large amounts of images from medical exams are being stored in databases, so developing retrieval techniques is an important research problem. Retrieval based on the image visual content is usually better than using textual descriptions, as they seldom give every nuance that the user may be interested in. Content-based image retrieval employs the similarity among images for retrieval. However, similarity is evaluated using numeric methods, and they often orders the images by similarity in a way rather distinct from the user’s intention. In this paper, we propose a technique to allow expressing the user’s preference over attributes associated to the images, so similarity queries can be refined by preference rules. Experiments performed over a dataset with CT lung images shows that correctly expressing the user’s preferences, the similarity query precision can increase from an average of 60% up to close to 100%, when enough interesting images exists in the database.

1 Introduction

Evaluating radiological images may be an arduous task for the specialists when analyzing certain diseases. The existence of several complex patterns and correlations between radiological and pathological findings is the main cause for the variations of radiologists’ interpretations. Computer-aided diagnosis (CAD) systems, which provides a second opinion for specialists, have proved to be highly efficient in increasing both diagnosis sensibility and specificity. However, when designing efficient systems to aid the radiologist, the diversity of radiological finding patterns and their different interpretations should also be supported.

Currently, the technology available to build analysis support systems able to recover image information from previous cases is incipient, creating a semantic gap between what the computational system and a human being recognize in the images. Several groups researching in the Content-Based Image Retrieval (CBIR) area are developing algorithms that take into account the radiologists’ similarity perception to compare images [6, 7, 8, 10]. This is a challenging task because either different users or the same user under different circumstances analyze/classify the same image in different ways. Thus, it is interesting to develop tools that include the specialists’ knowledge and their preferences when evaluating the query answers.

When searching for images in a CBIR system, the radiologists’ knowledge and how they indicate preferences is fundamental. For example, suppose that a specialist is searching for pulmonary diseases in an image, and asks for similar previous cases images as shown in Query Q1.

Q1: “Among the 10 images most similar to this lung XRay having the word ‘Consolidation’ in its report, the radiologist prefers images obtained in the driest seasons.”
In this example, the query can not be posed to the CBIR system because the preference for “the driest season” can not be directly expressed in the query expression. Thus, it is needed a way to provide to the system to know how to qualify one season as drier than another. The specialist’s background over what a “dry scale” means and his/her suspected diagnosis should be used to refine the query, bringing the query answer closer to what the radiologist expects. Therefore, to answer queries involving subjective factors, the CBIR system should both consider and use semantic background and preferences to obtain results closer to the users’ expectations. For example, in the Query Q1 the specialist expects to acquire first lung images of bronchiolitis exams, because the occurrence of this illness is favored with dry and cold weather; and then exams with other diseases.

Besides helping in diagnosing, preference similarity queries can also be used in school hospitals. Many of them are interested in clinical researches, thus they need similar image cases to allow quantitative and qualitative analyzes of signs and symptom behaviors. It is not easy to create an image database able to be queried by similarity using only textual information, due to the subjectivity of the textual image annotations. Preference similarity queries are well-suited to solve this problem, as they can search the database based on the diseases patterns (based on image content) and answer semantic queries (based on users’ preferences).

In this paper, we define an extension to the SQL query language that enables the user to express preferences over the answers of similarity queries. Similarity queries allow users to retrieve a number of images similar to a given query reference. Using preferences, it is possible to increase the number of answer images that in fact meet the user’s expectations, even when some of them are less similar regarding only the numeric evaluation of the similarity function.

The remainder of this paper is structured as follows. Section 2 presents background and useful related concepts. Section 3 describes our main contribution, showing how to extend a similarity-enabled query processor to also take into account user’s preferences, improving query answers quality. Section 4 reports experiments on real datasets obtained from the Clinical Hospital at Ribeirão Preto (HCRP) of the University of São Paulo - Brazil, showing that the quality of similarity queries are much improved by the proposed technique. Finally, Section 5 concludes the paper.

