A Multi-Similarity Algebra *

S. Adalı
Rensselaer Polytechnic Institute
sibel@cs.rpi.edu

P. Bonatti and M.L. Sapino
Università di Torino
{bonatti,mlsapino}@di.unito.it

V.S. Subrahmanian
University of Maryland
vs@cs.umd.edu

Abstract

The need to automatically extract and classify the contents of multimedia data archives such as images, video, and text documents has led to significant work on similarity based retrieval of data. To date, most work in this area has focused on the creation of index structures for similarity based retrieval. There is very little work on developing formalisms for querying multimedia databases that support similarity based computations and optimizing such queries, even though it is well known that feature extraction and identification algorithms in media data are very expensive. We introduce a similarity algebra that brings together relational operators and results of multiple similarity implementations in a uniform language. The algebra can be used to specify complex queries that combine different interpretations of similarity values and multiple algorithms for computing these values. We prove equivalence and containment relationships between similarity algebra expressions and develop query rewriting methods based on these results. We then provide a generic cost model for evaluating cost of query plans in the similarity algebra and query optimization methods based on this model. We supplement the paper with experimental results that illustrate the use of the algebra and the effectiveness of query optimization methods using the Integrated Search Engine (I.SEE) as the testbed.

1 Introduction

The need to automatically extract and classify the contents of multimedia data archives such as images, video, and text documents has led to significant work on similarity based retrieval of data. To date, most work on similarity based retrieval has focused on the creation of index structures for similarity based retrieval [9, 4, 15], with the exception of a very general theory of similarity developed in [12]. However, to date, there has been no work on developing an algebra (similar to the relational algebra) for similarity based retrieval. Given the fact that most feature extraction and identification algorithms in media data are very expensive, the need for query optimization to such databases is critical – yet, it is impossible to create principled query optimization algorithms without an initial underlying algebra within which similarity based retrieval operations are executed.

Our approach in this paper is to start with an arbitrary multimedia database **MDB** that has been organized according to any arbitrary, but fixed *set* of index structures and algorithms for retrieval using these index structures. Note that such a database may access a variety of specialized data structures each organized to facilitate one or more operations. Starting from this point, we define (Section 2) a *Multi-Similarity Algebra*, $\mathcal{MSA}$. Unlike previous works, $\mathcal{MSA}$ may be used to access not just one notion of similarity, but multiple similarity measures, encoded, by different algorithms, and to combine them in almost any way that the user wants. We then establish (Section 3) a set of equivalence, as well as containment results for $\mathcal{MSA}$ which provide the basis for query optimization techniques for processing similarity based queries given in Section 4. Before proceeding any further, we present two simple motivating examples from two different domains.

Example 1 (Face Recognition) Consider a set of images depicting faces of individuals. We might have two databases of such faces – a black and white image database **BW_DB** that comes accompanied by similarity based retrieval algorithms based on a holistic approach to face recognition (e.g. Chellappa's work at UMD), and a color image database **COL_DB** that uses eigenface based face retrieval algorithms (e.g. Pentland's work at MIT). While **COL_DB** can be used to retrieve images in **BW_DB**, the converse is not true. Furthermore, using **COL_DB** to retrieve images from **BW_DB** is less accurate than using **BW_DB**. Suppose now a user has a query image im_Q, and she wants to find images in **BW_DB** and **COL_DB** that are "similar" to im_Q. For this purpose, the user may specify the following types of queries:

1. Find the n "closest matches" to im_Q in **BW_DB** that match im_Q.
2. Find the n "closest matches" to im_Q in **BW_DB** $\cup$ **COL_DB** where the distance between an image im

*Supported by ARO grants DAAH-04-95-10174, DAAH-04-96-10297, and DAAH04-96-1-0398, by ARL contract DAAL01-97-K0135, and by NSF grant IRI-93-57756.

in **BW_DB** from im_Q is the average of the distances between im, im_Q reported by **BW_DB** and **COL_DB** respectively.

3. Find the n "closest matches" to im_Q in **BW_DB** ∪ **COL_DB** under the same conditions as before, taking into account, only texture properties.
4. Find the n "closest matches" to im_Q in **BW_DB** ∪ **COL_DB** under the same conditions as before, taking into account, images snapped after January 1, 1997.

Example 2 (Integrating multiple web search engines) Internet search engines manage keyword indexes of text documents. Given a queries, they return a set of documents together with an associated similarity "match". These similarity measures reflect the percentage of matched terms with associated weights that depend on the frequency of occurrence of each term in the document. Search engines such as HotBot and Excite support full text search whereas Infoseek supports searching within a different part of document such as the title, abstract, URL and etc. Suppose we are conducting an investigation into a suspected plagiarized research paper d_q. We ask for all documents that contain keywords similar to the keywords of d_q in all of the four above search engines. We want to find all documents d such that:

– d is returned by either HotBot or Excite with at least 65% percent similarity, and
– d is returned by AltaVista within the first 100 documents, and
– d is returned by Infoseek in the documents containing the words in the title of d_q with at least 70% similarity.
We can refine this query further by joining all the documents returned by Infoseek for the title, Excite and Hotbot for the full text with at least 60% similarity and choose those documents returned by Excite and at least one other search engine.

2 Heterogeneous Architecture for Similarity Search

Figure 1 shows our proposed similarity based retrieval architecture. The main features of our architecture are as follows:

Autonomous/Legacy Implementations: In our architecture, we assume the existence of certain legacy implementations of similarity whose APIs provide certain functions. This is certainly realistic – to date, most similarity based retrieval algorithms have been developed using special purpose techniques that vary widely (e.g. Internet search engines use a variety of strategies, as mentioned in the preceding example, while image matching products use diverse techniques for matching as well). Similarity implementations are defined in Section 2.2.

Multi-Similarity Algebra ($\mathcal{MSA}$): provides a way to reason about multiple similarity implementations through a single query language and a single set of algebraic operations. The multi-similarity algebra is introduced in Section 2.3. As there is often a *mismatch* between individual similarity implementations and the $\mathcal{MSA}$, similarity implementations need a "wrapper" that can be used to present a homogeneous interface between the similarity

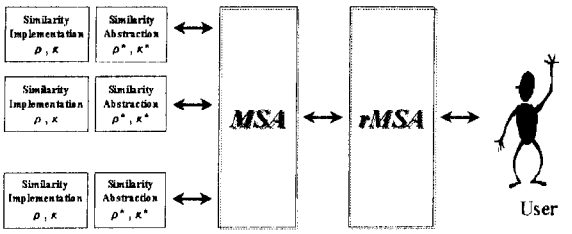

Figure 1: Architecture of Multi-Similarity Index Structure

implementations and the $\mathcal{MSA}$. Similarity abstractions are defined in Section 2.4.

Relational Multi-Similarity Algebra ($r\mathcal{MSA}$): provides a relational interface through which the user may query a set of similarity implementations. In effect, the $r\mathcal{MSA}$ "hides" implementation details from the user. It also gives him the ability to express complex queries and complex similarity strategies based upon his knowledge of the domain he is searching and his knowledge of the similarity engines (e.g. he may choose to disregard some of the similarity engines). The relational multi-similarity algebra is introduced in Section 2.6.

Query Optimization: We will develop techniques to optimize queries expressed both in $\mathcal{MSA}$ and in $r\mathcal{MSA}$. In particular, Section 3 describes various query equivalences, and how these equivalences may be used to create query plans. Using the $r\mathcal{MSA}$, the user may create views which may then be used for further query optimization. Section 4.2 defines similarity views, and shows how we may use a cost model (defined in section 4) for re-using views to optimize queries.

