Image Domain Formalization for Content-Based Image Retrieval

Caetano Traina Jr.
Computer Science
Department - ICMC
University of Sao Paulo at Sao Carlos - USP
Brazil
ciaetano@icmc.usp.br

Josiel M. Figueiredo
Computer Science
Department - ICET
Federal University of Mato Grosso
Brazil
josiel@icmc.usp.br

Agma J. M. Traina
Computer Science
Department - ICMC
University of Sao Paulo at Sao Carlos - USP
Brazil
agma@icmc.usp.br

ABSTRACT
This paper proposes a formal representation of the operations required to perform content-based image retrieval (CBIR) in large relational databases, using similarity queries. In this paper, we consider similarity as a numerical value obtained comparing a pair of images, which is calculated by a distance (dissimilarity) function. Distance functions usually rely on a set of features extracted from each image through a set of image processing algorithms called feature extractors. Before extracting features, other image processing algorithms are usually employed to pre-process each image, preparing it for the extractors. Usually there are several criteria that can be considered when measuring how much two images are similar. Therefore, to compare images in current CBIR environments one must define (1) the criteria, (2) the image pre-processing needed before the extractors can be executed, (3) which are those extractors, (4) which features must be considered, (5) and which distance function must be used. All of these definitions must have been set before a comparison can be performed. The complexity of defining how to compare images has lead to the development of systems aiming CBIR that allow relatively few options to configure the image comparison operations. Moreover, no formal representation of the entire CBIR process exists. In this paper we present such a formal environment, where all above-mentioned definitions are represented, entailing the development of flexible and highly-configurable CBIR systems. We also report a system developed using this formalism that enables the content-based retrieval of medical images from a hospital database, thus showing results of applying the presented formalism in a real environment.

Categories and Subject Descriptors
H.2.4 [Database Management]: Systems—multimedia databases, relational databases; H.2.8 [Database Management]: Database Applications—image databases; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—information filtering, query formulation, retrieval models, search process, selection process

General Terms
Algorithms, Management, Performance, Design, Experimentation, Languages.

Keywords
content-based image retrieval, relational databases, query algebra, similarity queries

1. INTRODUCTION
Comparing images is a complex process, aggravated by the lack of a precise definition of how to perform the comparison. There are many aspects of images that can be considered in the comparisons, thus leading to several comparison criteria. For example, one could be interested in images with similar colors, or with similar distribution of colors, or that have the same objects, among others.

Content-Based Image Retrieval Systems (CBIR) aims at searching in large image databases, retrieving those images whose contents meet a given search predicate. The search predicate usually means comparing the stored images with a image given as part of the predicate, called the query center. Comparisons are not performed on the images, but rather on features extracted from them and arranged as vectors, following the same structure for every image in the database[9]. Therefore, the resulting set of feature vectors can be compared using a dis-similarity function, also called a distance function. To speed up the searches the features are inserted in the database. Thus, at query time, only the feature vectors from the query center needs to be calculated. Using different sets of features from each image in a search operation leads to different comparison criteria.

There already exists several CBIR systems, and albeit they have differences in their functionality, everyone follows the same sequence of steps to execute a retrieval operation [7]: preprocess the images, extract the features, create the feature vectors, compare pairs of feature vectors. No one of the existing systems presents those four operations separately as described here, but rather treat the entire process as a monolithic unit. Nonetheless, each one of these four operations can have a large impact in the process of retrieving images by content from large image databases, and many researches are actively studying and improving them. Therefore, the identification of the basic components involved in a
CBIR process and a formal model of the whole process are important steps in developing more flexible and more easily configurable systems.

CBIR is a rising research area, with several different approaches, and its integration with the relational database technology has demanded several efforts. Major approaches in commercial systems include CBIR functionalities as extensions to the core engine, where the most known examples are the QBIC system included in DB2 Digital Library [4], and the Virage in Oracle [2]. Some academic efforts store the feature vectors in relations [5] while others integrate the use of a spatial [6] or a metric [3] access method.

Every existing strategy concentrates in processes occurring at the query time, disregarding the control over the whole CBIR process. Probably this is due to the vision of the similarity queries being focused only on the definition of operators considering data that are already stored in the database [5]. This direct approach simplify the overall complexity of CBIR processing, but tends to support adequately only rigidly predefined similarity queries.

In this paper we present a formal model of the whole CBIR process, aiming at including the image retrieval in a relational database management system, so the retrieval of images can be executed just as one more comparison predicate. The proposed solution can employ the resources of existing databases, creating an open architecture where image retrieval can be easily and powerfully managed.

