Efficient Techniques For Relevance Feedback Processing In Content-based Image Retrieval

2009

Danzhou Liu

University of Central Florida

Find similar works at: https://stars.library.ucf.edu/etd

University of Central Florida Libraries http://library.ucf.edu

Part of the Computer Sciences Commons, and the Engineering Commons

STARS Citation

Liu, Danzhou, "Efficient Techniques For Relevance Feedback Processing In Content-based Image Retrieval" (2009). Electronic Theses and Dissertations. 4001.

https://stars.library.ucf.edu/etd/4001

STARS Citation

Liu, Danzhou, "Efficient Techniques For Relevance Feedback Processing In Content-based Image Retrieval" (2009). Electronic Theses and Dissertations. 4001.

https://stars.library.ucf.edu/etd/4001

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact lee.dotson@ucf.edu.
Efficient Techniques for Relevance Feedback Processing in Content-based Image Retrieval

by

DANZHOU LIU
B.E. Beijing University of Aeronautics and Astronautics, 1994
M.E. Nanyang Technological University, 2002
M.S. University of Central Florida, 2005

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the School of Electrical Engineering and Computer Science in the College of Engineering and Computer Science at the University of Central Florida
Orlando, Florida

Summer Term
2009

Major Professor: Kien A. Hua
ABSTRACT

In content-based image retrieval (CBIR) systems, there are two general types of search: target search and category search. Unlike queries in traditional database systems, users in most cases cannot specify an ideal query to retrieve the desired results for either target search or category search in multimedia database systems, and have to rely on iterative feedback to refine their query. Efficient evaluation of such iterative queries can be a challenge, especially when the multimedia database contains a large number of entries, and the search needs many iterations, and when the underlying distance measure is computationally expensive. The overall processing costs, including CPU and disk I/O, are further emphasized if there are numerous concurrent accesses. To address these limitations involved in relevance feedback processing, we propose a generic framework, including a query model, index structures, and query optimization techniques. Specifically, this thesis has five main contributions as follows.

The first contribution is an efficient target search technique. We propose four target search methods: naïve random scan (NRS), local neighboring movement (LNM), neighboring divide-and-conquer (NDC), and global divide-and-conquer (GDC) methods. All these methods are built around a common strategy: they do not retrieve checked images (i.e., shrink the search space). Furthermore, NDC and GDC exploit Voronoi diagrams to aggressively prune the search space and move towards target images. We theoretically and experimentally prove that the convergence speeds of GDC and NDC are much faster than those of NRS and recent methods.

The second contribution is a method to reduce the number of expensive distance computation when answering k-NN queries with non-metric distance measures. We propose an efficient distance mapping function that transfers non-metric measures into metric, and still preserves the original distance orderings. Then existing metric index structures (e.g., M-tree) can be used to reduce the computational cost by exploiting the triangular inequality property.
The third contribution is an incremental query processing technique for Support Vector Machines (SVMs). SVMs have been widely used in multimedia retrieval to learn a concept in order to find the best matches. SVMs, however, suffer from the scalability problem associated with larger database sizes. To address this limitation, we propose an efficient query evaluation technique by employing incremental update. The proposed technique also takes advantage of a tuned index structure to efficiently prune irrelevant data. As a result, only a small portion of the data set needs to be accessed for query processing. This index structure also provides an inexpensive means to process the set of candidates to evaluate the final query result. This technique can work with different kernel functions and kernel parameters.

The fourth contribution is a method to avoid local optimum traps. Existing CBIR systems, designed around query refinement based on relevance feedback, suffer from local optimum traps that may severely impair the overall retrieval performance. We therefore propose a simulated annealing-based approach to address this important issue. When a stuck-at-a-local-optimum occurs, we employ a neighborhood search technique (i.e., simulated annealing) to continue the search for additional matching images, thus escaping from the local optimum. We also propose an index structure to speed up such neighborhood search.

Finally, the fifth contribution is a generic framework to support concurrent accesses. 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, indicate that the proposed optimization can significantly reduce average response time while achieving better precision and recall, and is scalable to support a large user community. This latter performance characteristic is largely neglected in existing systems making them less suitable for large-scale deployment. With the growing interest in Internet-scale image search applications, our framework offers an effective solution to the scalability problem.
ACKNOWLEDGMENTS

I would like to thank the University of Central Florida (UCF) for offering assistantships that support my PhD study.

I would like to express my deep gratitude to my advisor, Dr. Kien A. Hua. His inspiring and constructive supervision has been a constant source of encouragement for my study. I acknowledge his patience, sincerity and all his help extended to me throughout the six years.

Next, I would like to thank Dr. Charles E. Hughes for his insight and vision on this fascinating research field. He has always kept his door open and provided me valuable advice for my research. Thanks also go to other committee members: Dr. J. Michael Moshell, Dr. Annie S. Wu, and Dr. Ning Jackie Zhang for their wonderful inputs to refine this work.

Thanks should also go to all DSGers. They have shared their knowledge, provided useful comments on my research and presentations, and established a supportive environment. The period of time we were with together is filled with happiness and is unforgettable. It will be one of my best memories in the future.

I thank my parents, elder sister and younger sister. Also, I thank my parents-in-law. Special thanks go to my wife, Jianqi Miao. Their unconditional love and constant support keep me going even in the most miserable days. I thank my son, Gavin Liu. Although he messed up my life a little bit at the beginning, his smile is the best reward for all my efforts.

Last but not least, I would also like to extend my thanks to those who have cared and helped, in one way or another, in making this dissertation possible.
# TABLE OF CONTENTS

LIST OF FIGURES xi

LIST OF TABLES xii

LIST OF ACRONYMS xiii

CHAPTER 1: INTRODUCTION 1

1.1 Motivations ......................................................... 1
1.2 Related Work ...................................................... 2
1.3 Summary ........................................................... 6

CHAPTER 2: TARGET SEARCH 7

2.1 Target Search Methods ................................................ 9
  2.1.1 Naïve Random Scan Method .................................. 11
  2.1.2 Local Neighboring Movement Method ...................... 12
  2.1.3 Neighboring Divide-and-Conquer Method ................. 13
  2.1.4 Global Divide-and-Conquer Method ....................... 16
2.2 Handle Inaccurate Relevance Feedback .......................... 18
2.3 Query Processing Technique for Target Search ................. 21
  2.3.1 Index Structure for Target Search ......................... 21
  2.3.2 Efficient Query Processing for Target Search .......... 24
2.4 Experiments ......................................................... 27
  2.4.1 Simulated Experiments ...................................... 27
  2.4.2 Realistic Experiments ....................................... 34
  2.4.3 Query Processing Technique .............................. 39
2.5 Summary ........................................................... 41
LIST OF FIGURES

1.1 Single-point movement query shapes .................................. 2
1.2 Multiple-point movement query shapes ................................. 2

2.1 Local maximum trap in existing approaches ........................... 8
2.2 Slow convergence in existing approaches ............................. 8
2.3 Overview of the target search systems ................................. 10
2.4 Naïve Random Scan Method .............................................. 11
2.5 Neighboring Divide-and-Conquer Method .............................. 14
2.6 Example of NDC ............................................................ 16
2.7 Example of GDC ............................................................ 17
2.8 One case and the weighted centroid .................................... 18
2.9 Inaccurate relevance feedback and linear regression ............... 18
2.10 Our new index structure .................................................. 21
2.11 Query Processing Technique ............................................. 25
2.12 False Hit Ratio ............................................................. 28
2.13 Average Iterations ........................................................ 28
2.14 Maximum Iterations ....................................................... 29
2.15 Minimum Iterations ........................................................ 29
2.16 Standard Deviation of Iterations ....................................... 30
2.17 Average Aggregate Recall ............................................... 30
2.18 Average Aggregate Precision .......................................... 31
2.19 Average Total Checked Images ........................................ 31
2.20 CPU Time ................................................................. 32
2.21 GDC Average Iterations Comparison .................................. 34
2.22 Target Search GUI Interface ............................................. 35
2.23 Number of Iterations Comparison (with our index structure vs. without our index structure) with $k=50$ .................................................. 36
2.24 Sampling Queries ................................................................. 37
2.25 Constrained Sampling Queries .................................................. 37
2.26 Constrained Sampling Queries in Feedback Iterations ($k = 50$) ........... 38
2.27 Constrained $k$-NN Queries in Feedback Iterations ($k = 50$) ............... 39

3.1 Chamfer distance ................................................................. 43
3.2 Optimize mapping function ...................................................... 46
3.3 Algorithm for mapping function optimization .................................... 47
3.4 Results of Corel image data set .................................................. 49
3.5 Results of synthetic polygon data set ............................................ 50

4.1 SVMs ................................................................................. 53
4.2 KDX .................................................................................. 55
4.3 Bounding box in the original space ............................................... 57
4.4 SVM Query processing technique ............................................... 60
4.5 Algorithm for multiple range queries ......................................... 62
4.6 Recall for top-$k$ uncertain queries ............................................. 64
4.7 Recall for top-$k$ relevant queries ............................................... 65
4.8 Disk accesses of $k$ sampling queries .......................................... 65
4.9 Disk accesses of top-$k$ uncertain queries .................................... 66
4.10 Disk accesses of top-$k$ relevant queries ..................................... 66
4.11 Execution time for range queries ................................................. 67

5.1 Local optimum trap .................................................................. 71
5.2 Our new index structure ........................................................... 73
5.3 Algorithm for handling local optimum traps .................................... 75
5.4 Local optimum trap ratio .................................................. 77
5.5 Precision \((k = 100)\) ...................................................... 78
5.6 Recall \((k = 100)\) .......................................................... 78
5.7 Comparison of disk accesses \((k = 50)\) ................................. 79

6.1 The proposed framework overview ..................................... 83
6.2 Global Divide and Conquer Algorithm for Target Search ........ 86
6.3 Query Decomposition Algorithm for Category Search .......... 87
6.4 Our index structure ......................................................... 89
6.5 Using the Hilbert curve for disk allocation ......................... 89
6.6 Index tuning and group access .......................................... 90
6.7 Group Access Algorithm .................................................. 91
6.8 Average Iterations ......................................................... 96
6.9 Constrained sampling queries .......................................... 97
6.10 Multiple constrained sampling queries ............................... 97
6.11 \(k\)-NN queries in Qcluster ............................................ 98
6.12 Constrained \(k\)-NN queries .............................................. 98
6.13 Multiple localized \(k\)-NN queries .................................... 99
LIST OF TABLES

6.1 Precision Comparison (without our index structure vs. with our index structure) 95
LIST OF ACRONYMS

$Q$  a query
$k$  the number of data points to be retrieved with $Q$
$n_Q$  the number of query points in $Q$
$P_Q$  a set of $n_Q$ query points in $Q$
$W_Q$  a set of weights associated with $P_Q$
$D_Q$  the distance function for $Q$
$Q_s$  the starting query
$Q_r$  a refined query at a feedback iteration
$S_k$  the query result set
$S$  the whole space (i.e., the whole image database)
$|S|$  the cardinality of $S$
$S'$  the current search space, where $S' \subseteq S$
$p_s$  the starting query point
$p_t$  the target point (i.e., the target image)
$M$  the node capacity (i.e., fanout)
$m$  the minimum number of entries in a node
CBIR  content-based image retrieval
GDC  the global divide-and-conquer method
LNM  the local neighboring movement method
MBB  the minimum bounding box
NDC  the neighboring divide-and-conquer method
NRS  the naive random scan method
RF  relevance feedback
SVMs  support vector machines
CHAPTER 1: INTRODUCTION

1.1 Motivations

Content-based image retrieval (CBIR) has received much attention in the last decade, which is motivated by the need to efficiently handle the immensely growing amount of multimedia data. Many CBIR systems have recently been developed, including QBIC [24], Photo-book [70], MARS [66, 71, 74], NeTra [60], PicHunter [19], Blobworld [13], VisualSEEK [84], SIMPLIcity [93] and others [79, 5, 10, 29, 64, 92]. In a typical CBIR system, low-level visual image features (e.g., color, texture and shape) are automatically extracted for image descriptions and indexing purposes. To search for desirable images, a user presents an image as an example of similarity, and the system then returns a set of similar images based on the extracted features. In CBIR systems with relevance feedback (RF), the user can mark returned images as positive or negative, which are fed back into the systems as a new, refined query for the next round of retrieval. The process is repeated until the user is satisfied with the query result. Relevance feedback helps bridge the semantic gap between the descriptive limitations of low-level features and human perception of similarity [82]. Such systems achieve high effectiveness for many practical applications [27].

There are two general types of search: target search and category search [19, 27]. The goal of target search is to find a specific (target) image (e.g., a registered logo, a historical photograph or a painting), which can be determined based on low-level features. The goal of category search is to retrieve a particular semantic class or genre of images (e.g. scenery images or skyscrapers). Target search corresponds to known-item search in information retrieval, whereas category search corresponds to high-precision search. Due to semantic gaps, images in a semantic category might scatter in several clusters in low-level feature space.
Figure 1.1: Single-point movement query shapes

Figure 1.2: Multiple-point movement query shapes

Unlike queries in traditional database systems, users in most cases cannot specify an ideal query to retrieve the desired results for either target search or category search in multimedia database systems, and have to rely on iterative feedback to refine their query. Efficient evaluation of such iterative queries can be a challenge, especially when the multimedia database contains a large number of entries, and the search needs many iterations, and when the underlying distance measure is computationally expensive. The overall processing costs, including CPU and disk I/O, are further emphasized if there are numerous concurrent accesses. Thus, efficient techniques for relevance feedback processing are highly demanded.

1.2 Related Work

In this section, we survey existing techniques for relevance feedback processing.

Two well-known techniques for target search were proposed in QBIC [24] and PicHunter [19]. IBM’s QBIC system allows users to compose queries based on visual image features
such as color percentage, color layout, and texture present in the target image, and ranks retrieved images according to those criteria. QBIC, however, is not an RF technique, so that it is difficult for users to define the ideal queries in the first try (because this system does not allow them to refine their queries as in recent RF systems). To lessen the burden on users, PicHunter proposes to predict query’s intents by using a Bayesian-based RF technique to guide query refinement and target search. PicHunter’s performance, however, depends on the consistency of users’ behavior and the accuracy of the prediction algorithm. More importantly, both QBIC and PicHunter do not guarantee to find target images and suffer local maximum traps.

Techniques for category search can be divided into two groups: single-point and multiple-point movement techniques. A technique is classified as a single-point movement technique if the refined query $Q_r$ at each iteration consists of only one query point. Otherwise, it is a multiple-point movement technique. Typical query shapes of single-point movement and multiple-point movement techniques are shown in Figures 1.1 and 1.2, where the contours represent equi-similarity surfaces. Single-point movement techniques, such as MARS [66,74] and MindReader [40], construct a single query point close to relevant images and away from irrelevant ones. MARS uses a weighted distance (producing shapes shown in Figure 1.1.2), where each dimension weight is inversely proportional to the standard deviation of the relevant images’ feature values in that dimension. The rationale is that a small variation among the values is more likely to express restrictions on the feature, and thereby should carry a higher weight. On the other hand, a large variation indicates this dimension is not significant in the query, thus should assume a low weight. MindReader achieves better results by using a generalized weighted distance, see Figure 1.1.3 for its shape. Ostensive relevance feedback [9] can be used to adjust the weights based on the checked images, while the length of time since an image was checked is used in a decay function to modulate the impact of those already checked images.
In multiple-point movement techniques such as Query Expansion [14], Qcluster [46], and Query Decomposition [38], multiple query points are used to define the ideal space that is most likely to contain relevant results. Query Expansion groups query points into clusters and chooses their centroids as $Q_r$’s representatives (see Figure 1.2.1). The distance of a point to $Q_r$ is defined as a weighted sum of individual distances to those representatives. The weights are proportional to the number of relevant objects in the clusters. Thus, Query Expansion treats local clusters differently, as opposed to the equal treatment in single-point movement techniques.

In some queries, clusters are too far apart for a unified, all-encompassing contour to be effective; separate contours can yield more selective retrieval. This observation motivated Qcluster to employ an adaptive classification and cluster-merging method to determine optimal contour shapes for complex queries. Qcluster supports disjunctive queries, where similarity to any of the query points is considered as good, see Figure 1.2.2. To handle disjunctive queries both in vector space and in arbitrary metric space, a technique was proposed in FALCON [96]. It uses an aggregate distance function to estimate the (dis)similarity of an object to a set of desirable images. To bridge the semantic gap more effectively, we recently proposed Query Decomposition [38]. Based on the user’s relevance feedback, this scheme automatically decomposes a given query into localized subqueries, which more accurately capture images with similar semantics but in very different appearance (e.g., the front view and side view of a car), see Figure 1.2.3. Other techniques [87, 93, 27] are also available to address the semantic gap. In general, the above category search techniques do not guarantee to find target images and still suffer slow convergence, local maximum traps and high computation overhead.

Unlike queries in traditional database systems, users in most cases cannot specify an ideal query to retrieve the desired result in multimedia database systems, and have to rely on iterative feedback to refine their queries. Target search and category search may involve four typical types of queries: sampling queries [14, 19, 24, 38, 40, 46, 56, 66], constrained sampling
queries [56], $k$-NN queries [66, 40] and constrained $k$-NN queries [56]. Among all aforementioned techniques, only Chakrabarti et al. discussed how to efficiently evaluate $k$-NN queries in the Query Expansion model [14]. They observed that the refined queries in the Query Expansion model are not modified dramatically from one iteration to another. Instead evaluating each refined query from scratch, they proposed several techniques to save most of the I/O cost and CPU cost by appropriately exploiting the information generated during the previous iterations. However, Chakrabarti et al. did not systematically address how to efficiently evaluate other queries and how to support concurrent accesses. Although there have been many research efforts in supporting concurrent queries in traditional databases [78], continuous web queries [16], and spatio-temporal queries [12], those approaches cannot be directly applied to CBIR systems due to the special characteristics of CBIR queries as discussed above.

Some research efforts have been devoted to sampling for selectivity estimation [97, 98], real-time CPU scheduling for mobile multimedia systems [100], and efficiently evaluating $k$-NN queries [35, 73, 91, 42] and constrained $k$-NN queries [23] without relevance feedback, but much less has been reported on efficiently answering sampling queries, constrained sampling queries and constrained $k$-NN queries involved in target search and category search, where we need to consider iterative feedback and users’ inaccurate relevance feedback. Most existing hierarchical index structures (e.g., R-tree [30], R*-tree [6], and A-tree [76]) were not designed specifically for relevance feedback processing, which typically cannot be answered in one iteration and may require auxiliary information (e.g., sampling points) to answer sampling queries and constrained sampling queries. Collecting auxiliary information on the fly during each feedback iteration causes overheads on CPU and disk I/O. Therefore, efficient techniques for relevance feedback are highly demanded.
1.3 Summary

This dissertation contains the following five main topics:

- How to efficiently support target search,
- How to reduce the number of expensive distance computation when answering $k$-NN queries with non-metric distance measures,
- How to efficiently evaluate queries involved in SVMs,
- How to avoid local optimum traps,
- How to efficiently support concurrent queries.