2.1 Preliminaries on Objects

In this section, we will introduce the notion of objects. Intuitively, an object is any abstract data type with associated properties. Objects can be images, text documents, video segments, etc. Properties model the attributes that similarity implementations can search on. As objects can have many different attributes, we will assume that any similarity measure is defined with respect to a subset of these properties. For this reason, we will introduce the notion of a type as an identifier for an object with respect to a different similarity measure. Below, we formalize these notions.

Definition 2.1 A *property* is a pair $(PropId, VSet)$, where $PropId$ is the property identifier and $VSet$ is the set of possible values for that property. We will use $PropId$ to identify the property. $PropId.VSet$ denotes the second component of the property. A *type* is a finite set of property identifiers.

For example, suppose `keyword-list` is a property and $VSet$ contains the *power-set* of all words in a document archive. Similarly, in the case of *thumbnail* images, properties might include `avg-R`, `avg-G`, `avg-B`, `avg-texture`, each having the set of reals as the associated $VSet$, specifying the average red-green-blue values and texture, respectively. Now, given a database of thumbnail images,

we may only want to consider the `avg-R`, `avg-G`, `avg-B`, `avg-texture` properties as significant values. Then the set of these four properties (namely {`avg-R`, `avg-G`, `avg-B`, `avg-texture`}) is a type.

Definition 2.2 An *object* of type $\tau = \{p_1, \ldots, p_n\}$ is a pair consisting of an *object template*, and a set $\{p_1 = v_1, \ldots, p_n = v_n\}$, where $v_i \subseteq p_i.VSet$.

Returning to our thumbnail sketch domain, an example object with $oid = o_1$ may have the values `avg-R`= [0.2, 0.3], `avg-B`= [0, 0], `avg-G`= [0.7, 0.9] and `avg-texture` = 1. In some cases, templates correspond to the actual attributes of objects stored in a database. In cases when the user does not have access to the actual attributes, properties may denote a region in the similarity search space. In search engines, an object template can be a list of keywords or a conjunction of query attributes as in the I.SEE system (Integrated Search Engine [1]). An example template can be a list of keywords such as "PostModern Culture Netzine". A document matching such a template may have properties such as `URL`, `title`, `abstract`. This is consistent with image retrieval implementations that retrieve pictures that are similar to a sketch input by the user. In this paper, we do not differentiate between these two types of templates.

2.2 Similarity Implementations

Suppose **MDB** is any set of media objects (the "multimedia database"), organized according to some indexing structure. Most existing similarity based retrieval implementations on these objects provide functionalities of similar nature. We group these functionalities into two categories. In the first category, similarity implementations may allow the similarity measures that they compute to be queried directly. In some cases, such measures may not be visible or even available to an external source. In the second category, we will consider the similarity implementations that will retrieve documents in decreasing order of similarity. These implementations will allow different ranges in the ranking with respect to similarity to be retrieved. The two abstract functions below model the behavior of these two different similarity queries.

Definition 2.3 (Similarity Implementation) Let *Obj* be a given set of objects. A *similarity implementation* (SIMP for short) for *Obj* is a nonempty set of functions of the following types:

$$\rho : 2^{Obj} \times Obj \times \mathcal{N} \rightarrow 2^{Obj \times \mathcal{R}^k},$$

$$\kappa : Obj \times Obj \rightarrow \mathcal{R}^k$$

where $card(\rho(\mathbf{MDB}, O, k)) \leq k$. If $k \leq l$ then $card(\rho(\mathbf{MDB}, O, k)) \leq card(\rho(\mathbf{MDB}, O, l))$. Moreover, if $(O', m) \in \rho(\mathbf{MDB}, O, n)$, then $O' \in \mathbf{MDB}$.

Intuitively, κ takes objects $O, O' \in \mathbf{MDB}$ and returns a vector of real numbers $r = \langle v_1, \ldots, v_k \rangle$. Usually, r contains only one value, such as a "distance" between O and O' – the smaller the distance, the more "similar" these objects are considered to be.

Likewise, the function ρ takes a database of objects, a search object *Obj* and a number N as input. It returns as output the first N objects in similarity ranking with respect to *Obj* from the input database. *Note that a similarity implementation may contain several functions – however, the signatures (input-output types) of these functions must fall into one of the two categories above.*

Example 3 Recall the example about the Internet Search Engines. We assume that there exists a similarity implementation ρ_X for each search engine X. Each implementation takes as value the proprietary database of the search engine, a search query containing keywords, connectives and other attributes and a number denoting the number of documents requested in the answer. The description of the generic query language for Internet search engines and query specification in this generic language can be found in [1]. An expression of the form $\rho\_E(EDB, Q1, 10)$ requests the first 10 documents that are most similar to the description $Q1$ in the database for Excite. Excite returns a vector of two values associated with each answer to this query, the first number corresponds to the rank of the document and the second to the similarity measure expressed as a percentage value. Each object returned by this similarity implementation will have the properties title, URL, abstract and last modification date.

We assume that all similarity implementations use a special constant *all*. A call of the form $\rho\_E(EDB, Q1, all)$ is requests all tuples from the database EDB in decreasing order of similarity w.r.t. $Q1$. Technically, *all* denotes the number corresponding to the size of the input database for any similarity implementation.

2.3 Similarity Measures

Similarity measures refine similarity implementations as follows. First, we will allow a similarity measure to not just take two objects, but also a type as input. Intuitively, this measures the similarity of two objects with respect to a specific type. (Later, the similarities between O, O' w.r.t. different types may be *merged*). Second, we will assume that the answer returned is not a single value – but a *bag* of values. This approach not only generalizes the point based approach as a special case, it also provides a declarative definition language for expressing similarity computations that manipulate multiple similarity implementations and various interpretations of the similarity values returned by them. Finally, for the sake of convenience (this will make later definitions easier to read), we will represent similarity measures by sets of tuples (the reader will readily believe that all functions can be represented as relations, e.g. $f(x) = y$ can be represented as a relation with the tuple $f(x, y)$).

Definition 2.4 (Similarity Measure) A *similarity measure* m is a set of tuples of the form $\langle O, O', \tau, v \rangle$, where O, O' are objects, τ is a type and v is a bag of real numbers. A similarity measure m should be such that $\langle O, O', \tau, v \rangle \in m$ and $\langle O, O', \tau, v' \rangle \in m$ imply $v = v'$, i.e. identical bags.

Example 4 m_1 and m_2 below are two similarity measures.

m_1

o_1	o_2	τ_1	$\{0.8, 0.9\}$
o_1	o_2	τ_2	$\{0.5, 0.7\}$
o_1	o_3	τ_1	$\{0.2, 0.9\}$
o_2	o_3	τ_2	$\{0.5, 0.8\}$

m_2

o_1	o_2	τ_1	$\{0.4, 0.6\}$
o_1	o_3	τ_1	$\{0.5, 0.6\}$
o_1	o_3	τ_2	$\{0.4, 0.5\}$
o_2	o_3	τ_1	$\{0.1, 0.2\}$

2.4 Similarity Abstractions

The concept of a *similarity abstraction* ties together the concept of similarity implementations and similarity measures by mapping the values obtained from an implementation to the appropriate bag representation.

