http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 8 (8): 1115-1128, 2009
ISSN 1812-5638

Processing Techniques for Querying Multimedia Contents

Zhongsheng Cao, Zongda Wu, Yuanzhen Wang and Guiling Li
Institute of Database and Multimedia Technology, College of Computer Science and Technology,
Huazhong University of Science and Technology, Wuhan 430074, People Republic of China

Abstract: In our earlier studies, we have designed a general-purpose multimedia query language called UMQL, which allows users to query multimedia data based on their content information and then for its internal query representation, we have also designed an operator-based internal query algebra called UMQA, which has equivalent ability with UMQL on multimedia query specification, but focuses on internal query processing implementation. In this study, we discuss the query processing techniques for querying multimedia contents efficiently, namely, how to interpret and implement a UMQA-based query plan to obtain target multimedia data from a database efficiently. More specifically, we first of all discuss the efficient implementations of main UMQA operators. Then, we in theory analyze the execution costs for the implementation algorithms of UMQA operators and present the experimental results of performing these implementation algorithms on a prototype information system. Finally, the acceptable experimental results show that all the processing techniques proposed in this study for querying multimedia contents are feasible and applicable.

Key words: Multimedia database, multimedia query language, query algebra, query processing

INTRODUCTION

In the past decade, with the development of the internet and the availability of digital multimedia capturing devices such as image scanners, digital cameras and digital video cameras, the size of digital multimedia collection is increasing rapidly. Consequently, effective multimedia information retrieval and management techniques become more and more important. Multimedia content information represents what people sense when looking or listening, namely what are included by multimedia data and what features they behave themselves with, so it is more comprehensible for users and very important for content-based multimedia information retrieval. In recent years, many methods on extracting content information from multimedia original data have been proposed by Liu *et al.* (2007) and Christel and Hauptmann (2005). Presently, some high-level content information still can not be extracted effectively, but we believe that in the future more effective content extraction techniques will be proposed and more abundant multimedia contents will be extracted. Therefore, there should be a strong need to store, query and play multimedia content information from the multimedia databases.

A multimedia query language is a useful facility to specify users' multimedia query requirements and therefore is one of the most essential components in a multimedia database system. Hence, for querying multimedia contents effectively, a powerful and friendly query language must be supplied first for users. Although, traditional database query languages (e.g., SQL, OQL, etc.) have acquired great success, they do not suit uniform multimedia information retrieval, because the complex spatial and temporal relationships inherent in the wide range of multimedia data types make a multimedia query language different from its counterparts in traditional database management systems. In recent years, there have been many multimedia query language proposals (Tian *et al.*, 1999; Balkir *et al.*, 2002; Li and Ozsoyoglu, 1996; Lee *et al.*, 1999a, b, 2000), most of which are either designed for one particular medium (e.g., images), or specialized for a particular application (e.g., digital libraries), therefore also not competent for uniform multimedia information retrieval (Cao *et al.*, 2009).

In an earlier study (Cao *et al.*, 2007) of our project group, we have given a semi-structured data organization model and discussed a general-purpose multimedia query language called UMQL. It allows users to query various multimedia data uniformly based on their content information such as structure, feature, spatial relationship and temporal relationship. Subsequently, to supply a friendly interface for users, we designed and implemented a graphical environment (Wu *et al.*, 2008) which uses

Corresponding Author: Zhongsheng Cao, Institute of Database and Multimedia Technology,
College of Computer Science and Technology, Huazhong University of Science and Technology,
Wuhan 430074, People Republic of China

UMQL as its internal language and to check the correctness for any UMQL query given by users, we proposed a grammar analysis model and implemented a grammar analyzer (Cao *et al*., 2008; Huang, 2008). However, UMQL is a declarative textual query language as SQL or OQL, designed only for users to use and not suitable for internal implementation, so its query should be converted into some internal representation for being optimized and implemented efficiently. For the internal query implementation, we have also described an operator-based algebraic language called UMQA (Wu *et al*., 2009) which has equivalent capability with UMQL on multimedia query specification. UMQA is an internal algebra, designed for internal implementation, i.e., not for users to use directly. However, in the previous research, we haven't discussed the efficient implementations of UMQA operators.

Therefore, we in this study mainly discuss the processing techniques for querying multimedia contents efficiently, i.e., how to interpret and implement each operator in a UMQA-based query plan in order to obtain target multimedia data from a database efficiently. However, presently, most of query processing techniques (Brinkhoff *et al*., 1993; Graefe, 1993; Graefe *et al*., 1998; Chaudhuri, 1998; Papadias *et al*., 2003) are designed for the traditional database query algebras or themselves applications. Although, all these processing techniques have acquired well application results, they cannot be applied for multimedia query implementations, due to multimedia query particularities. So, the efficient implementations of some UMQA operators (especially for structure expansion and binary selection) need to be restudied. In the study, we propose a code index scheme for all objects in a multimedia database. Using the code index scheme, it is efficient to obtain all child objects for any object and it is also efficient to judge whether there is an ancestor-child relationship between two given objects. Then, we present two implementations for structure expansion operator, with and without the code index scheme and analyze and compare both execution costs in theory. We use a graph theory to illuminate the essential on evaluating a binary selection operator and present an implementation algorithm of binary selection and its improvement algorithm. Then, we in theory analyze and compare the execution costs for the two implementations. Lastly, we give the experimental results of performing all the four implementation algorithms on top of a prototype information system.

BACKGROUND: UMQL AND UMQA

UMQL (Cao *et al*., 2007, 2008; Wu *et al*., 2008; Huang, 2008) and its internal algebra UMQA (Wu *et al*., 2009) are both based on a semi-structured data organization model, which includes such basic notions as constructed data type, collection data type, object, child object and so on. These notions are briefly described below:

- A constructed data type is a composite structure of predefined data types (e.g., FLOAT, INTEGER etc.) collection data types and other constructed data types, whose instance is an object
- An instance of a collection data type is a composite value of one or more elements of the same data type
- Each basic item of an object is an attribute, whose value is called an attribute value of the object and also called child objects of the object, if the data type of the basic item is a collection data type or a constructed data type. To simplify presentation, a constructed data type represents both the data type name and the structure of objects belonging to the data type

The UMQL is a powerful multimedia query language, which allows users to query a variety of multimedia data uniformly based on their content information such as structure, feature, spatial relationship and temporal relationship. As SQL, UMQL uses the SELECT-FROM-WHERE statement for querying, but it extends the WHERE clause with new structure expression, feature expression and spatio-temporal expression, in order to accommodate to complex multimedia query requirements. Moreover, UMQL uses a variable to represent a group of content objects of the same data type, satisfying the same conditional restrictions. To show the syntax features of UMQL relevant to the use and querying of multimedia contents, we consider the following example of querying movies based on video contents. More specifically, the query retrieves Ang Lee-directed movies, each video of which contains one video clip that has not only more than fifty five video frames, but also three salient content objects: two horse and one sun, where the horse have more than 75% color feature similarity with the image horse.bmp and the sun is located above the horse. This query requirement can be described by UMQL as follows:

Example 1:

SELECT m. name, m. video **FROM** MOVIE m, PERSON d
WHERE clip (1) I N m. video. clips **AND** horse (2), sun (1) IN clip. objects # structure expression
AND d. id = m. director **AND** d. name = Ang Lee # join expression and normal expression
AND frame (clip) > 55 **AND** is (sun, sun) **AND** is (horse, horse) # feature expression
AND color (horse, horse. bmp) > 0.75 # feature expression
AND horse BEFORE [Y] sun. # spatio-temporal expression

Note that both m.video.clips and clip.objects, whose values are comprised of video clips and video salient objects, respectively are attributes of collection data types. In this example, firstly, we use a structure expression to define the inner structure of target multimedia data by declaring some new variables (i.e., clip, horse and sun) and containing relationships among these variables (i.e., clip contained by m; horse and sun contained by clip). And then we use a feature expression based on some feature functions (i.e., frame, is and color) to define the semantic notions and bottom-level features for salient objects represented by variables. Finally, we use a spatio-temporal expression to define the temporal relationship between the variables horse and sun (i.e., horse located under sun).

Below, we give a brief presentation on the semantics of structure expression, feature expression and spatio-temporal expression. A structure expression is used to support querying on the structure of multimedia contents, whose basic conditional item is described as $F = d_1(a_1), d_2(a_2), \ldots, d_n(a_n)$ IN $d.q_1.q_2.\ldots.q_m$, where d is a variable, d_i (i = 1, 2, …, n) is a new variable declared in F and contained by d, a_i is a positive integer and $d.q_1.q_2.\ldots.q_m$ is a path expression. Let $P_1, P_2, \ldots, P_n$ be the conditional restrictions operating on $d_1, d_2, \ldots, d_n$. Then any object o in the variable d satisfies F if and only if there exist $(a_1+a_2+\ldots+a_n)$ child objects in the structure expansion of o based on the path $d.q_1.q_2.\ldots.q_m$, where, a_1 child objects satisfy P_1, a_2 child objects of the rest satisfy P_2 and so on. A feature expression, whose implementation generally depends on a series of feature functions defined by system or users, is used to support querying based on bottom-level feature, semantic notion or other multimedia feature, through describing feature conditional restrictions for variables. A spatio-temporal expression is used to support querying based on spatial and temporal information inherent in a variety of multimedia objects, through defining spatio-temporal restrictions among variables. Its implementation depends on a set of spatio-temporal operators with a parameter, which can be used to represent the spatial relationships among spatial objects and the temporal relationships among temporal objects uniformly. Moreover, UMQL also includes join expression and normal expression (example 1) so as to keep compatible with SQL.

