in the top level.

```

HCTtracer (ArrayQP[], levelNo, item-MS)
➤ Let cell-MS be the owner cell of item-MS.
➤ If (levelNo = 0) then do: // if this is ground level
  o Append all items in cell-MS into ArrayQP[].
  o Return.
➤ Else do: // if this is an intermediate level
  o Create the priority queue of cell-MS: queue-MS.
  o For  $\forall O_N^i \in \text{queue-MS}$ , do:
    ▪ HCTtracer (ArrayQP[], levelNo-1,  $O_N^i$  )
➤ Return.

```

Note that this algorithm is executed as a separate process (thread) and can be paused externally from the main PQ process when the time comes for the next PSQ update. Once the QP segments are formed, PQ operation that is executed over HCT body becomes similar to the sequential PQ.

IV. EXPERIMENTAL RESULTS

The experiments are carried out using MUVIS system [9], [10] applications under which HCT Browsing and PQ over HCT are primarily developed and tested. All experiments are carried out on a Pentium-4 3.06 GHz computer with 2048 MB memory. The evaluations of the retrieval results are performed subjectively using ground-truth method. For (HCT) indexing of the databases used in this section the following regularization function is used:

$$CF_c = f(\mu_c, \sigma_c, r_c, \max(w_c), N_c) = K\mu_c\sigma_c r_c \max(w_c) \sqrt{N_c} \quad (3)$$

The performance evaluation is about the speed (or timing) of PQ over HCT operation compared both with the Sequential PQ and Normal Query (NQ). For this we performed 10 visual and aural retrieval experiments using all three query methods and we measured the time to retrieve at least 90% of the all relevant items that are subjectively determined using the *ground-truth* method within each database. We used PQ period, $t_p \leq 3\text{sec}$, and the results are presented in Table I. As

expected, over 20 query operations performed, PQ over HCT achieves the fastest retrievals and it yields the retrieval result in the first periodic update of PQ except 4 aural retrievals. In Figure 2 top plot; a query image has a group of 97 relevant images among 1000 images in the database and in the bottom plot; a query video has a group of 21 relevant video clips among 200 video clips. It can be seen from the figures that HCT tracer successfully captures all of the

relevant items at the beginning of QP . Therefore, PQ operation will be presenting them (first) to the user immediately after the query operation is initiated. Another important remark should be

made about the “trend” of the QP plots, that is, it traces along the increasing order of dissimilarity, as intended.

Table I: Retrieval times (in msec) for 10 visual and aural query operations performed per query type.

Query Genre	Query Type	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
Visual	NQ	19624	9407	5938	5922	7776	6774	7290	7799	4954	7039
	$Seq. PQ$	12007	5974	3997	6000	6003	4004	6002	2003	2001	3002
	PQ over HCT	3003	2003	2002	3139	2501	1501	3002	2003	1504	2001
Aural	NQ	37675	29820	31977	36652	54869	39905	48553	58897	84921	61080
	$Seq. PQ$	21004	21005	18006	36023	45003	18050	30023	24003	82998	57057
	PQ over HCT	3001	3000	12003	1501	1500	12033	3002	9000	12000	9002

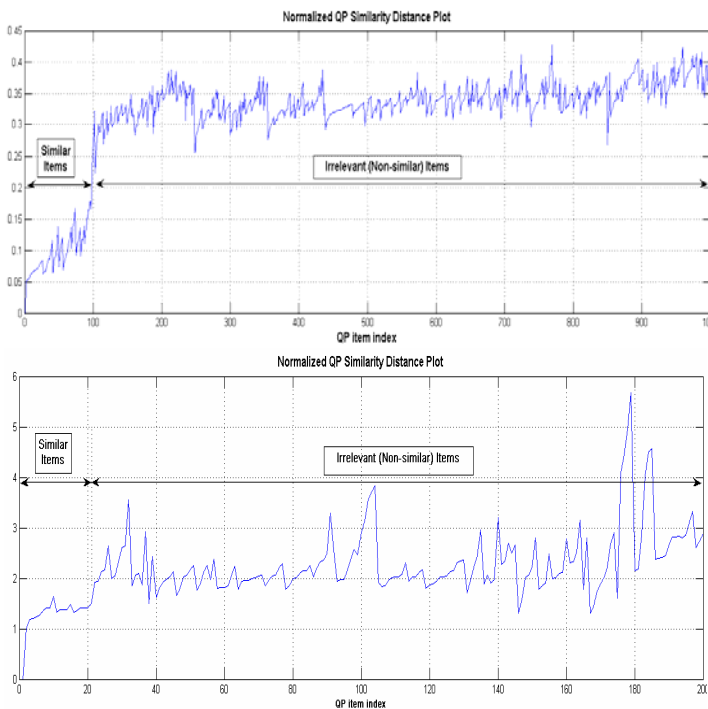


Figure 2: QP plots obtained for sample image (top) and video (bottom) PQ operations.