Definition 2.5 (Similarity Abstraction) A *similarity abstraction* of a similarity implementation SI w.r.t. τ and $f : \mathcal{R}^k \rightarrow \mathcal{R}$, is a set of functions contained in $\{\rho^*_{\tau,f}, \kappa^*_{\tau,f} | \tau$ is a type and f is a function from vectors of reals of arity k to reals $\}$, corresponding to the functions of SI, defined as follows:

1. If $\rho \in SI$ and $\rho(DB, O, n) = \{ (O_1, v_1), \ldots, (O_n, v_n) \}$, then $\rho^*_{\tau,f}(DB, O, n) =$
$\{\langle O, O_1, \tau, \{ f(v_1) \} \rangle, \ldots, \langle O, O_n, \tau, \{ f(v_n) \} \rangle\}$.
2. If $\kappa \in SI$, then
$\kappa^*_{\tau,f}(O, O') = \{ \langle O, O', \tau, \{ f(\kappa(O, O')) \} \rangle \}$.

The type associated by an abstraction may be the identifier of an implementation allowing us to combine similarity measures depending on which source they came from. Other type information may include different fields considered by an implementation, or the significant properties of objects for which an implementation gives better results. It is also possible to write relevant algebraic operations that focus on similarity values associated with a specific type.

Example 5 In this example we consider the similarity implementations for search engines introduced in Example 3. Suppose now for each similarity implementation ρ_X for a given search engine X, we introduce a new type X corresponding to the similarity values returned by this engine. Hence, a simple query of the form $\rho\_H^*_{\{H\},y/100}(HDB$, "Postmodern Culture Netzine",20) returns the first 20 documents that contain the keywords "Postmodern Culture Netzine". The function $y/100$ changes percentages to values between 0 and 1.

2.5 Similarity Algebra

We have come to the second level of abstraction in our multi-similarity algebra. Recall that in the first level, we abstracted similarity operations with respect to a given type to similarity measures. In this section, we introduce a set of algebraic operations on similarity measures that allow formulation of different methodologies for combining multiple similarity computations in a declarative language. This will make it possible to develop query reformulation and optimization methods on top of similarity computations. In the following let $\mathcal{F}$ be a family of computable functions over sets of reals, i.e. each $f \in \mathcal{F}$ takes as input, a set of real numbers, and returns as output, a single real number.

Definition 2.6 (Collect Function) Let m_1 and m_2 denote similarity measures, $f \in \mathcal{F}$, and $\circ \in \{\cup, \cap, -\}$. Then the collect function computes bags of bags as defined below:

$$Coll_\circ(m_1, m_2, O, O', \tau) = \{v \mid \exists \tau_1, v_1, \tau_2, v_2 \text{ such that } \langle O, O', \tau_1, v_1 \rangle \in m_1, \langle O, O', \tau_2, v_2 \rangle \in m_2, \tau_1 \circ \tau_2 = \tau, v = v_1 \cup v_2\}$$

Example 6 Recall the similarity measures given in Example 4. The following is returned by the $Coll_\circ$ function:

$$Coll_\cup(m_1, m_2, o_1, o_2, \tau_1 \cup \tau_2) = \{ \{ 0.4, 0.6, 0.5, 0.7 \} \}$$

Definition 2.7 (Similarity Algebra Operations) Suppose P is a computable predicate over types and bags of reals, and $f \in \mathcal{F}$. Then, the similarity algebra operations are defined as shown in Figure 2.

Example 7 Recall the similarity measures m_1 and m_2 given in Example 4. The following are examples of the results of the basic similarity algebraic operations on these two similarity measures.

$m_1 \bowtie_\cup m_2 = \{\langle o_1, o_2, \tau_1, \{ 0.4, 0.6, 0.8, 0.9 \} \rangle,$
$\langle o_1, o_2, \tau_1 \cup \tau_2, \{ 0.4, 0.5, 0.6, 0.7 \} \rangle,$
$\langle o_1, o_3, \tau_1, \{ 0.2, 0.5, 0.6, 0.9 \} \rangle,$
$\langle o_1, o_3, \tau_1 \cup \tau_2, \{ 0.2, 0.4, 0.5, 0.9 \} \rangle,$
$\langle o_2, o_3, \tau_1 \cup \tau_2, \{ 0.1, 0.2, 0.5, 0.8 \} \rangle\}$.
$m_1 \bowtie_- m_2 = \{\langle o_1, o_2, \emptyset, \{ 0.4, 0.6, 0.8, 0.9 \} \rangle,$
$\langle o_1, o_2, \tau_2 - \tau_1, \{ 0.4, 0.5, 0.6, 0.7 \} \rangle,$
$\langle o_1, o_3, \emptyset, \{ 0.2, 0.5, 0.6, 0.9 \} \rangle,$
$\langle o_1, o_3, \tau_1 - \tau_2, \{ 0.2, 0.4, 0.5, 0.9 \} \rangle,$
$\langle o_2, o_3, \tau_2 - \tau_1, \{ 0.1, 0.2, 0.5, 0.8 \} \rangle\}$.
$m_1 \oplus m_2 = \{\langle o_1, o_2, \tau_1, \{ 0.4, 0.6, 0.8, 0.9 \} \rangle,$
$\langle o_1, o_2, \tau_2, \{ 0.5, 0.7 \} \rangle, \langle o_1, o_3, \tau_1, \{ 0.2, 0.9, 0.6, 0.5 \} \rangle,$
$\langle o_2, o_3, \tau_2, \{ 0.5, 0.8 \} \rangle, \langle o_1, o_3, \tau_2, \{ 0.4, 0.5 \} \rangle,$
$\langle o_2, o_3, \tau_1, \{ 0.1, 0.2 \} \rangle\}$.
$m_1 \setminus m_2 = \emptyset$

Definition 2.8 (Similarity Algebra) Suppose we have some arbitrary, but fixed set of similarity implementations SI. $\mathcal{MSA}$ consists of (1) the space of all possible similarity measures, and (2) the set of all similarity abstractions $\rho^*_{\tau,f}$ and $\kappa^*_{\tau,f}$ (associated with the similarity implementations SI being considered). An expression in the similarity algebra is inductively defined as follows:

- $\rho^*_{\tau,f}(DB, O, n)$, is an expression in the similarity algebra where DB is a set of objects, O is an object (not necessarily in DB) and n is an integer;
- $\kappa^*_{\tau,f}(O, O')$, is an expression in the similarity algebra where O and O' are objects;
- $op(e_1, \ldots, e_n)$, is an expression in the similarity algebra where op is one of the similarity algebraic operators and $e_1, \ldots, e_n$ are expressions of the similarity algebra.

Some simple examples involving the database **BW_DB** introduced at the beginning of this paper are as follows. The query $\kappa^*_{\text{texture},+}(im_Q, im)$ first determines the similarity of im_Q and im based on texture, using the similarity implementation provided by **BW_DB**. The resulting set is then combined by merely adding the values in

Operation Name	Notation	Definition
Sim-Union Join	$m_1 \bowtie_\cup m_2$	$\{\langle O,O',\tau,v\rangle \mid v = \bigcup Coll_\cup(m_1,m_2,O,O',\tau)\}$
Sim-Intersection Join	$m_1 \bowtie_\cap m_2$	$\{\langle O,O',\tau,v\rangle \mid v = \bigcup Coll_\cap(m_1,m_2,O,O',\tau)\}$
Sim-Difference Join	$m_1 \bowtie_- m_2$	$\{\langle O,O',\tau,v\rangle \mid v = \bigcup Coll_-(m_1,m_2,O,O',\tau)\}$
Sim-Merge	$m_1 \oplus m_2$	$\{\langle O,O',\tau,v\rangle \mid v = \bigcup\{v' \mid \langle O,O',\tau,v'\rangle \in m_1 \cup m_2\}\}$
Sim-Subtract	$m_1 \setminus m_2$	$\{\langle O,O',\tau,v\rangle \mid \langle O,O',\tau,v\rangle \in m_1,\ \nexists\langle O,O',\tau',v'\rangle \in m_2\}$
Sim-Select	$\sigma_P(m)$	$\{\langle O,O',\tau,v\rangle \mid \langle O,O',\tau,v\rangle \in m \text{ and } P(O,O',\tau,v') = \text{true}\}$
Sim-Map	$map_f(m)$	$\{\langle O,O',\tau,f(v)\rangle \mid \langle O,O',\tau,v\rangle \in m\}\quad (f \in \mathcal{F})$

Figure 2: Similarity Algebra Operations

the set and this is the output. The query $\rho *_{\text{texture},+} (\mathbf{BW\_DB}, im_Q, 5)$ on the other hand, returns the five closest matches to im_Q using the above criterion.