The rest of this dissertation is organized as follows. In Chapter 2, we present a highly efficient target search technique. Chapter 3 provides an optimization technique for reduction in the number of expensive distance computation in multimedia retrieval. We discuss an incremental query processing technique for Support Vector Machines in Chapter 4. Chapter 5 is dedicated to a method to avoid local optimum traps. We present a framework to support concurrent accesses in Chapter 6. Finally, we conclude the dissertation in Chapter 7.
CHAPTER 2: TARGET SEARCH

In this chapter, we outline the four proposed methods for target search. Again, the goal of category search is to retrieve a particular semantic class or genre of images (e.g. scenery images or skyscrapers). Existing target search techniques re-retrieve previously examined images (i.e., those retrieved in the previous iterations) when they again fall within the search range of the current iteration. This strategy leads to the following disadvantages:

- No guarantee that the target can be found. The search operation generally takes several iterations of relevance feedback to examine a number of regions in the feature space, before it reaches the target image. During this iterative process, the search advancement might get trapped in a region as illustrated in Figure 2.1. It shows $s$ and $t$ as the starting point $p_s$ and the target point $p_t$, respectively. Initially, the 3-NN search with $p_s$ as the query point yields three points $p_s$, $p_1$, and $p_2$ as the query result. Let us say, the user marks points $p_1$ and $p_2$ as relevant. This results in point $p_r$, their centroid, as the new query point. With $p_r$ as the refined query, the next 3-NN computation again retrieves points $p_1$, $p_2$, and $p_s$ as the result. In this scenario, the search process is trapped in this local region, and can never reach the target point $p_t$. Although, the system can escape the local maximum trap with a larger $k$, it is difficult to guess a proper threshold ($k = 14$ in this example). Consequently, the user might not even know a local maximum trap is occurring.

- Slow convergence. Including previously examined images in the computation of the current centroid results in repeat retrieval of some of the images. This prevents a more aggressive movement of the search in the feature space. This drawback is illustrated in Figure 2.2, where $k = 3$. It shows that it takes six iterations for the search operation
Figure 2.1: Local maximum trap in existing approaches

Figure 2.2: Slow convergence in existing approaches

to reach the target point \( p_t \). This slow convergence incurs longer search time, and significant computation and disk access overhead.

To address the above limitations, we propose four target search methods: naïve random scan (NRS), local neighboring movement (LNM), neighboring divide-and-conquer (NDC), and global divide-and-conquer (GDC) methods. All these methods are built around a common strategy: they do not retrieve checked images (i.e., shrink the search space). Furthermore, NDC and GDC exploit Voronoi diagrams to aggressively prune the search space and move towards target images. We theoretically prove that the convergence speeds of GDC and NDC are much faster than those of NRS and recent methods. Results of extensive
experiments confirm our complexity analysis and show the superiority of our techniques in both the simulated and realistic environments.

The remainder of this chapter is organized as follows. The proposed methods for target search are presented in Section 2.1 in detail. Handling inaccurate user relevance feedback is discussed in Section 2.2. We introduce a new index structure and query processing technique for target search in Section 2.3. Our experimental results are given in Section 2.4. Finally, Section 2.5 offers some concluding remarks.

2.1 Target Search Methods

In this section, we present the four proposed target search methods. Again, the goals of our target search methods are avoiding local maximum traps, achieving fast convergence, reducing resource requirements, and guaranteeing to find target images.

Reconsidering already checked images is one of the several shortcomings of existing techniques that leads to the local maximum trap problem and slow convergence; the idea of leaving out checked images is our chief motivation for a new design principle. To simplify discussion, we assume that users are able to accurately identify the most relevant image from the returned images, and this most relevant image is the closest to the target image among the returned ones.

In target search, the ultimate goal is to locate the target images, and if none is found, the final precision and recall of the search is zero. In CBIR with RF, the traditional recall and precision can be computed for individual iterations. For target search, we will use the so-called ‘aggregate’ recall and precision: if after several, say $i$, iterations the target image is found, the average precision and recall are $1/(i \cdot k)$ and $1/i$, where $k$ is the fixed number of images retrieved at each iteration. In short, the number of iterations to find a target image is not only the most significant measure of efficiency, but also the most significant indicator.
of precision and recall. Therefore, we use the number of iterations as the major measure for theoretical analysis and experimental evaluation of the four proposed target search methods.

A query for target search is defined as $Q = \langle n_Q, P_Q, W_Q, D_Q, S', k \rangle$, where $n_Q$ denotes the number of query points in $Q$, $P_Q$ the set of $n_Q$ query points in the current search space $S'$, $W_Q$ the set of weights associated with $P_Q$, $D_Q$ the distance function, and $k$ the number of points to be retrieved in each iteration (see Figure 2.3). As discussed in Section 1.2, various techniques have been proposed to automatically determine $n_Q$ and $P_Q$ as well as adjusting $W_Q$ and $D_Q$ for improved retrieval effectiveness. For single-point movement techniques, $n_Q = 1$; for multiple-point movement techniques, $n_Q > 1$. Now we illustrate below how to use this model to represent the four typical types of queries:

- For a sampling query, we set $n_Q = 0$ and $S' = S$, which signify that this query is to randomly retrieve $k$ points in the whole image database $S$.

- For a constrained sampling query, we set $n_Q = 0$.

- For a $k$-NN query with single-point movement techniques, we set $n_Q = 1$ and $S' = S$;
  For a $k$-NN query with multiple-point movement techniques, $n_Q > 1$ and $S' = S$.

- For a constrained $k$-NN query with single-point movement techniques, we set $n_Q = 1$ while for a constrained $k$-NN query with multiple-point movement techniques, $n_Q > 1$.  

Figure 2.3: Overview of the target search systems
Naive Random Scan Method

\textbf{Input:} 
set of images $\mathbb{S}$
number of retrieved images at each iteration $k$

\textbf{Output:} 
target image $p_t$

01 $Q_s \leftarrow \langle 0, P_Q, W_Q, D_Q, \mathbb{S}, k \rangle$
02 $S_k \leftarrow \text{EVALUATEQUERY}(Q_s) /*$ randomly retrieve $k$ points in $\mathbb{S} */$
03 $\mathbb{S}' \leftarrow \mathbb{S} - S_k$
04 \textbf{while} user does not find $p_t$ in $S_k$ \textbf{do}
05 $Q_r \leftarrow \langle 0, P_Q, W_Q, D_Q, \mathbb{S}', k \rangle$
06 $S_k \leftarrow \text{EVALUATEQUERY}(Q_r) /*$ randomly retrieve $k$ points in $\mathbb{S}' */$
07 $\mathbb{S}' \leftarrow \mathbb{S}' - S_k$
08 \textbf{endo}
09 \textbf{return} $p_t$

Figure 2.4: Naïve Random Scan Method

This definition is a generalized version of $Q = \langle n_Q, P_Q, W_Q, D_Q \rangle$ defined in [14], where the search space is assumed to be the whole database for every search. In our generalized definition, $\mathbb{S}'$ is included to account for the dynamic size of the search space, which shrinks gradually after each iteration. Let $Q_s$ denote the starting query, $Q_r$ a refined query at a feedback iteration, $Q_t$ a target query which results in the retrieval of the intended target, and $S_k$ the query result set.

2.1.1 Naïve Random Scan Method

The NRS method randomly retrieves $k$ different images at a time until the user finds the target image or the remaining set is exhausted, see Figure 2.4. Specifically, at each iteration, a set of $k$ random images are retrieved from the candidate (i.e. unchecked) set $\mathbb{S}'$ for relevance feedback (lines 2 and 6), and $\mathbb{S}'$ is then reduced by $k$ (lines 3 and 7). Clearly, the naïve scan algorithm does not suffer local maximum traps and is able to locate the target image after some finite number of iterations. In the best case, NRS takes one iteration, while the worst case requires $\left\lceil \frac{|\mathbb{S}|}{k} \right\rceil$. On average NRS can find the target in $\left\lceil \sum_{i=1}^{\left\lceil \frac{|\mathbb{S}|}{k} \right\rceil} i / \left\lceil \frac{|\mathbb{S}|}{k} \right\rceil \right\rceil = \left\lceil \left( \left\lceil \frac{|\mathbb{S}|}{k} \right\rceil + 1 \right) / 2 \right\rceil$.
iterations. In other words, NRS takes $O(|\mathcal{S}|)$ to reach the target point regardless of data distribution. Therefore, NRS is only suitable for a small database set.

### 2.1.2 Local Neighboring Movement Method

Existing techniques allow already checked images to be reconsidered, which leads to several major drawbacks. We apply our non-re-retrieval strategy to one such method, such as MindReader [40], to produce the LNM method. LNM is similar to NRS except lines 5 and 6 as follows:

```plaintext
05 \quad Q_r \leftarrow \langle n_Q, P_Q, W_Q, D_Q, S', k \rangle \text{ based on the user’s relevance feedback}
06 \quad S_k \leftarrow \text{EVALUATEQUERY}(Q_r) \quad // \text{perform a constrained } k\text{-NN query} */
```

Specifically, $Q_r$ is constructed such that it moves towards neighboring relevant points and away from irrelevant ones, and a $k$-NN query is now evaluated against $S'$ instead of $S$ (lines 5 and 6). When LNM encounters a local maximum trap, it enumerates neighboring points of the query, and selects the one closest to the target. Therefore, LNM can overcome local maximum traps, although it could take many iterations to do so.

Again, one iteration is required in the best case. To simplify the following worst-case and average-case complexity analysis, we assume that $S$ is uniformly distributed in the $n$-dimensional hypercube and the distance between two nearest points is a unit.

**Theorem 2.1** For LNM, the worst and average cases are $\left\lceil \sqrt{n} \sqrt{|\mathcal{S}|}/\lfloor \log_2 n \rfloor \right\rceil$ and $\left\lceil \frac{(\sqrt{n} \sqrt{|\mathcal{S}|})}{\lfloor \log_2 n \rfloor} + 1\right\rceil/2$, respectively, assuming $S$ is uniformly distributed.

**Proof:** The hypercube’s edge length is $\sqrt{|\mathcal{S}|} - 1$, and the diagonal’s $\sqrt{n}(\sqrt{|\mathcal{S}|} - 1)$. Let the distance between the initial query point and the target point be $l$, then $l \leq \sqrt{n}(\sqrt{|\mathcal{S}|} - 1) < \sqrt{n} \sqrt{|\mathcal{S}|}$. Note that the expected radius for $k$-NN search in $S$ is $r = \lfloor \log_2 n \rfloor$ because the distance between two nearest points is a unit as given above. Since $S' \subset S$, $k$-NN search in LNM requires a radius larger than $r$, but less than $2r$. In other words, at each iteration,
LNM moves towards the target image at an average speed of $cr$ where $1 \leq c < 2$. It follows that the number of iterations needed to reach the target is $\lceil l/(c \log_2 n) \rceil$, which is bounded by $\left\lceil \sqrt{n} \sqrt{|S|}/[\log_2 n] \right\rceil$. Then, the worst and average cases are $\left\lceil \sqrt{n} \sqrt{|S|}/[\log_2 n] \right\rceil$ and $\left\lceil (\sqrt{n} \sqrt{|S|}/[\log_2 n] + 1)/2 \right\rceil$, respectively.

If the data were arbitrarily distributed, then the worst case could be as high as NRS’s, i.e. $\left\lceil |S|/k \right\rceil$ iterations (e.g., when all points are on a line). In summary, in the worst case LNM could take anywhere from $O(\sqrt{|S|})$ to $O(|S|)$.

### 2.1.3 Neighboring Divide-and-Conquer Method

Although LNM can overcome local maximum traps, it does so inefficiently, taking many iterations and in the process returning numerous false hits. To speed up convergence, we propose to use Voronoi diagrams [72, 3] in NDC to reduce search space. The Voronoi diagram approach finds the nearest neighbors of a given query point by locating the Voronoi cell containing the query point. Specifically, NDC searches for the target as follows, see Figure 2.5. From the starting query $Q_s$, $k$ points are randomly retrieved (line 2). Then the Voronoi region $VR_i$ is initially set to the minimum bounding box of $S$ (line 3). In the while loop, NDC first determines the Voronoi seed set $S_{k+1}$ (lines 6 to 10) and $p_i$, the most relevant point in $S_{k+1}$ according to the user’s relevance feedback (line 11). Next, it constructs a Voronoi diagram $VD$ inside $VR_i$ using $S_{k+1}$ (line 12). The Voronoi cell region containing $p_i$ in $VD$ is now the new $VR_i$ (line 13). Because only $VR_i$ can contain the target (as proved in Theorem 2.2), we can safely prune out the other Voronoi cell regions. To continue the search in $VR_i$, NDC constructs a $k$-NN query using $p_i$ as the anchor point (line 15), and evaluates it (line 16). The procedure is repeated until the target $p_t$ is found. When NDC encounters a local maximum trap, it employs Voronoi diagrams to aggressively prune the search space.
**NEIGHBORINGDIVIDECONQUER**(S, k)

**Input:**
- set of images $S$
- number of retrieved images at each iteration $k$

**Output:**
- target image $p_t$

```plaintext
01 $Q_s \leftarrow \langle 0, P_Q, W_Q, D_Q, S, k \rangle$
02 $S_k \leftarrow \text{EVALUATEQUERY}(Q_s)$ /* randomly retrieve $k$ points in $S$ */
03 $VR_i \leftarrow$ the minimum bounding box of $S$
04 $iter \leftarrow 1$
05 **while** user does not find $p_t$ in $S_k$ **do**
06     **if** $iter \neq 1$ **then**
07         $S_{k+1} \leftarrow S_k + \{p_i\}$
08     **else**
09         $S_{k+1} \leftarrow S_k$
10     **endif**
11     $p_i \leftarrow$ the most relevant point $\in S_{k+1}$
12     construct a Voronoi diagram $VD$ inside $VR_i$ using points in $S_{k+1}$ as Voronoi seeds
13     $VR_i \leftarrow$ the Voronoi cell region associated with the Voronoi seed $p_i$ in $VD$
14     $S' \leftarrow$ such points $\in S$ that are inside $VR_i$ except $p_i$
15     $Q_r \leftarrow \langle 1, \{p_i\}, W_Q, D_Q, S', k \rangle$
16     $S_k \leftarrow \text{EVALUATEQUERY}(Q_r)$ /* perform a constrained $k$-NN query */
17     $iter \leftarrow iter + 1$
18 **endo**
19 **return** $p_t$
```

Figure 2.5: Neighboring Divide-and-Conquer Method

and move towards the target image, thus significantly speeding up the convergence. Therefore, NDC can overcome local maximum traps and achieve fast convergence. We prove the following invariant.

**Theorem 2.2** The target point is always contained inside or on an edge (surface) of $VR_i$, the Voronoi cell region enclosing the most relevant point $p_i$.

**Proof:** This theorem can be proved by contradiction. First, note that according to the properties of the Voronoi cell construction, if $VR_i$ contains the most relevant point (i.e. the closest point) $p_i$ to the target point $p_t$, its seed $p_i$ is the nearest neighbor of $p_t$ among $S_{k+1}$. Suppose $p_t$ is inside $VR_j$, $i \neq j$. Then there exits another point in $S_{k+1}$ closer to $p_t$ than $p_i$, a contradiction.
Figure 2.6 explains how NDC approaches the target. In the first iteration, $S_k = \{p_1, p_2, p_s\}$ is randomly picked by the system, assuming $k = 3$. The user identifies $p_s$ as $p_i$ (the most relevant point in $S_k$). NDC then constructs a Voronoi diagram based on those three points in $S_{k+1} = S_k$, partitioning the search space into three regions. According to Theorem 2.2, the target must be in $VR_i$. NDC thus ignoring the other two regions, performs a $k$-NN query anchored at $p_s$ and retrieves $S_k = \{p_3, p_4, p_5\}$, the three closest points inside $VR_i$. Again, the user correctly identifies $p_5$ as the most relevant point in $S_{k+1} = \{p_s, p_3, p_4, p_5\}$. The system constructs a Voronoi diagram and searches only the Voronoi cell associated with $p_5$. The search continues and, finally, at the fourth iteration, the target point is reached as the result of a $k$-NN query of $p_6$, the most relevant point in $\{p_5, p_6, p_7, p_8\}$ retrieved in the third iteration. We now determine the worst-case complexity for NDC, assuming that $S$ is uniformly distributed.

**Theorem 2.3** Starting from any point in $S$, NDC can reach any target point in $O(\log_k |S|)$ iterations.

**Proof:** At the first iteration, $S$ is divided into $k$ Voronoi cells. Since the points are uniformly distributed from which $k$ points are randomly sampled, each $VR$ is expected to contain $\left\lceil \frac{|S|}{k} \right\rceil$ points. According to Theorem 2.2, we only need to search one $VR$, which contains about $\left\lceil \frac{|S|}{k} \right\rceil$ points. In the second iteration, the searched $VR$ contains $\left\lceil \frac{|S|}{k} - 1 \right\rceil / k \approx \lceil |S|/k^2 \rceil$ points. In the $i^{th}$ iteration, each $VR$ contains about $\left\lceil \frac{|S|}{k^i} \right\rceil$ points. Since $\frac{|S|}{k^i} \geq 1$, NDC will stop by $i \leq \log_k |S|$. Hence, NDC reaches the target point in no more than $O(\log_k |S|)$ iterations.

When $S$ is arbitrarily distributed, the worst case could take up to $\left\lceil \frac{|S|}{k} \right\rceil$ iterations (e.g., all points are on a line), the same as that of NRS. In other words, NDC could still require $O(|S|)$ iterations to reach the target point in the worst case.
2.1.4 Global Divide-and-Conquer Method

To reduce the number of iterations in the worst case in NDC, we propose the GDC method. Instead of using a query point and its neighboring points to construct a Voronoi diagram, GDC uses the query point and $k$ points randomly sampled from $VR_i$. Specifically, GDC replaces lines 15 and 16 in NDC with:

15 $Q_r \leftarrow (0, P_Q, W_Q, D_Q, S', k)$
16 $S_k \leftarrow \text{EVALUATEQUERY}(Q_r)$ /* randomly retrieve $k$ points in $S'$ */

Similar to NDC, when encountering a local maximum trap, GDC employs Voronoi diagrams to aggressively prune the search space and move towards the target image, thus significantly speeding up the convergence. Therefore, GDC can overcome local maximum traps and achieve fast convergence.

Figure 2.7 shows how the target could be located according to GDC. In the first iteration, $S_k = \{p_1, p_2, p_s\}$ is the result of $k = 3$ randomly sampled points, of which $p_s$ is picked as $p_i$. Next, GDC constructs a Voronoi diagram and searches the $VR$ enclosing $p_s$. At the second iteration, $S_{k+1} = \{p_s, p_4, p_5, p_6\}$ and $p_5$ is the most relevant point $p_i$. In the third and final iteration, the target point is located; GDC takes 3 iterations to reach the target point. We prove that the worst case for GDC is bounded by $O(\log_k |S|)$ regardless of data distribution.

