Batch Nearest Neighbor Search for Video Retrieval

Jie Shao, Zi Huang, Heng Tao Shen, Xiaofang Zhou, Ee-Peng Lim, and Yijun Li

EDICS: 4-KEEP

***Abstract*—To retrieve similar videos to a query clip from a large database, each video is often represented by a sequence of high-dimensional feature vectors. Typically, given a query video containing m feature vectors, an independent Nearest Neighbor (NN) search for each feature vector is often first performed. After completing all the NN searches, an overall similarity is then computed, i.e., a single content-based video retrieval usually involves m individual NN searches. Since normally nearby feature vectors in a video are similar, a large number of expensive random disk accesses are expected to repeatedly occur, which crucially affects the overall query performance. Batch Nearest Neighbor (BNN) search is stated as a batch operation that performs a number of individual NN searches. This paper presents a novel approach towards efficient high-dimensional BNN search called Dynamic Query Ordering (DQO) for advanced optimizations of both I/O and CPU costs. Observing the overlapped candidates (or search space) of a pervious query may help to further reduce the candidate sets of subsequent queries, DQO aims at progressively finding a query order such that the common candidates among queries are fully utilized to maximally reduce the total number of candidates. Modelling the candidate set relationship of queries by a Candidate Overlapping Graph (COG), DQO iteratively selects the next query to be executed based on its estimated pruning power to the rest of queries with the dynamically updated COG. Extensive experiments are conducted on real video datasets and show the significance of our BNN query processing strategy.**

***Index Terms*—Content-based retrieval, multimedia databases, high-dimensional indexing, query processing**

I. Introduction

Recently with the rapid increase of both centralized video archives and distributed video resources on WWW, the research on Content-based Video Retrieval (CBVR) has become very active. As shown in Fig. 1, besides the database and Graphic User Interface (GUI), a generic CBVR system contains three major modules: *Video Segmentation & Feature Extraction*, *Feature Vector Organization* and *Video Search Engine*. Videos have to be pre-processed to realize the functionality of retrieval. A video is first divided into a number of elementary segments, and the feature of each segment is then extracted [1]. Videos can have both spatial and temporal features. Spatial features include still image features, such as color, shape, texture, etc. Temporal features consist of object motion, camera operation, etc. Normally, all the features are represented by high-dimensional vectors. In short, each video is translated to a sequence of high-dimensional feature vectors in *Video Segmentation & Feature Extraction*. For a large video database, scanning on all these vectors is strongly undesirable due to the high complexity of video features. By video summarization [2], [3], the complexity can be reduced to a level that can be efficiently managed. In *Feature Vector Organization*, compact video representations are organized in a way such that fast retrieval is feasible. *Video Search Engine* searches the underlying indexing structure with some high-dimensional access method [4]–[11] to speed up the query processing.

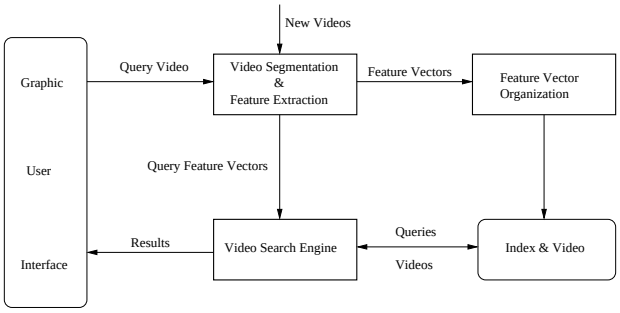


Fig. 1. A generic CBVR system architecture.

In conventional content-based similarity search, a query consumes a single NN search by traversing the indexing structure once. However, a distinguishing characteristic of video retrieval is that, each video is described by a sequence of feature vectors, so as to the query. Denote a query clip as $Q = \{q_1, q_2, \ldots, q_m\}$ and a database video as $P = \{p_1, p_2, \ldots, p_n\}$, i.e., Q and P have m and n feature vectors (or representatives, if some summarization is applied), to identify whether P is similar to Q or contains Q, typically for each $q_i \in Q$, a search is first performed in P to retrieve the similar feature vectors to q_i. A typical video similarity measure is to compute the percentage of similar feature vectors shared by two videos [2], [3], [12], [13]. Given Q and P, their similarity is defined as:

$$Sim(Q, P) = \frac{\sum_{i=1}^{m} T(q_i)}{m}, \tag{1}$$

where $T(q_i) = 1$ if q_i is relevant to some $p_j \in P$ and $T(q_i) = 0$ otherwise. Retrieval of similar feature vectors is processed as a range or kNN search in high-dimensional space. The answers of all query vectors are then integrated to determine the final result.

Now the problem comes on the way: totally the similarity search has to be performed for m time since there are m feature vectors in Q. For large video databases, due to the large number of feature vectors and the high dimensionality, it is a great challenge. We address this problem as Batch Nearest Neighbor (BNN) search, which is defined as *a batch operation that can efficiently perform a number of individual NN searches on the same database simultaneously*[1]. Content-based video retrieval is one of its typical applications. Given a query clip, a series of separate NN searches will incur lots of expensive I/O cost for random disk accesses and CPU cost for distance computations thus most existing CBVR systems are constrained by testing on small video databases. The efficiency of CBVR systems can be enhanced with a BNN solver as shown in Fig. 2. We assume that a maximal number of m queries can be fit in

Jie Shao is with School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane QLD 4072 Australia. Email: jshao@itee.uq.edu.au. Tel: 61-733651153.

Zi Huang is with School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane QLD 4072 Australia. Email: huang@itee.uq.edu.au. Tel: 61-733653476. Fax: 61-733654999.

Heng Tao Shen is with School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane QLD 4072 Australia. Email: shenht@itee.uq.edu.au. Tel: 61-733658359. Fax: 61-733654999.

Xiaofang Zhou is with School of Information Technology and Electrical Engineering, The University of Queensland, Brisbane QLD 4072 Australia. Email: zxf@itee.uq.edu.au. Tel: 61-733653248. Fax: 61-733654999.

Ee-Peng Lim is with Division of Information Systems, School of Computer Engineering, Nanyang Technological University, Singapore. Email: aseplim@ntu.edu.sg. Tel: 65-67904802. Fax: 65-67926559.

Yijun Li is with Nielsen Media Research, Australia. Email: yijun.li@acnielsen.com.au.

[1]For the purpose of easy illustration, we use NN search (k=1 in kNN search) for discussion. The extension to kNN search is straightforward.

main memory and processed in a batch. At current state, BNN search is studied when the underlying access mechanism is not equipped with parallel processing capability.

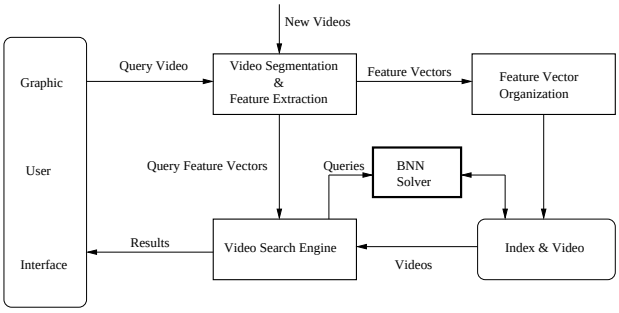


Fig. 2. A generic CBVR system architecture enhanced with BNN solver.

