A NOVEL MULTIMEDIA RETRIEVAL TECHNIQUE: *PROGRESSIVE QUERY* (WHY WAIT?)

Serkan Kiranyaz and Moncef Gabbouj

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

ABSTRACT

One of the challenges in the development of content-based multimedia indexing and retrieval application is to achieve an efficient retrieval scheme. The developers and users who are accustomed to making queries and thus retrieving any multimedia item from a large scale database can be frustrated by the long query times and memory and minimum system requirements. This paper presents a novel retrieval technique, which is designed to bring an effective solution especially for queries on large-scale multimedia databases and furthermore to provide instantaneous query retrievals along with the ongoing query process. In this way it achieves a series of sub-query results that will finally be converging to the full-scale search retrieval in a faster and significantly lower memory with no minimum system requirements. Experimental progressive query retrieval results show that the intermediate sub-query retrieval results might achieve such a retrieval performance that requires no further query processing time.

1. INTRODUCTION

There are several content-based multimedia indexing and retrieval systems such as MUVIS system [1], [7], Photobook, VisualSeek, Virage, and VideoQ [2],[3],[4],[5], some of which are designed to bring a framework structure for handling and especially the retrieval of the multimedia items such as digital images, audio and video clips. The most common retrieval scheme is query-by-example (QBE) and the query is usually performed via an exhaustive search over the entire database due to lack of an efficient indexing scheme. The indexing efficiency is especially reduced if the system supports several aural and visual features along with the user interaction options such as feature enabling and weighting set-up during the retrieval. The basic QBE operation works as follows: using the available aural or visual features (or both) of the queried multimedia item (i.e. an image or video clip) and all the database items, the similarity distances are calculated and then merged to obtain a unique similarity distance per database item. Sorting according to the similarity distances over entire database yields the query result.

There are several drawbacks of such QBE scenario: First of all the user has to wait till all the similarity measures are calculated and all the database items are sorted accordingly. Naturally this might take a significant time if the database size is big and the database contains a rich set of aural and visual features, which might further reduce the efficiency on the indexing process. In addition to such drawbacks any abrupt stopping during the query process will cause total loss of retrieval information and essentially nothing can be saved out of query operation so far performed. In order to speed up the query process it is a common retrieval application design to hold all the features of all the multimedia items of the database into the system memory first and then perform the calculations. Therefore, the increase in the database size or its feature sets will not only (proportionally) increase the query time (the time needed for completing a query) but it will also increase the system memory requirement.

In order to eliminate such drawbacks and provide a faster query scheme, we develop a novel retrieval scheme, the *Progressive Query* (*PQ*), which is implemented under MUVIS system to provide a solid basis and to test the performance of the technique. As its name implies *PQ* provides the 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. An illustration of *PQ* process is shown in Figure 1. It is an expected fact that *PQ* and a normal query will converge to the same (final) retrieval results at the end but even *PQ* will then perform overall query process faster (within a less total query time) than the normal query. Since *PQ* provides a series of sub-query results, each of which belongs to a smaller sub-set within the database, the chance of retrieving the relevant database items that would not be retrieved otherwise via a normal query, can be increased. Approvingly some experimental results show that it is quite probable to achieve even better retrieval performance within an intermediate sub-query than the final query state.

The rest of this paper is organized as follows: in section 2 we introduce the generic *PQ* design philosophy and implementation details. Section 3 presents the experimental results and some example demonstrations. Finally conclusions are given in section 4.

2. PROGRESSIVE QUERY OVERVIEW

Basically *PQ* performs over a series of sub-queries. A sub-query is a fractional query process that is performed over a sub-set of database items. The items within a sub-set can be chosen by any convenient manner such as randomly or sequentially but the size of each sub-set should be chosen via a suitable (to human perception) unit such as time (period). The other alternative is to fix the size of each sub-set to a pre-defined value by the user. However, this brings the problem of uncertainty since the user cannot know how much time a sub-query will take beforehand. The sub-query time will vary due to the number of features present in the database and the speed of the computer where it is running. In order to avoid such uncertainties,

the proposed *PQ* scheme is designed over periodic sub-queries as shown in Figure 1 with a user defined period value ($t = T_p$).