16
Theorem 2.4  Starting from an initial point in $S$, GDC can reach any target point in $O(\log_k |S|)$ iterations.

Proof: We will focus our attention on the size of $VR$ at each iteration, keeping in mind that points are randomly sampled for Voronoi diagram construction. Thus, at the first iteration, the searched $VR$ contains $\left\lceil \frac{|S|}{k} \right\rceil$ points; at the second iteration, it contains $\left\lceil \frac{|S|}{k(k+1)} \right\rceil$ points; and so on. At the $i^{th}$ iteration, it contains $\left\lceil \frac{|S|}{k(k+1)^{i-1}} \right\rceil$ points. Because $\frac{|S|}{k(k+1)^{i-1}} > 1$, that is, it requires that $i < \log_k |S|$. In other words, GDC can reach any target point in no more than $O(\log_k |S|)$ iterations.

Theorem 2.4 implies that for arbitrarily distributed datasets, GDC converges faster than NDC in general, although NDC might be as fast as GDC in certain queries, e.g., if the starting query point is close to the target point. In the previous example (Figure 2.6), NDC could also take three iterations, instead of four, to reach the target point if the initial $k$ points were the same as in Figure 2.7, as opposed to Figure 2.6.
2.2 Handle Inaccurate Relevance Feedback

Users’ inaccurate relevance feedback is a major issue for CBIR systems with RF. We need to make our system less sensitive to users’ uncertainty. For simplicity, we have assumed that users accurately picked the most relevant image out of the returned images for each iteration in the above discussion. In practice, however, users could make a wrong choice, or they might pick several seemingly good choices instead of settling on one in a target search query. Hence, we should not assume the system is always presented with correct queries.

To deal with this situation, we construct, in each iteration, a single query point that is a weighted centroid of all the picked images, as in MARS and MindReader. For example, the visual difference between images 1 and 2 (illustrated in Figure 2.8.1) could be so small that there is a high probability that users select the wrong image (i.e., image 1) for the next iteration. If this happens, the target image may never be found unless backtracking is allowed in NDC and GDC (NRS and LNM still work). When a single good choice is uncertain such as in this case, users are allowed to mark those images as relevant, and our system will choose their weighted centroid as the refined query point, shown in Figure 2.8.2.
Detecting inaccurate relevance feedback is also desirable. The following theorem and lemma can facilitate the detection.

**Theorem 2.5** If $\cos(\alpha) < 0$ where $\alpha$ is the angle between one vector from the previous query point $p_o$ to the new query point $p_n$ and the other from $p_o$ to the target $p_t$, the user must be giving inaccurate relevance feedback.

**Proof:** When the user is giving inaccurate relevance feedback, the new query point $p_n$ is farther from the target $p_t$ compared with the previous query point $p_o$. Because $-180^\circ \leq \alpha \leq 180^\circ$ and $\cos(\alpha) < 0$, then $90^\circ < |\alpha| \leq 180^\circ$. When $|\alpha| = 180$, $p_n$ heads in the direction away from the target, and therefore the user must have given inaccurate relevance feedback. When $90^\circ < |\alpha| < 180^\circ$, $|\alpha|$ should be the largest angle in the triangle, made by $p_o$ (e.g., point 2 in Figure 2.9.1), $p_n$ (e.g., point 3 in Figure 2.9.1) and $p_t$. This is because it is impossible to have two obtuse angles in this triangle. Then the edge connecting $p_n$ and $p_t$ is the longest edge in the triangle based on triangle’s properties. This means $p_n$ is farther from $p_t$ compared with $p_o$, indicating the user must have given inaccurate relevance feedback.

**Lemma 2.1** The relevance feedback must be inaccurate if

$$c_1\overrightarrow{p_o p_n} \cdot c_2\overrightarrow{p_o p_t} = c_1 c_2 |\overrightarrow{p_o p_n}| |\overrightarrow{p_o p_t}| \cos(\alpha) < 0$$

where $c_1$ and $c_2$ are two positive constants.

Lemma 1 simplifies the detection of inaccurate relevance feedback since only the vectors’ directions, and not their magnitudes matter. Even though the exact location of the target might be unknown, but its position relative to other results can be inferred from its visual features the user already knows. Thus, for a given target image, the user knows how to
move towards the target in the search space. Based on the user’s feedback, our target search technique is able to zoom in a narrower space the target must be in. In other words, we now know the approximate whereabouts of the target, though not its exact location (within that space). Lemma 1 relaxes the requirement of the exact location (i.e., only the direction of $\vec{p_0p_t}$ is needed to approximate). Assume that most of user’s behavior is consistent; i.e., the probability of accurate relevance feedback is larger than 0.5. This approximation problem can be treated as a probability problem. Basically, the more query points, the better the approximation. One way to estimate the direction is to use linear regression. For example, suppose the user has made four feedback iterations (see Figure 2.9.1), moving the query point from $p_s, p_1, p_2$ to $p_3$. Then the direction of $\vec{p_0p_t}$ (i.e., $\vec{p_2p_t}$ in this case) can be approximated by vector $\vec{v_1}$, which moves towards the search space containing $p_t$ and is a linear regression of points $p_s, p_1$ and $p_2$. In other words, $\alpha$ is approximated by $\beta$, where $\beta$ is the angle between $\vec{p_2p_3}$ and $\vec{v_1}$. If the dot product of $\vec{p_2p_3}$ and $\vec{v_1}$ is less than 0, it suggests that the user is likely to have given inaccurate relevance feedback, and a warning should be issued. However, if the relevance feedback is in fact accurate and the dot product is less than 0, this indicates that the direction approximation of $\vec{p_0p_t}$ is not accurate and we need to adjust it. For example, the dot product of $\vec{p_2p_3}$ and $\vec{v_1}$ is less than 0 in Figure 2.9.2 while the user’s relevance feedback is accurate. We can replace $p_2$ with $p_3$ for linear regression (i.e., only use points $p_s, p_1$ and $p_3$ for linear regression, omitting $p_2$), producing $\vec{v_2}$ for the approximation of $\vec{p_0p_t}$ in the next iteration.

Detecting users’ inaccurate relevance feedback is a difficult and open problem. We rely on short-term memory—the last few relevance feedbacks—to predict the general direction towards the destination, and focus on warning users if their feedbacks seem to be contradictory (our technique is only able to give a summary warning to the user, who may not be able to tell which one among the previous steps is inaccurate). Such a warning points out to users that their consecutive feedbacks appear contradiction, and is helpful to users in providing a better relevance feedback for the subsequent rounds. Actually, we have taken the following
steps to ensure our system is less sensitive to users’ inaccurate relevance feedback, in design and in implementation. First, we still keep LNM besides GDC and NDC in our prototype. Although converging slowly, LNM is robust against inaccurate relevance feedback because it basically enumerates the candidate images. Second, we use the above proposed method to automatically monitor users’ feedbacks, and issue warnings if inconsistent behaviors are detected. These warnings prompt the users to re-evaluate their feedbacks. Finally, our prototype allows users to backtrack their selections if missteps have been made.

### 2.3 Query Processing Technique for Target Search

In this section, we will discuss how to construct the index structure and to efficiently evaluate the three typical types of queries (i.e., sampling queries, constrained sampling queries and constrained $k$-NN queries) involved in the proposed target search methods.

#### 2.3.1 Index Structure for Target Search

Our index structure (see Figure 2.10) is constructed in two stages as follows:

**Hierarchical Clustering:** A hierarchical clustering technique, similar to the R*-tree [6], is used to organize the entire image database into a hierarchical tree structure. With each
node in this hierarchy representing a cluster, we extend the original node structure of the R*-tree to include also information to identify the images in their children nodes. We selected the R*-tree for our study because it is well known and has been widely used in practice, although other hierarchical clustering techniques [8] can be used as well. The hierarchical clustering is constructed as follows. When a new element (i.e., an image represented as a high-dimensional point) is inserted into the tree, this element is placed in a leaf node that requires the least enlargement in its bounding box, and a leaf node’s MBB is based on all dimensions of its contained image points. If a leaf node overflows, this node is split (i.e., a portion of its entries are removed from the node and reinserted into the tree), and such splits propagate up the tree [6].

**Information Augmenting:** We traverse the tree in a postorder fashion. In the original R*-tree, an *internal node* contains an array of *node entries*. Each node entry is a pair (*mbb*, *node-id*), where *mbb* is the minimum bounding box (MBB) that spatially contains the MBBs in the child node, with *node-id* as the child node address. In our index structure, each node entry is extended to be a tuple (*mbb*, *node-id*, *imageID-range*), where *imageID-range* refers to the range of image identifications contained in the pointed child node and *imageID-range* ⊆ [1, |S|]. Let us describe how to augment the structure illustrated in Figure 2.10. We start from the root node (i.e., Node 1) which has three node entries. We first visit the first node entry which points to Node 2. Node 2 has two node entries, pointing to leaf nodes 5 and 6 in order. Our depth-first traversal leads us to Node 5, which contains 3 image points. Then we set the *imageID-range* of the first entry in Node 2 to [1, 3], and each image contained in this node can randomly pick an exclusive ID within this range. That is, the three images in Node 5 can be assigned IDs 1, 2, and 3, respectively. We of course need to build up a one-one mapping between image IDs and exact image names, such as building a B+-tree index on the image ID field, or simply changing the image names to their corresponding image IDs as in our current implementation. Similarly, we set the *imageID-range* of the second entry in Node 2 to [4, 6]. As Node 2 doesn’t have any more node entries, we track
back to Node 1 and set *imageID-range* to encompass the ranges of all its children, which is \([1, 3] \cup [4, 6] = [1, 6]\) in this example. The above procedure is repeated for the second entry of Node 1. The *imageID-range* values in different internal node entries are shown in Figure 2.10.

When a new image is inserted, the structure has to be rebuilt. Because image databases are fairly static [27], the reconstruction is still acceptable considering the performance gains (shown in Section 2.4) we obtain. Our index structure has two properties as stated in the following theorems. Let \(M\) (i.e., node capacity or fanout) denote the maximum number of entries that can fit in a node, \(m\) the minimum number of entries in a node (we set \(m = M/2\) assuming \(M\) is an even number), \(L\) the total number of leaf nodes in our index structure, and \(L_Q\) the total number of leaf nodes related to a user query \(Q\) (assuming all related leaf nodes are contained in \(S'\) of \(Q\)).

**Theorem 2.6** For sampling queries, if \(k < L/2\), no retrieved points will be sampled from the same leaf node.

**Proof:** Recall that R*-tree is a height-balanced tree and the number of image points in each leaf node is between \(m\) and \(M\). For sampling queries, \(S\) is the search space, therefore the corresponding *imageID-range* is \([1, |S|]\). One possible solution is the following sampled image ID set \(\{1, M + 1, 2M + 1, 3M + 1, \ldots, (k - 1) \ast M + 1\}\). We first prove that \((k - 1)M + 1 \leq |S|\), which means the highest ID in the above set is a valid ID. Obviously, \(|S| \geq L \ast m = L \ast M/2\). On the other hand, \((k - 1)M + 1 < (L/2 - 1) \ast M + 1 \leq L \ast M/2\). It follows that \(|S| > (k - 1)M + 1\). Next we prove that no two images in this set are from the same leaf node. Clearly, if two images are in the same node, the difference between the values of the two corresponding IDs should be less than \(M\) (see Figure 2.10 for an example, \(M = 3\)). The difference between any two IDs in the above set is \(cM\) where \(c\) is a natural number, therefore no two images in the above set are from the same leaf node.  

23
Theorem 2.7  For constraint sampling queries, if $k > 2L_Q$, each leaf node related to $Q$ will be sampled.

Proof: Each relevant leaf node corresponds to an imageID-range. We union those imageID-ranges, and transform the union into a consecutive range $[1, |S'|]$ for analysis simplicity. One possible solution is: we sample the consecutive range at fixed interval $M/2$, then we obtain the following sampled image ID set

$$\{M/2, 2M/2, 3M/2, \ldots, s \ast M/2\},$$

where $s = \left\lfloor \frac{|S'|}{M/2} \right\rfloor$. Obviously, $L_Q \ast M/2 \leq |S'| \leq L_Q \ast M$, from which we have $L_Q \leq s \leq 2L_Q$. Since $k > 2L_Q$, then $L_Q \leq s < k$, which implies the number of sampled points in the above set is even fewer than $k$. Each relevant leaf node contains a imageID-range $\subseteq [1, |S'|]$ with length at least $M/2$, therefore its imageID-range will contain at least one sampled point in the above set. Hence, each leaf node related to $Q$ will be sampled.

The above desirable properties show that our index structure can facilitate sampling as many relevant leaf nodes as possible, and the sampled points can better capture the data distribution, thus are more representative. The results of our empirical study in Section 2.4 confirm that our index structure, in fact, can help improve both retrieval effectiveness and retrieval efficiency.

2.3.2 Efficient Query Processing for Target Search

We discuss our query processing technique \textsc{EvaluateQuery}(Q) on top of the above index structure for the three types of queries (i.e., sampling queries, constrained sampling queries and constrained \textit{k}-NN queries) based on our four target search methods. \textit{k}-NN queries are omitted because our target search methods do not use them. The query cost is the sum of
EVALUATEQUERY(Q)

Input:
given query Q
Output:
query result S_k

01  S_k ← ∅
02  if Q is a sampling query then /* based on n_Q and S' */
03      read the root node of our index structure
04      if k < L/2 then
05          choose a random non-negative integer n such that imageIDset =
06          \{1 + n, M + 1 + n, 2M + 1 + n, 3M + 1 + n, \ldots, (k - 1) * M + 1 + n\} ⊆ \{1, \ldots, |S|\}
07          else
08              imageIDset ← randomly sample k IDs from \{1, \ldots, |S|\} stored in root
09          endif
10  S_k ← retrieve images whose ID ∈ imageIDset
11  elseif Q is a constrained sampling query then
12      IDset ← All image IDs whose corresponding points ∈ S' by performing a modified
13              range query
14      if k > 2L_Q then
15              transform IDs ∈ IDset into a consecutive range [1, |S'|]
16              choose a random non-negative integer n such that imageIDset =
17              \{[M/2] + n, [2M/2] + n, [3M/2] + n, \ldots, s * M/2 + n\} ⊆ \{1, \ldots, |S'|\}
18              transform all IDs ∈ imageIDset back to the original ones
19              add some different IDs ∈ IDset − imageIDset into imageIDset such that
20              |imageIDset| = k
21      else
22              imageIDset ← randomly sample k IDs from IDset
23          endif
24  S_k ← retrieve images whose ID ∈ imageIDset
25  else /* Q is a constrained k-NN query */
26      Queue ← NEWPRIORITYQUEUE()
27      Insert nodeSet into Queue, where nodeSet is pruned by mbb(S')
28      while not ISEMPTY(Queue) and |S_k| < k do
29          Element ← DEQUEUE(Queue)
30          if Element is an image object and Element is inside mbb(S') then
31              S_k ← S_k ∪ \{the image corresponding to Element\}
32          elseif Element is a leaf node then
33              for each Object in leaf node Element do
34                  ENQUEUE(Queue, Object, OBJDIST(P_Q, Object))
35          enddo
36          else /* Element is a non-leaf node */
37              for each Child node of node Element do
38                  ENQUEUE(Queue, Child, DIST(P_Q, Child))
39          enddo
40      enddo
41      return S_k

Figure 2.11: Query Processing Technique
disk seek (including cylinder seek and rotation), data transfer and CPU time, in which seek time dominates the total query cost. Figure 2.11 lays out our query processing technique, designed to minimize the disk I/O cost.

For sampling queries, we just need to retrieve the root node (in line 3), which is stored in memory to reduce the disk I/O cost for the subsequent sampling queries. The root node contains all possible image IDs. If \( k \) is relatively small (i.e., \( k < L/2 \)), we will choose the technique based on Theorem 2.6 to guarantee that no images will be sampled from the same leaf node in order to make the sampled images as representative as possible (in line 5). Otherwise, random sampling can be performed to retrieve the query result (in line 7).

For constrained sampling queries, we will use the minimum boundary box of \( S' \), denoted by \( mbb(S') \), as the range, then perform a modified range query on our index structure to collect sampling image IDs (in line 11). Since in our index structure, each internal node entry is a tuple \((mbb, node-id, imageID-range)\), thus if \( mbb(S') \) contains a node’s \( mbb \), we can put the node’s \( imageID-range \) into \( IDset \) without visiting its children. If \( mbb(S') \) overlaps with or is contained in a node’s \( mbb \), we will visit its children recursively. Consequently, we can prune a lot of nodes to answer a constrained sampling query, therefore significantly reducing the disk I/O accesses. If \( k \) is relative large (i.e., \( k > 2L_Q \)), the technique based on Theorem 2.7 will guarantee that each leaf node related to \( Q \) will be sampled in order to make the sampled images more representative (from line 13 to 16). Otherwise, random sampling over \( IDset \) is performed to retrieve the query result (in line 18). To take advantage of the shrinking of the search space \( S' \) after each iteration (e.g., in GDC), we can recycle the nodes visited in the previous iterations to avoid re-reading those nodes from the disk. That is, we can put those visited nodes into \( nodeSet \) residing in memory, and perform the range query over \( nodeSet \).

For constrained \( k \)-NN queries, we extend the well-known \( k \)-NN algorithm proposed in [35]. Considering the search space \( S' \) is shrunk after each iteration and the results of two consecutive constrained \( k \)-NN queries may overlap, we recycle the visited nodes in the previous
iterations as does for constrained sampling queries. We first prune nodeSet by mbb(S′) (in line 23); that is, only nodes that either overlap with or are contained in mbb(S′) are kept. To answer a given constrained k-NN query, all nodes in nodeSet are inserted into a new priority queue Queue (in line 23) while nodeSet only contains the root node in the first iteration. In the while loop (from line 24 to 37), an element is dequeued from Queue (in line 22). If the element is an image object and is contained in mbb(S′), this image object is put into the query result Sk and the next element is dequeued from Queue until all k nearest neighbors are found or Queue is empty. If the element is a leaf node, all image objects in it are inserted into Queue (in line 30). If the element is an internal node, its child nodes are inserted into Queue (in line 34). In sum, we reuse the visited nodes in the previous iterations and prune nodeSet by mbb(S′) to reduce the disk I/O cost for answering constrained k-NN queries.

2.4 Experiments

In this section, we present experimental results for target search in both simulated and realistic environments, and evaluate the effectiveness of the query processing technique described in Section 2.3. Our dataset consists of more than 68,040 images from the COREL library. 37 visual image features divided into three main groups were used: colors (9 features) [85], texture (10 features) [83], and edge structure (18 features) [101]. The combination of those features captures essential image characteristics and facilitates effective similarity comparison. Our experiments were run on Sun UltraSPARC with 2GB memory.