Towards effective and efficient manipulations of very large video databases, we first present a generalized query processing strategy called Sharing Access (SA), which can be deployed with any arbitrary underlying access method to eliminate repeated random disk accesses. Reduced I/O operations are achieved by merging the candidates (or search spaces) of all queries together. More importantly, we show that benefit from a previous query, the pruning conditions of subsequent queries may become even tighter. Therefore, a more sophisticated technique called Dynamic Query Ordering (DQO) is introduced to exploit the common candidates among queries to perform BNN search in an order such that the total number of unnecessary candidate accesses is further reduced maximally in the batch. By a dynamically updated Candidate Overlapping Graph (COG), DQO iteratively selects a query which has the maximal estimated pruning power to the rest to execute next. The property of triangle inequality is also utilized to reduce some unnecessary distance computations for CPU optimization.

A preliminary version of this work appears in [14] as a poster paper, where we present the basic idea of BNN search. In this paper, we mainly make the following additional contributions:

- We summarize the existing proposals of optimizing a number of similarity searches simultaneously based on different access methods, and consider them in a uniform manner since their intrinsics are actually same.
- Besides identify pruning condition tightening, the opportunity to further reduce the total number of candidates in BNN search, we use formal proofs to justify progressively finding a query order to fully utilize it, and illustrate the CPU optimization as well.
- We report a more extensive set of experimental studies on the datasets generated from a large collection of real videos, and results demonstrate the effectiveness of proposed DQO.

The rest of paper is organized as follows. Section II reviews some related work, and Section III provides the preliminaries of our proposal. DQO algorithm is introduced in Section IV, followed by extensive performance studies in Section V. Finally, we conclude in Section VI.

II. Related Work

Conventionally, NN query refers to find the nearest data point to a query under a distance function. It is a primary operation for content-based similarity search that has been studied as one of the major topics for multimedia information retrieval. Existing high-dimensional access methods[2] can be roughly classified into categories: scan-based indexing such as VA-file [5] and LPC-file [6], tree-based indexing such as X-tree [7] and iDistance [8], and hybrid structure by utilizing the advantages of both scan-based and tree-based approaches such as IQ-tree [9], GC-tree [10], LDC-tree [11], etc. VA-file is particularly important to our work since we mainly adopt it to illustrate our ideas. It divides the data space into 2^b rectangular cells where b denotes a user-specified number of bits (typically 4-8 bits per dimension). VA-file allocates a unique bit-string of length b for each cell and approximates data point that falls into the cell by a bit-string. VA-file itself is simply an array of these compact geometric approximations. NN search is performed by scanning the approximation file. With the lower bound and upper bound distances of point approximation, majority of points are excluded (filtering step). The idea behind is if the lower bound distance of a point is already greater than the NN upper bound determined so far, this point can be filtered safely. Otherwise, it is considered as a candidate. All these candidates are then fetched by random disk accesses to check their actual distances (refinement step). Some improvement of VA-file, such as VA$^+$-file [18], has also been proposed to handle non-uniform and clustered datasets. LPC-file [6] is also proposed to better locate data points in vector approximation approach by only adding a small amount of polar coordinate information which is independent of the dimensionality, instead of increasing the number of bits b. Both of these techniques are orthogonal to our approach and can be employed to further reduce the query cost.

Complementary to exact NN query, ε-NN query [12], [19], [20] returns a point whose distance to query is no more than $(1+\varepsilon)$ times of exact NN distance to achieve a tradeoff between accuracy and efficiency. Recently, some variant NN query types have also been studied in database literature. *Reverse NN* query [21] retrieves the point whose NN is the query point. As data tend to be more dynamic, moving queries in the environment of dynamic data have also been investigated. *Continuous NN* query [22] is to continuously query the NN results until a future time. Particularly, the style of moving queries issued over stationary objects, e.g., continuously searching the nearest gas stations when a driver is moving the car, is somewhat similar to BNN query. However, in continuous NN query the trajectory of moving object is constrained, e.g., all the query points are restricted in a line segment. *Group NN* query [23] deals with a query with several points. It finds a database point with the smallest *sum* of distances to all the points in the given query set, e.g., finding a most representative frame that has the smallest sum of differences to all the frames of query clip. *Aggregate NN* query [24] is another query type that has been further relaxed to support other aggregate functions such as *max* or *min*. It is worthwhile to point out that, in BNN query, each point in query set independently looks for its own exact NN, which is fundamentally different from aggregations of multiple query results to determine the top k answers [25]. In spirit, the goal of BNN search is similar to kNN join processing [26], i.e., finding the NN (or kNN) for each point in one set from another set. However, kNN join deals with two large datasets which cannot fit into main memory, while in BNN search the query set is usually small, e.g., the query video clip is relatively short. If the query clip is represented by 1,000 frames, and the feature dimensionality used is 64, since each dimension is represented by a float number of 4 bytes, the size of feature vectors to be loaded for query video is 64×4×1000 bytes = 256 KB, so all the query points can resident in main memory pretty easily. Moreover,

[2]Here we focus on indexing in vector space since it contains more information and therefore allows a better structuring of data. Only when objects cannot be mapped into feature vectors but only similarities between objects exist, alternatively metric space indexing methods such as M-tree [15], Omni-technique [16] and HCT [17] will be employed.

BNN search is a time-critical process which demands immediate result, while similarity join normally is an offline pre-process that serves various knowledge discovery applications [27].

Optimizations of multiple queries have been well studied in the context of relational databases [28]–[31]. The proposed techniques include elimination of common sub-expressions, re-ordering of query plans, using materialized views, pre-fetching and caching of input data values, etc. Once common sub-expressions have been detected for a batch of queries, individual query plans can be merged or modified, and queries can be scheduled for execution so that the amount of data retrieved from the database is minimized, and multiple executions of common paths are avoided. Conceptually, some simple optimization strategies for multiple similarity queries, which we summarize as Sharing Access (SA) next, share the similar rough idea. Specifically, several views of subsets of database relations are maintained so that queries can be evaluated with materialized views, to minimize the number of accesses to database relations. The candidate sets of queries in BNN search can be regarded as the materialized views that are the intermediate and final results computed during the execution of queries. However, in high-dimensional datasets for multimedia applications, the technique used for minimizing the number of accesses to database points is different from that of minimizing the number of accesses to database relations.

BNN search in two-dimensional spatial databases has been suggested in [32] (where the term used is *all nearest neighbors queries*) by ordering the queries with a space filling curve, e.g., Hilbert, to maximize the point locality. Though it is efficient to schedule the queries by the spatial proximity to maximize the buffer hit ratio, in high-dimensional space which is typical for multimedia applications, this approach becomes no longer applicable, since the dataset is presumed to be either indexed with R-tree or spatial hashing, i.e., it does not scale well with dimensionality.

One approach to relevance feedback in content-based image retrieval is the continuous query point movement towards the center for the selected positive image vectors [33]–[35]. Each iteration involves a kNN search. In our earlier work SaveRF [36], through forecasting the positions and search spaces of subsequent queries, the common candidates can be exploited among the iterations by predicting the potential candidates to be searched for the next iteration and maintaining a small set for efficient sequential scan. Therefore, some repeated candidate accesses can be saved. This technique belongs to the category of simple Sharing Access (SA) strategy as well. In this paper, we investigate how to efficiently execute a batch of NN queries in high-dimensional space by an iterative process. Besides utilizing the intersections among consecutive queries to save random candidate accesses, our method can further progressively derive tighter pruning conditions for subsequent queries to discard some unpromising candidates directly. Unlike the fixed query order by nature in relevance feedback, actually in BNN search, all the query points are already on hand. Therefore, at each iteration, the query to be executed can be dynamically selected towards minimizing the overall cost. By utilizing the common candidates between a previous query and subsequent ones, significant overall search cost reduction can be further achieved. To the best of our knowledge, no existing work has investigated the I/O cost further reduction by *pruning condition tightening technique* and *dynamic ordering of query execution* to optimize BNN search in high-dimensional space.