The remainder of this paper is organized as follows. Section 2 describes the proposed formal model for the CBIR process. Section 3 presents how this model can be implemented in a system centered in a commercial database management system. It also reports a solution adopted to implement it in a system that is being used to perform content-based retrieval of images of medical exams in a hospital database, and Section 4 reports experimental results obtained using that system. Finally, Section 5 presents conclusions and discusses the main contributions of this work.

2. CONCEPTS

This section specifies a model to represent processes performing content-based image retrieval from image databases, considering that each process is composed of the following four basic components: image preprocessors, feature extractors, constructors for feature vectors, and distance functions. The model is detailed in the next subsections.

2.1 Images as an Attribute Type

In this section we consider that images should be stored in relations as values of attributes of the data type “StillImage” [1]. As an example, suppose a health care center storing the images from medical exams of its patients. It is common that a given type of exam includes more than one image. For example, a chest X-Ray exam usually has two frontal views of the lungs, and a brain tomography exam includes an array of several axial views of the head. Those data could be stored in a database relation having a schema as follows.

\[ \text{Patient} = \{ \text{Name, age, ChestV1XR, ChestV2XR, BrainCT} \} \]

Images stored in the ChestV1XR and ChestV2XR are in the same domain, so they can share the same comparison rules, but the BrainCT images are in a distinct domain, so they should potentially have comparison rules distinct from the other two. In fact, comparing two images from different domains is as meaningless as comparing two integer numbers from distinct domains, say age and weight.

To express the rules that must be followed when images from a given domain are compared, we defined the concept of a “domain expression”. The next section develops the concept of domain expressions, aiming to define how to compare and index images of a given domain in a relational database.

2.2 Domain Expressions

A domain expression \( \exp \text{(domain)} \) states a sequence of processes that must be executed over each image from a given domain when a pair of them must be compared. It requires the concepts described as follows.

An image environment is a pair \( \Phi = (\mathbb{I}, L) \), where \( \mathbb{I} = \{ I_0, I_1, I_2, \ldots \} \) is an unbounded array of images, and \( L \) is the index of one of such images, called the ‘current image’. Every image \( I_l \) \( l \leq 0 \) is always a null image. The image environment at the beginning of each expression execution always has \( L = 1 \) and every image \( I_l \in \mathbb{I} \) is set to the null image.

A feature set is represented as \( S = \{ a_1, a_2, \ldots, a_n \} \), where each feature \( a_i \) is a pair \( \text{feature name, feature value} \) describing meaningful properties of an image.

An image operand \( \lambda \) is a pair \( \lambda = (\Phi, S) \), where \( \Phi \) is an image environment, and \( S \) is a feature set.

An image processor represents the basic component for both image preprocessors and feature extractors from a CBIR process and is defined as a function \( \theta(\arg \bot s_i) : \lambda \rightarrow \lambda \) that changes any number of images \( I_l, i > L \), and add any number of new features into \( S \). The particular changes made in the images in \( \mathbb{I} \) and features added to \( S \) depend on the definition of each processor \( \theta \). Notice that \( \arg \) represents input arguments that govern the execution of the processing, and \( s_i \) represent the features extracted.

There are at least four kinds of predefined image processors, as follows. An image processor is called a feature extractor when no image in \( \mathbb{I} \) is changed but new features, extracted from \( I_L \), are added to \( S \). An image processor is called an image synthesizer when only the image \( I_{L+1} \) is changed, without depending on any other image \( I_l \), and \( S \) is unchanged. An image processor is called an unary image operator when only the image \( I_{L+1} \) is changed, the changes depend on the current image \( I_L \), and any number of new features are added to \( S \). An image processor is called a binary image operator when only the image \( I_{L+1} \) is changed, the changes depend on the current image \( I_L \) and \( I_{L-1} \), and any number of new features are added to \( S \).

Other image processors can be defined, but those four are the most common, and they preserve important properties that can be used to warranty consistence in database data, as we will show further.

Figure 1 presents a summary of the changes that each kind of image processor causes on the image operand, and the data it depends on. In this figure, \( f_2() \) represents a function that returns numerical values, and \( f_1() \) represents a function that returns one image. They specify the existing data demanded to process each function.

A domain expression can be as simple as an image operand \( \lambda \) or it can be the result of executing several image processors sequentially as in \( \theta_1 : \theta_2 : \ldots \), where each image processor
works on the image environment resulting from its predecessor. We use the ‘:\’ symbol to express the sequence of image operands, so that \( \theta_1 : \theta_2 \) indicates that \( \theta_2 \) is processed after processing \( \theta_1 \).