2.4.1 Simulated Experiments

In these experiments, we evaluated the performances of MARS [66, 74], MindReader [40], and Qcluster [46] against our techniques (NRS’s results are omitted since its performance can be statistically predicted). All the data resided in memory. The performance metrics of interest are the average total visited images, precision, recall, computation time and the
Figure 2.12: False Hit Ratio

Figure 2.13: Average Iterations
Figure 2.14: Maximum Iterations

Figure 2.15: Minimum Iterations
Figure 2.16: Standard Deviation of Iterations

Figure 2.17: Average Aggregate Recall
Figure 2.18: Average Aggregate Precision

Figure 2.19: Average Total Checked Images
number of iterations (average, maximum, minimum, and their variance) needed for each method to retrieve an intended target. These were measured as $k$ takes different values in $\{5, 15, 30, 50, 75, 100\}$. There were 100 pairs of starting points-target points selected randomly for the experiments.

In order to accurately evaluate the prime metrics, relevance feedback in these experiments was simulated: the point in the retrieval set closest to the target point was automatically selected as the most relevant point. To save computation overhead for NDC and GDC, we constructed the Voronoi region $VR_i$ containing the most relevant point instead of the whole Voronoi diagram, and approximated $VR_i$ by its minimum boundary box if $VR_i$ contains too many surfaces.

To illustrate the common problems of slow convergence and local maximum traps with the existing approaches, we demonstrate that MARS, MindReader and Qcluster have poor false hit ratios for small $k$. Figure 2.12 shows that when $k$ is small, their performance is affected by local maximum traps, i.e., their false hit ratios are very high even for a fairly
large $k$. For example, when $k = 100$, MARS’s false hit ratio is about 20% and Qcluster’s exceeds 40%, while the best performer MindReader is just below the 20% mark. As a result, users of these techniques have to examine a large number of returned images, but might not find their intended targets.

In the experiments that produced the number of iterations, we had to make sure that the compared techniques could successfully reach the intended targets. We thus used LNM in place of MindReader (LNM is an improved version of MindReader, see Section 2.1). The experimental results for LNM, NDC and GDC are shown in Figures 2.13 to 2.20. They show that NDC and GDC perform more efficiently when $k$ is small, with GDC being slightly better than NDC. Specifically, when $k = 5$, the average numbers of iterations for LNM, NDC and GDC (see Figure 2.13) are roughly 21, 10 and 7, respectively (compared to $\frac{68040}{5} = 13608$ iterations in NRS); the maximum numbers are 58, 20 and 11, respectively (see Figure 2.14); and the minimum numbers are 7, 4 and 4, respectively (see Figure 2.15). The results also confirm our analysis of GDC complexity (see Figure 2.13): GDC can reach the target point in $O(\log_k |S|) = (\log_5 68040) = 6.9141 \approx 7$ iterations.

The standard deviations of the iterations are shown in Figure 2.16. GDC and NDC are much more stable than LNM, with GDC’s slightly more uniform than NDC’s. This indicates that GDC and NDC can achieve fast convergence even with a poor selection of initial query points.

The average ‘aggregate’ recalls and precisions, defined in Section 2.1, are shown in Figures 2.17 and 2.18 respectively. Again, experimental results show that NDC and GDC achieve better retrieval effectiveness (precision and recall) when $k$ is small compared to LNM, with GDC being slightly better than NDC.

The average total checked images for LNM, NDC, and GDC in the experiments are plotted in Figure 2.19. The figure shows that GDC and NDC examined fewer than half of the total checked images of LNM (compared to $\frac{68040}{2} = 34020$ images need to be checked in NRS). In terms of CPU time, GDC is the most efficient, although the difference is smaller
Figure 2.21: GDC Average Iterations Comparison

as \( k \) increases (see Figure 2.20). This is because NDC and GDC take some computation overhead to construct \( VR_i \), while LNM requires more iterations and associated computation time for adjusting the generalized distance function. Overall, GDC and NDC significantly outperform LNM, with GDC slightly outdoing NDC. Figure 2.21 shows that using our index structure, the average iterations of GDC over 100 simulated target searches can be reduced by 1 when \( k = 5 \) and \( 15 \), although the difference is smaller as \( k \) increases. The reason is that our index structure is designed to sample as many relevant leaf nodes as possible and to better capture the data distribution, thus facilitating GDC to prune the search space.

### 2.4.2 Realistic Experiments

In simulated experiments, the most relevant points were assumed to be accurately selected among the returned points. In practice, however, this cannot be easily achieved by human
evaluators, unless the most relevant images are distinctly stood out. To evaluate our methods’ performance in realistic environments, we have extended the previous prototype [56] (based on ImageGrouper [62]) to couple with our index structure. Our prototype, shown in Figure 2.22, allows users to pose queries by dragging and grouping multiple relevant images on the work space (i.e., the right pane), choose discriminative visual features, and select one of the three retrieval methods (LNM, NDC and GDC). It monitors users’ feedback and prompts users to reexamine their relevance feedback if certain conditions are true as discussed in Section 2.2. It also allows users to rollback their feedback in the previous iteration if they wish. Thus, for instance, if there are several relevant images, the user can group them together to form a query, and if he reaches a dead-end without finding the target image, he can rollback.

We trained 20 graduate students (i.e., 15 engineering students and 5 art students) to use the target search system and asked them to find 36 given target images from different semantic categories in both situations (with or without our index structure). In Figure 2.23,
we show the results for finding the given 36 target images with $k = 50$ (i.e., 50 images were retrieved at each feedback iteration). The two images (i.e., an ancient building and race cars) took, on average, more iterations than the others to retrieve, mainly because many similar images exist in the collection. Even so, only 6 iterations on average were needed to locate them, while 7 iterations were needed without our index structure [56]. The results illustrate that our index structure can help reduce the number of iterations. The reason is that our index structure is designed to sample as many relevant leaf nodes as possible, and the sampled images can be very representative, which facilitates target search. To evaluate our target search technique in more practical real world scenarios, we conducted experiments on the collection through Google Image Search (randomly chose Corel category descriptions as queries). This collection contains the same number (i.e., 68040) of images
Figure 2.24: Sampling Queries

Figure 2.25: Constrained Sampling Queries
as the COREL dataset, and we also randomly picked the same number (i.e., 36) of target images. The results show that 8 iterations on average were needed to locate them. The log information indicates that LNM has been used by most users probably because some of the Google images have low image quality, and are hard to give the accurate relevance feedback.

After analyzing the experimental results, we also found out that art students on average took fewer iterations than the engineering ones in both experimental settings (using Corel images, and using Google images), probably because the former are better at recognizing the visual features, and then gave more accurate relevance feedback.

Observe that users’ inaccurate response may compromise the benefits of any CBIR systems with relevance feedback. To minimize its effects, we make our system, in design and in implementation, less sensitive to users’ inaccurate relevance feedback. First, our prototype (see Figure 2.22) still keeps LNM as a useful option. This is because LNM is robust against inaccurate relevance feedback as mentioned before, although converging slowly. Based on our observations, users in practice can use GDC or NDC to prune a lot of non-target images at
the first few iterations, and use LNM to finally locate the target. Second, in the experimental study, our system monitored users’ feedback, and issued warnings in the Status window if inconsistent behaviors were being detected (discussed in Section 2.2). These warnings prompt the users to re-evaluate their feedback. Finally, our prototype allows users to backtrack their selections if missteps have been made. The results were satisfactory overall, indicated by the successful finding of the intended targets. Of course users’ inaccurate relevance feedback is a difficult and open problem but our results are encouraging.

2.4.3 Query Processing Technique

In this section, we evaluate the effectiveness of the proposed query processing technique described in Section 2.3.2. The node size of the original R*-tree and our index structure were both set to 4KB, and both had three levels in our experimental settings. Following the suggestion of the R*-tree [6], the minimum utilization parameter of each node was set to 40% and the reinsert fraction parameter was set to 30% for all index structures. We compare
the performance of the proposed query processing technique on top of our index structure (denoted as \textit{QNEW}) against the existing technique with R*-tree (denoted as \textit{QOLD}). Specifically, our methods to evaluate sampling queries, constrained sampling queries, constrained k-NN queries are from line 2 to 9, 10 to 20, and 21 to 38 in Figure 2.11, respectively. The existing counterparts are proposed in [65], [65] with a straightforward extension by integrating the constraint, and [23], respectively. As mentioned in Section 2.3.2, the new index structure is introduced specifically for our query processing technique. To be fair, we also utilize an index structure (i.e., R*-tree) for the existing counterparts. It is not our claim that the proposed index structure is better than R*-tree. Actually, we claim that our query processing technique coupled with the new index structure outperforms existing ones with R*-tree. We use the number of disk accesses as the main measure of performance to compare \textit{QNEW} and \textit{QOLD}. Sampling queries, constrained sampling queries, and constrained k-NN queries were executed in these experiments; they were randomly generated, and relevance feedback was simulated as in Section 2.4.1. For constrained sampling queries, \textit{mbb}(S') was randomly chosen up to 75\% of \textit{mbb}(S). The dataset and image features were those used in Section 2.4.1. The results are averaged over 100 runs.

Figure 2.24 depicts that \textit{QNEW} significantly outperforms \textit{QOLD} for answering sampling queries in terms of disk accesses. For example, \textit{QOLD} performs about 5 times more disk accesses than \textit{QNEW} when \( k = 5 \), 150 times when \( k = 50 \), and 300 times when \( k = 100 \). This figure shows that \textit{QNEW} is independent of the number of sample points (i.e., \( k \)) because \textit{QNEW} just needs to access the root node of our index structure, resulting in only one disk access for answering a sampling query. On the other hand, \textit{QOLD} is proportional to \( k \), which is because \textit{QOLD} has to traverse the R*-tree to obtain sample points one by one, incurring almost 3 disk accesses per sample point. Figure 2.25 compares the performance of both approaches for answering constrained sampling queries. \textit{QNEW} is again independent of \( k \) because \textit{QNEW} just needs to perform a modified range query instead of sampling one by one as done by \textit{QOLD}. Although \textit{QOLD} slightly outdoes \textit{QNEW} when \( k \) is very small,
QNEW is superior when \( k > 8 \) and the performance gap widens as \( k \) increases. Specifically, QOLD requires about 7 times more disk accesses than QNEW when \( k = 50 \), and almost 15 times when \( k = 100 \). In feedback iterations, QNEW can reduce the cost further as shown in Figure 2.26 while QOLD cannot. The reason is that QNEW reuses the visited nodes in the previous iterations, instead of reloading them from the disk. Similarly, QNEW significantly outperforms QOLD by almost two orders of magnitude for answering constrained \( k \)-NN queries in terms of the overall I/O cost, as illustrated in Figure 2.27.

The performance difference between QNEW and QOLD confirms that the proposed query processing technique reduce the disk I/O cost significantly by taking advantage of our index structure, reusing the visited nodes in the previous iterations, and pruning non-relevant nodes as early as possible.

### 2.5 Summary

In this chapter, we proposed four target search methods using relevance feedback for content-based image retrieval systems. Our research was motivated by the observation that revisiting of checked images can cause many drawbacks including local maximum traps and slow convergence. Our methods outperform existing techniques including MARS (employing feature weighting), MindReader (employing complex feature weighting), and Qcluster (employing probabilistic models). All our methods are capable of guaranteeing finding intended target images, with NDC and GDC converging faster than NRS and LNM (which represents an improved version of MindReader). Simulated experiments have shown that NDC and GDC work more efficiently and effectively when \( k \) (i.e., the number of allowed returned images) is smaller, and GDC achieving \( \mathcal{O}(\log_k |S|) \) iterations is slightly better than NDC. We also proposed an index structure and efficient query processing technique. Experiments with our prototype show that our approach can achieve fast convergence (i.e. \( \mathcal{O}(\log_k |S|) \) iterations) even in the realistic environments, and is very promising for large CBIR systems.
CHAPTER 3: TRANSFER NON-METRIC MEASURES INTO METRIC FOR SIMILARITY SEARCH

Similarity search is widely used in multimedia retrieval systems to find the most similar ones for a given object. Some similarity measures, however, are not metric, leading to existing metric index structures cannot be directly used. To address this issue, we propose a simulated-annealing-based technique to derive optimized mapping functions that transfer non-metric measures into metric, and still preserve the original similarity orderings. Then existing metric index structures can be used to speed up similarity search by exploiting the triangular inequality property. The experimental study confirms the efficacy of our approach.

3.1 Introduction

Similarity search, which refers to finding the most similar objects to a given query object, is widely used in multimedia retrieval systems. Efficient evaluation of such similarity queries is a challenge when the multimedia database contains a large number of objects, and the underlying similarity (or distance) measure is computationally expensive. Although numerous index structures (e.g., R*-tree [6], SR-tree [44], A-tree [76], M-tree [17], and iDistance [43]) have been proposed to speed up similarity search, those methods typically assume that similarity measures are metrics such that the search space can be pruned by taking advantage of the triangular inequality property. This assumption, however, does not hold in various applications where the similarity measures are non-metric and objects do not have a fixed dimensionality. Such non-metric similarity measures include the Chamfer distance for comparing shapes [4], the Kullback-Leibler distance for comparing probability distributions [18], the dynamic time warping distance for comparing time series [45], and the edit distance for comparing strings [61]. For example, the Chamfer distance between two edge point sets $A$
and $B$ is the mean of the distances between each point in $A$ and its closest point in $B$, plus the mean of the distances vice versa. As illustrated in Figure 3.1:

\[ s(A, B) = \frac{(\sqrt{2} + 1 + \sqrt{2})}{3} + \frac{(1 + \sqrt{2} + 1)}{3} \approx 2.4, \]
\[ s(A, C) = \frac{(\sqrt{10} + \sqrt{5} + \sqrt{8})}{3} + \frac{\sqrt{5}}{1} \approx 5.0, \]
\[ s(B, C) = \frac{(2 + 1 + \sqrt{2})}{3} + \frac{1}{1} \approx 2.5. \]

Then $s(A, C) > s(A, B) + s(B, C)$, which means that the Chamfer distance does not follow the triangular inequality property and is therefore non-metric. Moreover, calculating this measure is costly because its time complexity is $O(d^2)$ if we assume that each edge point set has the same number of points $d$. In order to reduce the number of expensive measure calculation in similarity search, we therefore propose a simulated-annealing-based technique to derive optimized mapping functions that transfer non-metric measures into metric without changing the original similarity orderings, and then the existing index structures such as $M$-tree [17] can be employed to speed up similarity search.

The rest of this chapter is organized as follows. We give a brief review of the related work in Section 3.2. Section 3.3 describes in detail our technique, and Section 3.4 presents the experimental results. Finally, we conclude the chapter in Section 3.5.
3.2 Related Work

In this section, we review the existing techniques for handling non-metric measures in similarity search. These techniques can be divided into three categories: space-embedding techniques, classification techniques, and distance-mapping techniques.

Space-embedding techniques typically embed (i.e., transform) non-metric spaces into a Euclidean one, where the computation of Euclidean distance measures is very cheap compared to non-metric distance measures. The embedding is distance-preserving; that is, two objects are close to each other in the original space should be more likely close to each other in the embedding Euclidean space. Afterwards, existing multidimensional index structures can be used to facilitate similarity search. Existing space-embedding techniques include Lipschitz embeddings \([34]\), FastMap \([21]\), MetricMap \([94]\), and BoostMap \([1]\). Specifically, BoostMap \([1]\) transfers the embedding construction problem into the classical boosting problem that combines various weak classifiers into a strong one to achieve the embedding optimization. The classification accuracy of the resulting classifier indicates the embedding quality. These techniques, however, may lead to false dismissals due to the embedding, and are only suitable for static data set. If a new object is inserted, the embedding and the index structure have to be re-constructed from scratch.

Classification techniques \([41, 28]\) categorize objects into classes, choose representative objects for each class, and answer similarity search by performing classification of the query object. For example, DynDex \([28]\) employs a statistical approach including distance-based classification and bagging. Specifically, DynDex first employs the pairwise distance-based clustering, and each cluster is then stored in a sequential file to minimize disk latency. Second, DynDex transfers similarity search into a classification problem. Given a query object \(q\), DynDex estimates \(q\)'s class membership, and returns several candidate classes (i.e., clusters). From these candidate classes, DynDex finally finds those objects with the
shortest pairwise distance to $q$. The Classification techniques suffer the same problems as the space-embedding techniques: the approximate result set and static indexing.

Distance-mapping techniques [81], on the other hand, employ increasing functions to turn non-metric distance measures into approximated metric, while preserving the original distance orderings. Then existing metric index structures (e.g., M-tree) can be used to reduce the number of expensive distance computation by exploiting the triangular inequality property. Specifically, the TriGen algorithm [81] is introduced to derive an efficient mapping function among a set of concave functions by using the distance distribution in a fraction of the database, and by tuning the corresponding parameters. Distance-mapping techniques can obtain a more accurate result set, and can support dynamic indexing. However, the mapping functions derived by the TriGen algorithm are not optimal, which impairs the retrieval efficiency. Note that the retrieval efficiency is heavily affected by the *intrinsic dimensionality*, defined as $\rho(\mathcal{S}, d) = \frac{\mu^2}{2\sigma^2}$, where $\mu$ and $\sigma$ are the mean and the variance of the distance distribution in the data set $\mathcal{S}$, respectively. We therefore propose a technique to optimize the mapping functions in this chapter.

### 3.3 Mapping Function Optimization

In this section, we discuss the proposed technique to optimize the mapping functions that transfer non-metric similarity measures into metric without changing the original similarity orderings. Many concave functions can be used as the candidate mapping functions, such as $f(x) = x^{\frac{1}{1+w}}$ where $w > 0$, $f(x) = -(\psi-x+wx-aw)\cdot(-2bwx+2bw^2x-2abw^2+2bw-x+wx-aw+\psi(1-2bw))/(1-2aw-4awx-2aw^2+4aw^2x+2wx-2w^2x+2\psi(1-w))$ where $\psi = \sqrt{-x^2+x^2w^2-2aw^2x+a^2w^2+x}$ and $w, a, b > 0$, and $f(x) = sin(\frac{\pi}{2}x) \forall x \in [0, 1]$. Note that a function is a concave function if and only if $f\left(\frac{x+y}{2}\right) \geq \frac{f(x)+f(y)}{2}$ for any two points $x$ and $y$ in its domain. For the case shown in Figure 3.1, if $f(x) = \sqrt{x}$ is employed and the values of the Chamfer distance are normalized to be in the range $[0, 1]$ (e.g., divided
by 5.0), then $f(s(A, C)) < f(s(A, B)) + f(s(B, C))$, $f(s(A, B)) < f(s(A, C)) + f(s(B, C))$, and $f(s(B, C)) < f(s(A, B)) + f(s(A, C))$. That is, $f(x) = \sqrt{x}$ transfers the non-metric Chamfer distance measure into the metric one for the case shown in Figure 3.1.