III. Preliminaries

In this section, we introduce some preliminaries of our proposal. First, a generic framework is presented to summarize the common design philosophy of all NN access methods. Then a simple strategy for multiple similarity queries is shown as the baseline used in the experiments for performance comparison.

A. NN Search Framework

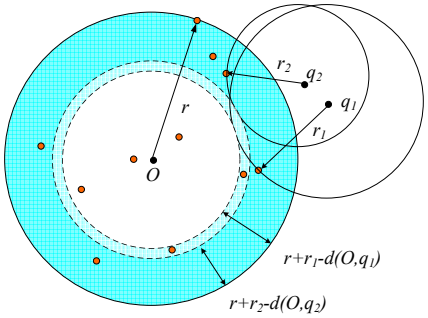
Generally, the NN query processing contains two steps:

- First, data space is filtered based on a certain technique to obtain some candidates. This technique prunes the database points by either a search radius typically in tree-based indexing structures, or a lower bound distance to query typically in scan-based approaches. For tree-based structures such as iDistance [8] which transforms each high-dimensional point to a one-dimensional distance value (with respect to a selected reference point) to be indexed by B^+-tree, such candidates are all the points in the relevant data pages corresponding to the leaf nodes of indexing structure that intersect with search space. For the scan-based approaches such as VA-file [5], such candidates are the points whose lower bound distances are not greater than the smallest upper bound distance determined so far.
- Then, the much smaller number of candidates are accessed from secondary memory and refined by computing the actual distances to query. The results are ranked for returning the NN. Though it is more clear to differentiate the terms of *random page accesses* used for tree-based indexing (I/O cost is gauged by the number of pages accessed) and *random candidate accesses* used for scan-based indexing (I/O cost is gauged by the number of candidates accessed), intrinsically they are similar. Note that I/O cost in VA-file comes from fetching both the approximation file and candidates into main memory. Since sequential I/O operation on the approximation file is much cheaper than random I/O operation on the candidates, and actually in BNN search, sequential I/O cost consumed in the filtering phase for multiple queries can be simply shared by loading the approximation file only once, we focus on investigating how to reduce the random I/O cost.

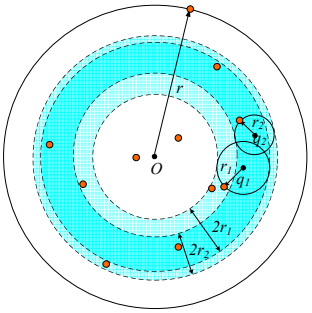
B. Sharing Access (SA)

Given a query video clip $Q = \{q_1, q_2, \ldots, q_m\}$, the naive way to find the NN of each q_i is to search separately, which is termed as *Sharing Nothing* (SN). According to the continuity of video sequences, consecutive queries describing the same shot/scene are usually close to each other in vector space and their search spaces are expected to overlap to some extent. Given a corresponding access method, compared with SN, performing them in a batch usually leads to a much shorter overall query response time. Batch execution provides a great room for query optimizations and the overall response time could be much shorter than the sum of independent NN searches. Considering I/O cost of random disk accesses is quite expensive, it usually becomes the bottleneck of whole query performance [6], [18]. The effect of optimization can be indicated by the number of random disk accesses saved.

If a candidate of current query will be re-accessed by some subsequent queries, it is preferred to utilize this candidate already in main memory and directly apply distance computations to avoid re-fetching. For instance, Fig. 3 show two cases of two queries q_1 and q_2 processed in the tree-based indexing iDistance, where O is the reference point and r is the radius of data space. In Fig. 3(a), all the points in the annulus of $r + r_1 - d(O, q_1)$ will be accessed by q_1, and those in the annulus of $r + r_2 - d(O, q_2)$ will be accessed by q_2. In Fig. 3(b), all the points in the annulus of $2r_1$ will be accessed by q_1, and those in the annulus of $2r_2$ will be accessed by q_2. Compared with SN that performs two NN queries separately, in both cases, actually there is no need to re-access the points in the



(a) Query points outside the data space but query spheres intersect the data space.



(b) Query points are contained in the data space.

Fig. 3. Repeated candidate accesses in iDistance.

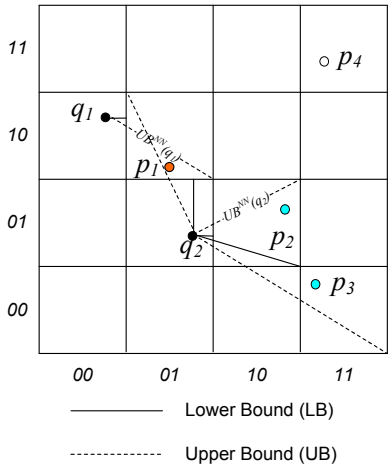


Fig. 4. Repeated candidate accesses in VA-file.

overlapped annulus. Query composition introduced in our early work [3] analyzes the search ranges and composes the overlapped ranges into a single one to eliminate the duplicated accesses. The strategy introduced in [37] illustrated with X-tree which loads a page once and immediately processes it for all the queries which consider it as a candidate page is also similar to this.

For another instance, in the scan-based indexing VA-file, it is likely that some candidates can be avoided being accessed for multiple times. Since in BNN search, each $q_i \in Q$ is resident in main memory, the candidate sets of all queries can be merged together and only their union set needs to be loaded once. The common candidates are computed with multiple queries for actual distances. Fig. 4 is a 2-dimensional VA-file (each dimension with 2 bits) for illustration, where the smallest upper bound of each query, denoted as $UB^{NN}(q_i)$, is marked. As can be seen, the candidate set C_1 of q_1 includes p_1 only and the candidate set of q_2 includes p_1, p_2, and p_3. First, the points in C_1 are accessed and processed (p_1 in Fig. 4). For another query q_2 with $C_2 = \{p_1, p_2, p_3\}$, actual distance computations can be directly applied to the common candidates shared by C_1 and C_2, since they are already available in main memory (p_1 in Fig. 4). p_1 is avoided being re-accessed by computing its distances to both q_1 and q_2. Through the merging of candidate sets, any duplicated candidate will be only accessed once.

The common essence between the query composition and merging operation strategies described above can be generalized as *Sharing Access* (SA). SA eliminates the redundant random disk accesses by the intersections among queries and loads the repeated candidates in a single pass for the whole query set. Any potential redundancy can be eliminated by the sharing among individual NN determinations with a *union* operation on the candidates. The reduction effect of SA R_{SA} is measured by:

$$R_{SA} = \sum_{i=1}^{m} |C_i| \Big/ |\bigcup_{i=1}^{m} C_i|, \tag{2}$$

where for tree-based indexing C_i is the considered page set of q_i and for scan-based indexing C_i is the candidate set of q_i. Since intrinsically they are same, and normally it is observed that all the tree-based indexing techniques still fail to improve the performance of sequential scan in higher dimensional space because they have to access most of the data points for a single query (the expected NN distance has also been theoretically proven to be even larger than the length of a dimension of data space in [5], [38] with uniform distribution assumption), in the following, we adopt scan-based approach VA-file, which is more popular and suitable for multimedia databases to illustrate our ideas.

SA is a static union of the known candidate sets. One question emerges: is it possible to further reduce the size of this union for faster query response? The answer is definite. In the next section, we show that in VA-file, a tighter pruning condition may be derived for the subsequent query thus some unpromising candidates can be further identified and discarded directly. Therefore, the candidate overlapping relationship among queries is exploited to dynamically select the next query to be executed that potentially filters the rest candidate sets maximally. By doing so, the overall cost can be further reduced significantly.

IV. Batch Nearest Neighbor Search

In this section, we introduce a novel BNN search algorithm called Dynamic Query Ordering (DQO), which dynamically schedules the queries towards the overall minimal I/O cost together with optimization of CPU cost. The key idea is inspired from the observation that the overlapped candidates already accessed by a previous query may help to derive tighter pruning conditions of subsequent queries, thus their candidate sets can be further reduced. First we re-examine the running example of VA-file.