2.6 Relational Multi-Similarity Algebra $r\mathcal{MSA}$

In this section, we will augment the algebra $\mathcal{MSA}$ so that it presents a relational interface to the user, even though it manipulates non-relational structures underneath. We will add two new operators to the standard relational algebra. The relational similarity operator *RSO* will give a relational view over the similarity measures computed by the similarity algebra. The *Best* operator which is the relational analog of the ρ operator in $\mathcal{MSA}$ will make it possible to ask rank related queries over multiple similarity abstractions.

Definition 2.9 (Relational Similarity Operator) Let f be a function from bags of reals to reals. Then the relational similarity operator *RSO* is defined as follows:

$$RSO_f(m,\tau) = \{\langle O,O',f(v)\rangle \mid \langle O,O',\tau,v\rangle \in m\}.$$

Example 8 For example, consider the similarity measure m_1 in Example 4. Then
$RSO_{min}(m_1,\tau_1) = \{\langle o_1,o_2,0.8\rangle, \langle o_1,o_3,0.2\rangle\}$.

Definition 2.10 (Best n operator) The best n operator $Best : N \times Table \times N \rightarrow Table$ selects tuples from a given table. The expression $Best(n,T,c)$ returns the table containing the maximal n tuples of T, w.r.t. the value of the cth column of T. If there exist two tuples $\langle t_1,\ldots,t_k\rangle$ and $\langle t'_1,\ldots,t'_k\rangle$ with $t_c = t'_c$, then the two tuples are ordered according to lexicographic ordering.

Example 9 Returning to the example of Example 8, we notice that $Best(1, RSO_{min}(m_1,\tau_1), 3)$ yields the single tuple $(o_1,o_2,0.8)$.

Definition 2.11 (Relational Multi-Similarity Algebra) The *relational multi-similarity algebra* consists of the space of possible relational tables, together with the relational similarity operator *RSO*, the standard relational operators, and the aggregate operator *Best*. An expression in the similarity relational algebra is inductively defined as follows:

- $RSO_f(se,\tau)$, is an expression where se is a similarity algebraic expression;
- e, is an expression where e denotes a relational table,
- $op(e_1,\ldots,e_n)$, is an expression where op is one of the standard n-ary relational algebraic operators and $e_1,\ldots,e_n$ are expressions of the similarity relational algebra.
- $Best(n,T,c)$, is an expression where n and c are natural numbers, and T is an expression of the similarity relational algebra.

Example 10 In this example, we consider several $r\mathcal{MSA}$ expressions involving the Internet search engines. We assume that we have access to three engines, namely Excite, HotBot and Infoseek with corresponding similarity abstractions $\rho\_E^*, \rho\_H^*, \rho\_I^*$ as introduced in Example 5. Then, we write the following queries in the $\mathcal{MSA}$ algebra:

$$q_1 \equiv \Pi_{\$1,\$3}(RSO_{avg}(Best^{avg}_{5,\{E,I\}}(\rho^*_{\{E\},x/100}(EDB,O,all) \bowtie_\cup \rho^*_{\{I\},x/100}(IDB,O,60)),\{E,I\}),3)).$$

$$q_2 \equiv \Pi_{\$1,\$3}(\sigma_{\$3\geq 0.8}(RSO_{min}(m_2,\{E,I\}))) \text{ where}$$
$$m_2 \equiv \rho^*_{\{E\},x/100}(EDB,O,all) \bowtie_\cup \rho^*_{\{I\},x/100}(IDB,O,all).$$

$$q_3 \equiv \Pi_{\$1,\$3}(RSO_{avg}((\rho^*_{\{E\},x/100}(EDB,O,20) \bowtie_\cup \rho^*_{\{H\},x/100}(IDB,O,20)),\{E,H\}) \cup RSO_{x/2}((\rho^*_{\{E\},x/100}(EDB,O,20) \setminus \rho^*_{\{H\},x/100}(IDB,O,20)),\{E\}) \cup RSO_{x/2}((\rho^*_{\{H\},x/100}(EDB,O,20) \setminus \rho^*_{\{E\},x/100}(IDB,O,20)),\{H\}))$$

The first query asks for the first 5 documents with respect to the average similarity measure computed by both Excite and Infoseek and all the documents should appear in the first 60 documents for Excite. The second query returns all documents that have 0.8 similarity with respect to both Excite and Infoseek. The last query merges the first 20 answers from Excite and HotBot. The similarity is averaged if the document appears in first 20 of the both engines, otherwise it is divided by half.

3 Equivalences/Containments in $\mathcal{MSA}$ and $r\mathcal{MSA}$

In the previous sections, we introduced a new algebra that allowed us to relate various similarity implementations with relational tables and relational operations. In this section, we will investigate the conditions under which expressions in the similarity relational algebra can be rewritten to equivalent or more restrictive (through containment) expressions. The results of this section will be used to develop query optimization techniques to lower

the cost of query processing by pushing selections down and reordering costly joins. We will also use these results to find alternate ways of evaluating queries by reusing the results of cached items and by accessing pre-defined views over the sources.

When considering equivalences, it is particularly important to note that an algebraic expression of the form $\rho^*(\mathbf{MDB}, o, all)$ says "find the top *all* matches for object o from **MDB**", which of course causes all objects in **MDB** to be returned in the appropriate order. In general, similarity implementations are the most costly operations in the $\mathcal{MSA}$. Hence the size of the input to these implementations must be reduced by either reducing the database **MDB** by pushing selections down or by putting additional bounds on the number of objects (*all*) requested whenever possible. It will turn out that selections over objects, values and types have different properties and we examine them separately. Let Po, Pv and Pt be predicates over the objects O, O', values v and types τ respectively of the members $\langle O, O', \tau, v \rangle$ of the similarity measures. $Po(O)$ denotes the unary predicate obtained by letting the first argument of Po be O.

3.1 Pushing Selection into Similarity Implementations

The following theorem tells us that selection is commutative. Furthermore, it tells us that when we consider selection conditions on values, then selections cannot be pushed inside the ρ operator. However, when selections are made using predicates on objects this is possible and sometimes, the selection can even be eliminated altogether. In addition, when selections are performed on types, then the selection can *always* be eliminated.

Theorem 1 Consider a multimedia database, **MDB**.
1. $\sigma_P \sigma_Q m = \sigma_Q \sigma_P m$ for any similarity measure m.
2. $\sigma_{Pv}(\rho^*_{\tau,f}(DB, O, n))$ does not change since σ_{Pv} cannot be "pushed inside" $\rho^*_{\tau,f}$, (no values in DB, O, n).
3.

$$\sigma_{Po}(\rho^*_{\tau,f}(DB,O,n)) = \begin{cases} \rho^*_{\tau,f}(DB,O,n) & \text{(i)} \\ \emptyset & \text{(ii)} \\ \rho^*_{\tau,f}(\sigma_{Po(O)}(DB),O,n) & \text{(iii)} \end{cases}$$

$$\sigma_{Pt}(\rho^*_{\tau,f}(DB,O,n)) = \begin{cases} \rho^*_{\tau,f}(DB,O,n) & \text{if } Pt(\tau) \text{ hold} \\ \emptyset & \text{otherwise.} \end{cases}$$

where the above conditions are (i) if Po depends only on O and $Po(O)$ holds, (ii) if Po depends only on O and $\neg Po(O)$ holds, (iii) if Po depends on both O and O' and $n = all$. Also note that $\rho^*_{\tau,f}(\sigma_{Po(O)}(DB), O, n) = \rho^*_{\tau,f}(\sigma_{Po(O)}(DB), O, |\sigma_{Po(O)}(DB)|)$.