The proposed approach employs the simulated annealing (SA) technique [47] for mapping function optimization. SA was developed to deal with highly nonlinear problems. Informally, we can view this scheme as a bouncing ball. Initially the “temperature” is high and the ball can bounce very high, over any mountain to reach any valley, given enough bounces. As the temperature cools gradually over time, the ball can only bounce lower and it eventually settles to become trapped in a relatively small region of valleys. It has been proven that this strategy can find the global optimum for many different applications. We use this technique to optimize mapping functions as shown in Figure 3.2. For a given candidate mapping function (i.e., the original function), we first uniformly sample $n$ points. Our approach, with a certain probability, moves up or down the $n$ sample points within a user-specified threshold. The probability of such moves is a function of the temperature as well as the
MappingFunctionOptimization($S$, $s$, $f(x)$)

**Input:**
- data set $S$
- similarity measure $s$
- candidate mapping function $f(x)$

**Output:**
- optimized mapping function $f'(x)$

01 $S \leftarrow \text{InitializeState}(f(x))$
02 $T \leftarrow T_0$
03 $i \leftarrow 1$
04 $S_{\text{best}} \leftarrow S$
05 $G_{\text{best}} \leftarrow \text{Goodness}(S, S, s)$
06 **while** $i \leq \text{Iter}_{\text{max}}$ and $\rho$ is large **do**
07 $S' \leftarrow \text{PerturbState}(S)$
08 $\Delta G \leftarrow \text{Goodness}(S', S, s) - \text{Goodness}(S, S, s)$
09 **if** $\Delta G > 0$ **then**
10 $S \leftarrow S'$
11 **else**
12 $S \leftarrow S'$ with probability of $e^{-\Delta G/T}$
13 **endif**
14 **if** $\text{Goodness}(S', S, s) > G_{\text{best}}$ **then**
15 $S_{\text{best}} \leftarrow S'$
16 $G_{\text{best}} \leftarrow \text{Goodness}(S', S, s)$
17 **endif**
18 $i \leftarrow i + 1$
19 $T \leftarrow \alpha \times T$
20 **endo**
21 $f'(x) \leftarrow \text{linear interpolation of } S_{\text{best}}$
22 **return** $f'(x)$

Figure 3.3: Algorithm for mapping function optimization

difference in quality between the new set of points and the previous set. Finally, we can derive the optimized function by linear interpolation of these sampling points.

The detailed procedures are presented in Figure 3.3: MappingFunctionOptimization(). In this algorithm, InitializeState() generates an initial state by uniformly sampling $n$ points of $f(x)$ (in line 1). PerturbState() creates a new state by randomly moving the $n$ sampling points up or down within a user-specified threshold (in line 7). In order to keep $f'(x)$ increasing, we need to make sure that $n$ sampling points in $S_{\text{best}}$ are in increasing order; that is, $f'(x_1) > f'(x_2)$ when $x_1 > x_2$ for all the sampling points. Goodness() evaluates the mapping quality by calculating the intrinsic dimensionality $\rho(S, d) = \frac{n^2}{2\sigma^2}$. The lower
intrinsic dimensionality, the better mapping quality. $S_{\text{best}}$ is used to keep track of the best solution seen so far (in lines 4 and 15). The temperature $T$ is initially set to be high (in line 2), and is decreased by some factor $\alpha$ (in line 19). Typical values of $\alpha$ lie between 0.8 and 0.99, and we choose $\alpha$ to be 0.9. At high temperatures, our approach tends to accept most of the new states (even worse ones), while at low temperatures, the probability of accepting the worse ones becomes low (in line 12). This algorithm terminates when the maximum number of iterations is reached or $\rho$ becomes very small (in line 6). The optimized functions derived by the above algorithm may not be concave, but still have the following desired property.

**Theorem 3.1** Given a similarity measure $s$ and the optimize function $f'(x)$ derived by the above algorithm, the similarity orderings are preserved.

**Proof:** The $n$ sampling points in $S_{\text{best}}$ are obtained by the two functions in the above algorithm: $\text{InitializeState}()$ and $\text{PerturbState}()$. For the first case, $n$ sampling points are in increasing order because the original mapping function $f(x)$ is a increasing function. For the second case, $\text{PerturbState}()$ guarantees that $n$ sampling points are in increasing order as well. For both cases, we linearly interpolate $S_{\text{best}}$ to derive $f'$ that is increasing. As $f'$ is increasing, then $\forall Q, O_i, O_j \in S$ follows that $s(Q, O_i) > s(Q, O_j) \iff f'(s(Q, O_i)) > f'(s(Q, O_j))$. This means that the similarity orderings are preserved after the mapping function $f'$ is employed.

### 3.4 Experiments

In this section, we present the experimental results. All experiments were performed on a 2.5-GHz Pentium IV-based computer with 1GB of RAM. Two data sets were used. The first data set consists of 68040 images from the Corel library, and 64-level gray-scale histograms were extracted from those images. We examined five non-metric similarity measures on the images: the 5-median $L_2$ distance (denoted as $5\text{-medL2}$), the squared $L_2$ distance (denoted
as $L2square$), and two fractional $L_p$ distances (denoted as $FracLp0.25$ for $p = 0.25$ and $FracLp0.75$ for $p = 0.75$). The second data set consists of 100,000 2D synthetic polygons with 5 to 10 vertices. We examined 4 non-metric similarity measures on the polygons: the 3-median and 5-median Hausdorff distances (denoted as $3-medHausdorff$ and $5-medHausdorff$), and the time warping distance with $\delta = L_2$ (denoted as $TimeWarpL2$), and with $\delta = L_\infty$ (denoted as $TimeWarpLmax$) [81]. All the distances were normalized to be in the range of $[0,1]$.

To evaluate the effectiveness of our approach, we compare the performance of our approach against the state-of-one one (i.e., TriGen) [81] in terms of the intrinsic dimensionality $\rho$. For fair comparison, we adopted almost the same setting as in [81], and the candidate mapping function used in our approach is $f(x) = x^{\frac{1}{1+w}}$ where $w > 0$. Figure 3.4 shows that our approach significantly reduces $\rho$ for all the four similarity measures on the Corel image data set: $5-medL2$, $L2square$, $FracLp0.25$ and $FracLp0.75$. The values of $\rho$ are reduced by about 38%, 12%, 23%, and 15%, respectively. Figure 3.5 also illustrates that our
approach reduces $\rho$ for all the other four similarity measures on the synthetic polygon data set: 3-medHausdorff, 5-medHausdorff, TimeWarpL2 and TimeWarpLmax. Specifically, the reduction can be up to 35% for TimeWarpL2. As mentioned before, the smaller $\rho$, the better mapping quality.

The performance gain is due to the fact that TriGen can only yield concave mapping functions, which limits its ability to reduce $\rho$. Our approach, on the other hand, can produce more generalized increasing functions that are optimized with the simulated annealing technique to achieve a smaller $\rho$.

### 3.5 Summary

In this chapter, we propose a simulated-annealing-based technique to derive optimized mapping functions that transfer non-metric measures into metric, and still preserve the original similarity orderings. Then existing metric index structures can be used to speed up similarity
search by exploiting the triangular inequality property. The experimental study shows that the optimized functions can significantly reduce the intrinsic dimensionality, thus improving the retrieval efficiency. Future research may investigate techniques for the improvement of the convergence speed because simulated annealing is a random search technique and the computational cost for the optimized mapping functions may be expensive.
CHAPTER 4: INCREMENTAL QUERY EVALUATION FOR SUPPORT VECTOR MACHINES

Support vector machines (SVMs) have been widely used in multimedia retrieval to learn a concept in order to find the best matches. In such a SVM active learning environment, the system first processes $k$ sampling queries and top-$k$ uncertain queries to select the candidate data items for training. The user’s top-$k$ relevant queries are then evaluated to compute the answer. This approach has shown to be effective. However, it suffers from the scalability problem associated with larger database sizes. To address this limitation, we propose an incremental query evaluation technique for these three types of queries. Based on the observation that most queries are not revised dramatically during the iterative evaluation, the proposed technique reuses the results of previous queries to reduce the computation cost. Furthermore, this technique takes advantage of a tuned index structure to efficiently prune irrelevant data. As a result, only a small portion of the data set needs to be accessed for query processing. This index structure also provides an inexpensive means to process the set of candidates to evaluate the final query result. This technique can work with different kernel functions and kernel parameters. Our experimental results indicate that the proposed technique significantly reduces the overall computation cost, and offers a promising solution to the scalability issue.

4.1 Introduction

Support Vector Machines (SVM) [11,90] have been widely used in various applications such as face detection in computer vision [67], and information retrieval [86,66,27,87,22,68,69,101,48,32,63,88,25,36]. In particularly, to address the semantic gap and the user’s subjectivity in multimedia retrieval, relevance feedback coupled with SVMs is typically used to learn a classifier for each user’s query. For example, in content-based image retrieval (CBIR) systems,
low-level visual image features (e.g., color, texture, and shape) are automatically extracted for image descriptions and indexing purposes. To search for desirable images, a user can mark returned images as positive or negative, which are then fed back into the system to train a SVM classifier. The system returns a set of images based on its best estimate for further feedback. This process is repeated until the user is satisfied with the query result. Such systems have been shown to effective for many practical CBIR applications [66,27,87].

Using SVMs for relevance feedback processing, however, suffers from the scalability problems associated with larger database sizes. This can be explained as follows. SVM is a maximum margin classifier. For the linearly separable case, SVM looks for the hyperplane that separates the relevant training instances from the irrelevant ones with the largest margin. This is illustrated in Figure 4.1, where $W$ is the normal to the hyperplane, the dark points are the relevant training instances, the white points are the irrelevant training instances, and the gray points are the testing instances. Support vectors are those training instances
closest to the separating hyperplane (e.g., Points 3, 4, 7, 8 and 9 in Figure 4.1) [11]. For the non-linear separable case, kernel functions can be used to transform data into a higher dimensional (feature) space where the training instances can be linearly separated. In such a SVM active learning environment, the following three typical queries are involved: $k$ sampling query, top-$k$ uncertain query, and top-$k$ relevant query. Specifically, $k$ sampling query is to randomly retrieve $k$ testing instances for user’s feedback; and it is desired that these sampled instances capture the data distribution well (i.e., they are good representatives). Top-$k$ uncertain query is to retrieve the $k$ testing instances closest to the hyperplane (e.g., Points 5 and 10 in Figure 4.1). Since these instances are considered most uncertain and informative, they are strongly recommended for the next round of feedback. Top-$k$ relevant query is to retrieve the $k$ farthest instances from the hyperplane in the relevant half-plane. These instances are the most relevant instances based on the learned SVM classifier. Since each feedback iteration typically changes both the transformed space and the separating hyperplane, traditional query evaluation techniques [65, 8, 73, 35] cannot be straightforwardly applied. As a consequence, most existing CBIR systems with SVMs [66, 27, 87] have to linearly scan the entire image set to evaluate both top-$k$ uncertain and top-$k$ relevant queries, resulting in the scalability issue for large collections of multimedia data.

To address this scalability issue, we propose an incremental query evaluation technique for the aforementioned three types of queries. These problems have not been well studied in the literature. The contributions of this chapter lie, not only in the individual algorithms implemented, but also at the system level by proposing a novel framework to improve the overall performance of the system. Due to this innovation, the proposed approach can significantly reduce the computation cost for query evaluation, and the performance study confirms that the proposed technique is highly efficient.

The remainder of this chapter is organized as follows. In Section 4.2, we review related work on query evaluation for SVMs. Our incremental query evaluation technique for the three types of queries involved in SVM active learning is introduced in Section 4.3. We
Figure 4.2: KDX

discuss our experimental study in Section 4.4. Finally, we conclude the chapter in Section 4.5.

4.2 Related Work

In this section, we review existing research on answering three aforementioned queries: \( k \) sampling query, top-\( k \) uncertain query, and top-\( k \) relevant query.

To answer a \( k \) sampling query, the acceptance/rejection sampling technique [65] first randomly picks up a path through the index such as R*-tree [6], and then the inclusion probability of an instance from the leaf node is proportional to some weight such as the number of instances in the same node. This technique, however, suffers from disk I/O overhead because it performs an index traversal from the root node to the leaf node for each sampled instance. To address this issue, our technique takes advantage of the proposed index structure to efficiently get auxiliary information (e.g., sampling instances) on the fly, resulting in only one disk page access in most cases.
Traditional top-$k$ queries (i.e., $k$ nearest neighbor queries or similarity queries) have been studied extensively [8, 73, 35]. Roussopoulos et al. [73] proposed an algorithm that retrieves the $k$ nearest neighbors from a multidimensional index by pruning away nodes that cannot lead to the $k$ nearest neighbors. The algorithm was further extended to reduce more unnecessary disk accesses by Hjaltason and Samet [35]. In addition, there are many research efforts in support top-$k$ queries in spatio-temporal databases and streaming databases. Those traditional top-$k$ query evaluation technique assume that the query concept can be represented as a simple query point in a metric space while the query concept is represented as a hyperplane in the transformed space in SVM active learning. Therefore, existing techniques can not be directly applied to top-$k$ uncertain and top-$k$ relevant queries.

To facilitate the evaluation of top-$k$ relevant queries, Panda and Chang proposed the kernel indexer (KDX) [69]. By exploiting geometrical properties of the SVM, KDX (see Figure 4.2) is constructed as follows: 1) find the instance $\phi(x_c)$ that is the approximate center in the transformed space, where $\phi$ maps $x_c$ from the original space into the transformed space (i.e., the feature space); 2) separate the instances into equi-count rings based on their angular distance from $\phi(x_c)$ (e.g., Figure 4.2 shows four rings, each containing the same number of instances); 3) construct an intra-ring indexer (i.e., a distance matrix) for each ring; and finally 4) create an inter-ring index. To answer a top-$k$ relevant query, KDX performs inter-ring and intra-ring pruning to retrieve the result set. Specifically, KDX first identifies the farthest ring (e.g., Ring 2 in Figure 4.2) from the hyperplane, calculates the candidate set in this ring, and then moves to the next ring, and so forth, until the remaining rings can be safely pruned. KDX, however, can not be easily extended to support top-$k$ uncertain queries, and may suffer from the dimensionality curse especially when the kernel function incurs the dimensionality of the transformed space to be infinite. To address these limitations, Panda and Chang mapped top-$k$ uncertain and top-$k$ relevant queries into range queries in the original space allowing for the reuse of existing index structures [68]. This approach obtains the approximate results. Our query processing technique (see Section 4.3)
is inspired by this work, and is more efficient by reusing the results of previous queries instead of query evaluation from scratch, optimizing the underlining index structures, estimating the bounding box without expensive SVM clustering, and proposing faster query evaluation for range queries. Experimental results in Section 4.4 show that our approach achieves significant savings in terms of disk I/O costs and execution time compared to the above approach.

4.3 The Proposed Incremental Query Evaluation Technique

In this section, we discuss the proposed query evaluation technique. Specifically, we first briefly introduce our approach in Section 4.3.1. Then we discuss in details our incremental query evaluation technique in Section 4.3.2.

4.3.1 Overview of Our Approach

We focus on the two-class SVM active learning in this chapter. Given a data set $X$ that consists of vectors in a metric space $M$. Among $X$, the training data set is denoted as
$X_t = \{x_{t,1}, \cdots, x_{t,n}\}$ with the corresponding labels $Y_t = \{y_1, \cdots, y_n\}$, where $y_i \in \{-1, 1\}$.
The testing data set is denoted as $X_u = \{x_{u,1}, \cdots, x_{u,m}\}$, and $X_u = X \setminus X_t$. During the query-concept learning phase, SVM typically transfers $X_t$ from $M$ into a feature space $F$, and derives a hyperplane separating the relevant training instances (i.e., those with the label 1) from irrelevant ones (i.e., those with the label -1), and achieving the largest margin [11]. The weights $W = \{\alpha_1, \cdots, \alpha_n\}$ associated with $X_t$ are determined accordingly. Those testing instances with $\alpha_i > 0$ are called support vectors, and they are in fact the closest points to the hyperplane. The class membership of a testing instance $x_{u,j}$ can be predicted by the following function: $S(x_{u,j}) = \sum_{i=1}^{n} \alpha_i y_i K(x_{t,i}, x_{u,j}) + b$, where $K$ is a kernel function. If $S(x_{u,j}) \geq 0$, $x_{u,j}$ is classified as +1; otherwise, -1. In fact, a top-$k$ relevant query is to retrieve $k$ instances with the largest values of $S$ in $X$, and a top-$k$ uncertain query is to retrieve the $k$ instances with the smallest absolute values of $S$ in $X_u$.

Different kernel functions can implicitly project $X$ into different feature spaces. A hyperplane in $F$ typically corresponds to a complex nonlinear decision boundary in the original space $M$. Mathematically, $K(x, x') = \langle \phi(x), \phi(x') \rangle$, where $\phi$ is the function to map instances from $M$ into $F$. With the kernel trick, we can use $K$ to compute the similarity between two instances in $M$ instead of explicitly using $\phi$. Some widely-used kernel functions are as follows: Polynomial kernel $K(x, x') = (x \cdot x' + 1)^d$, Gaussian kernel $K(x, x') = \exp \left( -\frac{\|x-x'\|^2}{2\sigma^2} \right)$, and Laplacian kernel $K(x, x') = \exp(-\gamma\|x-x'\|^2)$.

In such a SVM active learning environment, $k$ sampling query and top-$k$ uncertain query are used to retrieve instances in $X_u$ for labeling, and top-$k$ relevant query is used to retrieve the most relevant instances based on the learned query concept to the user. In order to achieve fast evaluation of these three types of queries, we construct and tune a multidimensional index in the original space $M$ as discussed below. Then we transform both top-$k$ uncertain query and top-$k$ relevant query into range queries in $M$ by determining relevant regions, and approximating these regions with bounding boxes (see Figure 4.3). We take advantage of our index structure (see Section 6.2) to efficiently evaluate $k$ sampling queries.
and range queries. After obtaining the candidates, we can calculate the corresponding values of \( S \), and return the approximate results for both top-\( k \) uncertain and top-\( k \) relevant queries. More importantly, based on the observation that most queries are not revised dramatically during the iterative evaluation, the proposed technique reuses the results of previous queries to reduce the computation cost, as explained in Section 4.3.2.

### 4.3.2 Efficient Query Evaluation

We discuss our query processing technique \textsc{EvaluateQuery}(Q) on top of the tuned index structure (see Section 6.2) for the three types of queries (i.e., \( k \) sampling queries, top-\( k \) uncertain query, and top-\( k \) relevant queries) used in SVM active learning. The query cost is the sum of disk seek (including cylinder seek and rotation), data transfer and CPU time, in which seek time dominates the total query cost. Figure 4.4 lays out our query processing technique, designed to minimize the disk I/O cost.

For \( k \) sampling queries, we just need to retrieve the root node (in line 1) that contains all possible image IDs. If \( k \) is relatively small, we sample instances from \( k \) different leaf nodes in order to make the sampled instances as representative as possible (in line 6). Otherwise, random sampling can be performed to retrieve the query result (in line 8).