A. Pruning Condition Tightening

In VA-file, a tighter pruning condition (smaller $UB^{NN}(q_i)$) informed from a previous query can safely discard some false admits in the original candidates set, therefore, full utilization of the candidates available in main memory may further reduce I/O cost. Consider the example in Fig. 5, interestingly, $d(q_2, p_1)$ is smaller than the original NN upper bound $UB^{NN}(q_2)$. Therefore, it can be used as a new NN upper bound of q_2, and random candidate accesses are only required for the updated C_2 which has excluded the points whose lower bounds are now greater than the updated pruning condition $d(q_2, p_1)$ (p_3 in Fig. 5). Though the new candidate set becomes smaller, the correctness of query result is still guaranteed. Since some

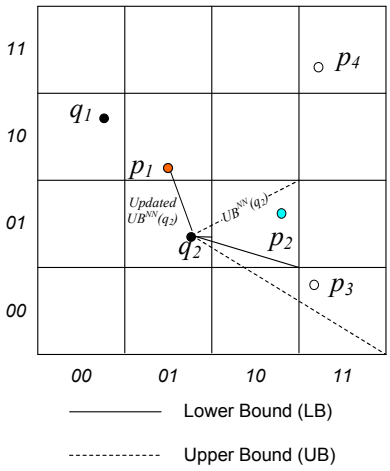


Fig. 5. Pruning condition tightening in VA-file.

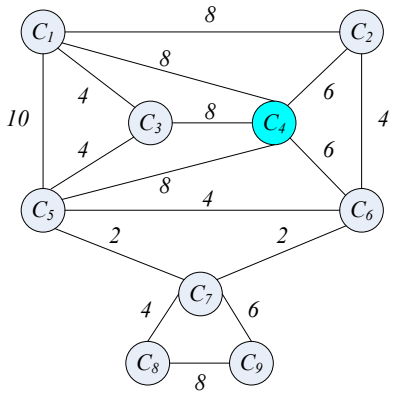


Fig. 6. A candidate overlapping graph.

actually unpromising candidates are identified, the random accesses for them can be saved.

Fully unitizing the candidates available in main memory accessed by a previous query may lead the overall candidates for the whole batch becomes only a subset of the union of all candidate sets. By the actual distance computations on the overlapped candidates with a previous query, a tighter upper bound may be updated as a new pruning condition. Therefore, some random candidate accesses which still will be conduced according to SA are now confirmed as unpromising false admits and can be disqualified safely. We distinguish this exploration as *saving actually unpromising random candidate accesses* with *eliminating repeated random candidate accesses* described in SA. Though it may incur some additional computations, compared with reduced I/O operations, potentially increased CPU time can be accepted. Moreover, as discussed in Subsection IV-D, not all the overlapped candidates need to be tested based on the property of triangle inequality.

B. Candidate Overlapping Graph (COG)

Since the tentative evaluation of smaller NN upper bound based on the overlapped candidates can further save I/O cost, we prefer an execution plan that benefits from the overall effect of pruning condition tightenings for all the queries in the batch most. Compared with SA which loads the candidates corresponding to $\bigcup_{i=1}^{m} C_i$ for refinement, the sequence of processing $C_1, C_2, \ldots, C_m$ now becomes crucial, since the execution merely according to the plain order $q_1, q_2, \ldots, q_m$ cannot always maximize the effect of memory utilization thus it is not optimal with respect to I/O cost. If there is a schedule τ that processes $C_1, C_2, \ldots, C_m$ as $C_{\tau 1}, C_{\tau 2}, \ldots, C_{\tau m}$ to utilize the overlapped candidates among queries to the maximal extent, its chance of further reducing I/O cost can be maximized.

Definition 1 (Optimal Query Order): Among all possible permutations of C_i, the optimal query order τ is a sequence of $C_{\tau 1}, C_{\tau 2}, \ldots, C_{\tau m}$ that minimizes the overall I/O cost by utilizing the overlapped candidates accessed by a previous query.

To maximize the overall benefit from the partial candidate prefetching accumulated during each round of NN searches, we introduce a process that dynamically finds the optimal query order τ by Candidate Overlapping Graph (COG). Before moving to the detailed algorithm, we first show how to model the candidate overlapping relationship among queries.

Definition 2 (Candidate Overlapping Graph): Given m queries with their candidate sets, the information of their overlapping relationship can be captured by their intersections. A candidate overlapping graph $G = \{V, E, \omega\}$ is a weighted graph representing the overlapped candidates. The vertex set $V = \{C_1, C_2, \ldots, C_m\}$ is a set of candidate sets of individual queries. The edge set E is defined as: for each vertex pair C_i and C_j, there is an edge E_{ij} if $C_i \cap C_j \neq \emptyset$, and its weight ω_{ij} is the size of intersection $|C_i \cap C_j|$.

Fig. 6 serves as an example of COG of 9 queries. The edges reflect how the two vertices they connect approximate each other in vector space, i.e., how similar they are. Note that, since some queries may be quite different from others, their candidate sets may not overlap with others. Therefore, some vertices may be isolated and it is possible that G is not a connected graph. For COG, we have the following propositions.

Proposition 1: The potential pruning condition tightening to derive a smaller NN upper bound only occurs between a vertex pair of a former query q_i and a latter query q_j where there is an edge E_{ij} connecting them.

Proof: Given a candidate $p_k \in C_i$ of q_i but $p_k \notin C_j$ of q_j, assume p_k tightens the NN upper bound of q_j, i.e., the distance $d(p_k, q_j)$ is smaller than $UB^{NN}(q_j)$. Since the lower bound of p_k is not greater than the distance between p_k and q_j, based on the pruning condition that any point whose lower bound is less than $UB^{NN}(q_j)$ will be included in C_j (due to the property of completeness), we have $p_k \in C_j$. This contradicts with $p_k \notin C_j$. Therefore, C_i and C_j have at least one common candidate p_k. By the edge definition of COG, we have the proof. ■

Proposition 1 suggests that a tighter pruning condition can only be informed from the overlapped candidates of another query. However, not all the overlapped candidates contribute to the pruning condition tightening. Naturally, we have the following proposition.

Proposition 2: Given a COG G, if a vertex C_i (and its associated edges) is removed, i.e., q_i is processed, only the vertices (and their associated edges) connecting to C_i are possible to be reduced.

Proof: According to Proposition 1, only the vertices connecting to the removed vertex are possible to reduce their sizes. Accordingly, the weights of all edges having at least one vertex connecting to the selected vertex are probably to become smaller as well. ■

Fig. 7 shows an example of removing C_4 from Fig. 6, where the shaded vertices and their associated edges are possible to be updated. Proposition 2 suggests that only a subgraph of G needs to be updated after a query is processed. Note that in Fig. 7, due to updating C_5 and C_6, some of their candidates are excluded and they become sharing no common candidates with C_7 now. Therefore, COG may become a unconnected graph.

Proposition 3: Given a COG G, if its each maximum connected subgraph $G^{'}$ gets its own optimal query order $\tau^{'}$, then serially

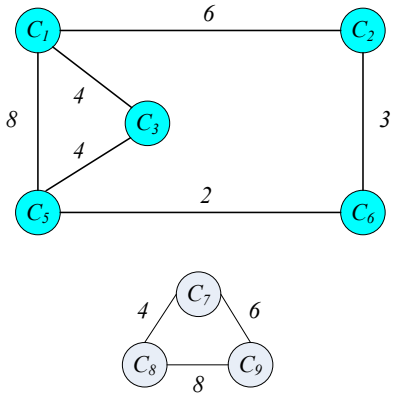


Fig. 7. Updated candidate overlapping graph after removing a selected vertex.

executing each $\tau^{'}$ in any arbitrary order is an optimal query order τ.