The UMQA, including a complete set of operators and translation formulas defined on these operators, is an internal algebra designed for internally representing and processing UMQL query. MQA includes the following six operators: join (Π), normal selection (σ^{NL}), structure selection (σ^{SE}), feature selection (σ^{FE}), spatio-temporal selection (σ^{SP}) and structure expansion (η). The first 5 operators are the logic implementations of join expression, normal expression, structure expression, feature expression and spatio-temporal expression, respectively and the sixth operator is used to expand objects to obtain their child objects based on a structure expression. Therefore, UMQA has the equivalent ability with UMQL in multimedia query specification. Using the mapping algorithm (Wu *et al*., 2009), we give a UMQA expression (i.e., a query plan) equivalent with the UMQL query in Example 1 as follows:

EXAMPLE 2:

π [m. name, m. video] (σ^{SE} [clip (1) **IN** m.video.clips **AND** horse (2), sun (1) **IN** clip.objects](
σ^{SP} [horse **BEFORE** [Y] sun]
$\sigma^{?FE}$ [frame (clip) > 55 **AND** is (horse, horse) **AND** is (sun, sun) **AND** color (horse, horse.bmp) > 0.75](
σ^{NL} [d.name = Ang Lee] (η [clip (1) **IN** m.video.clips **AND** horse (2), sun (1) **IN** clip.objects](
Π [d.id = m.director](ϵ [m←MOVIE](∅), ϵ [d←PERSON](∅))))))))

In this example, firstly, two scan operations (ϵ) are used to obtain two objects' sets named by MOVIE and PERSON from a multimedia database and bind them into the variables m and d, respectively. Secondly, structure expansion is used to expand the variable m based on the path expression m.video.clips so as to obtain all child objects for every object in m and bind these child objects into the variable clip and then the operator deals with clip similarly and produces the new variables horse and sun. Thirdly, normal selection, feature selection, spatio-temporal selection and structure selection are used to filter variables to remove those objects not satisfying the corresponding conditional restrictions. Finally, all objects in the variable m are target ones. In other words, for each object in m, there should be an child object in clip and for each object in clip, there should be two child objects in horse and one child object in sun. And such checks of residual objects in every variable are implemented by structure selection operations.

Moreover, to internally represent many collections of objects retrieved from a multimedia database, UMQA uses a query generation graph (Wu *et al*., 2009). In a query generation graph G (V, E), each vertex u of V (G) represents a variable bound with a collection of content objects and is comprised of a three-tuple form (N, O, R), where u [N] denotes the name of the variable, u [O] is a set of objects of the same data type bound to the variable and, u[R] employed to point out the immediate ancestor for each object in u [O] and each edge of E (G) connecting two vertices represents the ancestor-child relationship

between the variables denoted by the two vertices. The operands of UMQA operators are query generation graph. When a UMQA plan is implemented, all the operators act on a query generation graph, in which scan and structure expansion are used to obtain content information from a multimedia database and to construct a query generation graph for storing the information; then normal selection, structure selection, feature selection and spatio-temporal selection are used to remove the content objects stored in the vertex variables but not satisfying the given conditional restrictions.

UMQA OPERATOR IMPLEMENTATIONS

For a given UMQL query, to evaluate the query for obtaining target multimedia content information from a database, it only need to generate an equivalent UMQA query plan for the query and then to interpret and implement each operator of the query plan. A UMQA query plan mainly consists of the following seven types of operators, i.e., structure expansion, normal selection, structure selection, feature selection, spatio-temporal selection, scan and join. However, according to the amount of variables contained in the input operating expressions, UMQA selection operations can be separated into two categories (Wu *et al*., 2009), i.e., (1) monadic feature selection and monadic spatio-temporal selection (also called monadic selection together) and (2) binary feature selection and binary spatio-temporal selection (also called binary selection together). For the join, scan and monadic selection operators, their operations are identical approximately with those of the traditional relational algebraic system and thus their implementation algorithms are also identical with those of the relational algebraic system. Therefore, we in this study only discuss the efficient implementations for those new UMQA operators, including structure expansion, binary feature selection and binary spatio-temporal selection.

IMPLEMENTATION OF STRUCTURE EXPANSION

Here, we discuss the evaluation of structure expansion and gives two implementation algorithms, one that does use a code scheme and create index on the codes and the other that does not. We then compare the two implementations by computing both execution cost formulas.

We use the following physical storage model for a query generation graph, which is separated into two parts, one in the main memory and another in the outer hard disk. In memory, we store the main graph framework, including all edges and all vertices but only including the vertex name (u [N]) and the pairs of oids (u [R]), without the set of objects (u [O]). All the content objects of each vertex are stored in the hard disk and we only store one copy for all the variables that are expanded from the same structure expansion operator, i.e., for the variables d_1, d_2, …, d_n, that expanded from the same operator of structure expansion, we only in the hard disk store a union set of content objects, (d_1 [O]∪d_2[O]∪…∪d_n[O]).

Immediate implementation without coding: For the implementation of a structure expansion η [$d_1(a_1)$, $d_2(a_2)$, …, $d_n(a_n)$ IN $d.q_1.q_2.\ldots.q_m$], it first needs to expand each object in the variable d of the input query generation graph to obtain child objects, then bind all these child objects to every child variable d_1, d_2, …, d_n and last put these child variables into the input graph to output a new query generation graph. Hence, if the path expression $d.q_1.q_2.\ldots.q_m$ contains many attributes of collection data types, when the structure expansion operation is implemented immediately, it is required to access all these objects collections in a multimedia database. We below note the data type of the path $d.q_1.q_2.\ldots.q_m$ as TYPE ($d.q_1.q_2.\ldots.q_m$).

Algorithm 1: Structure expansion without coding

INPUT: a structure conditional item $v_1(a_1)$, $v_2(a_2)$, …, $v_n(a_n)$ **IN** $v.s_1.s_2.\ldots.s_m$ and a query generation graph G.
Read all the objects from the vertex named by "v" of the input graph G into the memory buffer; Name the memory partition with B1; { comment: all the content objects in each vertex are stored in the disk. }
Refresh G through adding n new vertices that are respectively named by v_1, v_2, … and v_n and adding n edges that respectively start from each of these new vertices to the vertex with the name v; k ←0;
for i←1 to m do k←k + 1;
if TYPE ($v.s_1.s_2.\ldots.s_i$) is a collection data type then
Read all pages from the collection named by TYPE ($v.s_1.s_2.\ldots.s_i$) of the multimedia database into the memory buffer; Name the memory partition with B2;
E_k←ø {**comment:** E_k is an assistant memory-based set comprising of pairs of oids, used to point out the parent content objects in B1 for every object in B2}.
for each content object a2 in B2 **do**
if there exists a content object a1 in B1, making that a1 contains a2. then
Put a pair (&a1, &a2) of oids into E_k; {comment: represents, a2 is a first generation child of a1}.
else {**comment:** there doesn't exist the target content object in B1}.
Remove the content object a2 from B2;
end if.
end for. Free the memory partition B1; Rename the memory partition B2 with B1;
end if.
end for. {**step 1** (from the line 1 to 15): obtain the oids of child objects for each object in the variable v}.
D←{(a, b)| ∃(a_1, b_1)∃(a_2, b_2∃...∃ (a_k,b_k) ((a_1, b_1)∈E_1, (a_2, b_2)∈ E_2,....,(a_k, b_k) ∈E_k, $b_1 = a_2$, $b_2 = a_3$,..., $b_{k-1} = a_k$, a_1, $b = b_k$)};
Copy D to the vertices of the graph G, with name v_1, v_2, …, or v_n;
for each content object a1 in B1 **do**
Output a1 into the "output page" B0 in the main memory buffer;
if B0 is full **then** {**comment:** B0 is a single page}.
Empty B0 by copying it into the outer disk as a page of the content objects' set of the vertices named by v_1, v_2, …, or v_n of the graph G;
end if.
end for. {**step 2** (from the line 16 to 24): construct and output the result query generation graph G.}
Copy B0 into the disk as the last page of the objects' set of the vertices "v_1", "v_2", …, or "v_n" of G;

We now analyze the execution cost in theory, by considering two essential factors: the number of disk accesses and the number of comparisons. We first assume that the expanded variable v of the input graph G is of size P_0 pages and each page owns N_0 objects. The objects' collection named by TYPE (v) in the multimedia database is of size P_0 pages (Obviously, each page also contain N_0 content objects). Assume that, the path expression $v.s_1.s_2.....s_m$ contains k ($k \geq 1$) attributes of collection data type and each corresponding collection in the multimedia database is of size P_i ($k \geq i \geq 1$) pages and each page owns N_i' objects. If the main memory buffer is of size P_M pages, then in Algorithm 1, we have made the assumption that, $P_M \geq (P_0 + P_1' + 1)$ and $P_M \geq (P_i' + P_{i+1}' + 1)$ for $k-1 \geq i \geq 1$. Then, we present the execution cost analysis for algorithm 1 as follows.