2 Background

In the database area, the problem of enhancing widely-used query languages with preference features has been tackled in several works [2, 3, 5]. In [2], a simple logical framework is proposed to express preferences by preference formulae. The formulae are incorporated into both the relational algebra and SQL, through the operator winnow parameterized by a preference formula. Kiessling [5] introduced Preference-SQL, which extends SQL with built-in base preference constructors and a pareto accumulation and a prioritized accumulation constructors. In [3], an extension of the standard SQL, called CPrefSQL, to express conditional preference queries was presented. CPrefSQL was designed to support a large variety of users’ preferences expressed by a set of conditional preference rules (cp-rules) of the form [9]:

\[ r : B_1 = b_1 \land \cdots \land B_k = b_k \rightarrow (A = a_1) > (A = a_2), \]

where \( \{B_1, \ldots, B_k, A\} \) is a set of attributes of the relation \( R \), \( b_i \in \text{dom}(B_i) \), and \( a_1, a_2 \in \text{dom}(A) \). Here, \( \text{dom}(B) \) denotes the attribute domain \( B \). The left side of the rule \( r \) is called the antecedent and the right side is called the consequent of \( r \). A set of cp-rules determine the preference order (partial order) over the set of tuples from the relation \( R \). The semantics of a cp-rule is the following: let \( u, v \) be tuples over the relation \( R \); we say that \( u \) is preferred to \( v \) according to the cp-rule \( r \) if \( u[B_i] = v[B_i] = b_i \), for \( i \in \{1, \ldots, k\} \), \( u[A] = a_1 \) and \( v[A] = a_2 \). Tuples can be compared using the transitivity property of the partial order associated to the set of cp-rules. For more details on this partial order see [3].

Similarity queries search the database for images similar to a given image query using a similarity measure, called distance function, to gauge their dissimilarity; the more similar the images, the smaller the distance between them. The Similarity Retrieval Engine (SIREN) is a service implemented between a relational database management system (RDBMS) and the application software that allows the execution of similarity queries in SQL [1].

SIREN intercepts and analyzes every SQL command sent from an application, treating the similarity-related constructions and references to complex data. When the command has neither similarity-related operations nor complex data, SIREN is transparent and the command is directly relayed to the RDBMS. If similarity-related constructions are found, SIREN executes the similarity-related operations and rewrites the command, calling the underlying RDBMS to execute the traditional operations. Similarity-related constructions are predicates expressed by similarity operators over complex data. The two most common similarity query operators are (a) range query (\( R_n \)), and (b) k-nearest neighbor query (\( kNN_n \)).

In this work, we extend the SIREN system adding a preference module to its architecture, which extends the CPrefSQL language introduced in [3] to include user knowledge and preferences into similarity queries. We choose to adopt this language due to the cp-rules expressiveness power. Thus, in this paper we describe how to use the CPrefSQL language to rewrite similarity queries, making them more semantically adequate to meet the users’ expectation.
3 Preference Similarity Queries

SIREN is a similarity retrieval engine working on top of RDBMS that extends SQL to enable expressing and evaluating similarity queries. However, to answer preference similarity queries targeting images that meet the users’ expectations, where users’ expectations are expressed as a set of qualitative preference rules (cp-rules), two new resources must be developed: first, SIREN needs to be expanded to store users’ preferences, whose rules can be used to rewrite similarity queries; and second, the SQL extension must have new commands added to allow expressing how to use the preference rules. In this section, we describe how both resources can be developed.

3.1 SIREN Preference Module

The SIREN query preference module rewrites similarity queries using similarity algebra [4], as shown in Figure 1. It explores several algebraically equivalent alternative plans, which leads to equivalent results but different execution costs. The part of the queries corresponding to non-similarity based operations are relayed to the underlying RDBMS.

![Figure 1. SIREN optimizer architecture with the preference module.](image)

The preference module is developed as a new SIREN functional component. It gets the preferences expressed in the user profile, and generates cp-rules, correlating the semantic information from preference rules and images. Together with the similarity algebra, the rules are used by SIREN to rewrite the similarity queries. Although the similarity algebra allows SIREN to obtain several equivalent plans that gives the same answer, applying preference rules can lead to plans that gives distinct results. Thus, our proposal is to use preference rules aiming either at simplifying the query execution, enabling to find answers faster, or at better following the users’ expectations, enabling to find better answers. In fact, SIREN is able to achieve both intents, improving both efficiency and efficacy of similarity query answering.