Proof: According to Proposition 1, a query in an isolated subgraph of G can never contribute to or informed by the pruning condition tightening of a query in another isolated subgraph, i.e., there is no interaction of smaller NN upper bound deriving to further reduce random candidate accesses. ■

With COG and its propositions, we are now ready to discuss the dynamic query ordering strategy.

C. Dynamic Query Ordering (DQO)

The framework of our BNN search algorithm Dynamic Query Ordering (DQO) is first presented below, and then the detail of each step is discussed.

Algorithm 1 Dynamic Query Ordering (DQO)

Input:
G - a candidate overlapping graph
Output:
NN results of all queries
Description:
1: **while** G is not empty **do**
2: select a vertex C_i from G;
3: execute q_i corresponding to C_i;
4: update G;
5: **end while**
6: **return** NN results of all queries;

DQO is an iterative process consisting of three steps:

- **Select a vertex:** In this step, a vertex C_i is dynamically selected from the iteratively updated graph G for execution. This step is the core of our algorithm, which determines the query order. Finding an optimal query order τ is a combinational optimization problem since whether the pruning condition can be tightened for each single query can not be anticipated. Thus, effective heuristic methods are needed to find near-optimal orders. A general guideline is to process the queries which are expected to reduce the candidate sets of others at maximal degrees earlier. Executed in this way, the NN upper bound $UB^{NN}(q_i)$ will approach to the actual NN distance $dist^{NN}(q_i)$ faster.
- **Execute the query:** Once a vertex is selected, its candidates are randomly accessed from secondary memory for actual distance computations. This step determines the I/O cost. The more candidate set size is reduced, the less I/O cost.
- **Update** G**:** Once the query q_i has been executed, its candidates are available in main memory. Before they are flushed away, the vertices connecting to C_i can be updated based on Proposition 2, so as to their associated edges. To do so, for each C_j with $\omega_{ij} \neq 0$, the minimal distance between q_j and its overlapped candidates in C_i, i.e., $min(d(q_j, p_i), \forall p_i \in C_i \cap C_j)$, is computed. If this minimal distance is smaller than the current NN upper bound $UB^{NN}(q_j)$, then update $UB^{NN}(q_j) = min(d(q_j, p_i), \forall p_i \in C_i \cap C_j)$. All the candidates in C_j whose lower bounds are now greater than the smaller $UB^{NN}(q_j)$ can be pruned safely, thus $|C_j|$ is reduced. Correspondingly, the weights of edges connecting to C_j are also updated.

The process is repeated until all the vertices have been processed and their NN results are returned. When G becomes not a connected graph, the algorithm applies to its each maximum connected subgraph. Obviously, the property that changing the query order will not affect the correctness of search result still holds. The query order determined in the first step affects the degree of candidate set reduction in the third step. As explained, finding the exact optimal order is not realistic unless all possible permutations have been tried. We propose two efficient heuristic methods to find near-optimal orders.

Since the future pruning condition and actual candidate reduction size are unpredictable, utilizing available information at current stage is reasonable and promising. Based on Proposition 1, processing a query only affects its connecting neighbors. Our first intuition is that a query order maximizing the total volume of overlap in the neighborhoods of consecutive queries may reduce the candidates most. This leads to the first heuristic for dynamic query selection.

Heuristic 1: Given a COG G of m queries, a vertex C_i is selected to be processed next if its $\sum_{j=1}^{m} \omega_{ij}$ $(j \neq i)$ is maximal.

Heuristic 1 suggests to greedily select a vertex with the largest number of total overlapped candidates with others as a pioneer. More common candidates not only mean more repeated random candidate accesses can be shared with its neighbors as in SA, but also imply a larger chance to derive tighter pruning conditions. In other words, a larger number of common candidates shared with others mean the query is much more *influential*. In G, to process a query that has the most number of its candidates being taken by other queries as their candidates as well is equal to select the vertex with the maximal total weights of associated edges from the remaining graph[3]. Recall the example in Fig. 6, this vertex corresponds to C_4. After processing q_4, all the associated edges are removed and the sizes of its neighboring vertices (with edge connections) are consequently reduced, as shown in Fig. 7. The reason is two folds. First, as in SA, the overlapped candidates between C_i and its connected vertex C_j, i.e., $C_i \cap C_j$, are also compared with q_j for actual distances. More importantly, the minimal actual distance computed between a candidate in $C_i \cap C_j$ and q_j can be used to tighten the pruning distance $UB^{NN}(q_j)$, as illustrated in the third step of DQO. Though $C_i \cap C_j$ may not be a completed candidate set of q_j, as long as q_i and q_j are nearby, a tighter $UB^{NN}(q_j)$ is highly possible to be updated as a by-product.

Heuristic 1 works with the assumption that the volume of overlap is proportional to the volume of candidates pruned. However, in real situation, this may not always be valid. After C_i is processed, the size of C_j itself also affects the number of candidates to be pruned from C_j. Meanwhile, if there are more candidates in C_j whose lower bounds are greater than the lower bound of an overlapped candidate, more candidates are potentially pruned from C_j. That is, the relative position of an overlapped candidate in C_j ranked by lower bound also affects the number of candidates to be pruned. Taking both the candidate set size and the relative position of an overlapped candidate into consideration, we have the second heuristic for dynamically

[3]When there is a tie, select the vertex whose size is smaller.

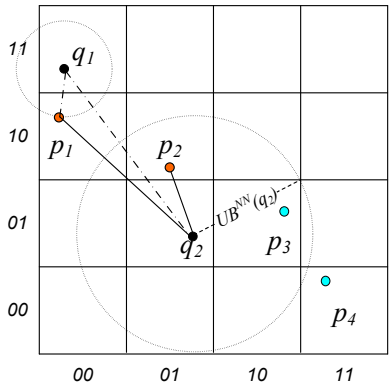


Fig. 8. An illustration for CPU optimization.

selecting the next query.

Heuristic 2: Given a COG G of m queries, a vertex C_i is selected to be processed next if its $\sum_j \sum_k (|C_j| - Position(p_k, C_j))$ is maximal, where C_j is a neighboring vertex of C_i, $p_k \in C_i \cap C_j$, and function $Position(p_k, C_j)$ returns the relative position of p_k in C_j ranked by lower bound in the ascending order.

$|C_j| - Position(p_k, C_j)$ is the anticipated maximal number of candidates that can be pruned by p_k in C_j, i.e., all the candidates having greater lower bounds than that of p_k would be pruned. Thus in Heuristic 2, a query is selected based on its maximal pruning power. The performance of both heuristics will be studied in experiments.

In summary, towards the optimal query order, in each iteration DQO selects a query to be processed based on the proposed heuristics by the dynamically updated COG, with the incremental random candidate accesses and tentative pruning condition tightenings.

D. CPU Optimization

In high-dimensional NN search, CPU cost is mainly due to the expensive distance computations in full dimensionality. This occurs when the actual distances are computed between the query and its candidates. Thus reducing the number of candidates also leads to a smaller CPU cost. In DQO, reduction of candidates is achieved by finding a tighter pruning condition through computing the actual distances to the overlapped candidates (updating G in DQO). While I/O cost can be reduced significantly by DQO, CPU cost can also be further optimized. The testing order of common candidates is another important factor to be considered, because not all the common candidates in main memory can contribute to the tightening of pruning condition. The property of triangle inequality can be employed to avoid the actually unnecessary computations on some overlapped candidates.