3.2 Pushing Object Selections

It turns out that selections based on objects can always be pushed through all our $\mathcal{MSA}$ operations as well as the RSO operations.

Theorem 2 Selections can be pushed through all operations of the $r\mathcal{MSA}$ algebra, i.e.

$$\begin{aligned} \sigma_{Po}(RSO_f(m,\tau)) &= RSO_f(\sigma_{Po}m, \tau) \\ \sigma_{Po}(m_1 \bowtie_{op} m_2) &= (\sigma_{Po}m_1) \bowtie_{op} (\sigma_{Po}m_2) \\ \sigma_{Po}(m_1 \oplus m_2) &= (\sigma_{Po}m_1) \oplus (\sigma_{Po}m_2) \\ \sigma_{Po}(map_f(m)) &= map_f(\sigma_{Po}\, m) \\ \sigma_{Po}(m_1 \setminus m_2) &= (\sigma_{Po}\, m1) \setminus m_2 \\ &= (\sigma_{Po}\, m1) \setminus (\sigma_{Po}\, m_2)\,. \end{aligned}$$

This result has a significant impact on the evaluation of $\mathcal{MSA}$ and $r\mathcal{MSA}$ queries, as is apparent from the following example.

Example 11 Suppose we have a database DB of pictures which, among their attributes, have the photographer's name. We want to rank the pictures by Hamilton according to their similarity to a given query picture O. Moreover, suppose that we can use two similarity implementations ρ_A and ρ_B, that return similarity evaluations with respect to attributes a and b, respectively. We want to combine these measures by taking their average values. In the similarity algebra the combined measure can be expressed as follows:

$$m = map_{avg}(\rho\_A^*_{\{a\},id}(DB, O, all) \bowtie_{\cup} \rho\_B^*_{\{b\},id}(DB, O, all)).$$

If DB contains n pictures by Hamilton, thanks to the above equivalences and Theorem 1, the above expression can be rewritten as

$$m' = map_{avg}(\rho\_A^*_{\{a\},id}(\sigma_{author=Hamilton}(DB), O, n) \bowtie_{\cup} \rho\_B^*_{\{b\},id}(\sigma_{author=Hamilton}(DB), O, n)).$$

Note that, by anticipating the selection, we are able to restrict the search space of the similarity implementations.

3.3 Type Selections

In contrast to the case of value selections and object selections, type selections are less amenable to being "pushed." They can be pushed only under certain strong conditions. The following result shows that type selections can be pushed through the operator $\oplus$ that merges similarity measures, and the operator map_f that combines results of different similarity implementations.

Theorem 3 Suppose m_1, m_2 are similarity, $P(\tau)$ is a type selection, and f is a computable map in $\mathcal{F}$. Then:

$$\begin{aligned} \sigma_{Pt}(m_1 \oplus m_2) &= (\sigma_{Pt}m_1) \oplus (\sigma_{Pt}m_2) \\ \sigma_{Pt}(map_f(m)) &= map_f(\sigma_{Pt})m\,. \end{aligned}$$

In general, selection over types cannot be pushed through a join, i.e. the equality,

$$\sigma_{Pt}(m_1 \bowtie_{\cup} m_2) = (\sigma_{Pt}m_1) \bowtie_{\cup} (\sigma_{Pt}m_2)$$

does not generally hold, as is shown by the following example.

Example 12 Suppose that m_1 and m_2 are similarity measures containing only tuples with type $\{a\}$, and $\{b\}$ respectively. Secondly, suppose that $m_1 \bowtie_{\cup} m_2 \neq \emptyset$. We have $\sigma_{\tau=\{a,b\}}(m_1 \bowtie_{\cup} m_2) = m_1 \bowtie_{\cup} m_2$, but $(\sigma_{\tau=\{a,b\}}m_1) \bowtie_{\cup} (\sigma_{\tau\{a,b\}})m_2 = \emptyset \bowtie_{\cup} \emptyset = \emptyset$.

Nevertheless, the following theorem shows that under appropriate conditions, the result does hold.

Theorem 4 (Behavior of Type Selection w.r.t. $\bowtie_\cup$) Suppose we consider the case where similarity measures m_1 and m_2 have the same type τ. Then:

$$\sigma_{Pt}(m_1 \bowtie_\cup m_2) = (\sigma_{Pt} m_1) \bowtie_\cup (\sigma_{Pt} m_2)$$

The above result is very useful when we deal with similarity implementations that consider only one property (e.g. average RGB values of pixels). Dual results exist when we consider the behavior of type selection w.r.t. $\bowtie_\cap$ – we state these below for the sake of completeness.

Theorem 5 (Behavior of Type Selection w.r.t. $\bowtie_\cap$) If all tuples in the similarity measures m_1 and m_2 have the same type τ then

$$\sigma_{Pt}(m_1 \bowtie_\cap m_2) = (\sigma_{Pt} m_1) \bowtie_\cap (\sigma_{Pt} m_2).$$

We finally consider type selection w.r.t. $\bowtie_-$. Here again, similar equivalence results exist.

Theorem 6 (Behavior of Type Selection w.r.t. $\bowtie_-$) The following are the equivalence results for pushing type selections down the sim-difference operator:

1. If, for all τ, τ', $P(\tau - \tau') \equiv P(\tau) \wedge P(\tau')$, then

$$\sigma_{Pt}(m_1 \bowtie_- m_2) = (\sigma_{Pt} m_1) \bowtie_- (\sigma_{Pt} m_2).$$

2. If the type components in the similarity measures m_1 and m_2 are disjoint, then

$$\sigma_{Pt}(m_1 \bowtie_- m_2) = (\sigma_{Pt} m_1) \bowtie_- m_2.$$

3.4 Value Selection

In this section, we study the behavior of the similarity-selection operator when value conditions are evaluated. The following result shows that selection over values may be pushed inside the relational similarity operator, *RSO*.

Theorem 7 Let $\sigma_{Pv} \cdot f$ be the composition of σ_{Pv} and f (i.e. $(\sigma_{Pv} \cdot f)(v) = \sigma_{Pv}(f(v))$). Suppose $f \in \mathcal{F}$ and m is any similarity measure. Then:

$$\begin{aligned} \sigma_{Pv}(RSO_f(m, \tau)) &= RSO_f(\sigma_{Pv} \cdot fm, \tau). \\ \sigma_{Pv}(map_f(m)) &= map_f(\sigma_{Pv} \cdot f\, m). \end{aligned}$$

3.5 The Best Match Operator

Recall that the operator $Best(n, T, c)$ finds the best n matches in T w.r.t. the c'th column of T. Usually, T is a relational view of a non-relational source – for example, T may be $RSO_{min}(m_1, \tau_1)$ where m_1, τ_1 are as defined in Example 4. In this section, we will study various equivalences associated with this operator. In particular, what we would like to ensure – when later doing query optimization – is that the similarity implementations (which are usually slow) do as little work as possible. Our study of equivalences involving *Best* will be motivated by this need. The operator *Best* has good properties with respect to standard selections, projections, and unions and differences, as illustrated by the following result.