For top \( k \) uncertain queries, we first determine the positive and negative support vectors in \( \mathbb{X}_u \) (in line 12). Specifically, the positive support vectors are those testing instances with \( \alpha_i > 0 \) and \( y_i = 1 \) while the positive support vectors are those with \( \alpha_i > 0 \) and \( y_i = -1 \). Top \( k \) uncertain query aims to retrieve the \( k \) instances in \( \mathbb{X}_u \) closest to the hyperplane. These instances typically lie between positive support vectors and negative support vectors. Therefore, a bounding box \( B \) that covers both positive and negative support vectors has a high probability to cover the desired query result. After deriving \( B \), we can partition \( B \) into multiple ranges (in line 14) to eliminate the empty space by some partitioning strategies, such as Equi-Count, Equi-Area, Min-Skew and Min-Overlap. The Equi-Count partitioning
**EVALUATEQUERY**(*Q*)

**Input:**
- given query *Q*
- training dataset *X_t*
- training data labels *Y_t*
- SVM weights *W*
- kernel function *K*

**Output:**
- query result *S_k*

01 read the root node of our index structure
02  *S_k* ← ∅
03  *X_u* ← *X* \ *X_t*
04 if *Q* is a *k* sampling query then
05     if *k* is relatively small then
06         *imageIDset* ← *k* image IDs in *X_u* such that the difference between any two IDs is larger than the node capacity
07     else
08         *imageIDset* ← randomly-sampled *k* IDs in *X_u*
09     endif
10  *S_k* ← retrieve images whose ID ∈ *imageIDset*
11 elseif *Q* is a top-*k* uncertain query then
12     determine the positive and negative support vectors based on *Y_t* and *W*
13     estimate the bounding box *B* for this query
14     decompose *B* into a set of ranges *R*
15     *R_new* ← *R* \ *R_prev* (ranges within *R* of previous queries)
16     *S* ← **EVALUATEANGEQUERY**(*R_new*)
17     *S_k* ← *k* instances with the smallest absolute values of *S* in (*S* ∪ *S_prev*) \ *X_u*.
18     else /* *Q* is a top-*k* relevant query */
19     determine relevant instances based on *Y_t* and *W*
20     derive the minimum bounding box *B* of those relevant instances
21     expand *B*
22     decompose *B* into a set of range queries *R'*
23     *R'_new* ← *R'* \ *R'_prev* (ranges within *R'* of previous queries)
24     *S'* ← **EVALUATEANGEQUERY**(*R'*)
25     *S_k* ← *k* instances with the largest values of *S* in *S'* ∪ *S'_prev*
26 endif
27 return *S_k*

Figure 4.4: SVM Query processing technique
strategy creates ranges containing roughly the same number of instances. The Equi-Area partitioning strategy creates ranges having the same area. The Min-Skew partitioning strategy divides $B$ into ranges such that each range contains uniformly distributed instances. The Min-Overlap partitioning strategy creates ranges that have minimal overlaps among them. We can adopt any of the above partitioning strategies. In our experiments, we have implemented the Equi-Area partitioning strategy. After a set of ranges is determined, we can avoid some unnecessary range queries by eliminating those ranges within the ranges of previous queries (in line 15). Such incremental strategy works because of the observation that most queries are not revised dramatically during the iterative evaluation. Of course, this strategy incurs memory overhead because we need to buffer the previous results in memory. Considering the performance gain shown in Section 4.4, this overhead is still acceptable. Then, EvaluateRangeQuery() is called to perform multiple range queries to get the results $S$ (in line 16). Finally, we obtain the query result $S_k$ by calculating the values of $S(x_{u,j}) = \sum_{i=1}^{n} \alpha_i y_i K(x_{t,i}, x_{u,j}) + b$ for $\forall x_{u,j} \in (S \cup S_{prev}) \cap X_u$, and sorting these values to get $k$ instances with the smallest absolute values (in line 17).

To evaluate top $k$ relevant queries, we first determine the relevant instances (i.e., those training instances with $y_i = 1$) (in line 19). Then we derive the minimum bounding box $B$ for these relevant instances. Because top $k$ relevant query is to retrieve $k$ farthest instances to the hyperplane in the relevant half-plane, $B$ is needed to expand to cover these instances in some cases. One way is to expand $B$ at a given ratio, and another way is to use sampling instances to estimate the desired range. After expanding $B$, we can partition $B$ into multiple ranges, and achieve incremental query evaluation by removing some ranges within the ranges of previous queries (in line 23) similarly as for top $k$ uncertain queries. Then, we call EvaluateRangeQuery() to perform multiple range queries. Finally, query result $S_k$ is determined by calculating the values of $S(x_{u,j}) = \sum_{i=1}^{n} \alpha_i y_i K(x_{t,i}, x_{u,j}) + b$ for $\forall x \in S' \cup S'_{prev}$, and sorting these values to obtain $k$ instances with the largest values (in line 25).
EVALUATE\textsc{RangeQuery}(Q)

\textbf{Input:}
\begin{itemize}
  \item a set of ranges $\mathcal{R}$
\end{itemize}

\textbf{Output:}
\begin{itemize}
  \item query result $S$
\end{itemize}

\begin{algorithmic}
  \STATE $S \leftarrow \emptyset$
  \STATE $Queue \leftarrow \text{NEWQueue}()$
  \STATE $\text{ENQUEUE}(Queue, \text{root})$
  \WHILE {$Queue$ is not empty}
    \STATE $Element \leftarrow \text{DEQUEUE}(Queue)$
    \IF {$Element$ is a leaf node}
      \FOR {each $Instance$ in leaf node $Element$}
        \IF {$Instance$ is within any $r \in \mathcal{R}$}
          \STATE $S \leftarrow S \cup \{Instance\}$
        \ENDIF
      \ENDFOR
    \ELSE // $Element$ is a non-leaf node
      \FOR {each $Child$ node of node $Element$}
        \IF {$Child$ overlaps or contains with any $r \in \mathcal{R}$}
          \STATE $\text{ENQUEUE}(Queue, Child)$
        \ENDIF
      \ENDFOR
    \ENDIF
  \ENDWHILE
  \RETURN $S$
\end{algorithmic}

Figure 4.5: Algorithm for multiple range queries

In addition, the algorithm for multiple range queries (i.e., \textsc{EvaluateRangeQuery}(), see Figure 4.5) has some important features. Firstly, this algorithm accesses each node at most once for multiple ranges by performing the breadth-first search to traverse the proposed index structure. Secondly, such breath-first search can take advantage of our index structure to achieve sequential access during the search process, thus reducing the disk seek time significantly. Thirdly, this algorithm achieves optimality as explained below.

\textbf{Definition 4.1} \textit{Optimality}

An algorithm for multiple range queries is optimal if and only if it retrieves only once the disk nodes that overlap or contain at least one of the regions.

\textbf{Theorem 4.1} The proposed algorithm shown in Figure 4.5 is optimal.
Proof: We prove this by two steps. We first prove that each disk node in the proposed index structure is accessed at most once. In the proposed algorithm, we perform a breath-first search to traverse the proposed index structure, which guarantees that each disk node is accessed at most once. Then we prove that only the disk nodes that overlap or contain at least one of the regions are accessed. In our proposed index structure, a parent node contains all its child nodes. Therefore, if a parent node does not overlap or contain any region, all its child nodes does not either. That is, if the MBB of a node does not overlap or contain any region, the proposed algorithm does not insert it into the queue (see line 14 in Figure 4.5), avoiding unnecessary disk accesses of this node and its children. Hence, the proposed algorithm retrieves only once the disk nodes that overlap or contain at least one of the regions; that is, the proposed algorithm is optimal according to the definition of Optimality.

4.4 Experiments

In this section, we evaluate the effectiveness of the proposed query processing technique described in Section 4.3. Our dataset consists of more than 68,040 images from the COREL library. There are a total of 37 visual image features divided into three main groups: colors (9 features) [85], texture (10 features) [83], and edge structure (18 features) [101]. The COREL images have been classified into distinct categories by domain professionals, and each category contains about 100 images. For each chosen category, 50% of images were used as the training data. We chose LIBSVM [15] with the Gaussian kernel for SVM learning. The node size of the original R*-tree and our index structure were both set to 4KB, and both had three levels in our experimental settings. Following the suggestion of the R*-tree [6], the minimum utilization parameter of each node was set to 40% and the reinsert fraction parameter was set to 30% for all index structures. We compare the performance of the
proposed query processing technique on top of our index structure (denoted as \textit{TNEW}) against the existing technique with R*-tree (denoted as \textit{TOLD}). Specifically, our methods to evaluate $k$ sampling queries, top-$k$ uncertain queries, top-$k$ relevant queries are from line 4 to 10, 11 to 16, and 17 to 23 in Figure 4.4, respectively. The existing counterparts are proposed in [65], [68], and [68], respectively. As mentioned in Section 4.3, the new index structure is introduced specifically for our query processing technique. To be fair, we also utilize an index structure (i.e., R*-tree) for the existing counterparts. It is not our claim that the proposed index structure is better than R*-tree. Actually, we claim that our query processing technique coupled with the proposed index structure outperforms existing ones with R*-tree. The experiments were performed on a 3.4-GHz Pentium IV-based computer with 1.5GBytes of RAM, and the results are averaged over 100 runs.

Figures 4.6 and 4.7 show the recall of top-$k$ uncertain queries and top-$k$ relevant queries with different $k \in \{5, 15, 20, 25, 35, 50\}$, respectively. The precision of top-$k$ uncertain queries and top-$k$ relevant queries with different $k \in \{5, 15, 20, 25, 35, 50\}$ have the same figures as
Figure 4.7: Recall for top-$k$ relevant queries

Figure 4.8: Disk accesses of $k$ sampling queries
Figure 4.9: Disk accesses of top-$k$ uncertain queries

Figure 4.10: Disk accesses of top-$k$ relevant queries
the recall of their counterparts because here we assume the number of retrieved images (i.e., $k$) is equal to the number of relevant ones. Note that the proposed technique retrieves the approximate results, and the actual results are determined by calculating the values of $S$ for all instances in $\mathbb{X}$. These two figures show that the recall increases as $k$ increases, and the recall can be improved further by expanding the bounding box. However, the larger expansion, the more data are retrieved and the more computation cost is needed to determine the final results. For top-\(k\) uncertain queries, the recall without expansion is acceptable because the recall is about 0.72 when $k = 5$, and 0.82 when $k = 50$. This indicates that a bounding box $B$ that covers both positive and negative support vectors has a high probability to cover the desired query result. For top-\(k\) relevant queries, the recall is about 0.85 for $k = 50$ even after 15% expansion. This means that the originally estimated bounding box covering all relevant instances in the training dataset is either too tight or not accurate. To achieve a better recall, we need to expand $B$ more or use other sampling instances to estimate a more accurate bounding box. \(T_{NEW}\) can make the sampling instances very
representative as explained in Section 4.3.2, therefore it is desired to use both $k$ sampling queries and top-$k$ uncertain queries to retrieve the candidate training data for the next round feedback.

Figures 4.8 to 4.10 show that $TNEW$ significantly outperforms $TOLD$ for answering three types of queries in terms of disk accesses, and the performance gap widens as $k$ increases. Clearly, the total number of disk access increases as $k$ increases. As shown in Figure 4.8, $TOLD$ performs about 5 times more disk accesses than $TNEW$ when $k = 5$, 58 times when $k = 25$, and 120 times when $k = 50$ for $k$ sampling queries. This figure shows that $TNEW$ is independent of the number of sample points (i.e., $k$) because $TNEW$ just needs to access the root node of our index structure, resulting in only one disk access for answering a sampling query. On the other hand, $TOLD$ is proportional to $k$, which is because $TOLD$ has to traverse the R*-tree to obtain sample points one by one, incurring almost 3 disk accesses per sample point. If a sampling point is not met the criteria, another traversal is needed. For top-$k$ uncertain queries, $TOLD$ performs about twice disk accesses compared to $TNEW$ when $k = 5$, and about third times when $k = 50$ (see Figure 4.9). For top $k$ relevant queries, $TNEW$ performs up to about 3 times better (see Figure 4.10). Figure 4.11 illustrates the efficiency of the proposed algorithm for multiple range queries in terms of execution time. $TNEW$ can be up to about 4 times faster than $TOLD$. More importantly, the curve of $TNEW$ is quite low and flat, indicating that it can support a large number of range queries simultaneously. The performance gain is due to the fact that the proposed algorithm for multiple range queries minimizes disk accesses, and achieves sequential access to reduce the number of expensive random disk accesses, thus reducing the execution time significantly. The performance difference between $TNEW$ and $TOLD$ confirms that the proposed query processing technique reduce the disk I/O cost and execution time significantly by incremental query evaluation, taking advantage of our index structure, and evaluating efficiently those three types of queries.
4.5 Summary

Although support vector machines have been shown to be effective for multimedia retrieval, it suffers from the scalability problems associated with larger database sizes. This important limitation is addressed in this chapter by proposing a highly efficient query evaluation technique for SVMs. Taking advantage of an index structure tuned for better data clustering, the proposed technique answers $k$ sampling queries on the fly, transforms top-$k$ uncertain queries and top-$k$ relevant queries into range queries in the original space, and then evaluates these range queries efficiently. More importantly, by reusing the results of previous queries, the proposed technique can save query evaluation cost much more. This approach is not affected by the changes of kernels and kernel parameters of SVMs. The experimental results indicate that our approach significantly reduces the computation time and the number of disk accesses for query evaluation.
CHAPTER 5: HANDLE LOCAL OPTIMUM TRAPS

Existing CBIR systems, designed around query refinement based on relevance feedback, suffer from local optimum traps. That is, when the user is examining a relevant cluster surrounded by less relevant images, essentially the same set of images will be returned for the user to provide relevance feedback. Since the user would select the same query images again, the relevance feedback process gets trapped in a local optimum. This local-optimum trap problem may severely impair the overall retrieval performance of today’s CBIR systems. In this chapter, we therefore propose a simulated annealing-based approach to address this important issue. When a stuck-at-a-local-optimum occurs, we employ a neighborhood search technique (i.e., simulated annealing) to escape from the local optimum. We also propose an index structure to speed up such neighborhood search. Our experimental study confirms that our approach can efficiently address the local-optimum trap problem, and therefore can improve the effectiveness of existing CBIR systems.

5.1 Introduction

Content-based image retrieval (CBIR) has received a great deal of attention with many CBIR systems developed [27]. Nearest neighbor search is a popular technique for most of today’s CBIR systems. This is due to its simplicity and effectiveness with the following desirable properties: 1) $k$-nearest neighbor ($k$-NN) classifier scales well with the number of classes while other techniques (e.g., support vector machines [90]) can not straightforwardly support an arbitrary number of classes; 2) it can model complex and non-parametric distribution; and 3) its classification accuracy becomes asymptotically optimal as the training size approaches infinity [20]. In order to address the semantic gap and the subjectivity of human perception problems in CBIR, relevance feedback has been adopted. This scheme interacts with the
user. In each round, the user helps by identifying the relevant images within the set of images retrieved in the previous round. The system then utilizes this feedback to modify the current $k$-NN query (i.e., query point movement techniques discussed in Section 1.2) and thus to improve its retrieval results in the next round. This process is repeated until the user is satisfied with the results.

This scheme, however, can become stuck at local optima. Since query points in relevance feedback systems have to move through many regions before reaching the most relevant images the user would like to retrieve, it is possible that they get trapped in one of these regions. Figure 5.1 illustrates a possible scenario, where Cluster 2 contains the most relevant images. As a result of a 3-NN search at a starting query point $p_s$, the system returns points $p_1$ and $p_2$, in addition to $p_s$. Since both $p_1$ and $p_2$ are relevant, the refined query point $p_r$ is their centroid and the anchor of the next 3-NN search. However, the system will retrieve exactly the same set. In other words, the system can never get out of this region because the retrieval set is saturated with the $k$ checked images. Although, the system can escape with a larger $k$, it is difficult to guess a proper threshold (up to $k = 14$ in this example). Consequently, the user might not even know a local optimum trap is occurring.

Although this local-optimum trap problem may severely impair the overall retrieval performance of today’s CBIR systems, it has received rather limited attention. In this chapter,
we therefore propose a simulated annealing-based approach to address this important issue. The remainder of this chapter is organized as follows. Section 5.2 presents our simulated annealing-based approach. Section 5.3 describes our empirical results. Finally, Section 5.4 concludes the chapter with directions for future work.

5.2 The Proposed Approach

To address the local-optimum trap problem, our approach employs a probabilistic neighborhood search technique (i.e., simulated annealing) to escape from local optima. However, simulated annealing is a random search technique and it may take many iterations to converge. We need to leverage user’s relevance feedback to substantially reduce the number of iterations, and speed up the neighborhood search at each iteration. Therefore, we also propose an index structure, as discussed below, to reduce the search time.

5.2.1 Index Structure

Our index structure (see Figure 5.2) is constructed in three stages as follows:

**Distance Embedding:** BoostMap [1] is chosen for the distance embedding. Given an application-dependent distance measure that can be metric or non-metric, we first create a large pool of simple 1D embeddings by picking any image feature vector in the image database as a reference object. Then we start with weak classifiers corresponding to those 1D embeddings, and combine those classifiers into a single, optimized classifier using AdaBoost [77]. Finally we convert the optimized classifier into a multidimensional embedding (for more details refer to [1]). With BoostMap, all image feature vectors are offline embedded into a Euclidean space, in which measuring Euclidean distances is very cheap. Moreover, after a given expensive distance measure is turned into a Euclidean one (i.e., satisfying the triangular inequality property), existing multidimensional index structures [8] can be directly used to further reduce CPU and disk I/O overhead for k-NN evaluation.
Figure 5.2: Our new index structure

Hierarchical Clustering: A hierarchical clustering technique, similar to the R*-tree [6], is used to organize the entire image database into a hierarchical tree structure. As each node in this hierarchy represents a cluster, we extend the original node structure of the R*-tree to include also some auxiliary information, as discussed below, to facilitate query processing and simulated annealing. Without loss of generality, we select the R*-tree for our study because it is well known and has been widely used in practice while other hierarchical clustering techniques can be used as well [8].

Information Augmenting: We traverse the tree in a postorder fashion. In an original R*-tree, an internal node contains an array of node entries. Each node entry is a pair \((m.bb, node-id)\), where \(m.bb\) is the minimum bounding box (MBB) that spatially contains the MBBs in the child node, with \(node-id\) as the child node address. In our index structure, each node entry is extended to be a tuple \((m.bb, node-id, imageID-range, relevant-nodes)\), where \(imageID-range\) refers to the range of image identifications contained in the pointed child node and \(imageID-range \subseteq [1, |S|]\) where \(|S|\) is the cardinality of the whole image database, and \(relevant-nodes\) contains a set of relevant nodes on the same level. Let us describe how to build the structure illustrated in Figure 5.2. We start from the root node (i.e., Node 1) which has three node entries. We first visit the first node entry which points to Node 2. Node 2 has two node entries, pointing to leaf Nodes 5 and 6 in order. Our depth-first traversal leads
us to Node 5, which contains 3 image points. Then we set the imageID-range of the first entry in Node 2 to $[1, 3]$. We of course need to build up a one-one mapping between the image IDs and exact image names, such as by hashing, building a B+-tree index on the image ID field, or simply changing the image names to their corresponding image IDs as in our current implementation. Similarly, we set the imageID-range of the second entry in Node 2 to $[4, 6]$. As Node 2 doesn’t have any more node entries, we track back to Node 1 and set imageID-range to encompass the ranges of all its children, which is $[1, 3] \cup [4, 6] = [1, 6]$.