Fig. 8 shows not all the overlapped candidates can help to tighten the pruning condition, where q_1 and q_2 have candidate sets $\{p_1, p_2\}$ and $\{p_1, p_2, p_3, p_4\}$, respectively. The overlapped candidates are p_1 and p_2. Assume q_1 is processed first. Clearly, p_1 cannot further tighten the pruning condition of q_2 since its distance to q_2 is greater than current $UB^{NN}(q_2)$, while p_2 can be used to prune p_4 from C_2. Therefore, we want to avoid the unnecessary distance computations to further reduce CPU cost.

Proposition 4: Given two queries q_i and q_j and $p_k \in C_i \cap C_j$, assume q_i is processed, if $|d(q_i, q_j) - d(q_i, p_k)| \geq UB^{NN}(q_j)$, then $d(q_j, p_k) \geq UB^{NN}(q_j)$.

Proof: Based on the property of triangle inequality, $d(q_j, p_k) \geq |d(q_i, q_j) - d(q_i, p_k)|$. Given $|d(q_i, q_j) - d(q_i, p_k)| \geq UB^{NN}(q_j)$, we have $d(q_j, p_k) \geq UB^{NN}(q_j)$. ■

Bit number per dimension	6	7	8
Average number of candidates	3405	942	500

TABLE I
BIT NUMBER PER DIMENSION VS. AVERAGE NUMBER OF CANDIDATES.

Since both $d(q_i, p_k)$ and $d(q_i, q_j)$ are already computed, by Proposition 4, some overlapped candidates which cannot tighten the pruning condition can be avoided from actual distance computations. This further reduces the CPU cost.

V. EXPERIMENTS

This section reports the experiment results on the practical effectiveness of our proposed methods.

A. Setup

We use a real video database consisting of 2,168 video clips which are TV commercials captured from TV stations for evaluations. They are recorded with VirtualDub [39] at PAL frame rate of 25fps. The time length of each clip is about 60 seconds, therefore, the whole database consists of about 3,000,000 video frames. Four feature datasets in 8-, 16-, 32- and 64-dimensional RGB color spaces for this very large video collection were generated for test purpose, and the value of each dimension was normalized by dividing the total number of pixels. For each query clip, one frame is extracted from each second of it, so each query video consumes a BNN search with 60 sampled frames. By default, we set k=100 in kNN search for each query point in BNN search, i.e., each frame searches for 100 most similar frames from the database. Since the number of candidate accesses and the extent of correlations of query points also depend on the query video content and its variance, 10 video clips were randomly selected from the database and all the results reported are the average based on the 10 queries. All the experiments were performed on Window XP platform with Intel Pentium 4 Processor (3.0 GHz CPU) and 512 MB RAM. The page size used in the implementation is 4 KB.

Since the performance of kNN search depends heavily on the number of disk accesses, the total number of candidates of all queries in the batch is a good performance indicator. Moreover, the number of candidates sharing the same disk page decreases as the dimensionality increases and the disk access pattern is random. Thus, in fact, the candidate selectivity can be an estimator for the I/O cost in kNN search [5], [6].

Table I shows the average number of candidates per frame in the query clip by the naive approach where each query is independently performed on the 32-dimensional dataset. As more bits are allocated for each dimension, the number of candidates of each query frame drops quickly. This is obvious since VA-file computes tighter lower and upper bounds with more bits per dimension for better locating the data points. However, whenever more bits are used, the cost of scanning all the approximations also increases correspondingly. For example, the cost for scanning VA-file of 8 bits per dimension is doubled for that of 4 bits per dimension.

Table II provides the reduction effect of total number of candidate accesses that a simple SA strategy can achieve. As expected, for smaller bit number per dimension tested, the saving is more significant. Even 8 bits per dimension is used, the reduction effect of SA makes the total number of expensive random candidate accesses becomes only less than 40% of original. Considering the advantage of SA over the naive approach SN is very clear and has also been demonstrated with other access methods such as query composition technique with iDistance [3], in the following, we mainly compare DQO with the simple optimization strategy SA. Since two heuristic

Bit number per dimension	6	7	8
Total number of candidates	204280	56547	4795
Number of candidates of SA	14256	8199	1840
Reduction effect of SA	0.070	0.145	0.384

TABLE II
EFFECTIVENESS OF SA.

methods are introduced, we denote DQO using Heuristic 1 and Heuristic 2 as DQO1 and DQO2, respectively. To clearly see the further improvement achieved by DQO, SA is employed as the baseline for comparison, and we use the indicator *Improvement Ratio* (IR), which is defined as *the ratio of the total number of candidate accesses saved by DQO over the total number of candidates of SA*. IR of DQO shows the percentage of candidates that can be further pruned by utilizing the effect of pruning condition tightening by DQO from the already reduced overall candidate set of SA. Meanwhile, we test the elapsed time of DQO, which actually can reveal the CPU cost for ordering the queries and utilizing the effect of pruning condition tightening to reduce the sizes of candidate sets. Finally, through the comparison of overall query response time, we test how helpful the additional CPU time of DQO is to the I/O cost saving for the whole batch to the end.

B. Effect of Bit Number in VA-file

In the first experiment, we test the effect of bit number per dimension in VA-file on Improvement Ratio. The 32-dimensional dataset is used in this experiment.

Fig. 9 shows the effectiveness of DQO1 and DQO2 for different bit numbers. From Fig. 9, we have several observations. First, our DQO can further improve SA by pruning away more than 20% candidates of SA for various bit numbers per dimension. This confirms that the overlapped candidates from a previous query indeed help to tighten the pruning conditions of subsequent queries and our DQO is very effective in fully utilizing the overlapped candidates. Second, IRs of both DQO1 and DQO2 decrease as the bit number per dimension increases. This is reasonable since the total number of candidate accesses becomes much smaller as the bit number per dimension increases (cf. Table I and Table II), so as to the number of overlapped candidates. As a result, DQO has less opportunities for further pruning. Third, as expected, DQO2 consistently performs better than DQO1, however, such improvement is not very significant. This is probably due to nearly uniform distribution of overlapped candidates in most candidate sets. In this case, the advantage of Heuristic 2 would not be very obvious. Fourth, note that compared with our preliminary experiment results reported in [14] where successive video frames are used for the query video clip, the IRs of DQO conducted with sampled video frames here are a little smaller. The reason is that the correlations of query points representing the sampled frames from each second of query video is generally expected not as significant as that of successive video frames.

Table III shows the corresponding CPU time of DQO1 and DQO2 of different bit numbers for each dimension used in VA-file. It can be observed that the higher IR of DQO2 than that of DQO1 achieves at the expense of higher CPU cost. Also, IR has a positive correlation with the elapsed time of DQO. A general guideline is using moderate bit numbers can get better overall query performances. For the rest of experiments, we only use 7 bits and 8 bits per dimension for tests.

C. Effect of Dimensionality

In this experiment, we test the effect of dimensionality on Improvement Ratio with all four feature datasets. As shown in Fig. 10, again, DQO improves SA greatly. Very interestingly, as dimensionality goes

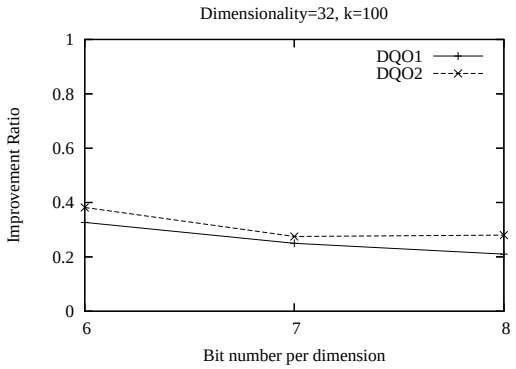


Fig. 9. IR vs. bit number per dimension.