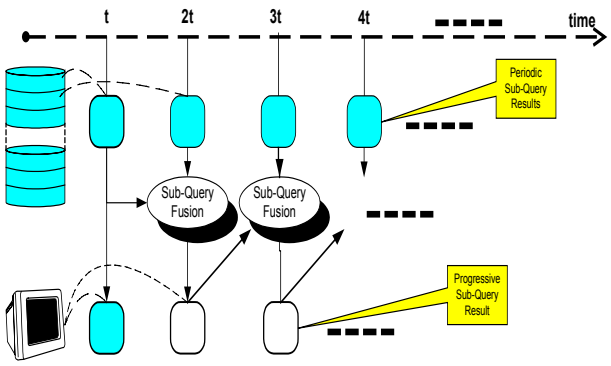

Figure 1: Progressive Query.

2.1. Sub-Query Fusion Operation

The overall *PQ* operation is achieved over progressive sub-queries (*PSQ*). One can state that *PQ* is a (periodic) series of *PSQ* results. Once a *PSQ* is realized it is rendered on the screen and kept intact along with the lifetime of the ongoing *PQ* so that the user can access it at any time. The first *PSQ* is the first periodic sub-query performed. After the first *PSQ*, the rest of the *PSQ*s are obtained by a fusion operation between the current periodic sub-query and the previous *PSQ*. The fusion operation is a process of fusing two of the sorted sub-query results to achieve one (fused) sub-query result. Since both sub-query results are already sorted with respect to the similarity distances of the items within the sub-sets, fusion can be performed simply by comparing the consecutive items in each of the sub-query lists. Let n_1 be the number of items in the first sub-set and n_2 be the number of items in the second one. Then the fusion operation will take $n_1 + n_2$ similarity distance comparisons.

2.2. Periodic Sub-Query Formation

In order to achieve periodic sub-queries we need to define some additional sub-query compositions:

- **Atomic Sub-Query**: The smallest sub-set size on which a sub-query is performed. Here we assume that atomic sub-query time is not significant compared to periodic sub-query time. Atomic sub-queries are the only sub-query types that have a fixed sub-set size (S_{ASQ}). They are only used during first periodic query and they are used in order to provide an initial sub-query rate (t_r), that is the time spent for the retrieval of a single database item, formulated as follows:

$$t_r = \frac{t_{ASQ}}{N_{ASQ}} \quad if \quad N_{ASQ} > 0 \qquad (1)$$

where t_{ASQ} is the total time spent for atomic sub-query and N_{ASQ} is the number of database items, which are involved (used) in query operation. Note that $0 \le N_{ASQ} \le S_{ASQ}$. In case $N_{ASQ} = 0$, one or more atomic sub-queries have to be performed until we get a valid t_r value (i.e. $N_{ASQ} > 0$).

- **Fractional Sub-Query:** This can be any sub-query performed over a sub-set whose size is smaller or equal to the sub-set size of the periodic sub-query. In other words a fractional sub-query time might be less than or equal to a periodic sub-query time.

As explained before periodic sub-queries are periodic over time and a mechanism is needed to ensure this periodicity. This mechanism works over atomic and fractional sub-queries; it performs fusion operation over as many atomic and fractional sub-queries as necessary. First it starts with an atomic sub-query to obtain a valid sub-query rate and it keeps going with atomic queries until a valid t_r value is obtained. Once a valid t_r value is obtained, then one or more fractional sub-queries will be performed to complete the first periodic sub-query. The size of the fractional query (N_{FSQ}) can then be estimated as:

$$N_{FSQ} = \frac{T_p}{t_r} \qquad (2)$$

and the fractional sub-query is performed within a sub-set of N_{FSQ} items. Once the fractional sub-query is completed, the total time ($t_\sum$) so far spent from the beginning of the operation till now is compared with the required sub-query time period, T_p . If $t_\sum$ value is not within a close neighborhood of T_p (i.e. $t_\sum < T_p - T_W$) then the operation continues with a new fractional sub-query until the condition is met. For the new fractional sub-query and for all the latter fractional sub-query operations t_r value is re-estimated (updated) from the former operations such as:

$$t_r = \frac{t_\sum}{\sum_{FSQ} N_{FSQ}} \quad if \quad \sum_{FSQ} N_{FSQ} > 0 \qquad (3)$$

The flowchart of the formation of a periodic sub-query is shown in Figure 2.

2.3. *PQ* versus Normal Query

Along with the ongoing *PQ* operation periodic sub-query results are used to obtain *PSQ* results. The first *PSQ* is equal to the first periodic sub-query. After that when a new periodic sub-query is performed then a new *PSQ* can be formed by fusing the current periodic sub-query with the last *PSQ* as shown in Figure 1. As mentioned before *PQ* is nothing but the time series of *PSQ* retrieval results.