The above procedure is repeated for the second entry of Node 1. The imageID-range values in different internal node entries are shown in Figure 5.2. relevant-nodes aims to capture the semantic linkage between nodes, and can be initialized to be a set of neighboring nodes on the same level. For example, the first entry in Node 2 will be set to $\{6, 7, 8\}$, indicating that Nodes 6, 7 and 8 are relevant to Node 5. Later on, we can update relevant-nodes by mining association rules [31] among nodes from the log data of users’ relevance feedback.

When new images are inserted, the structure has to be rebuilt. Because image databases are fairly static [27], the reconstruction is still acceptable considering the performance gains (shown in Section 5.3) we obtain.

### 5.2.2 Simulated Annealing-based Approach

The proposed approach is inspired by the simulated annealing (SA) technique for optimization [47]. SA was developed to deal with highly nonlinear problems. Informally, we can view this scheme as a bouncing ball. Initially the “temperature” is high and the ball can bounce very high, over any mountain to reach any valley, given enough bounces. As the temperature cools gradually over time, the ball can only bounce lower and it eventually settles to become trapped in a relatively small region of valleys. It has been proven that this strategy can find the global optimum for many different applications.
SIMULATED ANNEALING($S, Q$)

**Input:**
- data set $S$
- a query $Q$

**Output:**
- candidate images $S_{best}$

01 $S \leftarrow \text{INITIALIZESTATE}(S, Q)$
02 $P_k \leftarrow \text{SAMPLENODES}(S, k)$
03 Present images in $P_k$ for the user’s feedback
04 $T \leftarrow T_0$
05 $i \leftarrow 1$
06 $S_{best} \leftarrow S$
07 $G_{best} \leftarrow \text{GOODNESS}(S)$
08 while $i \leq \text{Iter}_{max}$ and and the user is not satisfied do
09 $S' \leftarrow \text{PERTURBSTATE}(S)$
10 $P_k \leftarrow \text{SAMPLENODES}(S', k)$
11 Present images in $P_k$ for the user’s feedback
12 $\Delta G \leftarrow \text{GOODNESS}(S') - \text{GOODNESS}(S)$
13 if $\Delta G > 0$ then
14 $S \leftarrow S'$
15 else
16 $S \leftarrow S'$ with probability of $e^{-\Delta G/T}$
17 endif
18 if $\text{GOODNESS}(S') > G_{best}$ then
19 $S_{best} \leftarrow S'$
20 $G_{best} \leftarrow \text{GOODNESS}(S')$
21 endif
22 $i \leftarrow i + 1$
23 $T \leftarrow \alpha * T$
24 enddo
25 return $S_{best}$

Figure 5.3: Algorithm for handling local optimum traps

In the CBIR environment, we use the same technique to avoid becoming stuck at a local optimum. When the feedback process seems to become trapped in a neighborhood of the embedding Euclidean space, our approach, with a certain probability, considers a random nearby neighborhood to continue the search for possibly additional matching images. The probability of continuing the search is a function of the temperature as well as the difference in quality between the new set of query images and the previous set. The detailed procedures are presented in Figure 5.3. Specifically, \textsc{InitializeState}() generates an initial state, by taking into account relevant nodes of those containing the current query images (in line
1). And the relevant nodes can be obtained from relevant-nodes in the node entries (or even recursively through their parent node entries to ensure that every node is reachable) of the above index structure. SampleNodes() randomly samples $k$ images from nodes in the current state by consulting imageID-range in the corresponding node entries (see lines 2 and 10). PerturbState() creates a new state by randomly swapping some nodes from the set of relevant nodes (in line 9). Goodness() calculates the quality of the retrieval results based on the user’s relevance feedback. $S_{\text{best}}$ is used to keep track of the best solution seen so far (in lines 6 and 19). The temperature $T$ is initially set to be high (in line 4), and is decreased by some factor $\alpha$ (in line 23). Typical values of $\alpha$ lie between 0.8 and 0.99, and we choose $\alpha$ to be 0.9. At high temperatures, our approach tends to accept most of the new states (even worse ones) hoping to get out of the local optimum traps, while at low temperatures, the probability of accepting the worse ones becomes low (in line 16). This algorithm terminates when the maximum number of iterations is reached or the user is satisfied with the results (in line 8), and then the normal image retrieval is continued.

5.3 Experiments

In this section, we present our experimental results. All experiments were performed on a 2.5-GHz Pentium IV-based computer with 1GB of RAM. The node size of our index structure and the R*-tree were both set to 4KB, and both had three levels in our experimental settings. Our dataset consists of more than 68,040 images from the Corel library, these images have already been classified into semantic categories by domain professionals, and each category contains about 100 images. A total of 128 image features divided into three groups were used: color histogram (100 features), texture (10 features), and edge structure (18 features). We employed the technique in [33] to learn an expensive distance measure for image retrieval, and then used BoostMap [1] to transfer this distance measure to a Euclidean one; that is, those 128-dimensional image feature vectors were embedded into a 30-dimensional Euclidean
space. Relevance feedback was simulated similarly as in [14]: images from the same category are automatically considered most relevant, and images from related categories (e.g., tigers and horses) are considered relevant. All the measurements averaged 100 queries.

To illustrate the local optimum trap problem with existing approaches, we demonstrate that MARS [74], Query Expansion [14] and Qcluster [46] have high ratios to be stuck at local optima for small $k$ (see Figure 5.4). Even for a fairly large $k$, the ratios still remain relatively high. For example, when $k = 100$, MARS’s ratio is about 20%; both Qcluster’s and Query Expansion’s exceed 25% with Qcluster’s being slightly better. As a result, the overall retrieval performance of these techniques will be severely impaired.

To show that our SA-based approach can escape from local optimum traps and improve the performance of existing CBIR systems, we randomly simulated 100 scenarios where local optimum traps happened, and chose Qcluster as an example. The experimental evaluation illustrates that our approach helps to improve Qcluster’s precision and recall by about 11% (see Figure 5.5) and 9% (see Figure 5.6), respectively with $k = 100$. Although there is sudden
Figure 5.5: Precision ($k = 100$)

Figure 5.6: Recall ($k = 100$)
degradations in both precision and recall at iterations 4 and 6, our SA-based approach still achieves fast convergence. This is because that INITIALIZEState() and SAMPLENodes() can help choose good candidates for SA.

To evaluate the effectiveness of our index structure, we compare the performance of our query processing techniques (e.g., SAMPLENodes() and PerturbState() in Figure 5.3) with our index structure (denoted as QNEW) against the ones with R*-tree (denoted as QOLD). Figure 5.7 shows that QNEW significantly outperforms QOLD in terms of the number of disk accesses. Specifically, QOLD requires about 8 times more disk accesses than QNEW for the first iteration, and the performance gap widens as the number of iterations increases when $k$ is set to 50. The larger $k$ is, the more we can save. The reason is that QOLD has to traverse the R*-tree to obtain sample points one by one, incurring almost 3 disk accesses per sample point. QNEW, on the other hand, saves disk I/Os by only accessing the auxiliary information (i.e., imageID-range and relevant-nodes) in internal nodes of our index structure.
5.4 Summary

In this chapter, we proposed an SA-based approach to address the local-optimum trap problem in CBIR systems. Our experimental study shows that our approach can achieve fast convergence, improve the performance of existing CBIR systems, and significantly reduce disk I/O overhead as well. Future research may investigate techniques to further improve the convergence speed. For instance, we will exploit the log data of users’ relevance feedback to predict the relevant nodes or even target images. We will also investigate a new way to interact with the user based upon our current prototype. In particular, when the system tries to escape from a local optimum trap, the user might experience a sudden degradation in the quality of the images returned for relevance feedback. We need a user interface that would communicate that this phenomenon is positive normal behavior, and not the result of a design flaw.
CHAPTER 6: SUPPORT CONCURRENT ACCESSES

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 chapter, we propose a generalized query model suitable for expressing queries of different types, and investigate efficient processing techniques for this new framework. We exploit sequential access and data sharing by developing new storage and query processing techniques to leverage inter-query concurrency. Our experimental results, based on the Corel dataset, indicate that the proposed optimization can significantly reduce average response time in a multiuser environment, and achieve better retrieval precision and recall compared to two recent techniques.

6.1 Introduction

Content-based image retrieval (CBIR) has received a great deal of attention with many systems implemented [38, 91, 2, 13, 19, 24, 46, 66, 80, 89, 95, 7, 37, 39, 74, 56, 84, 93, 99]. They are effective for many practical applications [82, 27].

There are two general types of search in CBIR: target search and category search [27]. The goal of target search is to find a specific (target) image (e.g., a registered logo, a historical photograph, or a painting). The goal of category search is to retrieve a particular semantic class or genre of images (e.g., rose images or skyscrapers). To bridge the semantic gap between the descriptive limitations of low-level features and human perception of similarity, many CBIR systems utilize relevance feedback to allow the user to mark some of the returned images as positive or negative. These user inputs are fed back into the system as a refined...
query for the next round of retrieval. This process is repeated until the user is satisfied with
the query results.

In order to support target search and category search, many techniques have been de-
developed for different specific 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 [38, 56, 14, 19, 24, 40, 46, 66]. In this chapter, we 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 lever-
age sequential data access and I/O sharing among concurrent queries. Thus, the original
problems addressed in this chapter are twofold:

(i) a generalized model for various types of CBIR queries, and

(ii) 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 numbers
of concurrent users.

Our experimental results, based on the Corel dataset, confirm that existing query pro-
cessing techniques aimed at reducing the cost of individual query independently, are not
scalable to support a multiuser environment. The performance study indicates that the pro-
posed framework achieves better retrieval effectiveness than those of two recent techniques
in terms of precision and recall, and can significantly improve average response time in a
multiuser environment. A preliminary version of this study was presented in [51]. This chap-
ter introduces index tuning and query optimization techniques, and presents more extensive
experiments.

The remainder of this chapter is organized as follows. The proposed framework is pre-
sented in Section 6.2 in detail, including the generalized query model, an index structure and
query processing techniques. Our performance study is discussed in Section 6.3. Finally, we conclude this chapter in Section 6.4.

### 6.2 The Proposed Framework

In this section, we discuss the proposed framework. As illustrated in Figure 6.1, this framework has a 4-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, 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
interface layer and query processing layer, chooses the best search algorithm based on the user’s goal (target search or category search). The query processing layer, including generic query model, query processing algorithm 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 structure and tuning those structures.

As discussed in Section 1.2, much work has been done for the interface layer [38,56,19,24] and search algorithm layer [38,56,14,19,24,40,46,66], but less work has been done for the last two layers (i.e., query processing layer and indexing layer). In this chapter, we therefore focus on the last two layers. Specifically, we will discuss in detail our query model, an index structure and our query optimization techniques.

### 6.2.1 Generic Query Model

In our query model, a user query is defined as

\[ Q = \{Q_i \mid i \in \{1, \ldots n\} \}, \quad (\text{Eq. 6.1}) \]

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

\[ Q_i = \langle n_{Q_i}, P_{Q_i}, W_{Q_i}, D_{Q_i}, S_{Q_i}, k_{Q_i} \rangle, \quad (\text{Eq. 6.2}) \]

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 \( Q_i \), \( S_{Q_i} \) 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. Unlike queries in relational database systems, the user in most cases can not specify an ideal query to retrieve the desired result in multimedia database systems, and has to rely on iterative feedback to refine his query. According to user’s feedback, various relevant feedback techniques (i.e., search algorithms) have been proposed to automatically determine \( n_{Q_i} \) and
$P_Q$, and adjust $W_Q$ and $D_Q$ for better retrieval effectiveness, which we have discussed in Section 1.2.

Now, we illustrate how to use this model to represent the six typical types of queries in CBIR systems:

- Sampling queries [38, 56, 14, 19, 24, 40, 46, 66] refer to randomly retrieving a specified number (denoted as $k$) of images from the whole image database. For a sampling query, we set $n = 1$, $n_Q = 0$, $k_Q = k$, and $S_{Q_i} = S$, where $S$ is the whole search space (i.e., the whole image database).

- Constrained sampling queries [56] refer to randomly retrieving a specified number of images within a constrained search space instead of the whole search space. For a constrained sampling query, we set $n = 1$, $n_Q = 0$ and $k_Q = k$, which signify that this query is to randomly retrieve $k$ points in $S_{Q_i}$.

- Multiple constrained sampling queries [38] refer to randomly retrieving a specified number of images within multiple constrained search spaces instead of a single constrained search space. For a multiple constrained sampling query, we set $n > 1$, $n_Q = 0$ and $k_Q = k/n$ (suppose $k = cn$, where $c$ is a natural number).

- $k$-NN queries refer to retrieving a specified number of images most similar to some given query images in the whole image database. If a query is a $k$-NN query with single-point movement techniques [40, 66], we set $n = 1$, $n_Q = 1$, $k_Q = k$, and $S_{Q_i} = S$; for a $k$-NN query with multiple-point movement techniques [14, 46], $n_Q$ is set differently (i.e., $n_Q > 1$).

- Constrained $k$-NN queries refer to retrieving a specified number of images most similar to some given query images within a constrained search space instead of the whole search space. If a query is a constrained $k$-NN query [56] with single-point movement
GLOBALDIVIDECONQUER\((S, k)\)

**Input:**
the whole image database \(S\)
number of retrieved images with a query \(k\)

**Output:**
target image \(p_t\)

01 \(Q_1 \leftarrow \langle n_{Q_1}, P_{Q_1}, W_{Q_1}, D_{Q_1}, S_{Q_1}, k_{Q_1} \rangle\), where \(n_{Q_1} \leftarrow 0, S_{Q_1} \leftarrow S\) and \(k_{Q_1} \leftarrow k\)
02 \(Q \leftarrow \{Q_i \mid i \in \{1, \ldots n\}\}\), where \(n \leftarrow 1\)
03 \(R \leftarrow \text{EVALUATEQUERY}\(Q\) /* \(Q\) is a random sampling query */
04 \(V R_i \leftarrow \text{the minimum bounding box of } S\)
05 **while** user does not find \(p_t\) in \(R\) according to the user’s relevance feedback **do**
06 \(p_i \leftarrow \text{the most relevant point } \in R\)
07 construct a Voronoi diagram \(VD\) inside \(VR_i\) using points in \(R\) as Voronoi seeds
08 \(VR_i \leftarrow \text{the Voronoi cell region associated with the Voronoi seed } p_i\)
in \(VD\)
09 \(S_{Q_1} \leftarrow \text{such points } \in S\ \text{that are inside } VR_i\ \text{except } p_i\)
10 \(Q_1 \leftarrow \langle n_{Q_1}, P_{Q_1}, W_{Q_1}, D_{Q_1}, S_{Q_1}, k_{Q_1} \rangle\), where \(n_{Q_1} \leftarrow 0\) and \(k_{Q_1} \leftarrow k\)
11 \(Q \leftarrow \{Q_i \mid i \in \{1, \ldots n\}\}\), where \(n \leftarrow 1\)
12 \(R \leftarrow \text{EVALUATEQUERY}\(Q\) /* \(Q\) is a constrained random sampling query */
13 **endo**
14 **return** \(p_t\)

Figure 6.2: Global Divide and Conquer Algorithm for Target Search

techniques, we set \(n = 1, n_{Q_i} = 1\) and \(k_{Q_i} = k\) while for a constrained \(k\)-NN query with multiple-point movement techniques, \(n_{Q_i} > 1\).

- Multiple localized \(k\)-NN queries [38] refer to retrieving a specified number of images most similar to some given query images within multiple constrained search spaces instead of a single constrained search. For a multiple localized \(k\)-NN query, we set \(n > 1, n_{Q_i} = 1\) and \(k_{Q_i} = k/n\).

In our generic model, we consider various types of queries. \(S_{Q_i}\) is also included to account for the dynamic change of search space, which may be reduced after each feedback iteration [38, 56]. Note that the proposed query model is to make the query interface more
**QUERY DECOMPOSITION** \((\mathbb{S}, k)\)

**Input:**
- the whole image database \(\mathbb{S}\)
- number of retrieved images with a query \(k\)

**Output:**
- category images \(R\)

01 \(I \leftarrow \emptyset\)
02 \(Q_{1} \leftarrow (n_{Q_{1}}, P_{Q_{1}}, W_{Q_{1}}, D_{Q_{1}}, S_{Q_{1}}, k_{Q_{1}}), \) where \(n_{Q_{1}} \leftarrow 0, S_{Q_{1}} \leftarrow \) representative images in root node and \(k_{Q_{1}} \leftarrow k\)
03 \(Q \leftarrow \{Q_{i} \mid i \in \{1, \ldots, n\}\}, \) where \(n \leftarrow 1\)
04 \(R \leftarrow \text{EVALUATEQUERY}(Q) */Q\ is a constrained random sampling query */\)
05 \(I \leftarrow I \cup \) relevant images \(\in R\) picked by user
06 \(h \leftarrow 2\)
07 **while** \(h < \) the height of the index structure **do**
08 \(N \leftarrow \) the nodes in the level \(h \) which contain at least one representative image in \(I\)
09 **for** each node \(N_{i} \in N** do
10 \(Q_{i} \leftarrow (0, P_{Q_{i}}, W_{Q_{i}}, D_{Q_{i}}, S_{Q_{i}}, k_{Q_{i}}), \) where \(S_{Q_{i}} \leftarrow \) representative images in \(N_{i}\) and \(k_{Q_{i}} \leftarrow k/|N|\)
11 **enddo**
12 \(Q \leftarrow \{Q_{i} \mid i \in \{1, \ldots, n\}\}, \) where \(n \leftarrow |N|\)
13 \(R \leftarrow \text{EVALUATEQUERY}(Q) */Q\ is a multiple constrained sampling query */\)
14 \(I \leftarrow I \cup \) relevant images \(\in R\) picked by user
15 \(h \leftarrow h + 1\)
16 **enddo**
17 **for** each image \(I_{i} \in I** do
18 \(Q_{i} \leftarrow (1, P_{Q_{i}}, W_{Q_{i}}, D_{Q_{i}}, S_{Q_{i}}, k/|I|), \) where \(S_{Q_{i}} \leftarrow \) the leaf node containing \(I_{i}\) and its sibling nodes
19 **enddo**
20 \(Q \leftarrow \{Q_{i} \mid i \in \{1, \ldots, n\}\}, \) where \(n \leftarrow |I|\)
21 \(R \leftarrow \text{EVALUATEQUERY}(Q) */Q\ is a multiple localized \(k\)-NN query */\)
22 **return** \(R\)

Figure 6.3: Query Decomposition Algorithm for Category Search

expressive, enabling the proposed framework to formulate a variety of queries. The detailed implementations of how to evaluate such queries depend on the query optimization module.

