

# Support Concurrent Queries in Multiuser CBIR Systems

Danzhou Liu      Kien A. Hua  
School of Electrical Engineering and Computer Science  
University of Central Florida  
Orlando, Florida 32816, USA  
{dzliu, kienhua}@cs.ucf.edu

## Abstract

*Various techniques have been developed for different query types in content-based image retrieval (CBIR) systems such as sampling queries, constrained sampling queries, multiple constrained sampling queries,  $k$ -NN queries, constrained  $k$ -NN queries, and multiple localized  $k$ -NN queries. In this paper, we propose a generalized query model suitable for expressing queries of different types, and investigate efficient processing techniques for this new framework. We develop new storage and query processing techniques to exploit sequential access and leverage inter-query concurrency to share computation. Our experimental results, based on the Corel dataset, indicates that the proposed optimization can significantly reduce average response time in a multiuser environment.*

## 1. Introduction

Content-based image retrieval (CBIR) has received a great deal of attention with many systems implemented [2]. There are two general types of search in CBIR: *target search* and *category search*. The goal of target search is to find a specific (target) image (e.g., a registered logo). The goal of category search is to retrieve a particular semantic class or genre of images (e.g., skyscrapers). To support target search and category search, many techniques have been developed for different types of queries such as sampling queries, constrained sampling queries, multiple constrained sampling queries,  $k$ -NN queries, constrained  $k$ -NN queries, and multiple localized  $k$ -NN queries [3, 4, 5, 6].

Although there have been many research efforts in supporting concurrent queries in traditional databases, continuous web queries, continuous streaming queries, and spatio-temporal queries, those approaches cannot be directly applied to CBIR systems due to the special characteristics of CBIR queries. In this paper, we therefore propose a generalized approach for processing these different types of queries

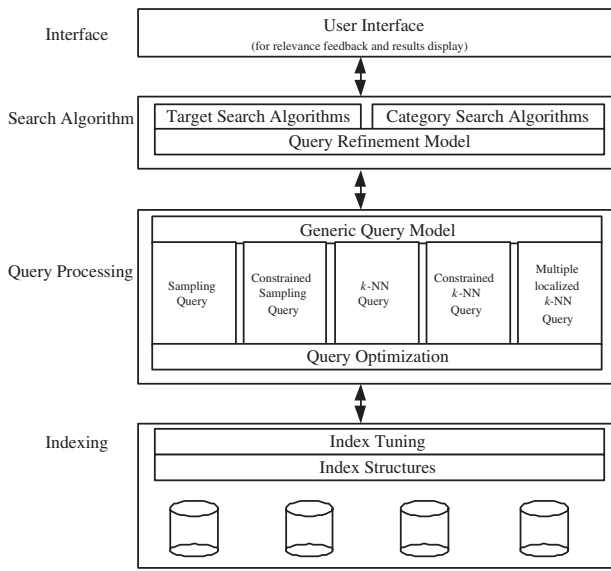
in a unified framework. Our contributions include new storage designs and query processing techniques to leverage sequential data access and I/O sharing among concurrent queries. Thus, the original problems addressed in this papers are twofold:

1. a generalized model for various types of CBIR queries, and
2. techniques for efficient support of concurrent queries in a multiuser environment.

To the best of our knowledge, these practical issues in CBIR applications have not been studied in the literature, and there is no commercial CBIR systems supporting large number of concurrent users. Our experimental results, based on the Corel dataset, confirm that existing query processing techniques aimed at reducing the cost of individual query independently, are not scalable to support a multiuser environment. The performance study indicates that the proposed framework can significantly improve average response time in a multiuser environment. With the growing interest in Internet-scale image search applications, our framework offers an effective solution to the scalability problem.

## 2. The Proposed Framework

In this section, we discuss the proposed framework. As illustrated in Figure 1, this framework has a four-layer structure constituted by the interface layer, search algorithm layer, query processing layer and indexing layer from top to bottom. This framework is designed to be generic enough to accommodate different target search and category search algorithms, query types, query optimization techniques, index structures and index tuning techniques. In the interface layer, a user can mark returned images as positive or negative, or give relevance values for the next round of retrieval. The search algorithm layer, connecting the inter-



**Figure 1. The proposed framework overview**

face layer and query processing layer, chooses the appropriate search algorithm based on the user's goal (target search or category search). The query processing layer, including generic query model, query processing algorithms for different query types, and query optimization, aims to efficiently evaluate the queries involved in the chosen search algorithm. The indexing layer is responsible for deciding appropriate index structures and tuning those structures. Specifically, we will discuss in detail our query model, indexing tuning and query optimization techniques.