The execution of an image processor \( \theta(\arg \perp s_i) \) in a specific place in the domain expression is controlled by its arguments \( \arg \). To assure consistency and repeatability regarding multiple comparisons of images stored in a database, the arguments must be only constants or feature values previously extracted in this execution of the domain expression.

Besides image processors, there are at least three other kinds of operators: the projection, the persistence processors and the two control processors Pipeline and Fork. The control processors organize the execution flow of the other processors, replicating the current image operand into one or more image operands, as follows. The Pipeline control processor, denoted as \( \lambda_0[\lambda_1] \), replicates \( \lambda_0 = (\langle I, L \rangle, S) \) into \( \lambda_1 = (\langle I, L + 1 \rangle, S) \). The Fork control processor \( \lambda(\lambda) \) replicates \( \lambda \rightarrow \{ \lambda \}^+ \), that is, any number of image operands equal to the original operand \( \lambda \).

The result \( \lambda_p \) of the processor Fork \( \lambda_0(\lambda_1, \lambda_2, \ldots, \lambda_n) \) is as follows: the resulting set of features \( S_p = \bigcup_{i=1}^{n} S_i \), and the image environment \( \Phi_p = (I_p, L) \) maintains the same current image \( I_p = I_0 \) from \( \lambda_0 \) and replaces the images following the current one such that \( I_{p+i} = I_{i+1} \) from \( \Phi_i \). \( I_L \) is the image \( I_{L+1} \) from \( \Phi_1 \). \( I_{L+2} \) is the image \( I_{L+2} \) from \( \Phi_2 \) and so on, up to \( I_{L+n} \) is the image \( I_{L+n} \) from \( \Phi_n \).

The Projection operator \( \Pi(S_i)(\lambda_0) \rightarrow \lambda_1 \) projects the feature set \( S_i \) from \( \lambda_0 \) into the feature set \( S_i \) from \( \lambda_1 | S_i \subseteq S_0 \).

There are two persistence processors. Their objective is to link the data generated in the image processing flow to the database as follows. The persistence processor FeatureVector \( \Delta_S(\text{AttributeName}) \) stores the current feature set, taken as a feature vector, as the value of the row-typed attribute named \( \text{AttributeName} \). The persistence processor SimilarityCriterion \( \Delta_X(\text{CriterionName}, df()) \) prepares the current feature set as a feature vector named \( \text{CriterionName} \), so it can be used to compare or index stored images following the \( \text{CriterionName} \) criterion, using the distance function \( df() \). Therefore, a \( \Delta_X \) processor defines the current set of features as the parameters to be processed by the distance function, to compare two images of the domain regarding the corresponding criterion.

The image processors are defined by domain specialists.
TheImage() : 

Histogram (256 ⊥ h[c][256]) : ∆X (Color, L1()) : 

TexturedShape (5 ⊥ e[5], t[5], v[5]) : 

[ Π(e[1 − 5], t[1 − 5]) : ∆X (Texture, L2()) , 

[ Histogram(5 ⊥ h[5]) : Π(h[5], h[1 − 5]) : ∆X (TexturedObjects, L1()) ] ]

This expression works as follows. Initially the image environment has a set of null images and L = 1. The synthesizer TheImage() gets the image to be analyzed, setting it as the current image I1.

The feature extractor Histogram(256 ⊥ h[c][256]) generates the color histogram of this image with 256 bins into the vector h[c]. The next operator is the SimilarityCriterion operator ∆X (Color, L1()). As the feature set S now holds the color histogram h[c], it is used to compare images regarding the Color criterion, using the L1 (Manhattan) distance function. The current image remains as the original image and thereafter the unary image operator TexturedShape(5 ⊥ e[5], t[5], v[5]) analyzes its texture to segment the image considering the 5 most distinct regions. The segmented image is set as I2, and the three five-dimensional vectors e[5], t[5] and v[5] are added to S.

The next operator is the fork operator || with two image operands. The two parallel processing paths start with the feature set S with the four vectors (h[c][256], e[5][5], t[5][5]) and the two images (the original image I1 and the segmented one I2). The first parallel path projects the feature set through Π(e[1 − 5], t[1 − 5]), retaining all the elements (of indexes 1 to 5) of vectors e and t. Thereafter, the SimilarityCriterion operator ∆X (Texture, L2()) directs these two vectors to be processed through distance function L2 (Euclidean) when comparing images regarding the criterion Texture.