Normal query (*NQ*) is the most basic approach in multimedia retrieval. It is performed over the entire database, loads and uses all the features available, calculates the similarity distance of each database item and ranks them. The final result is therefore the sorted array of database items, which represents the query retrieval with respect to their similarity to the queried multimedia item.

PQ and *NQ* eventually converge to the same retrieval result. Also in the abovementioned scenarios they are both designed to perform exhaustive search over the entire database within MUVIS. However PQ have several advantages over *NQ* in the following aspects:

• **Overall Retrieval Time (Query Speed):** There are three major processes in *NQ*: Loading feature vectors to the system memory, calculating the similarity distances and sorting the database items according to their similarity distances. The first two processes will spend the same time within *PQ* operation but the sorting will be faster due to the following fact: Let n be the number of database items in the database. If, for example, *Quick Sort* is applied, then the number of comparisons will be O($n \log n$) on average and O(n^2) in the worst case. Assume that we only perform *PQ* in two *PSQ* series: Let n_1 be the number of items in the first sub-set and n_2 be the number of items in the second one where $n = n_1 + n_2$. In both average and worst-case scenario:

$$O(n_1^2) + O(n_2^2) < O(n^2) \ (worst case)$$
$$O(n_1 \log n_1) + O(n_2 \log n_2) < O(n \log n) \ (avg.\ case) \quad (4)$$

So *PQ* will apply a faster sorting algorithm especially if the worst-case scenario is considered. It can be shown by deduction that *PQ* time will become significantly faster if the number of *PSQ* operation is getting bigger (i.e. with smaller sub-set size or T_p value).

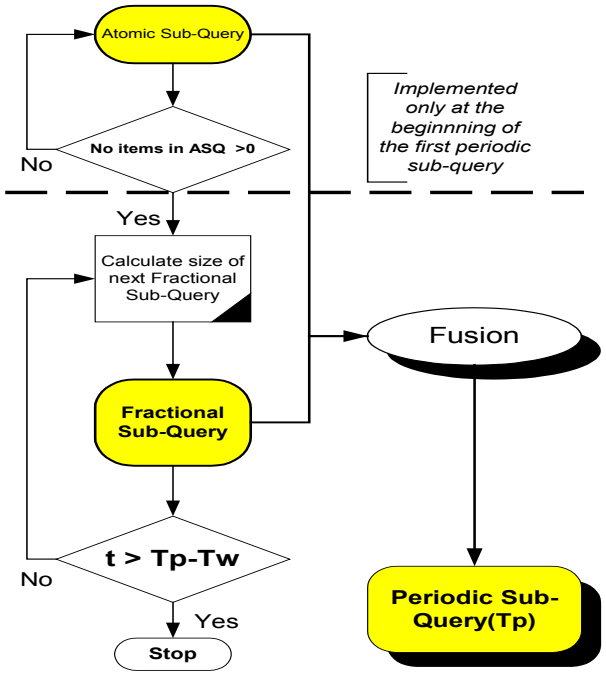

Figure 2: Flowchart of a Periodic Sub-Query.

PSQ No:	1	2	3	4	5	6	7	8	9	10	11	12
***PSQ* time (*msec*)**	4997	5337	4927	5128	4507	4907	6198	4516	5067	4516	4997	3786
***PQ* time (*msec*)**	4997	10334	15261	20389	24896	29803	36001	40517	45584	50100	55097	58883

Table 1: *PSQ* retrieval times for aural video retrieval shown in Figure 5

• **System Memory Requirement**: The memory requirement is proportional to the database size and the number of features present in a *NQ* operation. Due to the partitioning of the database into sub-sets, *PQ* will reduce the memory requirement by the number of *PSQ* operations performed. After each periodic sub-query operation, the memory used for feature vectors in that sub-set is no longer needed and can be used for the next periodic sub-query. Therefore, sub-query will significantly be a faster solution on such systems that due to the lack of memory capacity the virtual memory has to be used for a *NQ* operation and *PQ* might even become the only feasible query operation on such systems in which the system memory is no longer capable of performing a *NQ* on a massive size multimedia database. Figure 3 illustrates the memory usage of one retrieval example that is shown in Figure 5 by a *PQ* and a *NQ*.