We present two state-of-art algorithms for target search and category search, respectively, using our query model (see Figures 6.2 and 6.3). Existing target search techniques re-retrieve previously examined images (i.e., those retrieved in the previous iterations) when they again fall within the search range of the current iteration. This strategy leads to two major
disadvantages: 1) no guarantee that the target can be found because the search activity might get trapped in a region during the iterative process, and 2) slow convergence. To address the above limitations, GDC employs Voronoi diagrams to aggressively prune the search space and move towards the target image, thus significantly speeding up the convergence. As shown in Figure 6.2, GDC prunes the search space (see line 8) by employing Voronoi diagrams, and a constrained random sampling query is evaluated (see line 12) in the while loop (from line 5 to 13). This process is repeated until the target is located. As proved in [56], the worst case for GDC is bounded by $O(\log_k |S|)$, indicating that GDC achieves fast convergence. In addition, most existing category search techniques with relevance feedback confines the search result to a single neighborhood in the feature space. Unfortunately, no visual-based feature vector is sufficient to facilitate perfect semantic clustering, and semantically similar images with different appearances are always clustered into distinct neighborhoods in the feature space. Confinement of the search results to a single neighborhood is an inherent limitation of exiting techniques. To address this limitation, we proposed the Query Decomposition technique [38] to facilitate retrieval of semantically similar images from multiple neighborhoods in the feature space. As shown in Figure 6.3, the relevant clusters are derived based on the user’s relevance feedback (see lines 14 and 18), and the retrieval results are the $k$ most similar images from those relevant clusters via a multiple localized $k$-NN query (see line 21). As shown in Figures 6.2 and 6.3, it is the system that automatically decides which type of queries to use, and calculates the corresponding internal parameters while the user is required to provide feedback to the system either by marking the retrieved images as relevant or irrelevant, or via explicit weight modification of the retrieved images.
6.2.2 Index Tuning and Query Optimization Techniques

The purpose of our index structure is to facilitate efficient query evaluation and improve retrieval effectiveness as well for both target search and category search. Our index structure (see Figure 6.4) is constructed in two stages as follows: Hierarchical Clustering, and Information Augmenting (see Section 2.3.1).

To achieve efficient query evaluation and support concurrent queries in our framework, we propose three techniques; namely, index 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 (i.e., do not change often). The design of most existing hierarchical index structures (e.g., R*-tree) usually overlooks the differences between sequential and random accesses. 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. The query cost is the sum of disk seek (including cylinder seek and rotation), data transfer and CPU time. Due to the mechanical limitations of the disk head, seek time is usually an order of magnitude more expensive than transfer time and CPU time, and therefore dominates the total query cost. Hence, multiple random disk accesses are generally much more expensive than being able to retrieve the desired disk pages in one sequential access as in our design. To reduce the number of disk random accesses, we use the Hilbert curve [75] for disk page allocation. Hilbert curve is a space filling technique which maps a multidimensional data space into a one-dimensional data space; that is, it defines a linear order to visit every disk page in the multidimensional space exactly once. 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
GROUPACCESS(L)

**Input:**
node ID list $L$

**Output:**
relevant node list $N$

01 Remove duplicate node IDs in $L$
02 Sort all node IDs in $L$ according to their disk locations in ascending order
03 $i \leftarrow 1$
04 $j \leftarrow 1$
05 INSERTLIST($G_1$, $n_1$) // Put the first node ID in $L$ into the first disk access group $G_1$
06 while $i < L$.size do
07 if $n_{i+1}$ is not close to $n_i$ according to their disk locations then
08 $j \leftarrow j + 1$ // create a new disk access group
09 endif
10 INSERTLIST($G_j$, $n_{i+1}$)
11 $i \leftarrow i + 1$
12 enddo
13 $N \leftarrow \emptyset$
14 for each disk access group $G_k$ in \{ $G_1, \cdots, G_j$ \} do
15 Perform a sequential disk access from the first node ID to the last node ID in $G_k$
16 INSERTLIST($N$, only the relevant nodes)
17 enddo
18 return $N$

Figure 6.7: Group Access Algorithm

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. For example, for the children nodes 7, 8 and 9 of node 3 in Figure 6.4, the physical disk address order is nodes 7, 9 and 8 according to their Hilbert values (see Figure 6.5). Compared to the possible original disk allocation (see Figure 6.6a), the final disk allocation of the tuned index structure is: Node 1, Node 2, Node 3, Node 4, Node 5, Node 6, Node 7, Node 9, Node 8, Node 12, Node 11, and Node 10 (see Figure 6.6b). We also need to change node-id in each original node entry ($mbb$, node-id, imageID-range) accordingly.
**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 disk seeks at the cost a few extra cheap page transfer. We illustrate the advantage of this optimization with an example as follows. Consider two concurrent queries \( q_1 \) and \( q_2 \). \( q_1 \) requires nodes \( N_5, N_6, N_7 \) and \( N_8 \); and \( q_2 \) requires nodes \( N_7 \) and \( N_8 \). If we process them independently, it would incur six disk seeks or four disk seeks (\( N_7 \) and \( N_8 \) can be accessed once when caching is used, see Figure 6.6a). Group access, benefiting from the above index tuning, can reduce this cost significantly by retrieving the relevant nodes for the two queries together in only one sequential access (i.e., one disk seek): \([N_5, N_6, N_7, N_9, N_8]\) with the additional node \( N_9 \) (see Figure 6.6b). The detailed group access algorithm is illustrated in Figure 6.7. First, duplicate node IDs in \( L \) are removed in line 1, and the unique node IDs are sorted according to their disk positions in line 2. Then node IDs are divided into several disk access groups from line 6 to line 12 according to their locations with each other. For each group, a sequential access is performed. Of course, for each query in a multiuser environment, the server maintains an in-memory list of relevant node IDs. After retrieving relevant nodes for all concurrent queries during some time interval using the group access algorithm, the server can determine the relevant nodes for each query. Then, the server’s query processing layer can evaluate each query and produce the query results.

**Individual Query Optimization:** We discuss our query optimization strategies for six typical queries as follows.

- Sampling queries. We just need to retrieve the root node of our tuned index structure. The root node contains all possible image IDs, and a random sampling can be
performed. If $k$ is relatively small, we can try to sample those images from $k$ different leaf nodes in order to make the sampled images as representative as possible.

- Constrained sampling queries. We use $S_{Q_1}$ as the range, then perform a range query on our tuned index structure except all the leaf nodes to collect sampling images. The range query is implemented by using the breadth-first traversal to facilitate our group access algorithm because sibling nodes in our tuned index structure are physically consecutive. If $k$ is relatively large, we can try to sample all the leaf nodes related to $Q$ in order to make the sampled images more representative.

- Multiple constrained sampling queries. Similar to the above constrained sampling queries. All multiple constrained queries are performed on the same level of our tuned index structure, and all these queries can be answered simultaneously by using our group access algorithm.

- $k$-NN queries. 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 such $k$-NN queries when our group access algorithm is employed. 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.

- Constrained $k$-NN queries. Similar to $k$-NN queries except that we need to prune the nodes which are outside of $S_{Q_1}$.

- Multiple localized $k$-NN queries. Each localized $k$-NN is performed on the relevant 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.
The proposed query optimization techniques aim to reduce overheads of CPU and disk I/O by taking advantage of the proposed index structure to efficiently get auxiliary information (e.g., sampling points) on the fly, reusing the visited nodes in the previous iterations, pruning non-relevant nodes as early as possible, performing sequential disk accesses if applicable to reduce expensive random disk accesses, and leveraging computation sharing in inter-query concurrency. The novelty and contributions of this chapter lie not only in the individuals algorithms implemented, but also at the system level by proposing a novel framework to improve the whole system performance.

6.3 Experiments

In this section, we present our experimental results, involving target search (see Section 2.4), category search and query optimization. All experiments were performed on a 2.5-GHz Pentium IV-based computer with 1GB of RAM. For index structures, large node size reduces the number of disk pages accessed while incurring more CPU cost for query evaluation. On the other hand, small node size reduces the CPU cost, but increases the number of disk pages accessed for query evaluation [6]. As a result, a trade-off has to be made between the CPU cost and disk I/O cost. In our experiments, the node size of our tuned index structure and the R*-tree were both set to 4KB (based on empirical studies in [8,14]), and both had three levels in our experimental settings.

6.3.1 Category Search

In order to evaluate the performance of our proposed framework for category search, we extended our previous prototype accordingly. For comparison fairness, we adopted the same setting as in [38]: the test database includes 15,000 images taken from the Corel image database with a few hundreds new images we created to test the capability of the proposed techniques in handling the semantic gap in CBIR. The Corel images have been classified
Table 6.1: Precision Comparison (without our index structure vs. with our index structure)

<table>
<thead>
<tr>
<th>Query</th>
<th>Results in [38]</th>
<th>Results with our index structure</th>
</tr>
</thead>
<tbody>
<tr>
<td>A person (Hair-model, fitness, Kongfu)</td>
<td>0.81</td>
<td>0.85</td>
</tr>
<tr>
<td>Airplane (single, multiple)</td>
<td>0.85</td>
<td>0.91</td>
</tr>
<tr>
<td>Bird (eagle, owl, sparrow)</td>
<td>0.61</td>
<td>0.75</td>
</tr>
<tr>
<td>Car (modern sedan, antique car, steamed car)</td>
<td>0.85</td>
<td>0.88</td>
</tr>
<tr>
<td>Horse (polo, wild horse, race)</td>
<td>0.72</td>
<td>0.80</td>
</tr>
<tr>
<td>Mountain view (snow, with water)</td>
<td>0.46</td>
<td>0.65</td>
</tr>
<tr>
<td>Rose (yellow, red)</td>
<td>0.71</td>
<td>0.75</td>
</tr>
<tr>
<td>Water sports (surfing, sailing)</td>
<td>0.44</td>
<td>0.76</td>
</tr>
<tr>
<td>Computer (server, desktop, laptop)</td>
<td>0.86</td>
<td>0.90</td>
</tr>
<tr>
<td>Personal computer (desktop, laptop)</td>
<td>0.69</td>
<td>0.81</td>
</tr>
<tr>
<td>Laptop (with clear background, with</td>
<td>0.71</td>
<td>0.82</td>
</tr>
<tr>
<td>complicated background)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>0.70</td>
<td>0.81</td>
</tr>
</tbody>
</table>

into distinct categories by domain professionals. We also classified the additional images we created for this experimental study into the appropriate Corel categories. This is because Corel image database does not perfectly match our needs, therefore we added a few hundreds new images such as images of server, desktop and laptop (those are semantically similar images but with very different appearances). Since users search for images based on high level semantic concepts (as opposed to low level image features), we used the Corel category information as the ground truth in our experiments. To test our system’s capability of bridging the semantic gap, the eleven queries in Table 1 are carefully designed to investigate the impact on performance under both general (e.g., ”finding computers”) and more specific (e.g., ”finding laptop computers”) queries, and to make the evaluation extensive. The average results over all eleven test queries are presented at the bottom of Table 1.

The results, shown in Table 1, illustrate that using our index structure we can improve the average precision by 11%. For some queries (e.g., finding mountain view and water
sports), our index structure can help to improve the precision even more (i.e., by about 20%). Again, this is because our index structure can facilitate sampling as many relevant leaf nodes as possible, and help the user choose more representative images to capture the high level semantic concepts in Query Decomposition. The experimental results in [38] demonstrate that our Query Decomposition (QD) technique significantly outperforms the existing counterpart—Multiple Viewpoints [26] (interesting readers can refer to [38] for details). However, the main purpose of the experiments in this section is to show that the proposed index structure can help to improve the performance of QD, instead of showing the superiority of QD over other methods. Therefore, we do not include the comparison results with other methods.

### 6.3.2 Query Optimization

In this section, we evaluate the effectiveness of the proposed query optimization techniques described in Section 6.2. We compare the performance of the proposed technique with all
Figure 6.9: Constrained sampling queries

Figure 6.10: Multiple constrained sampling queries
Figure 6.11: \(k\)-NN queries in Qcluster

Figure 6.12: Constrained \(k\)-NN queries
the optimization features (denoted as $QOP$) with that of a similar technique with R*-tree and none of the optimization features (denoted as $QNO$). The experiments were based on all the typical types of queries discussed in Section 6.2, except sampling queries because even a large number of concurrent sampling queries can be answered very quickly by $QOP$. In this study, $k$ was set to 50, all queries were randomly generated, and relevance feedback was simulated similarly as in Section 2.4.1. For constrained queries, $S_{Q_i}$ was randomly chosen up to 10% of $S$. The dataset and image features were the same as in Section 2.4.1. The results are averaged over 100 runs.

Figures 6.9 to 6.13 show the effect of increasing the number of concurrent queries from 1000 to 10000 for five types of queries, respectively. Clearly, the total processing time increases as the number of concurrent queries increases. All five figures illustrate that $QOP$ significantly outperforms $QNO$, and the performance gap widens with the increases in the number of queries. Specifically, for constrained sampling queries (see Figure 6.9), $QOP$ is about 2 times faster than $QNO$ when the number of queries is 1000, and about 5 times faster when the number of queries is 10000. More importantly, the curve of $QOP$ is quite low and flat, indicating that it can support a large number of concurrent queries simultaneously. For
multiple constrained sampling queries (see Figure 6.10), \textit{QOP} performs up to 4 times faster. For \(k\)-NN queries in Qcluster (see Figure 6.11), \textit{QOP} performs up to 6 times faster compared to \textit{QNO}. The relatively larger savings in this case are due to the fact that Qcluster queries can benefit more from information generated during the previous iterations. For constrained \(k\)-NN queries (see Figure 6.12), \textit{QOP} performs up to 5 times faster; for multiple localized \(k\)-NN queries (see Figure 6.13), it performs up to 4 times faster.

The experimental results shows that \textit{QNO} is not scalable since its query processing time increases rapidly as the number of concurrent queries increases, while \textit{QOP} exhibits a very slow increasing rate for all five types of queries. The performance difference between \textit{QNO} and \textit{QOP} confirms that the proposed query optimization techniques can substantially improve performance and enhance system scalability to support a large user community, by taking advantage of the proposed index structure to efficiently get auxiliary information (e.g., sampling points) on the fly, reusing the visited nodes in the previous iterations, pruning non-relevant nodes as early as possible, performing sequential disk accesses if applicable to reduce expensive random disk accesses, and leveraging computation sharing in inter-query concurrency.

### 6.4 Summary

In this chapter, we presented a unified framework for processing generalized queries including various types of target search and category search queries. To the best of our knowledge, such a general model has not been studied in the literature. In terms of execution efficiency, we introduced query processing techniques to allow computation sharing among concurrent queries in a multiuser environment. Our experimental study, based on the Corel dataset, indicates that our system prototype provides significant savings in average response time while achieving better precision and recall, and is scalable to support a large user community. This latter performance characteristic is largely neglected in current systems making them
less suitable for large-scale deployment. With the growing interest in Internet-scale image search applications, our framework offers an effective solution for the scalability problem.
CHAPTER 7: CONCLUSION

In this dissertation, we propose a generic framework, including a query model, index structures, and query optimization techniques for efficient relevance feedback processing. Specifically, this dissertation has five main contributions as follows.

The first contribution is an efficient target search technique [56, 55, 57]. We propose four target search methods: naïve random scan (NRS), local neighboring movement (LNM), neighboring divide-and-conquer (NDC), and global divide-and-conquer (GDC) methods. All these methods are built around a common strategy: they do not retrieve checked images (i.e., shrink the search space). Furthermore, NDC and GDC exploit Voronoi diagrams to aggressively prune the search space and move towards target images. We theoretically and experimentally prove that the convergence speeds of GDC and NDC are much faster than those of NRS and recent methods.

The second contribution is a method to reduce the number of expensive distance computation when answering \(k\)-NN queries with non-metric distance measures [50]. We propose an efficient distance mapping function that transfers non-metric measures into metric, and still preserves the original distance orderings. Then existing metric index structures (e.g., M-tree) can be used to reduce the computational cost by exploiting the triangular inequality property.

The third contribution is an incremental query processing technique for Support Vector Machines (SVMs) [49]. SVMs have been widely used in multimedia retrieval to learn a concept in order to find the best matches. SVMs, however, suffer from the scalability problem associated with larger database sizes. To address this limitation, we propose an efficient query evaluation technique by employing incremental update. The proposed technique also takes advantage of a tuned index structure to efficiently prune irrelevant data. As a result, only a small portion of the data set needs to be accessed for query processing. This index
structure also provides an inexpensive means to process the set of candidates to evaluate the final query result. This technique can work with different kernel functions and kernel parameters.

The fourth contribution is a method to avoid local optimum traps [52]. Existing CBIR systems, designed around query refinement based on relevance feedback, suffer from local optimum traps that may severely impair the overall retrieval performance. We therefore propose a simulated annealing-based approach to address this important issue. When a stuck-at-a-local-optimum occurs, we employ a neighborhood search technique (i.e., simulated annealing) to continue the search for additional matching images, thus escaping from the local optimum. We also propose an index structure to speed up such neighborhood search.

Finally, the fifth contribution is a generic framework to support concurrent accesses [51, 53, 58, 54, 59]. 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, indicate that the proposed optimization can significantly reduce average response time while achieving better precision and recall, and is scalable to support a large user community. This latter performance characteristic is largely neglected in existing systems making them less suitable for large-scale deployment. With the growing interest in Internet-scale image search applications, our framework offers an effective solution to the scalability problem.

I outline below two interesting research directions for future work:

• Medical Data Management System for Pervasive Healthcare. With the advances in pervasive computing technologies, pervasive healthcare will become a reality sooner or later. Pervasive healthcare facilitates to offer a wide range of medical services, and improves healthcare quality while it raises many challenges for the management of large volume of complex medical data that consist of patient’s textual medical history, 2D and 3D medical images, surgical videos, and time-series data (e.g., pulse rate).
will investigate the technique to capture high-quality surgical videos, to analyze the video contents (e.g., to automatically detect what operations have been performed, and to measure the performance quality), and to construct indexing for efficient retrieval. Another research direction is to extend our work for efficient evaluation of $k$-NN, location-based and context-aware queries with friendly user interfaces and real-time response requirements. For example, when performing the surgery, a surgeon can just use gesture or voice to immediately access all the required information such as the patient’s related medical history, blood pressure with various resolutions, the nearest hospital having matched organs, and advice from other experts.

- 3-D Medical Imaging Visualization and Retrieval System. We have studied the problem of developing Internet-based interactive applications of high-resolution 3-D medical image data. We proposed an innovative storage and communication framework, and our experimental results indicate that this framework enables real-time interaction with remote high-resolution 3-D medical images. Future research may investigate more effective and intuitive user interfaces. Another possible extension is to study how to partition data and how to allocate these partitions among servers to achieve better performance for very large high-resolution 3-D medical image data. Furthermore, I will examine the research problems in developing a 3-D medical imaging retrieval system, such as how to efficiently identify salient objects in medical images, how to derive a good distance measure to improve retrieval accuracy, and how to answer concurrent queries efficiently in Internet-scale applications.
REFERENCES


[12] Y. Cai, K. A. Hua, and G. Cao. Processing Range-Monitoring Queries on Hetero-
gegeneous Mobile Objects. In International Conference on Mobile Data Management

using expectation-maximization and its application to image querying. IEEE Trans-

queries in top-k retrieval systems. IEEE Transactions on Knowledge and Data Engi-

URL:http://www.csie.ntu.edu.tw/~cjlin/libsvm/.

Query System for Internet Databases. In Proceedings of the ACM SIGMOD Interna-

search in metric spaces. In Proceedings of the International Conference on Very Large