The preference module acts in a way similar to the indexer module: whenever a new preference model is created, the associated relation is processed following the model definition, generating the corresponding cp-rules. Thereafter, whenever a similarity query is posed, the optimizer uses the rules to evaluate if any of them can be used either to improve the query quality (tailoring the execution to better meet the users’ expectations), or to improve the query execution efficiency (using the rules as screening predicates to prune part of the data that is known to have no interesting data).

3.2 Defining a Preference Model

As presented in Section 2, some authors proposed SQL-based language extensions for preference. However, none of them execute preference queries over complex data – that is, they process preference queries only over traditional query. In this paper, we use the CPrefSQL language [3] to express the user’s expectation and to execute preference queries over similarity queries.

When cp-rules are included to similarity queries, we need to define the preference model in SQL, which provides the specification of how the rules are stored in the database catalog and how they are used to create constraints to be included in the queries. Thus, the syntax of the specification commands should follow the data definition language (DDL) style. To handle a Preference Model (PM) in SQL, we included a new command following that style: the CREATE PREFERENCE MODEL (and the corresponding DROP PREFERENCE MODEL) command.

A PM defines the cp-rules in the user’s profile. A new PM is defined by the CREATE PREFERENCE MODEL statement, whose syntax in EBNF (Extended Backus-Naur Form) notation follows.

```
<create_preference_model_statement>::=:
  CREATE PREFERENCE MODEL <model_name>
  FROM <relation_name>
  AS <preference_list>
  ['[/<attribute_list>']

The syntax follows the traditional CREATE INDEX command. The new PM, which is called <model_name> assigns the list of cp-rules defined in <preference_list> to the relation <relation_name>. Each rule is declared following the IF <antecedent> THEN <consequent> syntax. Multiple rules are taken as a conjunction (connected by and). Cprules can be created with or without antecedents, which are terms in the form attribute = value connected by the AND keyword. The consequents are always a preference relation between values of the same attribute. The set of cp-rules is created as part of the the user’s profile. The optional parameter <attribute_list>, which is represented between brackets, states that the attributes in the list are not compared in the rule — non specified attributes must have the same value in both tuples compared.
For illustration purpose, we will use here, as a running example, a database of lung tomographies (CT) obtained from the Clinical Hospital at Ribeirão Preto (HCRP) of the University of São Paulo - Brazil. This is the database that will be used in Section 4 to evaluate the implemented prototype. Suppose that a radiologist defines his/her profile regarding lung sickness like bronchiolitis and pneumonia. Example 1 shows a preference model stating that when searching for CT lung exams, if the image classification has the "Consolidation" finding, then the user prefers images of the driest seasons.

Example 1
CREATE PREFERENCE MODEL LungPref
FROM LungExams AS
IF class = 'Consolidation' THEN
   season = 'winter' > season = 'autumn'
   [id, sex, age, date] AND
IF class = 'Consolidation' THEN
   season = 'autumn' > season = 'spring'
   [id, sex, age, date] AND
IF class = 'Consolidation' THEN
   season = 'spring' > season = 'summer'
   [id, sex, age, date];

where $c_1 > c_2$ means that a tuple meeting condition $c_1$ is preferred over those meeting condition $c_2$.

3.3 Setting a Preference Model

Having a PM created, the set of $cp$-rules are validated and the PM becomes ready to be used. However, real query rewriting is not enabled until the user explicitly assigns it to his/her own query MODIFICATION environment. This environment embodies a set of query modifications rules that can be employed together to rewrite similarity queries. The SET MODIFICATION statement controls what modifications are enabled. Its syntax is as follows.