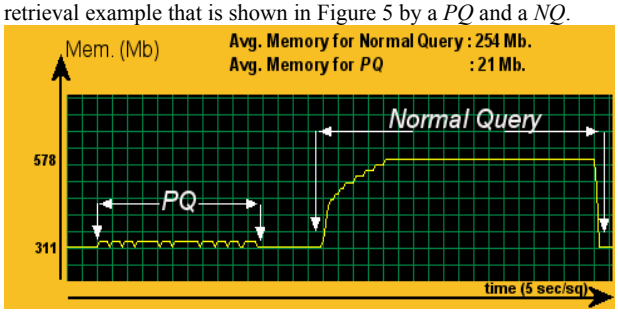

Figure 3: Memory usage for PQ and *NQ*.

• **Query Accessibility**: This is the major advantage that *PQ* provides. Along with the ongoing process *PQ* allows intermediate query results (*PSQ* results), which might sometimes show equal or even better performance than the final (overall) retrieval result as some typical examples given in Figure 4 and Figure 5. In this way user might get the relevant retrieval results in a fracture of the time that is needed in an *NQ* operation.

3. EXPERIMENTAL RESULTS

Several experiments are carried out to test the performance of *PQ* with respect to *NQ*. Visual and aural queries have been performed on both image and video databases. It is experimentally observed that *PQ's* overall operation is 0-15% faster than *NQ* retrievals (depending on the number of *PSQ* series) if *NQ* memory requirement does not exceed the system memory. If it exceeds then *PQ* can outperform *NQ* by over 60%. Note that the relevant query retrievals may eventually occur in an intermediate PSQ state as shown in Figure 5. It is also observed that the *PSQ* retrieval times are within 5% of T_p value in general. *PSQ* arrival times for the *PQ* example shown in Figure 5 are given in Table 1. In this example *PQ* operation total time is 58.883 seconds where *NQ* takes 63.87 seconds. Note that the relevant query results are already obtained within 28.8 seconds at the end of the 6.th *PSQ*.

In Figure 4, image retrieval via *PQ* using Canny Edge Histogram feature is shown. There are 1400 binary images in this database. We use $T_p = 0.2\,\text{sec}$ and *PQ* operation is completed in three *PSQ* series (i.e. *PQ* #1, #2 and #3). This is one particular example that an intermediate *PSQ* retrieval might yield a better performance than the final *PQ* retrieval (that is same as the retrieval result of *NQ*). In this example *PQ* #1 first 12-best retrieval is obviously better than the ones in PQ #3 (the final).

In Figure 5, video retrieval via aural *PQ* using *MFCC* (Mel-Frequency Cepstral Coefficients [9]) as the audio features is shown. There are 800 video clips in the database with a total duration of over 36 hours. We use $T_p = 5\,\text{sec}$ and *PQ* operation is completed in 12 *PSQ* series but only 3 *PSQ* retrievals (i.e. PQ #1, #6 and #12) are shown. Note that *PQ* #6 and the latter retrieval results till *PQ* #12 are identical, which means *PQ* operation produces the final retrieval result in an earlier (intermediate) *PSQ* retrieval.

In Figure 6, another video retrieval via visual *PQ* using several color (*YUV*, *HSV*, etc.), texture (*GLCM [8]*) and shape (Canny Edge Histogram) features is shown. There are 181 video clips in the database with a total duration of over 12 hours. We use $T_p = 3\,\text{sec}$ and *PQ* operation is completed in 4 *PSQ* series. This is a particular example that the relevant query results are retrieved in the last (final) *PSQ*.

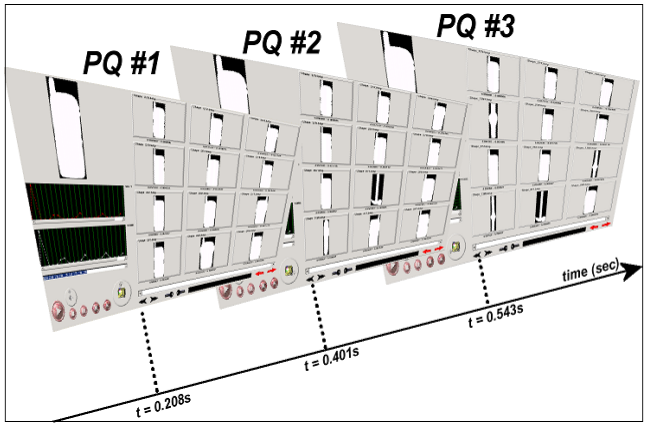

Figure 4: *PQ* image retrieval within 3 *PSQ*s. *Tp*=0.2sec.