V. CONCLUSIONS

In this paper, we proposed HCT indexing structure designed for multimedia databases to achieve the following innovative properties:

- HCT is a dynamic, parameter independent and flexible cell (node) sized indexing structure, which is optimized to achieve as focused cells as possible using the visual and aural descriptors.
- By means of the flexible cell size property, one or minimum number of cell(s) are used to store a group of similar items.
- During their life-time the cells are under the close surveillance of their levels in order to enhance the compactness using mitosis operations whenever necessary to rid of the dissimilar item(s) from the cell. Furthermore, for the item insertions, an optimum cell search technique (Pre-emptive) is used to find out the most suitable (similar) cell in each level.
- HCT is also intrinsically dynamic, meaning that the cell and level parameters and primitives are subject to continuous upgrade operations to provide most reliable environment. Such a dynamic

internal behavior keeps the HCT body intact by preventing the potential sources of corruption.

- By means of MST within each cell, the optimum nucleus item can be assigned whenever necessary and with no cost. Furthermore the optimum split management can be done when the mitosis operation is performed. Most important of all, MST provides a reliable compactness measure via “cell similarity” for any item instead relying on only a single (nucleus) item. By this way a better judgment can be done whether or not a particular item is suitable for a mature cell.

- HCT is particularly designed to work with PQ in order to provide the earliest possible retrievals of the relevant items.

Experimental results approve that HCT achieves all the abovementioned properties and capabilities in an automatic and parameter invariant way. The analysis obtained from different databases suggests that HCT usually yields better clustering performance when the discrimination factor of the features is sharper and it achieves more relevant cell structure for the user point of view.

REFERENCES

- [1] J. L. Bentley, “Multidimensional binary search trees used for associative searching”, Communications of the ACM, v.18 n.9, p.509-517, Sept. 1975.
- [2] A. Guttman, “R-trees: a dynamic index structure for spatial searching”, Proc. of ACM SIGMOD, pp. 47-57, 1984.
- [3] D. White and R. Jain, “Similarity Indexing with the SS-tree”, In Proc. Of the 12th IEEE Int. Conf. On Data Engineering, pp. 516-523, 1996.
- [4] K. Chakrabarti and S. Mehrotra, “The hybrid tree: An index structure for high dimensional feature spaces”, In Proc. Int. Conf. on Data Engineering, pages 440--447, February 1999.
- [5] X. Zhou , G. Wang , J. X. Yu , G. Yu, “M+-tree: a new dynamical multidimensional index for metric spaces”, Proc. of the Fourteenth Australasian database conference on Database technologies 2003, pp.161-168, Adelaide, Australia, February 01, 2003.
- [6] P. Ciaccia, M. Patella, and P. Zezula, “M-tree: an efficient access method for similarity search in metric spaces”, In International Conference on Very Large Databases (VLDB), pages 426-435, Athens, Greece, August 1997.
- [7] T. Bozkaya, Z. M. Ozsoyoglu, “Distance-Based Indexing for High-Dimensional Metric Spaces”, Proc. ACM-SIGMOD: pp.357-368, 1997.
- [8] P. N. Yianilos, “Data structures and algorithms for nearest neighbor search in general metric spaces”, Proc. of the fourth annual ACM-SIAM Symposium on Discrete algorithms, pp.311-321, Austin, Texas, US, January 25-27, 1993.
- [9] MUVIS. <http://muvis.cs.tut.fi>
- [10] S. Kiranyaz, M. Gabbouj, “A Novel Multimedia Retrieval Technique: Progressive Query (WHY WAIT?)”, WIAMIS Workshop, Lisboa, Portugal, April 2004.
- [11] S. Brin, “Near Neighbor Search in Metric Spaces”, VLDB, pp. 574-584, 1995