Theorem 8 (Interaction between *Best* and Projection,Union, Difference) For any $c \in \tau$, it is the case that:
$Best(n, \Pi_\tau T, c) = \Pi_\tau(Best(n, T, c'))$ where c' is the column of T corresponding to column c of $\Pi_\tau T$.
$Best(n, T_1 \cup T_2, c) = Best(n, Best(n, T_1, c) \cup Best(n, T_2, c), c)$.
$Best(n, T_1 - T_2, c) = Best(n, Best(n+ \mid T_2 \mid, T_1, c) - T_2, c)$.
$Best(n, RSO_g(m, \tau), c) = RSO_g(Best^g_{n,\tau} m)$.

We observe that $Best(n, T, c)$ belongs to the *relational* algebra; when it is pushed inside similarity expressions its parameters must be changed. In particular, there is no clear ordering function between bags of real values. We use the notation $Best^g_{n,\tau} m$ to denote the operator that selects the n best matches (w.r.t. similarity measure m and type τ) after executing function g on the bags of values of similarity measures. This will be the same function that is used to perform relational similarity operations. Given bags v, w of real numbers, we say that

$$v \leq_g w \iff g(v) \leq g(w).$$

Suppose now that T is a relational table. The following result presents some containment/equivalence results. The first says that pushing *Best* inside a selection yields a subset of the original query. The second result says that when *Best* and selection interact and the selection condition is a true type-selection, then the selection may be eliminated or set to the empty set (hence, this is a highly desirable optimization). On the other hand, when performing object selections and *Best* together, we have a containment result in part (1) below. Pushing *Best* inside a $\sigma_{Po}(m)$ query yields a subset of the answer to the original. Thus, if we had previously stored the view $Best^g_{n,\tau}(\sigma_{Po}(m))$, and we now want to execute the query $\sigma_{Po}(Best^g_{n,\tau} m)$, then we can execute this selection on the materialized view $Best^g_{n,\tau}(\sigma_{Po}(m))$, thus avoiding recomputations. This will be explained in further detail in Section 4.2. Finally, the fourth part of the theorem says that the same property holds for value-selections.

Theorem 9 (Interaction of *Best* and Selections)

$$\begin{aligned} Best(n, \sigma_P T, c) &\supseteq \sigma_P(Best(n, T, c)). \\ Best^g_{n,\tau}(\sigma_{\lambda\tau.Pt} m) &= \begin{cases} Best^g_{n,\tau} m & \text{if } Pt(\tau) \text{ holds,} \\ \emptyset & \text{otherwise} \end{cases} \\ Best^g_{n,\tau}(\sigma_{Po} m) &\supseteq \sigma_{Po}(Best^g_{n,\tau} m) \\ Best^g_{n,\tau}(\sigma_{Pv} m) &\supseteq \sigma_{Pv}(Best^g_{n,\tau} m). \end{aligned}$$

However, things become rapidly more complex when we consider the interactions between *Best* and the operations ρ^*. These interactions describe the conditions under which we may "push" the global "best" operation to ρ^* which applies to a single similarity implementation at a time. Before proceeding to define the interactions between *Best* and ρ^*, we define two generic conditions that we will use.

(C1) $\leq_g$ preserves the ordering of real numbers over singletons, i.e., $x \leq y \Rightarrow \{x\} \leq_g \{y\}$;

(C2) $(v_1 \leq_g v_2) \Rightarrow \forall v_3\ (v_1 \mid\mid v_3 \leq_g v_2 \mid\mid v_3)$.

Example 13 All monotonic functions of min/max satisfy the above conditions. For example, g can be c, $\max(v)$, $\min(v)$, $\frac{\max(v)+\min(v)}{c}$, $c^{\max(v)}$, etc., where c is a positive constant.

The first part of the following result tells us that under some easy to check conditions, the computation of *Best* is redundant and can be eliminated. The second result tells us that if both conditions (C1) and (C2) hold, then doing a *Best* computation on a merged pair of similarity measures is equivalent to performing the *Best* operation on each, and then taking the *Best* again. Which of these two execution orders is preferable will be studied through experiments in Section 5. The last result says that combining sets of rankings through function $f \in \mathcal{F}$ can be either done before executing *Best* or after.

Theorem 10 (Interaction of $Best, \rho^*$ and Similarity Measures)

$Best^g_{n,\tau}(\rho^*_{\tau,f}(DB,O,k)) =$
$= \rho^*_{\tau,f}(DB,O,k)$ if $n \geq k$,
$= \rho^*_{\tau,f}(DB,O,n)$ if $n < k$ and (C1) holds.
$Best^g_{n,\tau}(\rho^*_{\tau',f}(DB,O,k)) = \emptyset$ if $\tau \neq \tau'$
$Best^g_{n,\tau}(m_1 \oplus m_2) =$
$Best^g_{n,\tau}(Best^g_{n,\tau}(m_1) \oplus Best^g_{n,\tau}(m_2))$
if both (C1) and (C2) hold,
$Best^g_{n,\tau}(map_f m) = map_f(Best^{f\cdot g}_{n,\tau} m)$

The use of the above theorems for query optimization is illustrated by the following example.

Example 14

$$
\begin{aligned}
&Best(n, RSO_{\max}(\sigma_{\tau=\{a\}}(m1) \oplus \sigma_{\tau=\{b\}}(m2), \{a\}), 3) = \\
&= RSO_{\max}(Best^{\max}_{n,\{a\}}(Best^{\max}_{n,\{a\}}(\sigma_{\tau=\{a\}}(m1)) \oplus \\
&\quad Best^{\max}_{n,\{a\}}(\sigma_{\tau=\{b\}}(m2)), \{a\}) \\
&= RSO_{\max}(Best^{\max}_{n,\{a\}}(Best^{\max}_{n,\{a\}}(m1) \oplus \emptyset), \{a\}) \\
&= RSO_{\max}(Best^{\max}_{n,\{a\}}(Best^{\max}_{n,\{a\}}(m1)), \{a\})
\end{aligned}
$$

Note that the calls to the similarity implementations in m_2 and one of the selections have been removed.

The following result specifies the relationship between *Best* and the different types of join. It says that under certain conditions, performing *Best* after doing a similarity join is the same as performing *Best* first on each of the similarity measures being joined, then joining the results, and then repeating the *Best* operation.

Theorem 11 (Interaction of *Best* with Similarity Joins) Suppose f is monotone w.r.t. $\leq_g$ and (C1) and (C2) hold. Then:

$$
\begin{aligned}
&Best^g_{n,\tau}(m_1 \bowtie_{op} m_2) = \\
&\quad = Best^g_{n,\tau}(\bigoplus_{\tau=\tau_1\ op\ \tau_2} Best^g_{n,\tau_1}(m_1) \bowtie_{op} Best^g_{n,\tau_2}(m_2)). \\
&\quad = Best^g_{n,\tau}(\bigoplus_{\tau=\tau_1\ op\ \tau_2} Best^g_{n,\tau}(\ Best^g_{n,\tau_1}(m_1) \bowtie_{op} \\
&\qquad\qquad Best^g_{n,\tau_2}(m_2))).
\end{aligned}
$$

The following example shows some uses of these results in optimizing some simple queries.