```
<set_modification_statement> ::= SET MODIFICATION [ADD | REMOVE | UPDATE] [ALL | <model_name>]
```

This statement is used both to enable or disable using a specified PM in the user’s MODIFICATION environment (using the ADD or REMOVE clauses), or to update existing PMs (using the UPDATE clause). When SET MODIFICATION ADD <model_name> is posed, the current set of $cp$-rules from the <model_name> model is added to the user’s MODIFICATION environment. When SET MODIFICATION UPDATE <model_name> is posed, the PM associated to the <model_name> model is re-evaluated (for example, due to changing configurations in the profile). When SET MODIFICATION REMOVE <model_name> is issued, the current set of $cp$-rules from the <model_name> model is removed from the user’s MODIFICATION environment. The ALL option is used to add, remove or update all PMs from the user in the user’s MODIFICATION environment.

Whenever there are PMs enabled in the user’s MODIFICATION environment, all queries issued by the user are rewritten following those models. Therefore, the SET MODIFICATION command allows the user to control when queries should be modified, and which models must be employed to modify each query. If the user adds a model in his/her MODIFICATION environment and other model was already added, the SIREN query rewriter asserts that both models are consistent and uses them to rewrite queries as a conjunction; otherwise, only the last added model is used to rewrite queries. Example 2 adds the PM defined in Example 1 into the user’s MODIFICATION environment, enabling every subsequent similarity query to be rewritten according to the rules obtained from the corresponding PM.

Example 2
SET MODIFICATION ADD LungPref

3.4 Using a Preference Model

When there SET MODIFICATION commands active, each similarity query posed is automatically rewritten taking into account the rules enabled in the user’s MODIFICATION environment. The query rewritten is enabled adding the ACCORDING TO PREFERENCE clause after the WHERE condition in the SELECT statement, as defined following.

```
<according_clause> ::= ACCORDING TO PREFERENCES ([n,] <model_name_list>)
```

This clause allows performing an additional filtering over the tuples returned after the execution of the clauses FROM and WHERE. The remaining tuples are those satisfying the users’ preferences specified by the <model_name>. The optional parameter $n$, which is defined in the user’s profile, enables selecting the $n$ most preferred tuples, respecting the preference hierarchy. Therefore, it is possible to select images with varying similarity degrees. The semantic of a $k$-$NN$ query using parameter $n$ corresponds to: first select the $k$ images most similar to the image reference; thereafter, among them, select up to $n$ most preferred images following the user’s profile preference hierarchy.

For instance, to execute the Query Q1 presented in Section 1 in SIREN, the user’s MODIFICATION environment must to be enabled, otherwise, SIREN will not execute preference similarity queries, but just the similarity query. Query Q1 is posed to SIREN in the way following.

Example 3
SELECT id, age, Image
FROM LungExams
WHERE Image NEAR 'C:\PatientExam1.jpg' STOP AFTER $k$;

where $k$ is the number of images to be retrieved.

When the MODIFICATION environment is enabled, SIREN automatically rewrites this command using the ‘LungPref’.
preference model. SIREN answers this query returning the $k$ images most similar to 'C:\PatientExam1.jpg' that satisfy the preferences specified in the user’s profile.

4 Experimental Evaluation

In this section, we present experiments comparing SIREN executing a similarity query both using cp-rules to rewrite queries and not using it. SIREN and its preference module are implemented in C++, and the experiments were evaluated using an Intel Core 2 Quad 2.83GHz processor with 4GB of main memory, under the Windows XP operational system. The RDBMS used to process the traditional part of the query was PostgreSQL.

The experiments were performed using a database created with 246 lung images collected in 108 distinct computed tomography exams from HCRP patients. The exams were separated according to their description and each image were classified by a radiologist into six distinct class – Consolidation, Emphysema, Interlobular Septal Thickening, Honeycombing, Ground-glass Opacity and Normal (in average 40 images per class), according to the radiological finding contained in each image. We employed two user profiles, one more general another more specific, and compared both with a plain similarity query, with no preferences enabled. The more specific PM is the LungPref, already shown in Example 1. The generic preference model is called DrySeasonsPref, and it is defined in Example 4. In this profile, the user just defines that he/she prefers dry and cold weather, saying nothing about CT findings preference.

Example 4