### 2.1. Generic Query Model

In our query model, a user query is defined as

$$Q = \{Q_i \mid i \in \{1, \dots, n\}\},$$

where  $Q_i$  is a subquery and  $n$  is the user specified number of subqueries. Then a subquery  $Q_i$  is defined as

$$Q_i = \langle n_{Q_i}, P_{Q_i}, W_{Q_i}, D_{Q_i}, \mathbb{S}_{Q_i}, k_{Q_i} \rangle,$$

where  $n_{Q_i}$  denotes the number of query points in  $Q_i$ ,  $P_{Q_i}$  the set of  $n_{Q_i}$  query points in  $Q_i$ ,  $\mathbb{S}_{Q_i}$  the subspace to retrieve data points,  $W_{Q_i}$  the set of weights associated with  $P_{Q_i}$ ,  $D_{Q_i}$  the distance function, and  $k_{Q_i}$  the number of data points to be retrieved in this subquery. Now we illustrate how to use this model to represent the most typical queries (i.e., sampling queries, constrained sampling queries,  $k$ -NN queries, multiple constrained sampling queries,  $k$ -NN queries, constrained  $k$ -NN queries, and multiple localized  $k$ -NN queries). If a query is a sampling query, we set  $n = 1$ ,  $n_{Q_i} = 0$ ,  $\mathbb{S}_{Q_i} = \mathbb{S}$  (i.e., the whole search space) and

$k_{Q_i} = k$ . If a query is a constrained sampling query, we set  $n = 1$ ,  $n_{Q_i} = 0$  and  $k_{Q_i} = k$ , which signify that this query is to randomly retrieve  $k$  points in  $\mathbb{S}_{Q_i}$ . If a query is a multiple constrained sampling query, we set  $n \geq 1$ ,  $n_{Q_i} = 0$  and  $k_{Q_i} = k/n$ . If a query is a  $k$ -NN query with single-point movement techniques, we set  $n = 1$ ,  $n_{Q_i} = 1$ ,  $\mathbb{S}_{Q_i} = \mathbb{S}$  and  $k_{Q_i} = k$ ; if a query is a  $k$ -NN query with multiple-point movement techniques, we set  $n = 1$ ,  $n_{Q_i} \geq 1$ ,  $\mathbb{S}_{Q_i} = \mathbb{S}$  and  $k_{Q_i} = k$ . If a query is a constrained  $k$ -NN query, we set  $n = 1$ ,  $n_{Q_i} = 1$  and  $k_{Q_i} = k$ . If a query is a multiple localized  $k$ -NN query, we set  $n \geq 1$ ,  $n_{Q_i} = 1$  and  $k_{Q_i} = k/n$ .

In our generic model, we consider various query refinement models, multiple subqueries and various types of queries.  $\mathbb{S}_{Q_i}$  is also included to account for the dynamic change of the search space, which may be reduced after each feedback iteration [3, 5]. Therefore, our query model is so generic that it can be used to express all typical queries that we have known in CBIR systems.

### 2.2. Index Tuning and Query Optimization Techniques

To achieve efficient query evaluation and support concurrent queries in our framework, we propose three techniques; namely, indexing tuning, group access and individual query optimization. We discuss these schemes as follows:

**Index Tuning:** Tuning the index structures is particularly reasonable for CBIR systems where the image sets are fairly static. Since the disk pages allocated to sibling nodes are often not physically consecutive (typically a disk page contains only one node), a query may incur a large number of random accesses even for each feedback iteration. To reduce the number of disk random accesses, we use the Hilbert curve [1] for disk page allocation. The advantage of this data placement scheme is that nodes that are close together in the multidimensional space are usually close to each other in physical storage, allowing us to retrieve neighboring nodes using sequential access. Specifically, we create a tuned index structure as follows: we traverse the non-tuned index structure in a breadth-first fashion, and then create a tuned index structure with the disk page allocation almost following the traversal order except for the children nodes in the same node. For the children nodes in the same node, we allocate them to the disk in the order of the Hilbert curve values of their centers.

**Group Access:** In a multiuser environment, concurrent queries might have their relevant nodes overlapping each other. Even answering a single query may involve multiple sub-queries. For these cases, we can save disk activities by performing group access. That is, instead of retrieving disk pages for each of the queries independently, we allow them to share disk accesses and reduce the number of expensive