Number of disk accesses: In step 1 of algorithm 1, all the disk access operations are read ones. We first read all the content objects with P_0 pages in the variable v into the main memory buffer. Then, from the line 3 to 15, we in succession access the hard disk for every collection attribute of the path $v.s_1.s_2.....s_m$, so in all making $(P_1' + P_2' \ldots + P_k')$ disk read operations.

In step 2, there are all the disk writing operations. After step 1, all the objects in the memory partition B1 that need to written into the hard disk only are a child set of those in the collection TYPE ($v.s_1.s_2. \ldots .s_m$), so we need to estimate the number of the written objects. For every content object in the variable v, the average number of its child objects in the collection TYPE ($v.s_1.s_2. \ldots .s_m$) can be estimated as $(P_k'N_k')/(P_0'N_0)$. Thus, the number of the child objects in the memory partition B1 that need to be written is estimated as $(P_0N_0)(P_k'N_k')/(P_0'N_0)$, i.e., $(P_0P_k'N_k')/P_0'$.

So the total number of disk access operations in algorithm 1 is estimated as follows:

- **Step 1:** $P_0 + (P_1' + P_2' \ldots + P_k')$
- **Step 2:** $(P_0P_k'N_k') / P_0' / N_k' = (P_0P_k') / P_0'$; {one page can be written by N_k' content objects}

So in all, $P_0 + (P_1' + P_2' \ldots + P_k') + (P_0P_k') / P_0'$.

Number of comparisons: In step 1 of algorithm 1, all the comparisons concentrate on the sentences from the line 3 to 15. If using size (B) to represent the number of the elements in set B, then doing one judgment for the if condition in the line 8 need to make size (B1) comparisons and therefore from the line 7 to 13, in all we need to make such conditional comparisons with the number, size (B1)•log[2](size(B2)), because all the objects' collections are sorted as their oids.

In step 2, all the comparisons concentrate on the two sections, the line 16 to construct D and from the line 18 to 23 to write all the content objects in B1 into disk. To construct D, we need to join all the sets of E_1, E_2, ... and E_k.. The sets E_1, E_2, ... and E_k are well-sorted, so the equivalent join between arbitrary two E and E' of all these sets is only required to implement (size(E)+size(E')) comparison operations. For the for loop sentence, because the number of all the content objects on B1 is estimated as $(P_0P_k'N_k')/P_0'$, its required comparisons should be with the number 2 $(P_0P_k'N_k')/P_0'$, one for the for conditional judgment and another for the if one.

So the total number of comparisons in algorithm 1 is estimated as follows:

- **Step 1:** 2m {one for the for conditional judgment and another for the if conditional judgment} + $(P_0N_0)(\log[2](P_1'N_1') + 1)$ {in the first implementation, size(B1) = P_0N_0 and size(B2) = $P_1'N_1'$} + $(N_1'P_1'P_0'/P_0')(\log[2](P_2'N_2') + 1)$ {in the 2nd, size (B1) ≈ $N_1'P_1'P_0'/P_0'$ and size(B2) = $P_2'N_2'$. } + ... {in the 3rd implementation, in the 4th implementation, ...}, + $(N_{k-1}'P_{k-1}'P_0'/P_0')$ $(\log[2](P_k'N_k') + 1)$; {in the last, size (B1) ≈ $N_{k-1}'P_{k-1}'P_0'/P_0'$ and size(B2) = $P_k'N_k'$}
- **Step 2:** $(P_0/P_0')[(P_1'N_1' + P_2'N_2') + (P_2'N_2' + P_3'N_3') + \ldots + (P_{k-1}'N_{k-1}' + P_k'N_k')]$ {in the D construction, size (E_i) $(P_0/P_0')(P_i'N_i')$ for i = 1, 2, ..., k and we assume that $P_i'N_i' \geq P_{i-1}'N_{i-1}'$}, + 2 $(P_0P_k'N_k')/P_0'$; {to write all the content objects in B1 into disk}

So in all approximately equal to, $(P_0N_0)\log[2](P_1'N_1') + (N_1'P_1'P_0'/P_0')\log[2](P_2'N_2') + \ldots + (N_{k-1}'P_{k-1}'P_0'/P_0')\log[2](P_k'N_k') + (P_0/P_0')[(P_1'N_1' + P_2'N_2') + (P_2'N_2' + P_3'N_3') + \ldots + (P_{k-1}'N_{k-1}' + P_k'N_k')]$.

Implementation using code index scheme: Based on the observation that, all the content objects in a multimedia database don't hold too many generations children (generally not more than 10 generations), we propose a code scheme for each object in a multimedia database and then create index on these object codes. In this code scheme, we assign unique codes to every content object. Using the unique codes, it is effortless to judge whether there is an ancestor-child relationship between two content objects; and combining the index on the object codes, it is efficient to obtain all the child objects with any generation for any content object.

We now introduce how to assign unique codes for all the objects in a multimedia database. Assign unique codes for every content object in a multimedia database, making that, for every content object the prefix of its code equals to the code of any of its ancestor content objects.

More specifically, given a content object u, if its code is C(u), then the code for any immediate child object v of u is C(v) = C(u)• n (n≥1 and n is the serial number of the object v in all the immediate child objects of u). Therefore, it's the prefix code scheme also called Dewey decimal classification code (Chien *et al*., 2002), which has been widely applied. Utilizing the code scheme, we can judge whether two arbitrarily given objects contain an ancestor-child relationship; however, an operator of structure expansion is required to expand content objects to obtain all their child objects of any generation based on a path efficiently, hence, which yet can't be completed immediately by the code scheme. We below give some ourselves definition on top of the prefix code scheme and then based on it, give some useful properties.

Definition 1: Given two arbitrary content objects u and v in a multimedia database, if their prefix codes are respectively $C(u) = N_1 \bullet N_2 \bullet ... \bullet N_n$ and $C(v) = N_1' \bullet N_2' \bullet ... \bullet N_m'$ (n≥1, m≥1, N_i and N_j' are both positive integers for i = 1, 2, …, n and for j = 1, 2, …, m), then we define the arithmetic comparisons of the two codes as follows:

- C(u) is equal to C(v), i.e., C(u) = C(v), if and only if, n = m and, for i = 1, 2, …, n, have $N_i = N_i'$
- C(u) is greater than C(v), i.e., C(u) > C(v), if and only if, $N_1 = N_1'$, $N_2 = N_2'$, …, $N_m = N_m'$ (n > m), or there exists a non-negative integer t (t < n, t < m), making that, $N_1 = N_1'$, $N_2 = N_2'$, …, $N_t = N_t'$ and $N_{t+1} > N_{t+1}'$
- C(u) is less than C(v), i.e., C(u) < C(v), if and only if, $N_1 = N_1'$, $N_2 = N_2'$, …, $N_n = N_n'$ (n < m), or there exists a non-negative integer t (t < n, t < m), making that, $N_1 = N_1'$, $N_2 = N_2'$, …, $N_t = N_t'$ and $N_{t+1} < N_{t+1}'$

Remark 1: If given the prefix code of a content object, we can know the prefix codes of all its ancestor content objects, i.e., given an arbitrary content objects u, if its prefix code is $C(u) = N_1 \bullet N_2 \bullet ... \bullet N_n$ (n≥1), then the prefix code of its i-th generation ancestor object is $N_1 \bullet N_2 \bullet ... \bullet N_{(n-i)}$ (i = 1, 2, …, n - 1).

Remark 2: For two given content objects, there exists an ancestor-child relationship between them, if and only if, there exists a prefix relationship between them. Namely, for two arbitrarily given content objects u and v, if their prefix codes are $C(u) = N_1 \bullet N_2 \bullet ... \bullet N_n$ and $C(v) = N_1' \bullet N_2' \bullet ... \bullet N_m'$ (n≥1, m≥1), then the content object u is the ancestor of v, if and only if, n≤(m – 1) and $N_1 = N_1'$, $N_2 = N_2'$, …, $N_n = N_n'$.