Bit number per dimension	6	7	8
DQO1	12.2	2.4	1.8
DQO2	55.8	5.1	2.5

TABLE III
ELAPSED TIME OF DQO (IN SECONDS) VS. BIT NUMBER PER DIMENSION.

up to 64, DQO performs better. A known phenomenon 'dimensionality curse' in high-dimensional space is that a small increase of search radius will introduce a large number of candidates. On the other hand, a small decrease of search radius will filter a large number of candidates. Note that our DQO aims at providing a tighter pruning condition, and the performance indicator of vertical axis is the relative percentage of candidates that can be pruned, not the absolute number of candidate accesses. Therefore, a small tightening of pruning condition is expected to prune away more portion of candidates in a higher dimensional space. As we will also see later in Subsection V-E, more overlapped candidates are potentially used to further tighten the pruning condition in a higher dimensional space. This is another reason for the up trends of DQO1 and DQO2 in Fig. 10.

Table IV provides the corresponding elapsed time of DQO tested with different dimensional datasets. When the dimensionality increases, the candidate selectivity of each query usually becomes larger as well. Therefore, DQO also needs more computations to process the larger candidate sets.

D. Effect of k

It is straightforward to extend DQO to kNN search for each query point in BNN search by replacing $UB^{NN}(q_i)$ with $UB^{kNN}(q_i)$ as the pruning condition. In this experiment, we test the effect of k on Improvement Ratio. The 32-dimensional dataset is used. Fig. 11 indicates that DQO is slightly more effective for a larger k. The reason can be explained as that a larger k corresponds to larger candidate selectivity, therefore, the chance of a larger $UB^{kNN}(q_i)$ to be tightened is higher.

Table V provides the corresponding elapsed time of DQO of different k. Similarly, When k increases, the size of candidate set of each query becomes larger as well, therefore, DQO also needs more computations to process.

E. Effect of CPU Optimization

In this experiment, we test how many percentage of expensive distance computations for overlapped candidates can be saved based on Proposition 4. Here we use Saving Ratio (SR) which is defined as *the ratio of the total number of distance computations saved by CPU optimization over the total number of distance computations without CPU optimization for overlapped candidates in DQO*.

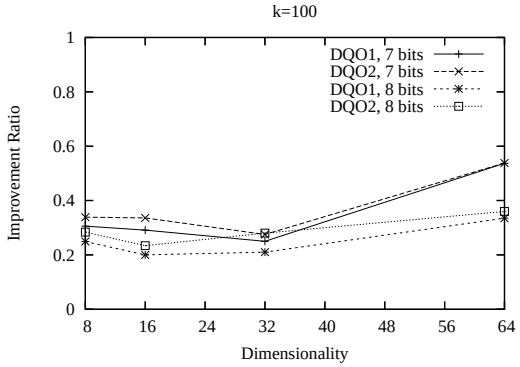


Fig. 10. IR vs. dimensionality.

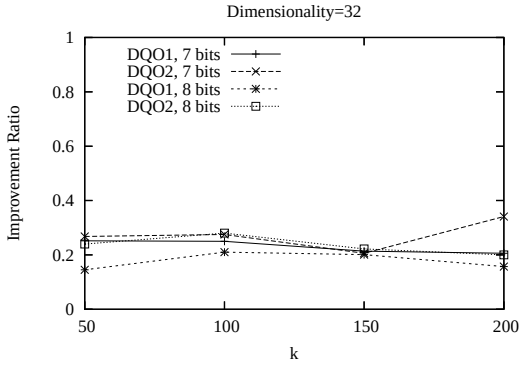


Fig. 11. IR vs. k.

Dimensionality	8	16	32	64
DQO1, 7 bits	2.2	2.3	2.4	20.3
DQO2, 7 bits	4.9	4.3	5.1	130.6
DQO1, 8 bits	1.9	1.7	1.8	3.3
DQO2, 8 bits	3.2	2.1	2.5	11.0

TABLE IV
ELAPSED TIME OF DQO (IN SECONDS) VS. DIMENSIONALITY.

k	50	100	150	200
DQO1, 7 bits	2.3	2.4	2.5	2.6
DQO2, 7 bits	4.4	5.1	5.7	5.7
DQO1, 8 bits	1.7	1.8	1.9	2.0
DQO2, 8 bits	2.3	2.5	2.9	3.1

TABLE V
ELAPSED TIME OF DQO (IN SECONDS) VS. k.

All four datasets are used in this experiment. As can be seen from Fig. 12, on average, the CPU optimization technique further reduces 10%-20% actual distance computations. When the dimensionality of feature space is smaller, the CPU optimization technique can save more actual distance computations for overlapped candidates. As the dimensionality increases, SR drops gradually. This indicates that in a higher dimensional space, smaller number of overlapped candidates can be avoided for actual distance computations, i.e., more overlapped candidates can potentially help to further tighten the pruning condition. This also explains the trend in Fig. 10 where the percentage of candidate accesses saved by DQO grows as the dimensionality increases.

F. Comparison of Query Response Time

In the previous experiments, our proposed DQO is mainly compared with other methods of BNN search (SN and SA) in terms of candidate selectivity. Though kNN search from a large dataset is generally regarded to be disk-bounded, and reducing the number of candidates also helps to reduce the CPU cost for actual distance computations, they may not fully reflect the reality. To demonstrate the practical effectiveness of DQO, we further perform a number of timing tests to compare the query response time of DQO with naive SN and simple SA executions. The 32-dimensional dataset is used in this experiment.

To be clearer, in Fig. 13 we show the average response time per query, i.e., the total response time divided by the number of individual queries in the batch (60 in our experiment setting) by SN, SA, DQO1 and DQO2 respectively, where both the cases of 7 bits and 8 bits per dimension are tested. As can be seen from Fig. 13, DQO can outperform SN by a factor up to 2, and is also better than SA substantially on the average. It also can be seen that the expense of elapsed time of DQO is usually worthwhile, considering its significant gain by reducing a large number of expensive random candidate accesses for the whole batch, which finally leads to shorter total response time. Actually for a batch of 60 queries in our experiment, it usually only takes several seconds (cf. Table III) for ordering the queries to fully utilize the effect of pruning condition tightening and estimating new upper bounds of subsequent queries, which leads to smaller candidate sets for exhaustive consideration. In other words, the response time saving from the much smaller I/O cost can be greatly benefit from the slightly additional CPU cost.

VI. CONCLUSIONS

This paper proposes Dynamic Query Ordering, a generic framework that facilities batch-oriented similarity query processing in high-dimensional space for video retrieval. For more sophisticated utilization of the commonalities across a series of NN searches, how to progressively choose an optimal execution order is exploited as the key to speed up BNN search. Experiments on real video datasets show that the proposed methods can outperform simple Sharing Access execution measured by the number of random candidate accesses, and the CPU cost can also be further reduced based on the property of triangle inequality. Our BNN search strategy is expected to enhance the efficiency of current CBVR systems significantly.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their comments, which led to improvements of this paper. This work has been supported by Australian Research Council under grant DP0663272.

REFERENCES

[1] H. Wang, A. Divakaran, A. Vetro, S.-F. Chang, and H. Sun, "Survey of compressed-domain features used in audio-visual indexing and analysis." *J. Vis. Commun. Image R.*, vol. 14, no. 2, pp. 150–183, 2003.

[2] S.-C. S. Cheung and A. Zakhor, "Efficient video similarity measurement with video signature." *IEEE Trans. Circuits Syst. Video Techn.*, vol. 13, no. 1, pp. 59–74, 2003.

[3] H. T. Shen, B. C. Ooi, X. Zhou, and Z. Huang, "Towards effective indexing for very large video sequence database." in *SIGMOD Conference*, 2005, pp. 730–741.