```
CREATE PREFERENCE MODEL Profile2
FROM LungExams AS
season = 'winter' > season = 'autumn'
[id, class, sex, age, date] AND
season = 'autumn' > season = 'spring'
[id, class, sex, age, date] AND
season = 'spring' > season = 'summer'
[id, class, sex, age, date];
```

The $k$ nearest neighbor query $Q1$, coded as shown in Example 3, was posed to SIREN asking for $k = 50$ for eighty-two distinct query references, covering 1/3 of the database. The same query set was posed with the PM disabled, and with the LungPref or the DrySeasonsPref PM enabled. Figure 2 presents the percentage of interesting answer, considering as interesting the images that have the same class of the query image, both in the similarity-only query and in the preference-enabled similarity queries. In this graph, the percentage of interesting answers is the proportion of relevant images obtained regarding the total number of images of the same class in the database.

Analyzing the obtained results, we can see that using preferences improves the amount of queries that return interesting images. In fact, only 37% of the plain similarity queries were able to retrieve 30% of the interesting answers, whereas 50% of the queries meeting the DrySeasonsPref PM and 77% of the queries using the LungPref PM retrieve at least 30% of the interesting answers. However, the figure also shows that if the PM is too generic, it is not able to improve answers for larger amounts of returned images. In fact, 90% of the similarity-only queries and 100% of the LungPref PM returns up to 50% of correct answers but DrySeasonsPref obtain only 61% of interesting answers. This is due to the generic preferences find preferred images in classes much distinct from the originally intended class.

Figure 2. Percentage of correct answer in the similarity and preference similarity queries.

Figure 3 is a graph showing the precision of the answer achieved for varying percentage of interesting answers filtered by the preference evaluation, again considering similarity-only queries and queries with LungPref or DrySeasonsPref PM enabled. When no preference is enabled, the graph shown for the similarity-only queries corresponds to a well-known Precision $\times$ Recall graph, where the similarity criterion employed achieves precision from 80% to 60% for up to 50% of recall (percentile of interesting answers). It must be remembered that this query searches
over the entire database. However, when PMs are enabled, the graphs show that the amount of interesting answers for the same percentile of precision increases considerably. In fact, for the more specific LungPref PM, it achieves 100% of precision for up to 30% of interesting answers. The less specific DrySeasonsPref PM also improves precision for most percentiles of interesting answers, but it reveals an interesting behavior for the range from 10% to 50% of interesting answers: the precision increases for increasing interesting answers. This is due the preference-based part of the query answering process working over the intermediate results obtained by the similarity-based part of the query answering process. Thus, as the amount of intermediate results increases, the preference-based part is able to find better answers. However, this effect does not continue indefinitely. As the similarity process continues to retrieve progressively farther answers, they do not meet the preference anymore, and the benefit obtained by any of them becomes null.

Figure 3 allows to observe another interesting behavior: more selective preference models improve answers for low-cardinality answers. In fact, the figure shows that, in the beginning, the more specific LungPref PM outperforms the DrySeasonsPref one. This effect remains whenever the database has sufficient number of interesting answers to be retrieved. This is why the precision of the more specific PM drops before the more generic does: the amount of images meeting the more specific PM is smaller than the amount of images that can meet the more generic one.

5 Conclusion

In this paper, we presented a novel technique to improve the quality of similarity queries over image databases, in order to obtain answers that fulfills the users’ expectations. Our technique also provides an intuitive way to express what the user intends to obtain. This is achieved allowing the user to define a profile stating what he/she considers interesting. Having the rules of what the user regards as interesting/important, similarity queries can be posed in the form “Find images similar to this given one that are like those of my preference.”. The experiments performed using the SIREN prototype over a dataset of 202 lung CT images allowed to improve the retrieval precision to close to 100%, for similarity queries that otherwise would achieve an average of 60% of precision.

The proposed technique uses cp-rules, expressed as user’s preference rules, to rewrite the similarity queries. The rules are thereafter employed by the query optimizer of the similarity-enabled query engine to rewrite the similarity queries posed. Our approach is non-invasive to the user, in the sense that the user can enable or disable the query rewriting, thus allowing the query to retrieve just the images that meet the user’s interest, or allowing just the plain similarity query to be executed. It also allows non-expert users to take advantage of the query rewritten resource, as the user does not need to change the query by himself/herself: an specialist can tune the query engine defining the user’s preferences and thereafter the system performs the rule-based query optimizations without any further user’s intervention.

References