Remark 3: If given the prefix code of a content object, we can know the domain of the prefix codes of its child content objects. Arbitrarily given a content objects u, if its prefix code is $C(u) = N_1 \bullet N_2 \bullet ... \bullet N_n$ (n≥1), then for any v of all its child content objects, have $N_1 \bullet N_2 \bullet ... \bullet N_n < C(v) < N_1 \bullet N_2 \bullet ... \bullet N_n + 1$, regardless of the generation of v to u.

Remark 1 and 2 are more obvious based on the construction of content object prefix codes, so below, we only give the proof for Remark 3 by using definition 1 on the arithmetic comparisons of two object codes.

Proof: For two arbitrarily given content objects u and v, we note their prefix codes as $C(u) = N_1 \bullet N_2 \bullet ... \bullet N_n$ and $C(v) = N_1' \bullet N_2' \bullet ... \bullet N_m'$ (n≥1, m≥1), respectively. If the content object v is one of all the child objects of u, then from the construction of assigning prefix codes for content objects, we first have n < m and $N_1 = N_1'$, $N_2 = N_2'$, …, $N_n = N_n'$. From Definition 1, we know that, $(C(u) = N_1 \bullet N_2 \bullet ... \bullet N_n) < C(v)$. From $N_1 = N_1'$, $N_2 = N_2'$, …, $N_n = N_n'$, we have $N_1 = N_1'$, $N_2 = N_2'$, …, $N_{n-1} = N_{n-1}'$, $N_n + 1 > N_n'$. So from Definition 1, we again have $C(v) < N_1 \bullet N_2 \bullet ... \bullet N_n + 1$. Therefore, we conclude that $N_1 \bullet N_2 \bullet ... \bullet N_n < C(v) < N_1 \bullet N_2 \bullet ... \bullet N_n + 1$. (END).

We note the prefix code of a given content object a as CODE(a). After completing to assign the prefix codes for all the content objects in a multimedia database, we create cluster index for every collection of content objects of the same constructed type, on the object prefix codes, based on the comparison relationships defined in Definition 1. We use the traditional B^+ tree for the cluster index creation.

In the implementation for expanding a given content object a based on a path "$v.s_1.s_2. \ldots .s_m$", we first obtain the code range (CODE(a), CODE(a) + 1) of the child objects of a and then combining the B^+ tree index, we can from the collection named by TYPE("$v.s_1.s_2. \ldots .s_m$") obtain all the content objects whose prefix codes are located in (CODE(a), CODE(a) + 1). Below, we give the particular implementation algorithm of structure expansion with using prefix codes and the index on them, by pseudocode.

Algorithm 2: Structure expansion with prefix code index

INPUT: a structure conditional item $v_1(a_1)$, $v_2(a_2)$, …, $v_n(a_n)$ **IN** "$v.s_1.s_2. \ldots .s_m$" and a query generation graph G.

Read the B^+ tree indexing structure for the collection named by TYPE("$v.s_1.s_2. \ldots .s_m$") from the multimedia database into the memory buffer; Name the memory partition with B0;

Read all the objects from the vertex named by "v" of the input graph G into the memory buffer; Name the memory partition with B1;

Refresh G through adding n new vertices that are respectively named by "v_1", "v_2", … and "v_n" and adding n edges that are respectively from each of the new produced vertices to the vertex with the name v;

E←ø {**comment:** used to point out the child objects in TYPE ($v.s_1.s_2. \ldots .s_m$) for every object in B1}

for each content object a1 in B1 **do**

Based on the B^+ tree index in B0, from the collection named by TYPE

Algorithm 2: Continued

($v.s_1.s_2. \ldots .s_m$) of the multimedia database, read all the content objects whose prefix codes are located in the code range (CODE(a1), CODE(a1) + 1), into the memory buffer; Name the memory partition with B2;
for each content object a2 in B2 do
Put a pair (&a1, &a2) of oids into E;
Output a2 into the "output page" B0 in the memory buffer;
if B0 is full **then** {**comment:** B0 is a single page}
Empty B0 by copying it into the outer disk as a page of the content objects' set of the vertices named by v_1, v_2, …, or v_n of the graph G;
end if.
end for. Free the memory partition B2;
end for.
Copy E to the vertices of the graph G, with name v_1, v_2, …, or v_n;
Copy B0 into the disk as the last page of the objects' set of the vertices v_1, v_2, …, or v_n of G;

Similarly, we assume that the expanded variable v in G is of size P_0 pages and each page owns N_0 content objects. Assume in the multimedia database, the collection named by TYPE (v) is of size P_0' pages (each page also contain N_0 objects) and the collection TYPE($v.s_1.s_2.\ldots.s_m$) is of size P_1 pages and each page owns N_1 objects. Obviously, $P_1 = P_k$' and $N_1 = N_k$'. Assume that, the B^+ tree index on the collection named by TYPE ($v.s_1.s_2. \ldots .s_m$) is of size P_2 pages and each page owns N_2 key words in average. If the main memory buffer is of size P_M pages, then in algorithm 2, we have made the assumption that, $P_M \geq (P_0 + P_2 + 1 + R/N_k')$ where, R represents the maximal number of child objects of each object in v. We below analyze the execution cost of algorithm 2 in theory.

Number of disk accesses: In algorithm 2, we first read all the P_0 pages of objects in the variable v and all the P_2 pages of the B^+ index tree into the main memory buffer. Then, we need from the collection TYPE ($v.s_1.s_2. \ldots .s_m$) to read all the content objects as the children of the objects in the variable v. Based on the B^+ index tree, we can immediately obtain the target child objects instead of reading all pages of TYPE ($v.s_1.s_2. \ldots .s_m$). For each object in v, the average number of its child objects in TYPE ($v.s_1.s_2. \ldots .s_m$) is $(P_k'N_k')/(P_0'N_0)$, so the number of the target child objects that need to be read into the memory buffer is estimated as $(P_0N_0)(P_k'N_k')/(P_0'N_0)$, i.e., $(P_0P_k'N_k') / P_0'$. Last, we need to write all these content objects again into the outer disk.

So the total number of disk accesses in algorithm 2 is estimated as follows:

- $P_0 + P_2 + 2 (P_0P_k' N_k') / P_0'/ N_k' = P_0 + P_2 + 2 (P_0P_k') P_0'$. { one page can be written by N_k' content objects}

Number of comparisons: In algorithm 2, the main comparison operations are to continuously search the B^+ index tree. For any B^+ tree, if its rank value is equal to m and the number of all its key words is equal to N, then the comparison number to search a key word is not more than $(\log[m/2]((N + 1)/2) + 1)$. For the B^+ index tree in algorithm 2, N is approximately equal to P_k' N_k' and, the total number for the search operations in all owns (P_0N_0).

So the total number of comparisons in algorithm 2 is estimated as follows:

- $P_0N_0 (\log[m / 2]((P_k' N_k' + 1) / 2) + 1) + 2 (P_0P_k'N_k') P_0'$. { for the 'for' loop sentence from the line 7 to 14}

Preliminary comparisons for two implementations: Now, we have given two implementation algorithms for structure expansion, one that uses the prefix codes and the index on them and one that does not. Both the algorithms have been coded into UMQL query processing. In the following experimental section, we will present the experimental results for performing the two algorithms on a prototype system; however, using the cost formulas derived for each algorithm, we here first make several observations.

- **Observation 1:** For an operator of structure expansion, the implementation algorithm with code index scheme always has smaller number of disk accesses than the implementation algorithm without using prefix codes
- **Observation 2:** For an operator of structure expansion, the implementation algorithm with code index scheme always has smaller number of comparisons than the implementation algorithm without using prefix codes

Observation 1 and 2 can be demonstrated effortless from the previous cost formulas. Compared with algorithm 1 that is an immediate implementation for structure expansion, algorithm 2 using the code index scheme can obtain all the child objects for a given expanded objects' collection by immediately accessing the target collection in the multimedia database, consequently avoiding to read those middle assistant collections. This reduces the number of disk accesses and the number of comparisons effectively.

IMPLEMENTATION OF BINARY SELECTION

Here, we discuss the evaluation for operators of binary feature selection and binary spatio-temporal selection and present two implementation algorithms, where the latter is the improvement of the former. Then by computing both execution cost formulas, we compare the two implementation algorithms.

Immediate implementation: From the definition on binary selection (Wu *et al*., 2009), we know that, for a binary selection σ [F], if u_1 and u_2 is the two variables contained in the binary conditional item F, then the binary selection operation can be illuminated briefly as follows. First all the content objects in u_1 or u_2 are classified into several groups, making in every group all the content objects of the same ancestor; Next for each group g_1 of objects in u_1 and each group g_2 in u_2, if both have the same ancestor, then generate a pair of object's groups (g_1, g_2); Last the binary selection predicate contained in F is applied to each pair of object's groups to remove all those objects not satisfying F. Hence, for the implementation of a binary selection operation, the essential problem is how to evaluate the binary selection function constructed based on the input binary conditional item. Below, we first use the graph theory to illuminate the essential on evaluating a binary selection function.