Example 15

$$
\begin{aligned}
&Best^{\min}_{5,\{a,b\}}(\rho\_A^*_{\{a\}}(DB,O,all) \bowtie_\cup \rho\_B^*_{\{b\}}(DB,O,all)) = \\
&\quad = Best^{\min}_{5,\{a,b\}}(\bigoplus_{\tau 1 \cup \tau 2 = \tau} Best^{\min}_{5,\tau 1}(\rho\_A^*_{\{a\}}(DB,O,all)) \bowtie_\cup \\
&\qquad Best^{\min}_{5,\tau 2}(\rho\_B^*_{\{b\}}(DB,O,all))) \\
&\quad = Best^{\min}_{5,\{a,b\}}(Best^{\min}_{5,\{a\}}(\rho\_A^*_{\{a\}}(DB,O,all)) \bowtie_\cup \\
&\qquad Best^{\min}_{5,\{b\}}(\rho\_B^*_{\{b\}}(DB,O,all))) \\
&\quad = Best^{\min}_{5,\{a,b\}}(\rho\_A^*_{\{a\}}(DB,O,5) \bowtie_\cup \rho\_B^*_{\{b\}}(DB,O,5))
\end{aligned}
$$

4 Cost Model for $\mathcal{MSA}$

In this section, we will consider a cost model for query processing using $\mathcal{MSA}$s. In this model, we make the following assumptions. First, we assume that all similarity implementations are linked to the query processor as foreign functions. On the average, the cost of executing similarity implementations is an order of magnitude higher than the cost of joining tables in the similarity algebra. In general, the number of similarity implementations executed per query is a negligible number in comparison with the number of objects. Hence, we will ignore the cardinality of the set of similarity values in similarity implementations for cost estimation purposes. This is certainly a reasonable assumption – for example, when considering Internet search engines as in I.SEE, the number of search engines is negligible when compared to the number of objects (web pages) accessible through the search engines.

4.1 Cost Estimation in the Similarity Algebra

First, we will estimate the cost of execution for similarity algebra operations. We recognize that the cost of similarity implementations is the bottleneck of the operations performed in the similarity algebra. For example, in the face recognition example, the face recognition software is the one that is a bottleneck - similarly, in the case of the Internet search engine, most computation time consumed is in the access and execution by the Internet search engine – this includes both the connect time and the execution time. We will model their cost with respect to the similarity abstractions computed on top of them.

Definition 4.1 (Cost Estimate) Suppose m is a similarity measure. A cost estimate of the tuples in m of the form $\langle O, O', \tau, s \rangle$ is a triple $\langle card, T, \sigma \rangle$ where *card* is a set of pairs of the form (τ, v) (denoted by $card_\tau$), such that v is the expected cardinality of the pairs of type τ, T is the expected time cost of computing all the tuples in m, and σ is the expected number of different objects O'.

Given a cost estimate $\langle \{(\tau_1, v_1), \ldots, (\tau_k, v_k)\}, T, \sigma \rangle$ of a similarity measure, the expected cardinality is given by the following expression $\Sigma_{i=1}^k v_i$.

o_1	o_2	τ_1	$\{0.8, 0.9\}$
o_1	o_2	τ_2	$\{0.5, 0.7\}$
o_1	o_3	τ_1	$\{0.2, 0.9\}$
o_2	o_3	τ_2	$\{0.5, 0.8\}$

Suppose m_1 is the similarity measure given above and $\langle\{(\tau_1,2),(\tau_2,3)\},20,2\rangle$ is a cost estimate for m_1. According to this estimate, we expect at most 2 objects of type τ_1 and 3 objects of type τ_2. The expected number $\sigma = 2$ indicates that there are at most two different objects, possibly corresponding to the sum of objects retrieved by similarity implementations. Finally, the expected time is $T = 20$ assuming 5 time units for each tuple in m_1 - note that all four tuples in m_1 are considered when deriving 20 here because each of them must be taken into account when retrieving objects associated with type τ_1. The expected cardinality of this cost estimate is $2 + 3 = 5$.
The cost estimate of a similarity abstraction $\rho^*_{\tau,f}(DB,O,n)$ is given by the expression:

$$\langle\{(\tau, min(n,|DB|))\}, f_\rho(\tau, min(n,|DB|)), n\rangle$$

for some function f_ρ that is monotonically non-decreasing in its second argument. Recall that $\rho^*_{\tau,f}(DB,O,n)$ is a request saying "Find the top n matches for object O in the database DB, using τ and f to define the concept of a best match (as described earlier in the paper)." Only a single type is returned by the similarity abstraction and the expected number of objects O' is $min(n,|DB|)$. Similarly, cost parameters for a similarity abstraction $\kappa^*_{\tau,f}(O,O')$ are given by the triple: $\langle\{(\tau,1)\}, c_\kappa, 1\rangle$. The explanation for this is that we merely want to see how closely objects O, O' are matched, and this can be done in constant time, for some constant, c_κ.

We now examine the cost of similarity algebra operations. Let m_1 and m_2 be similarity measures with associated cost vectors $\langle c_1, T_1, \sigma_1\rangle$ and $\langle c_2, T_2, \sigma_2\rangle$. Suppose $(\tau_1, v_1) \in c_1$ represents a tuple in c_1 and $(\tau_2, v_2) \in c_2$ represents a tuple in c_2. The following table shows how we can estimate the cardinalities and selectivities of algebraic operations applied to m_1, m_2. Due to space constraints, we will not go into their detailed derivations.

Operator	Cardinality	Selectivity
$m_1 \bowtie_{op} m_2$	$\bigcup\{(\tau, min(\sigma_1,\sigma_2)(\sum X))\}$ where $(\tau,v_1)\in c_1$, $(\tau,v_2)\in c_2$, $\tau_1\ op\ \tau_2 = \tau$, $X = min(\frac{v_1}{\sigma_1}, \frac{v_2}{\sigma_2})$	$min(\sigma_1,\sigma_2)$
$m_1 \oplus m_2$	$\bigcup\{(\tau, v_1+v_2)\}$ where $(\tau,v_1)\in c_1$ and $(\tau,v_2)\in c_2$	$max(\sigma_1,\sigma_2)$
$\sigma_P(m_1)$	$\bigcup\{(\tau,\sigma_P v)\}$ where $(\tau,v)\in c_1$, σ_P is the selectivity of P	$\sigma_1\sigma_P$
$map_f(m_1)$	c_1	σ_1

4.2 Query Optimization in $r\mathcal{MSA}$

In this section, we discuss the basics steps involved in optimizing queries in $r\mathcal{MSA}$. We first define the concept of a query tree, and then provide various operations that transform query trees into equivalent, simpler query trees.

4.2.1 Query Tree

Given a query formulated in the correct algebraic form, we construct a query tree for this expression. An $r\mathcal{MSA}$ query tree is very similar to a relational query tree, but it also includes the similarity algebraic operators.

Definition 4.2 (Query Tree) A query tree associated with an $r\mathcal{MSA}$ query Q is a labeled directed acyclic graph $G_Q = (V, L, E)$ with a unique root where V, L, E are set of vertices, labels and labeled edges respectively. Edges are denoted by $(n_1 \xrightarrow{e} n_2)$ for some $n_1, n_2 \in V$ and $e \in L$.

An admissible query tree G_Q satisfies the following conditions. (1) All leaves correspond either to a database table or a similarity abstraction. (2) All labels in G_Q contain expressions involving selection, projection, plus similarity abstraction, Sim-Select, Sim-Map, *Best* and *RSO* operators. Labels may also be equal to λ which indicates no selection operation to be performed at that label. (3) If v is a node with both incoming and outgoing edges ("interior node"), then v corresponds to either a similarity operator (with the exception of Sim-Select and Sim-Map), a relational join operation or a special operator called a similarity view (denoted by SV). Intuitively, a similarity view is a similarity algebra query that has been previously computed and stored. $T(v)$ denotes the set of edges contained in the subtree rooted at vertex v in a query graph G. Hence, an edge $(v_1 \xrightarrow{e} v_2) \in T(v)$ if vertex v_1 is reachable from vertex v. If v_1 is reachable from v and if there is an edge from v_1 to v_2, then this edge is in the subtree rooted at v.