The second parallel path promotes image I2 to be the current one using the pipeline operator [, so the feature extractor Histogram(5 ⊥ h[5]) operates over the segmented image, and the histogram h[5] counts the number of pixels covering each one of the five texture regions. The next projection Π(h[c][1], h[1 − 5]) retains the five elements of the h[c] histogram, and the first element h[c][1] of the color histogram — in medical images, the object under analysis is usually at the center of the image, so h[c][1] probably counts the color of the border of the image. Those six values are then submitted to the distance function L1 when comparing images regarding the criterion TexturedObjects, as defined by the ∆X (TexturedObjects, L1()) SimilarityCriterion operator.

2.4 Using Domain Expressions to Compare Images

Comparing two images requires the extraction of the data associated to the required criterion. Thus, the evaluation can be done without executing all processors in the domain expression. The domain expression must be evaluated following the path starting at the SimilarityCriterion operator associated with the required criterion, and backtracking to the point where every required data is available. The fundamental property to be warranted is that a data element can only be obtained if the function that generates it has its input data available. Although a detailed analysis of the process needed to evaluate the domain expression is beyond the scope of this paper, we present here its main idea.

From the several operators defined in Subsection 2.2, only image processors generate new images and/or features, as expressed by the functions f[i] and f[i] in Figure 1. Following these dependencies, it is possible to backtrack a domain expression regarding a comparison criterion starting at the corresponding SimilarityCriterion operator, to obtain the execution path defined as Path(CriterionName).

The goal of an image database structure is to find the best configuration to execute queries. Image processing algorithms are usually slow, so it makes sense to preprocess each image when it is stored, so the features are extracted during the insertion time and stored together with the image. However, not every extracted feature need to be stored, but just those required by at least one of the comparison criteria. The persistence processor FeatureVector ∆S(AttributeName) is employed to indicate those features.

When an execution path is extracting features of an image being stored in the database and a ∆S(AttributeName) is found, the corresponding features in S are stored in the same relation of the image, in the indicated attribute. When an execution path is getting features from an image already stored and an ∆S(AttributeName) is found, then the stored features are retrieved and the corresponding dependencies are cleared, avoiding the execution of the costly image processors that generate them. Therefore, it is highly recommended (but not required) that each SimilarityCriterion operator ∆X(CriterionName, df() i) be immediately preceded by a FeatureVector ∆S(AttributeName).

When an execution path is answering a query but the image is not stored in the database (e.g. when the query center is a new image), then the ∆S(AttributeName) persistence processors are not activated, the dependencies are not cleared and the image processors are executed. Notice however, that the domain expression helps to reduce the processing cost even in this case, because it allows selecting just the image processors that effectively contribute to generate the data needed by the required criterion.
3. IMPLEMENTATION ASPECTS

The domain expression was conceived to allow efficient content-based retrieval of images stored in a relational database. Hence the query should be, preferably, expressed in a extension of the SQL language and executed by the DBMS server or, otherwise, supported by a separated software module, if the DBMS does not provide that support. To implement those concepts, the following questions must be answered.

1. How to represent domains in the software system?
2. How to represent a domain expression in the software system?
3. How to store the images in the database?
4. How to store the feature vectors in the database?
5. How to define the image processors?
6. How to define the distance functions?
7. How to express and ask the similarity queries?

In this section we describe how these questions were answered in a prototype we implemented to answer similarity queries from medical exams stored in a large hospital database. This prototype uses the corporate DBMS (Oracle 9i) of the hospital as its supporting working storage.

In the prototype, the images are stored as attributes of type BLOB, as defined in the SQL:1999 standard. These attributes are placed in relations of a logical database distinct from those of the main application software, so the data corresponding to the image retrieval chores are hidden from the hospital applications. For each image domain DomId required, a new relation is created in the image database, defined by the following schema:

\[ < \text{DomId} >= \{ \text{ImgId}, \text{Img}, \text{Relation}, \text{AttrName}_1, \text{AttrName}_2, \ldots \} \]

The ImgId is a number uniquely identifying the image. When the main application defines an image attribute of type "StillImage", this type is translated to an integer type referencing the ImgId attribute of the hidden relation, which effectively stores the image in its Img attribute. The attribute Relation stores which is the relation and image attribute of the main application that holds this image. The attributes AttrName\textsubscript{1}, AttrName\textsubscript{2}, \ldots are row-typed attributes that stores the features extracted from image Img.

This framework answers questions 1, 3 and 4. The image processors are maintained in the database as stored procedures calling external functions that implement the image processing algorithms. The protocol of parameter passing is defined following the structure of the image environment concept, and the stored procedures calling them prepares the corresponding parameters obtained from the corresponding DomId relation. This framework answers question 5. The distance functions could be implemented in a similar way, but as our application always use an unchanging set of them, question 6 was answered implementing a set of distance functions directly in the prototype source code.