Before present discussion, we give some denotation specification that will be used in the following text:

- Z_1 and Z_2 are two groups of content objects and satisfying, any object in Z_1 is of the same ancestor with any in Z_2; (2) F is a basic binary conditional item and let VA(F) = {v_1, v_2} be the variables contained by F, $N(v_1)$ = n the correlative number of v_1 and $N(v_2)$ = m the correlative number of v_2; (3) P is the binary selection predicate contained by F; (4) Z_1'and Z_2' are the output of the binary function h ((Z_1, Z_2), (n, m), F). i.e., h((Z_1, Z_2), (n, m), F) = (Z_1', Z_2')

Definition 2: A unordered graph G = (V, E) is bipartite graph, if and only if, its vertices V can be partitioned into two disjoint subsets V_1 and V_2, such that, for the two vertices of every edge in G, one belongs to V_1 and another belongs to V_2. We note a bipartite graph G as G = (V_1, V_2, E). A bipartite graph G = (V_1, V_2, E) is a complete bipartite graph, if and only if, every vertex in V_1 are adjacent with every vertex in V_2. If $|V_1|$ = n and $|V_2|$ = m, then we call G as (nm) complete bipartite graph.

Based on the notions on bipartite graph and complete bipartite graph, we gives some useful properties, which denotes, to evaluate a binary selection function is equivalent to search complete bipartite child graphs from a bipartite graph.

Remark 4: Construct a unordered graph G_Z = (V, E) as follows: V←Z_1∪Z_2 (Please note that Z_1∩Z_2 = ∅); E ←{(o_1, o_2) | o_1∈Z_1, o_2∈Z_2, P(o_1, o_2) = true. }, then have that, G is a bipartite graph. In the following sections, we note the graph G_Z as G_Z= (Z_1, Z_2, E_Z).

Proof: Remark 4 is more obvious. From the construction for the graph G_Z (G_Z = (Z_1, Z_2, E_Z)), we know that, the vertices' set of G_Z is composed of two disjoint sets Z_1 and Z_2 and every edge of E_Z is connected from Z_1 to Z_2. So from Definition 2, we conclude that, G_Z is a bipartite graph.

Remark 5. If all the (nm) complete bipartite child graphs of G_Z (G_Z = (Z_1, Z_2, E_Z)) are listed as follows, G_1 = ($Z_1^{(1)}$, $Z_2^{(1)}$, E_1), G_2 = ($Z_1^{(2)}$, $Z_2^{(2)}$, E_2), …, G_r = ($Z_1^{(r)}$, $Z_2^{(r)}$, E_r) (r≥0, $Z_1^{(i)}$⊆Z_1, $Z_2^{(i)}$⊆Z_2, E_i⊆E_Z for i = 1, 2, …, r.), then we have the conclusions that, Z_1' = $Z_1^{(1)}$∪$Z_1^{(2)}$∪…∪$Z_1^{(r)}$; Z_2' = $Z_2^{(1)}$ ∪$Z_2^{(2)}$∪…∪$Z_2^{(r)}$.

Proof: First we prove Z_1'⊆$Z_1^{(1)}$∪$_1^{(2)}$∪…∪$Z_1^{(r)}$. For any content object o contained by V_1', from the definition on binary selection function (Wu *et al*., 2009), we know that, there exists a subset U_1 of Z_1, of size n objects and containing the content object o; there exists a subset U_2 of Z_2, of size m objects; and every content object in U_1 with every one in U_2 satisfy the binary predicate P. So we can construct a unordered graph G = (V, E) as follows, V←U_1∪U_2 (U_1∩U_2 = ∅); E←{(o_1, o_2) | o_1∈U_1, o_2∈U_2, P(o_1, o_2) = true}.We can conclude that, G is a (nm) complete bipartite graph and also a child graph of G_Z. Therefore, we have the conclusion that, the object o is also contained by $Z_1^{(1)}$⊆$Z_1^{(2)}$∪…∪$Z_1^{(r)}$, i.e., Z_1'∪$Z_1^{(1)}$∪$Z_1^{(2)}$∪…∪$Z_1^{(r)}$. Next we prove $Z_1^{(1)}$∪$Z_1^{(2)}$ ∪…∪$Z_1^{(r)}$⊆Z_1'. For any content object o contained by $Z_1^{(1)}$∪$Z_1^{(2)}$∪…∪$Z_1^{(r)}$, we know that, there certainly exists a (nm) complete bipartite graph G_i = ($Z_1^{(i)}$, $Z_2^{(i)}$, E_i) (r≥i≥0) containing o and $Z_1^{(i)}$⊆Z_1, $Z_2^{(i)}$ ⊆Z_2 and E_i⊆E_Z, i.e., there exist a subset $Z_1^{(i)}$ of Z_1 and a subset $Z_2^{(i)}$ of Z_2, making that, o∈$Z_1^{(i)}$, $|Z_1^{(i)}|$ = n, $|Z_2^{(i)}|$ = m and $\forall a \forall b\,(a \in Z_1^{(i)}, b \in Z_2^{(i)} \rightarrow P(a,b))$. Therefore we have the conclusion that, o is also contained by Z_1', i.e., $Z_1^{(1)}$ ∪$Z_1^{(2)}$∪…∪$Z_1^{(r)}$⊆Z_1'. Therefore, we have that, Z_1' = $Z_1^{(1)}$∪$_1^{(2)}$∪…∪$Z_1^{(r)}$. Similarly, we also can prove that, Z_2' = $Z_2^{(1)}$∪$Z_2^{(2)}$∪…∪$Z_2^{(r)}$ (END).

The implementation of a binary selection operator needs to continuously apply the binary selection function on every pair of groups of content objects for filtering all those non-target objects. However, remark 5 shows that, to evaluate the binary selection function is equivalent to search all the complete bipartite child graphs from a bipartite graph. Based on the discussion, we first give an immediate implementation algorithm for a binary selection operator.

Algorithm 3: Binary selection

INPUT: a basic binary conditional item F and a query generation graph G. Assume that, VA (F) = {v_1, v_2}, the correlative numbers of v_1 and $v_2$are n and m and the binary predicate contained by F is P.

Read all the content objects from the vertex named by v_1 of the query generation graph G into the memory buffer; Name the memory partition with B1;

Algorithm 3: Continued

Read all the content objects from the vertex named by v_2 of the query generation graph G into the memory buffer; Name the memory partition with B2;
for each group of objects Z1 in B1 with the same ancestor **do**
if there is a group of objects Z2 in B2, making any object in Z2 of the same ancestor with any in Z1 **then**
E1←ø; E2←ø; k←0;
for each child group Z1' with n elements in Z1 **do for** each child group Z2' with m elements in Z2 **do**
for each object a1 in Z1' **do for** each object a2 in Z2' **do**
if P(a1, a2) is true. then k←k + 1; **end if.** {**comment:** P is the binary predicate in the input F}
end for. end for.
if k = n • m then {**comment:** means that, Z1' and Z2' can construct a complete bipartite graph}
E1←E1∪{id | id∈&Z1'. }; E2←E2∪{id | id∈&Z2'};
end if. k←0;
end for. end for. {**comment:** from Z1 and Z2, search all the complete bipartite child graphs}
Remove from Z1 all the content objects whose oids are not contained by E1;
Remove from Z2 all the content objects whose oids are not contained by E2;
{**comment:** remove the objects in Z1 or Z2 will lead to remove their counterparts in B1 or B2}
end if. {**comment** (from the line 5 to 15): to evaluate the binary selection function}
end for. {comment: here, for the memory partitions B1 and B2, all their content objects that do not satisfy the input condition F have been removed}
Rewrite all the content objects in B1 and B2 into the disk as the objects' sets of the vertices named by v_1 and v_2 of the graph G; And refresh v_1[R] and v_2[R]; {comment: The graph G after refreshed becomes the final processing result of binary selection}

In algorithm 3, if the two vertices v_1 and v_2 of the input query generation graph are respectively of size M and N pages and after selected both are respectively of size M' and N' pages, then the number of accessing disk is (M + N + M' + N'). This can be seen as the minimal number for all the implementations of binary selection operators, so in the execution cost analysis for algorithm 3, we consider other two factors: the total number of evaluating the binary predicate P and the number of comparisons, which both can affect the algorithm efficiency essentially. We first assume that, in algorithm 3, the memory partitions B1 and B2 are respectively of size M_1 and M_2 content objects; every group Z1 of objects in B1 are of the same size, N_1 objects; every group Z2 of objects in B2 are also of the same size, N_2 objects; and in B1, in all there are M groups Z1 satisfying the if condition in the line 4 (Obviously, $0 \le M \le (M_1/N_1)$). Then, we present the execution cost analysis for algorithm 3 as follows:

- **Number of evaluating the binary predicate P:** For most of binary selection operations, especially for feature selection, the evaluations for their containing binary predicates (e.g., color and shape in example 1) are more expensive, so whose total evaluation numbers own great influence on the efficiency of implementation algorithm. In algorithm 3, we need to evaluate the binary predicate P(a1, a2) for any object a1 in Z1' and for any one a2 in Z2'; however, Z1' and Z2' are respectively of sizes, n and m, so, such one time implementation needs to make the predicate evaluation operations with the number (nm). For Z1, the number of its child groups Z1' is $\binom{N_1}{n}$ and for Z2, the number of Z2' is $\binom{N_2}{m}$, so for a pair of Z1 and Z2, the total number of evaluating the binary predicate P is $\binom{N_1}{n}\binom{N_2}{m}$

  So the total number of evaluating the binary predicate P in Algorithm 3 is equal to $\binom{N_1}{n}\binom{N_2}{m}$

- **Number of comparisons:** In algorithm 3, the evaluation for the binary predicate P in the line 8 is the most inner conditional judgment sentence and without other judgment sentences with the same level, so the number of comparisons in Algorithm 3 can be estimated approximately as $\binom{N_1}{n}\binom{N_2}{m}$

Implementation after improvement: The improvement of algorithm 3 is to reduce the execution cost as much as possible, namely to reduce the binary predicate evaluation number and the number of comparisons. For the former, we use a buffer of an adjacency matrix and for the latter, we give some new properties on bipartite graph and its complete bipartite child graphs to reduce the search space for evaluating the binary selection function.

Remark 6: For any content object o belonging to Z_1, if the total number of the content objects in Z_2 which satisfy the binary predicate P with o is r_1 ($0 \le r_1 \le |Z_2|$) and $r_1 \le (m-1)$, then we have that, $o \notin Z_1$'. Similarly, for any content object o belonging to Z_2, if the total number of the content objects in Z_1 which satisfy the binary predicate P with o is r_2 ($0 \le r_{2s} \le |Z_1|$) and $r_2 \le (n-1)$, then we have that, $o \notin Z_2$'.

Proof: Remark 6 is more obvious. From the definition on binary function, we have the conclusion that, for any content object o in Z_1', there must exist at least m content objects in Z_2, satisfying the binary predicate P with o, namely, if the total number of the content objects in Z_2 which satisfy the binary predicate P with o is less than m, then have $o \notin Z_1$'. Similarly, we can prove the later part of Remark 6 (END).

Remark 7: For any child bipartite graph G_C ($G_C = (V_1, V_2, E_C)$) of G_Z ($G_Z = (Z_1, Z_2, E_Z)$), satisfying that, $V_1 \subseteq Z_1$, $V_2 \subseteq Z_2$, $E_C \subseteq E_Z$, $|V_1| = n$ and $E_C = \{(o_1, o_2) \mid o_1 \in V_1, o_2 \in V_2, (o_1, o_2) \in E_Z\}$, if all the (nm) complete bipartite child graphs of G_C are $G_1 = (V_1^{(1)}, V_2^{(1)}, E_1)$, $G_2 = (V_1^{(2)}, V_2^{(2)}, E_2)$, …, $G_t = (V_1^{(t)}, V_2^{(t)}, E_t)$ ($t \geq 0$, $V_1^{(i)} = V_1$, $V_2^{(i)} \subseteq V_2$, $E_i \subseteq E_C$, for i = 1, 2, …, t.), then the graph G_M ($G_M = (V_1', V_2', E_M)$, $V_1' = V_1^{(1)} \cup V_1^{(2)} \cup \ldots \cup V_1^{(t)} = V_1$, $V_2' = V_2^{(1)} \cup V_2^{(2)} \cup \ldots \cup V_2^{(t)}$, $E_M = E_1 \cup E_2 \cup \ldots \cup E_t$.) is a (nu) ($u = |V_2'|$ and $u \geq m$) complete bipartite child graph of G_C and there doesn't exist a (nv) complete bipartite child graph of G_C making that $u < v$. So G_M is called one maximal (n) complete bipartite child graph of G_Z.

Proof: First we prove G_M is a complete bipartite child graph of G_C. Given any content object $o \in V_1'$, i.e. $o \in V_1$, $o \in V_1^{(1)}$, $o \in V_1^{(2)}$,…, $o \in V_1^{(t)}$, so there must exist m edges in E_1 that connect o with any object in $V_2^{(1)}$, m edges in E_2 that connect o with any object in $V_2^{(2)}$,… and m edges in E_t that connect o with any object in $V_2^{(t)}$, i.e., there must exist many edges in $E_M = E_1 \cup E_2 \cup \ldots \cup E_t$ that connect o with any object in $V_2' = V_2^{(1)} \cup V_2^{(2)} \cup \ldots \cup V_2^{(t)}$. Therefore, in the graph G_M, o is adjacent with any content object in V_2', namely, G_M is a complete bipartite graph. Besides, $V_1' \subseteq V_1$, $V_2' \subseteq V_2$ and $E_M \subseteq E_C$, so G_M is a complete bipartite child graph of G_C. Next we prove that, there must not exist a child complete bipartite graph G_O ($G_O = (V_1, V_2'', E_O)$, $V_2'' \subseteq V_2$, $E_O \subseteq E_C$) of G_C making that $|V_2''| > |V_2'|$. We assume G_O existent. Then, for any child set S with m elements of V_2'', we know that, in G_O every object in S are adjacent with every object in V_1 because G_O is a complete bipartite graph. Thus we have that, the child graph G (G = (V_1, S, E'), E' = { $(o_1, o_2) \mid o_1 \in V_1, o_2 \in S, (o_1, o_2) \in E_O$. }) is a (nm) complete bipartite graph, so $S \subseteq V_2'$, i.e., $V_2'' \subseteq V_2'$. This is incompatible with the previous assumption. So G_O is not existent, i.e., there doesn't exist a child complete bipartite graph G_O ($G_O = (V_1, V_2'', E_O)$) of G_C making that $|V_2''| > |V_2'|$ (END).

From Remark 7, we know that, to evaluate the binary selection function there is no need to search all the complete bipartite child graphs, but only need to search all the maximal complete bipartite child graphs. This is more efficient.

Definition 3: For a bipartite graph G = (V_1, V_2, E), $V_1 = \{a_1, a_2, \ldots, a_n\}$, $V_2 = \{a_1', a_2', \ldots, a_m'\}$, make M_{ij} be the number of the edges that connect a_i with a_j', then we call the matrix $(M_{ij})_{n \times m}$ as the adjacency matrix of G.

In Definition 3, we define the adjacency matrix for a bipartite graph, which is different with that of traditional graph matrix representation, because in a bipartite graph, every the vertex in V_1 or V_2 are not adjacent with any in the same group. We can produce the adjacency matrix M for Z_1 and Z_2, consequently avoiding to evaluate the binary predicate P repeatedly.

In the improvement algorithm of the immediate implementation for binary selection, we use remark 6 and 7 to reduce the search space for evaluating the binary selection function and use the adjacency matrix to store all the results of evaluating the binary predicate P, in order to reduce the predicate evaluation number. We below give the improvement implementation for a binary selection operator.

Algorithm 4: Binary selection after improvement

INPUT: a basic binary conditional item F and a query generation graph G. Assume that, VA(F) = {v_1, v_2}, the correlative numbers of v_1 and $v_2$are n and m and the binary predicate contained by F is P.
Read all the content objects from the vertex named by v_1 of the query generation graph G into the memory buffer; Name the memory partition with B1;
Read all the content objects from the vertex named by v_2 of the query generation graph G into the memory buffer; Name the memory partition with B2;
for each group of objects Z1 in B1 with the same ancestor **do**
if there is a group of objects Z2 in B2, making any object in Z2 of the same ancestor with any in Z1 **then**
Produce the adjacency matrix M for Z1 and Z2, namely, for any content object a1 in Z1 and any object a2 in Z2, if a1 and a2 satisfy P, then M[a1][a2] = 1, or else M[a1][a2] = 0;
for each object a1 in Z1 do if (M[a1][1] + M[a1][2] +…+ M[a1][size(Z2)]) < m **then**
Remove a1 from Z1; Remove the row corresponding to a1 from M;
end if. end for.
for each object a2 in Z2 do if (M[1][a2] + M[2][a2] +…+ M[size(Z1)][a2]) < n **then**
Remove a2 from Z2; Remove the column corresponding to a2 from M;
end if. end for. {**comment:** use Remark 5.6 to filter Z1 and Z2}
for each object a1 in Z1 **do**
for each child matrix M' of M with n rows, including the row corresponding to a1 **do**
for each column C of M' do if (M[1][C] + M[2][C] +…+ M[n][C]) = n **then**
Put the column number C into E;
end if. end for. {**comment:** search the maximal complete bipartite child graph G_M of M'}
if size(E) > m **then** {**comment:** means that, in M' there exist G_M}
Remove all the n content objects that correspond to each row of M' from Z1;
Remove all the size(E) objects that correspond to those columns in E of M' from Z2;
end if.
end for. Remove the row corresponding to a1 from M.
end for. {**comment:** for each object in Z_1, search its maximal complete bipartite graph}
end if. {**comment** (from the line 5 to 22): to evaluate the binary selection function}
end for.
Rewrite all the content objects in B1 and B2 into the disk as the objects' sets of the vertices named by v_1 and v_2 of the graph G; And refresh v_1[R] and v_2[R]; {**comment:** The graph G after refreshed becomes the final processing result of binary selection}

Similarly, we assume in algorithm 4 that, every group Z1 of objects in B1 are of the same size, N_1 ones; every group Z2 of objects in B2 are also of the same size, N_2 ones and in B1, in all there are M groups Z1 satisfying the

if conditional judgment in the line 4 sentence. Then, we present the execution cost analysis for Algorithm 4 as follows:

- **Number of evaluating the binary predicate P:** In algorithm 4, we only need to evaluate once the binary predicate P for any object a1 in Z1 with any object a2 in Z2; and, Z1 and Z2 are respectively of size, N_1 and N_2, so, such one time implementation need to make the predicate evaluation operations with the number $(N_1 \bullet N_2)$

So the total number of evaluating the binary predicate P in algorithm 4 is estimated as $(M \bullet N_1 \bullet N_2)$.