[4] G. Lu, "Techniques and data structures for efficient multimedia retrieval based on similarity." *IEEE Trans. Multimedia.*, vol. 4, no. 3, pp. 372–384, 2002.

[5] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces." in *VLDB*, 1998, pp. 194–205.

[6] G.-H. Cha, X. Zhu, D. Petkovic, and C.-W. Chung, "An efficient indexing method for nearest neighbor searches in high-dimensional image databases." *IEEE Trans. Multimedia.*, vol. 4, no. 1, pp. 76–87, 2002.

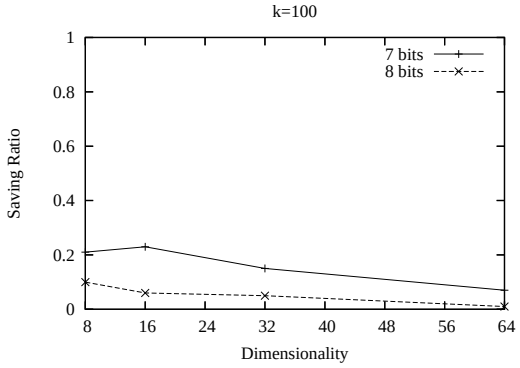


Fig. 12. SR by CPU optimization.

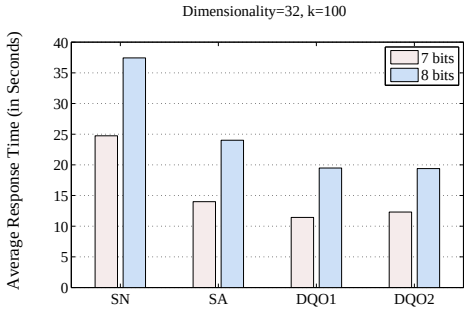


Fig. 13. Comparison of query response time.

[7] S. Berchtold, D. A. Keim, and H.-P. Kriegel, "The x-tree : An index structure for high-dimensional data," in *VLDB*, 1996, pp. 28–39.

[8] H. V. Jagadish, B. C. Ooi, K.-L. Tan, C. Yu, and R. Zhang, "idistance: An adaptive b^+-tree based indexing method for nearest neighbor search." *ACM Trans. Database Syst.*, vol. 30, no. 2, pp. 364–397, 2005.

[9] S. Berchtold, C. Böhm, H. V. Jagadish, H.-P. Kriegel, and J. Sander, "Independent quantization: An index compression technique for high-dimensional data spaces." in *ICDE*, 2000, pp. 577–588.

[10] G.-H. Cha and C.-W. Chung, "The gc-tree: a high-dimensional index structure for similarity search in image databases." *IEEE Trans. Multimedia.*, vol. 4, no. 2, pp. 235–247, 2002.

[11] N. Koudas, B. C. Ooi, H. T. Shen, and A. K. H. Tung, "Ldc: Enabling search by partial distance in a hyper-dimensional space." in *ICDE*, 2004, pp. 6–17.

[12] Z. Yang, W. T. Ooi, and Q. Sun, "Hierarchical, non-uniform locality sensitive hashing and its application to video identification." in *ICME*, 2004, pp. 743–746.

[13] H. Lu, B. C. Ooi, H. T. Shen, and X. Xue, "Hierarchical indexing structure for efficient similarity search in video retrieval." *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 11, pp. 1544–1559, 2006.

[14] J. Shao, Z. Huang, H. T. Shen, X. Zhou, and Y. Li, "Dynamic batch nearest neighbor search in video retrieval." in *ICDE*, 2007, pp. 1395–1399.

[15] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces." in *VLDB*, 1997, pp. 426–435.

[16] R. F. S. Filho, A. J. M. Traina, C. T. Jr., and C. Faloutsos, "Similarity search without tears: The omni family of all-purpose access methods." in *ICDE*, 2001, pp. 623–630.

[17] S. Kiranyaz and M. Gabbouj, "Hierarchical cellular tree: An efficient indexing scheme for content-based retrieval on multimedia databases." *IEEE Trans. Multimedia.*, vol. 9, no. 1, pp. 102–119, 2007.

[18] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. E. Abbadi, "High dimensional nearest neighbor searching." *Inf. Syst.*, vol. 31, no. 6, pp. 512–540, 2006.

[19] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions." *J. ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[20] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing." in *VLDB*, 1999, pp. 518–529.

[21] F. Korn and S. Muthukrishnan, "Influence sets based on reverse nearest neighbor queries." in *SIGMOD Conference*, 2000, pp. 201–212.

[22] Y. Tao, D. Papadias, and Q. Shen, "Continuous nearest neighbor search." in *VLDB*, 2002, pp. 287–298.

[23] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis, "Group nearest neighbor queries." in *ICDE*, 2004, pp. 301–312.

[24] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui, "Aggregate nearest neighbor queries in spatial databases." *ACM Trans. Database Syst.*, vol. 30, no. 2, pp. 529–576, 2005.

[25] R. Fagin, "Combining fuzzy information: an overview." *SIGMOD Record*, vol. 31, no. 2, pp. 109–118, 2002.

[26] C. Xia, H. Lu, B. C. Ooi, and J. Hu, "Gorder: An efficient method for knn join processing." in *VLDB*, 2004, pp. 756–767.

[27] C. Böhm, B. Braunmüller, F. Krebs, and H.-P. Kriegel, "Epsilon grid order: An algorithm for the similarity join on massive high-dimensional data." in *SIGMOD Conference*, 2001, pp. 379–388.

[28] T. K. Sellis, "Multiple-query optimization." *ACM Trans. Database Syst.*, vol. 13, no. 1, pp. 23–52, 1988.

[29] M. H. Kang, H. G. Dietz, and B. K. Bhargava, "Multiple-query optimization at algorithm-level." *Data Knowl. Eng.*, vol. 14, no. 1, pp. 57–75, 1994.

[30] K. Shim, T. K. Sellis, and D. S. Nau, "Improvements on a heuristic algorithm for multiple-query optimization." *Data Knowl. Eng.*, vol. 12, no. 2, pp. 197–222, 1994.

[31] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe, "Efficient and extensible algorithms for multi query optimization." in *SIGMOD Conference*, 2000, pp. 249–260.

[32] J. Zhang, N. Mamoulis, D. Papadias, and Y. Tao, "All-nearest-neighbors queries in spatial databases." in *SSDBM*, 2004, pp. 297–306.

[33] D. Tao, X. Tang, X. Li, and Y. Rui, "Direct kernel biased discriminant analysis: A new content-based image retrieval relevance feedback algorithm," *IEEE Transactions on Multimedia*, vol. 8, no. 4, pp. 716–727, 2006.

[34] D. Tao, X. Tang, X. Li, and X. Wu, "Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 7, pp. 1088–1099, 2006.

[35] Y. Rui, T. S. Huang, M. Ortega, and S. Mehrotra, "Feedback: A power tool in interactive content-based image retrieval," *IEEE Trans. on Circuits and Systems for Video Technology , Special Issue on Segmentation, Description, and Retrieval of Video Content*, vol. 8, no. 5, pp. 644–655, 1998.

[36] H. T. Shen, B. C. Ooi, and K.-L. Tan, "Saverf: Towards efficient relevance feedback search." in *ICDE*, 2006.

[37] B. Braunmüller, M. Ester, H.-P. Kriegel, and J. Sander, "Efficiently supporting multiple similarity queries for mining in metric databases." in *ICDE*, 2000, pp. 256–267.

[38] C. Böhm, "A cost model for query processing in high dimensional data spaces." *ACM Trans. Database Syst.*, vol. 25, no. 2, pp. 129–178, 2000.

[39] http://www.virtualdub.org.