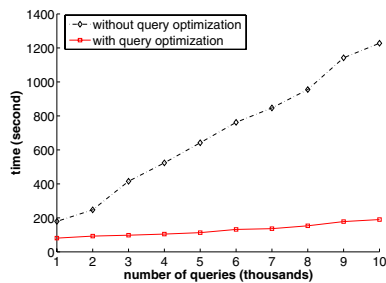


Figure 2.  $k$ -NN queries

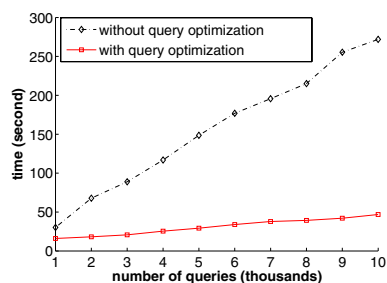


Figure 3. Constrained  $k$ -NN queries

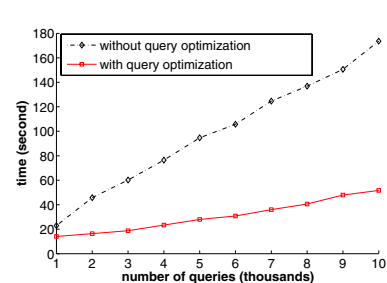


Figure 4. Multiple localized  $k$ -NN queries

disk seeks at the cost a few extra cheap page transfer.

**Individual Query Optimization:** Again, disk pages allocated to sibling nodes in the tuned index structure are physically consecutive. Therefore, the number of expensive disk seeks are expected to be reduced significantly for answering queries when our group access algorithm is employed. In addition, for the subsequent  $k$ -NN queries in the same search, we exploit the information generated during the previous  $k$ -NN queries to further reduce the disk I/O cost and CPU cost. Answering constrained  $k$ -NN queries is similar to answering  $k$ -NN queries except that we need to prune the nodes which are outside of  $S_{Q_1}$ . For multiple localized  $k$ -NN queries, each localized  $k$ -NN is performed on the related leaf node and its sibling leaf nodes. Our group access algorithm can be employed as well to take advantage of disk access locality and sharing.

### 3. Experiments

In this section, We evaluate the effectiveness of the proposed query optimization techniques described in Section 2.2. We compare the performance of the proposed techniques with all the optimization features (denoted as  $QOP$ ) with that of similar techniques with none of the optimization features (denoted as  $QNO$ ). Our dataset consists of more than 68,040 images from the Corel library, and 37 visual image features were used. All experiments were performed on a 2.5-GHz Pentium IV-based computer with 1GB of RAM. The node size of the index structures was set to 4KB, and the first two levels of the index structures resided in memory. In this study,  $k$  was set to 50, all queries were randomly generated, and relevance feedback was simulated. The results are averaged over 100 runs.

Figures 2 to 4 show the effect of increasing the number of concurrent queries from 1000 to 10000 for three types of queries (i.e.,  $k$ -NN queries, constrained  $k$ -NN queries, and multiple localized  $k$ -NN queries), respectively. The experimental results for constrained sampling queries and mul-

tiply constrained sampling queries share the similar effect, and are omitted. Clearly, the total processing time increases as the number of concurrent queries increases. All three figures indicate that  $QOP$  significantly outperforms  $QNO$ , and the performance gap widens with the increases in the number of queries. Specifically, for  $k$ -NN queries (see Figure 2),  $QOP$  performs up to 6 times faster compared to  $QNO$ . The relatively larger savings in this case are due to the fact that  $k$ -NN queries can benefit much from information generated during the previous iterations. For constrained  $k$ -NN queries (see Figure 3),  $QOP$  performs up to 5 times faster; for multiple localized  $k$ -NN queries (see Figure 4), it performs up to 3 times faster.

The experimental results shows that  $QNO$  is not scalable since its query processing time increases rapidly as the number of concurrent queries increases, while  $QOP$  exhibits a very slow increasing rate. The performance difference between  $QNO$  and  $QOP$  confirms that leveraging computation sharing in inter-query concurrency can substantially improve performance and enhance system scalability to support a large user community.

### References

- [1] C. Böhm, S. Berchtold, and D. A. Keim. Searching in High-dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001.
- [2] T. Gevers and A. Smeulders. Content-based Image Retrieval: An Overview. In G. Medioni and S. B. Kang, editors, *Emerging Topics in Computer Vision*. Prentice Hall, 2004.
- [3] K. A. Hua, N. Yu, and D. Liu. Query Decomposition: A Multiple Neighborhood Approach to Relevance Feedback Processing in Content-based Image Retrieval. In *ICDE*, 2006.
- [4] D.-H. Kim and C.-W. Chung. QCluster: Relevance Feedback Using Adaptive Clustering for Content-based Image Retrieval. In *SIGMOD*, pages 599–610, 2003.
- [5] D. Liu, K. A. Hua, K. Vu, and N. Yu. Fast Query Point Movement Techniques with Relevance Feedback for Content-based Image Retrieval. In *EDBT*, pages 700–717, 2006.
- [6] O.-B. Michael and S. Mehrotra. Relevance Feedback Techniques in the MARS Image Retrieval Systems. *Multimedia Systems*, (9):535–547, 2004.