- **Number of comparisons:** In algorithm 4, first based on remark 6 to filter Z1 and Z2 needs to traverse twice the adjacency matrix M, so whose total comparison number is approximately equal to 2 $(N_1 \bullet N_2)$. We assume the operation can't filter any object in Z1 or Z2, i.e., when entering into the loop sentence from the line 12 to 22, the sizes of Z1 and Z2 are still N_1 and N_2. And we assume the worst instance that for every child matrix M' of M, there doesn't exist any maximal complete bipartite graph. Then, for each object a1 in Z1 to search all the maximal (n) complete bipartite graphs that contains a1 need to make $\binom{N_1 - i}{n-1}$ $(N_1 \geq i \geq 1)$ attempts; however, each attempt needs to make (nN_2) comparisons

So in the worst instance, the number of comparisons in Algorithm 4 is estimated as follows:

$$2(MN_1N_2) + M(nN_2)\binom{N_1 - 1}{n-1} + M(nN_2)\binom{N_1 - 2}{n-1} + \ldots + M(nN_2)\binom{n}{n-1} + M(nN_2)\binom{n-1}{n-1} \approx M(nN_2)\binom{N_1}{n-1}$$

Preliminary comparisons for two implementations: In the previous two sections, we have given an implementation algorithm for a binary operator and its improvement algorithm. And both are coded into UMQL query processing. By using the cost formulas derived for each algorithm, we here first make several observations.

Observation 3: For the two implementation algorithms for an operator of binary selection, algorithm 3 and 4, the latter always has smaller number of evaluating the binary predicate P than the former.

Observation 4: For the two implementation algorithms for an operator of binary selection, algorithm 3 and algorithm 4, the latter always has smaller number of comparisons than the former.

Observation 3 and 4 can be proved from the previous cost formulas. Algorithm 4 is the improvement of algorithm 3, which use the graph properties to reduce the search space for evaluating the binary selection function and use the adjacency matrix to store all the results of evaluating the binary predicate P, to reduce the predicate evaluation number. These all make algorithm 4 more efficient.

SYSTEM AND EXPERIMENT

We have designed and implemented a prototype system (Wu *et al*., 2008; Cao *et al*., 2008; Huang, 2008) for UMQL-based multimedia information retrieval, running on Windows XP or Windows NT and developed using Visual C++ for its system server and Visual Basic for its some clients. The prototype system uses the client server methodology into its architecture, namely consists of a system server, many clients and socket application program interfaces used to connect the server with the clients (Fig. 1), where those functional components that are correlative with the processing techniques proposed in this study are shaded with points.

The system server (UMQL Server) is the essential component, whose main function is to process the UMQL queries from many clients. More specially, it would check the grammar correctness of a UMQL query; translate the UMQL query if owning correct syntax and semantics to its equivalent UMQA expression; optimize the UMQA expression to produce its query plan; interpret and implement all the operators of the UMQA query plan bottom-up; and last return the query results to the clients. All these functions are completed respectively by the child components as syntax analyzer, semantic analyzer, algebraic translator, algebraic optimizer and plan interpreter. Besides, the server is a multi-threaded application, namely for each request from client, it would create a service thread.

In the system, there may be many clients that all connect to the UMQL server using the socket application program interface. The common function of these clients is to supply a friendly interface for users, accept and transmit the users' input UMQL queries to the server and receive and display the multimedia presentations from the server. The clients that we presently have implemented include a UMQL textual environment and a VMQ visual one, where the former is a textual interface, whose implementation is relatively effortless and the latter is

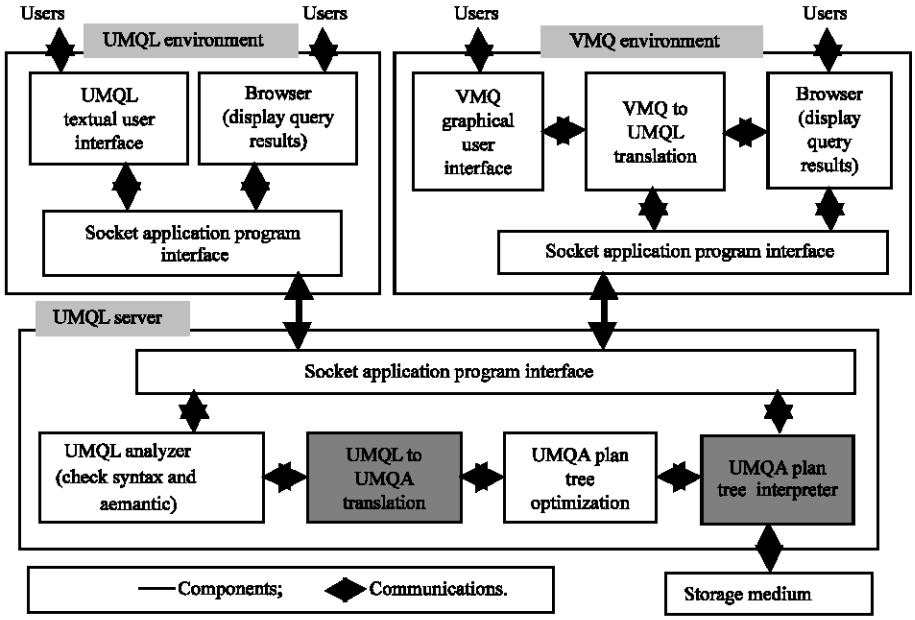

Fig. 1: Architecture of UMQL-based multimedia retrieval system

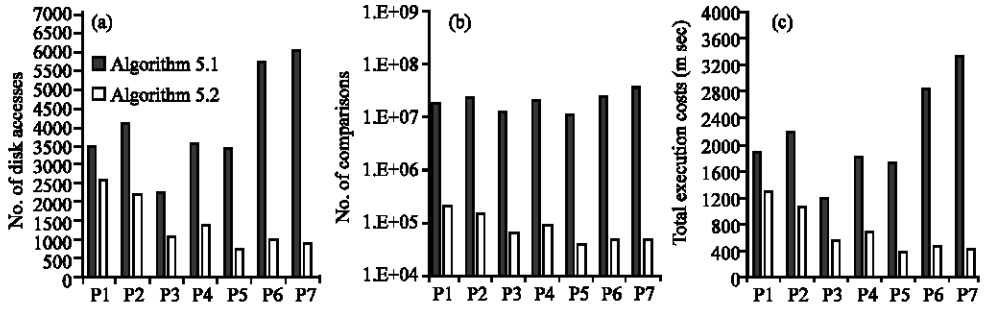

Fig. 2: Performance comparisons on structure expansion implementations, (a) performance on accessing disk, (b) performance on making comparisons and (c) efficiency performance

based on a visual query language called VMQ (Wu *et al*., 2008), of the equivalent ability with UMQL on multimedia query specification, which is a icon-based graphical query language.

Using the prototype system, we performed experiments on an Intel Pentium IV 1.6 GHz machine with 512 MB RAM and running Windows XP Professional OS, in order to assess the effectiveness of our implementation algorithms for new UMQA operators.

For the implementations of structure expansion operators, algorithm 1 and 2, their experiments were run on a series of randomly generated objects' collections, where collections sizes ranged from 10 to 2000 pages (each page with 32 kb) and each page of every collection assigned to contain 64 content objects. In each experiment, the two algorithms owned the same input objects' collections, but for algorithm 1, they were all sorted by oids and for algorithm 2, they were assigned with unique prefix codes and there was a B^+ tree cluster index with rank 50 on the codes. For our experiments, we generated seven structure expansion operations, P_1 to P_7, where P_n (n = 1, 2, …, 7) had (n + 1) collection attributes in its input path expression and the size of each objects' collection was increased gradually. Then, we experimented to see how the two implementation algorithms behave on accessing disk number, comparisons number and total execution costs. The experimental results are presented as Fig. 2.