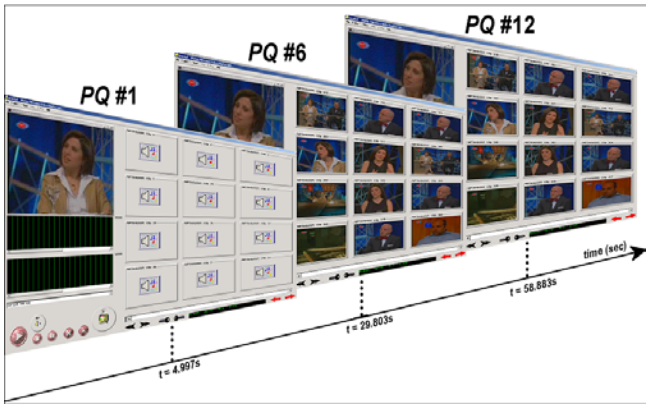

Figure 5: Aural video *PQ* retrieval within 12 *PSQ*s (only 1.st, 6.th and 12.th are shown). *Tp* = 5sec.

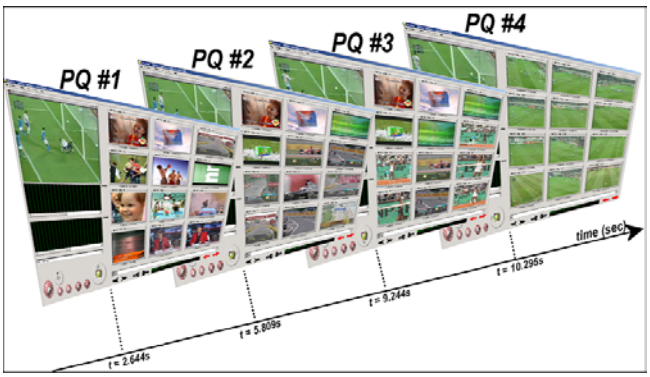

Figure 6: Visual video *PQ* retrieval in 4 *PSQ*s. *Tp*=3sec.

4. CONCLUSIONS

PQ is primarily developed to provide instantaneous and faster retrievals along with the ongoing query process. By this way the user can get an idea about the current status of the query, immediately evaluates the available retrieval results and if satisfactory results are already achieved, the user can even stop the query process without wasting further time. We confirm this with a significant number of experiments.

Another important objective achieved with the proposed *PQ* technique is that it avoids the implementation drawbacks, which *NQ* encounters. This is especially the case if the current system configuration does not match the minimum *NQ* requirements such as memory and speed. Experimental results show that *PQ* is not affected from such drawbacks and currently has no limitations whatsoever the system configuration presents.

5. REFERENCES

[1] S. Kiranyaz, K. Caglar, O. Guldogan, and E. Karaoglu, "MUVIS: A Multimedia Browsing, Indexing and Retrieval Framework", *Proc. Third International Workshop on Content Based Multimedia Indexing, CBMI 2003,* Rennes, France, 22-24 September 2003.

[2] A. Pentland, R.W. Picard, S. Sclaroff, "Photobook: tools for content based manipulation of image databases", Proc SPIE (Storage and Retrieval for Image and Video Databases II) 2185:34-37, 1994.

[3] J.R. Smith and Chang, "VisualSEEk: a fully automated content-based image query system", ACM Multimedia, Boston, Nov. 1996.

[4] Virage. URL:www.virage.com

[5] S.F. Chang, W. Chen, J. Meng, H. Sundaram and D. Zhong, "VideoQ: An Automated Content Based Video Search System Using Visual Cues", Proc. ACM Mult., Seattle, 1997.

[6] ISO/IEC JTC1/SC29/WG11, " Overview of the MPEG-7 Standard Version 5.0", March 2001.

[7] F.Alaya Cheikh, B.Cramariuc, C.Reynaud, M.Quinghong, B.Dragos-Adrian, B.Hnich, M.Gabbouj, P.Kerminen, T.Mäkinen and H.Jaakkola, "MUVIS: a system for content-based indexing and retrieval in large image databases", *Proceedings of the SPIE/EI'99 Conference on Storage and Retrieval for Image and Video Databases VII, Vol.3656*, San Jose, California, 26-29 January 1999.

[8] M. Partio, B. Cramariuc, M. Gabbouj, A. Visa, "Rock Texture Retrieval Using Gray Level Co-occurrence Matrix", *Proc. of 5th Nordic Signal Processing Symposium*, Oct. 2002.

[9] L. R. Rabiner and B. H. Juang, *"Fundamental of Speech Recognition"*, Prentice hall, 1993.