Similarity queries are requested as range or k-NN queries, submitted to the prototype as XML declarations. The domain expressions are also expressed as XML declarations stored in the database in a relation of domains. This relation has the attributes as follows: the name of the domain; the XML attribute that specify its expression (supplied by the analyst that designed the comparison criteria required for that domain); and a XML attribute for each execution path derived from the domain expression (obtained by the prototype by pre-processing the domain expression). This framework answers question 2 and 7. When the prototype receives a query as a XML declaration, it retrieves the corresponding execution path to identify the required features of type AttrName, retrieves them from the corresponding DomId relation, calls the functions that extracts the corresponding features from the query center, and compares the features to answer the requested query.

The same answers to questions 1 and 3 to 6 can be used to implement the domain expression concept in a DBMS server. Questions 2 and 7 can be answered in similar ways, but representing the domain expression as a new schema object in SQL (with the corresponding create, alter and drop commands), and representing the similarity queries as new predicates, assignable to image attributes.

4. EXPERIMENTAL RESULTS

Although image processing operations are costly considering processing time and memory requirements, the concept of the domain expression allows reducing the need to execute them at query time, yet maintaining a high level of flexibility and maintainability of the system. To evaluate this claim, we drilled the prototype for different workloads of queries and database sizes. We report here the results of comparing images following the domain expression stated in the example of Subsection 2.3. We measured the query time for the three criteria Color, Texture and TexturedObjects, and different number of required objects. Figure 4 shows the result of a k-NN query asking for the 10 images nearest to a given one shown as “Object Query” in the “Query Specification” box, regarding the Color criterion.

Figure 5 shows the average execution time when asking for different numbers k of neighbors in k - NN queries, in a relation containing 11,000 medical images. This set of images
occupies 5GBytes of disk space, and its features, extracted following that domain expression, occupies 4MBytes of disk space. The features from each comparison aspect were indexed using the Slim-tree [8] access method. In this figure, we show the total time to ask 500 k-NN queries for each value of $k$, which varies from 5 to 200, for each of the three similarity criteria expressed in our example.

As it can be seen, the processing time depends on the complexity of the features to be compared. The Color criterion depends on the 256-dimensional histogram, so it is more costly than the Texture and TexturedObjects criteria, based on features with 10 and 6 elements respectively. Remembering that Figure 5 shows the total time for 500 queries at each measured point, the time for every query is quite acceptable to make similarity queries a useful analytical tool for medical professionals. Moreover, the inclusion of new image processing algorithms as new image processors is as easy as to code them as external functions called from the stored procedure, making it possible to incorporate new ones to the already existing set of image processors. We do not present here the precision of the queries, as this parameters depends on the particular image processors employed, and not on the domain expression concept.

5. CONCLUSIONS
The main contribution of this paper is presenting a formal model of the whole content-based image retrieval process, aiming at having the image retrieval ability supported in relational database management systems. To this purpose we consider that a CBIR process is composed of the components as follows: (1) the comparison criteria that specify in which manner images are similar, (2) the preprocessing needed to be performed over an image before their features can be extracted, (3) the feature extractor algorithms, (4) the subset of extracted features that must be considered, (5) and the distance function employed to compare the feature.

The specification of the CBIR process as a composition of those components allow expressing comparison operations in a manner powerful and flexible. It allows configuring general-purpose feature extractors to be employed and reused in several domain expressions, leading the definition of how images from a specific domain can be compared as a programming activity that can be executed by experts in the application domain, with little knowledge of the area of image processing or databases. Therefore, the definition of the concepts of distinct image domains, domain expressions, comparison criteria, and the precise identification of the four kinds of operators (image processors, feature projection, persistence processors and control processors) involved the CBIR process are also novel contributions of this paper.

The concept of image domain expressions allows integrating the CBIR process into a database management system, integrating not only the storage of images, but also its content-based retrieval and image manipulation operations. To evaluate the applicability of the proposed concepts, we implemented the prototype of a CBIR tool to answer similarity queries from images of medical exams stored in a large hospital database. The success of the experiments demonstrates that the inclusion of CBIR operations in a DBMS core engine is doable. This fact represents a sensible evolution of the CBIR area in the challenge to reach a flexible and efficient CBIR environment.

6. ACKNOWLEDGMENTS
This work has been supported by CNPq (Brazilian National Council for Supporting Research) under grants 52.1685/98-6, 52.1267/96-0 and 860.068/00-7, and FAPESP (Sao Paulo State Research Foundation) under grant number 02/07318-1.

7. REFERENCES