In Fig. 2a, a comparison of the performance on accessing disk shows that algorithm 2 that is with code index scheme always has smaller number of disk accesses than algorithm 1 without using prefix codes, which is

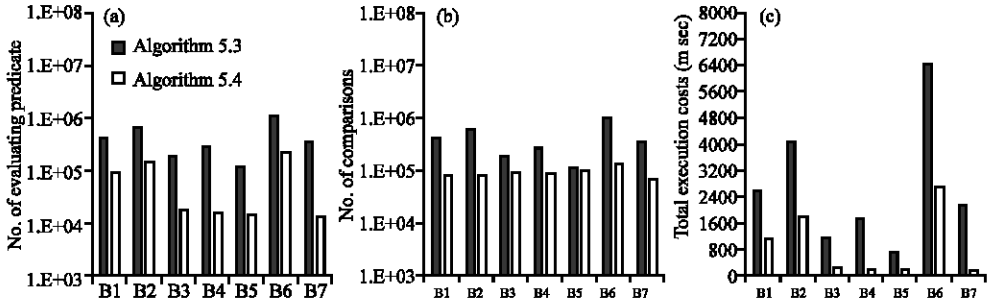

Fig. 3: Performance comparisons on binary selection implementations, (a) performance on evaluating P, (b) performance on making comparisons and (c) efficiency performance

accordant with observation 1 and besides, with the number increase of middle objects' collections (namely, with the increase of collection attributes in the input path), algorithm 2 behaves with better and better accessing disk performance. In Fig. 2b, the experiments on the number of making comparisons show that algorithm 2 always has smaller number of making comparisons than algorithm 1, which is accordant with observation 2 and besides, because to search all the children for a given content object is very time-consuming for algorithm 1 but just opposite for algorithm 2, the performances for the two implementations on comparisons number are with difference of several magnitudes. Therefore, in Fig. 2c, a comparison for the total execution performance shows that, for the two implementation algorithms of structure expansion operators, algorithm 1 and 2 both behave as acceptable performance (not more than 3000 m sec), but algorithm 2 always behaves with the better execution efficiency than algorithm 1 (not more than 1000 m sec) and besides, the total algorithm execution costs are mainly dominated by accessing disk operations.

For the implementations on binary selection operators, algorithm 3 and 4, each experiment of them was run on a pair of randomly generated objects' collections, where collections sizes ranged from 1000 to 2000 pages, each page also assigned to contain 64 content objects and all the content objects of the same collection sorted by their oids. For each pair of objects collections, every object's group of the same ancestor was of size varied randomly from five to ten objects and there was its counterpart in the other collection. For each experiment, the two algorithms owned a same input pair of objects' collections. For our experiments, we generated seven binary feature selection operations, B_1 to B_7, where in each selection operation, the variable correlative numbers n and m were varied randomly from one to three and the feature binary predicate was same, whose evaluation cost was approximately equivalent with 1000 normal comparison operations. Then we experimented to see how the two algorithms behave on predicate evaluation number, comparisons number and total execution costs. The experimental results are presented as Fig. 3.

In Fig. 3a, a comparison of the number on evaluating relatively more expensive binary feature predicate shows that for the two implementations of binary selection operators, algorithm 4 always has smaller predicate evaluation number than algorithm 3, accordant with observation 3 and the number difference is with some magnitudes, which denotes that the adjacency matrix buffer technique that is introduced into algorithm 4 is effective. In Fig. 3b, the experiments on the number of making comparisons show that algorithm 4 always has smaller number of making comparisons than algorithm 3, accordant with observation 4 and for algorithm 3, its comparisons number is approximately accordant with its predicate evaluation number. In Fig. 3c, a comparison of the total performance shows that, algorithm 1 and 2 both behave as acceptable performance (not more than 6000 m sec), but algorithm 4 always behaves as the better execution efficiency than algorithm 3 (not more than 2000 m sec). In the experiments, the correlative numbers and the size of objects' group pair have important influence on the algorithm total performance.

CONCLUSION

In this study, we have given the processing techniques for querying multimedia content information efficiently, i.e., mainly discussed the efficient implementations of UMQA new operators and given their implementation algorithms. These algorithms have been coded into the UMQL-based multimedia information system. Finally, the experimental results of performing these implementation algorithms show that the processing

techniques proposed in this study for querying multimedia content information are feasible and applicable.

ACKNOWLEDGMENTS

This study is partially supported by the National High-Tech Research and Development Plan of China under Grant Nos. 2006AA01Z430.

REFERENCES

Balkir, N., H.G. Ozsoyoglu and Z.M. Ozsoyoglu, 2002. A graphical query language: VISUAL and its query processing. IEEE Trans. Knowledge Data Eng., 14: 955-978.

Brinkhoff, T., H.P. Kriegel and B. Seeger, 1993. Efficient processing of spatiol joins using R-trees. Proceedings of ACM SIGMOD International Conference on Management of Data, June 1, Washington, DC., USA., pp: 237-246.

Cao, Z.S., Z.D. Wu and Y.Z. Wang, 2007. UMQL: A unified multimedia query language. Proceedings of the IEEE International Conference on Signal Image Technology and Internet Based Systems, Dec. 2007, Shanghai, China, pp: 101-107.

Cao, Z.S., Z.D. Wu and Y.Z. Wang, 2008. A grammar analysis model for the unified multimedia query language. J. Elect. Sci. Technol. China, 6: 87-93.

Cao, Z.S., Z.D. Wu and Y.Z. Wang, 2009. Multimedia query languages and their design criteria. Comput. Sci. China, 36: 9-14.

Chaudhuri, S., 1998. An overview of query optimization in relational systems. Proceedings of ACM SIGACT-SIGMOD-SIGACT Symposium on Principles of Database Systems, June 1-4, ACM, New York, pp: 34-43.

Chien, S.Y., Z. Vagena and D. Zhang, 2002. Structural joins: A primitive for efficient XML query pattern matching. Proceedings of the IEEE International Conference on Data Engineering, IEEEICDE'2002, San Jose, USA, pp: 141-152.

Christel, M.G. and A.G. Hauptmann, 2005. The Use and Utility of High-Level Semantic Features in Video Retrieval. In: Image and Video Retrieval, Leow, W.K. *et al.* (Eds.). LNCS., 3568, Springer Verlag, Berlin, Heidelberg, ISBN 978-3-540-27858-0, pp: 134-144.

Graefe, G., 1993. Query evaluation techniques for large databases. ACM Comput. Surveys, 25: 73-170.

Graefe G., R. Bunker and S. Cooper, 1998. Hash joins and hash teams in microsoft SQL server. Proceedings of International Conference on Very Large Databases, Aug. 24-27, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA., pp: 86-97.

Huang, D.W., 2008. Study on Query Analysis for a Multimedia Query Language UMQL. Huazhong University of Science and Technology, Wuhan, China.

Lee, T., L. Sheng, T. Bozkaya, N.H. Balkir, Z.M. Ozsoyoglu and G. Ozsoyoglu, 1999a. Querying multimedia presentations based on content. IEEE Trans. Knowledge Data Eng., 11: 361-385.

Lee, T., Z.M. Ozsoyoglu and G. Ozsoyoglu, 1999b. A graph query language and its query processing. Proceedings of the IEEE International Conference on Data Engineering, Mar. 1999, Sydney, Australia, pp: 1-1.

Lee, T., L. Sheng and N.H. Balkir, 2000. Query processing techniques for multimedia presentations. Multimedia Tools Appli., 11: 63-99.

Li, J. and Z.M. Ozsoyoglu, 1996. Processing OODB queries by o-algebra. Proceedings of the ACM International Conference on Information and Knowledge Management, Nov. 1996, Rockville, Maryland, USA, pp: 1-1.

Liu, Y., D.S. Zhang, G.J. Guo and W.Y. Ma, 2007. A survey of content-based image retrieval with high-level semantics. Pattern Recogn., 40: 262-282.

Papadias, D., J. Zhang, N. Mamoulis and Y. Tao, 2003. Query processing in spatial network databases. Proceedings of International Conference on Very Large Databases, Sept. 9-12, Berlin, Germany, pp: 802-813.

Tian, Z.P., H.R. Dang and A.Y. Zhou, 1999. Multimedia object query language and its query processing. Chinese J. Software, 10: 694-701.

Wu, Z., Z. Cao and Y. Wang, 2009. UMQA: An internal algebra for querying multimedia contents. Inform. Technol. J., 8: 411-426.

Wu, Z.D., Z.S. Cao and Y.Z. Wang, 2008. Design and implementation of a visual multimedia query language. J. Huazhong Univ. Sci. Technol., 36: 45-56.