The construction of an initial query tree from a given query is straightforward. However, in general this tree must be reorganized to lower the cost of query processing. To this end, we will introduce tree manipulation operators. Due to the high execution cost of similarity implementations, we want to make use of common subexpressions as much as possible. This requires identifying common equivalent subexpressions in a query, treating those common equivalent subexpressions as views, and materializing them to avoid redundant computation. The tree compaction operator introduced below handles this case.

Definition 4.3 (Tree Compaction) Let $G = (V, L, E)$ be a query tree and let the edges $(n_1 \xrightarrow{e} n_2)$, $(n_1' \xrightarrow{e'} n_2')$ be elements of G such that n_2 is not a leaf node. The relation $n_2 \equiv n_2'$ holds whenever the query corresponding to the subquery of n_2 is equivalent to the query corresponding to the subquery of n_2'. The tree compaction operator $TCO_\equiv$ returns the following graph whenever $n_2 \equiv n_2'$:

$$TCO_\equiv(E, (n_1 \xrightarrow{e} n_2), (n_1' \xrightarrow{e'} n_2')) =$$
$$(E \cup X) - T(n_2') - (\{(n_1' \xrightarrow{e'} n_2'), (n_1 \xrightarrow{e} n_2)\}).$$

$$X = \begin{cases} \{(n_1 \xrightarrow{e} SV), (n_1' \xrightarrow{e'} SV), (SV \xrightarrow{\lambda} n_2)\} & \text{if } e \neq e' \\ \{(n_1 \xrightarrow{\lambda} SV), (n_1' \xrightarrow{\lambda} SV), (SV \xrightarrow{e} n_2)\} & \text{if } e = e'. \end{cases}$$

Intuitively, the above tree compaction operator looks at two edges, $(n_1 \xrightarrow{e} n_2)$ and $(n_1' \xrightarrow{e'} n_2')$ in the graph where n_2 and n_2' are equivalent. We have already provided a comprehensive list of equivalent queries in earlier parts of this paper. The TCO operator then treats n_2 as a similarity view SV that is computed only once. It then reconstructs the tree by appropriately replacing explicit calls to n_2, n_2' by SV and ensuring that the view itself is computed in the tree once (and only once).

The significance of the tree compaction operator is the detection of common subexpressions that can be cached

and re-used for query processing. A similar tree compaction operator can also be defined for the subsumption relation ($\subseteq$) that does not guarantee equality. The $TCO_{\subseteq}$ operator can be used in conjunction with the containment preserving query rewriting methods to reduce the cost of retrieving first set of tuples. In either case, we need to use query rewriting methods that depend on syntactic or semantic description of query contents. Such methods were discussed in the literature for different applications [2, 16]. Extending these methods to the similarity algebraic operations is beyond the scope of this paper.

In addition to the tree compaction operator, we use the traditional selection push operators below join operations. The intended gain in pushing selections is the reduction of the number of tuples processed. The legal selection operations were discussed in Section 3. The important point to note is that a selection is pushed down a node n if n has a single incoming edge. If two nodes are pointing to the same common subexpression $T(n)$ when a selection operation was pushed down, then $T(n)$ must be duplicated and the expression must be pushed only for the appropriate node. Hence, pushing selections have the obvious effect of reducing the number of common subexpressions.

4.2.2 Computing with Similarity Abstractions

In this section, we discuss possible optimizations for the execution of similarity abstractions. We assume that for large numbers n, all answers to an expression of the form $\rho^*_{\tau,f}(DB, O, n)$ will be returned in a nontrivial time interval. As similarity computations are usually very costly, the rest of the selection operations can be interleaved with the retrieval of similarity attributes. In this model, the query processor should stop the execution of a similarity implementation when the necessary conditions have been met to evaluate the rest of the query. We first define the notion of a similarity abstraction filter.

Definition 4.4 (Similarity Abstraction Filter) Let σ be a selection operation involving sim-map, sim-select, and $Best^g_{n,\tau}$. A similarity abstraction filter associated with σ is a function of the form $z(v) \geq lb$ where lb is a constant and $z(v)$ preserves ordering of real numbers with respect to singleton sets such that whenever $z(v) \geq lb$ is false for a set of values v, then the selection condition σ evaluates to false for v.

Similarity abstraction filters need not always exist. For them to be calculated, σ should involve selections over values. As we will see in the implementation section, the existence of query filters introduce great savings in query response time. Suppose $cost(|DB|, n)$ characterizes the execution cost of $\rho^*_{\tau,f}(DB, O, n)$, $cost_{conv}(|DB|)$ is the cost of creating index structures necessary for executing ρ and the cost of σ is c_1. Pushing selections into similarity abstraction offers four possibilities:

1. Push selection if possible to create an intermediate table, execute ρ on this table. $\sigma(\rho^*_{\tau,f}(DB, O, n)) = \rho^*_{\tau,f}(\sigma(DB), O, n)$. Execution cost of $\rho^*_{\tau,f}(\sigma(DB), O, n)$ is $c_1 + cost_{conv}(|\sigma(DB)|, O, n) + cost(\sigma(|DB|), n)$.
2. Push selection if possible into ρ to create a modified similarity implementation execution. In this case, the selection is performed at the database that is computing the σ function, hence the query is optimized for execution as a whole. $\sigma(\rho^*(DB, O, n)) = \rho^*_{\tau,f}(\sigma(DB), O, n)$. Execution cost of $\rho^*_{\tau,f}(\sigma(DB), O, n)$ is $cost(\sigma(|DB|), n)$.
3. No push is possible since the database DB does not support selections and all the objects are retrieved from DB via similarity implementations. The execution cost of a naive implementation is then $\sigma(\rho^*_{\tau,f}(DB, O, n) = cost(|DB|, n) + c_1$.
4. Suppose selection cannot be pushed down, but there exists a similarity abstraction filter $z(v) \geq lb$ associated with σ. If f is an order preserving function, then ρ^* is computed until $z(v) \geq lb$ becomes false. We denote this operation with $\sigma([z, lb](\rho^*_{\tau,f}(DB, O, n))$ The worst case cost of this execution is the same as the previous case. Experimental results are provided to show the average behaviour of the filter operation.

Suppose now we are computing the following join expression $\sigma(\rho^*_{\tau_1,f_1} \bowtie_{op} \rho^*_{\tau_2,f_2})$. Two new optimizations for this operation are introduced below:

Join Cardinality Filter: Suppose $\sigma = Best^g_{n,\tau}$ such that g is an order preserving function over bags of two real numbers. In other words, whenever $v_1 \leq v_2$ and $v_3 \leq v_4$, then $g(\{v_1, v_3\}) \leq g(\{v_2, v_4\})$. Then, if both f_1 and f_2 are order preserving relations, replace the above join with the following filter join:

$$\sigma(\rho^*_{\tau_1,f_1} \overset{(g,|n|)}{\bowtie_{op}} \rho^*_{\tau_2,f_2}).$$

The join cardinality filter is executed as follows. The execution of similarity operations are performed in stages, where in each stage, the join is performed and the best elements are chosen after g is executed. Recall that g is a function that operates on bags of reals and returns a real as output. Suppose x is the nth highest value found so far in the join with respect to function g, $\{h_1\}$ is the highest element in $\rho^*_{\tau_1,f_1}$ that did not contribute to joined tuple, and $\{l_1\}$ is the lowest element in the bags computed by $\rho^*_{\tau_1,f_1}$. The same is true for $\{h_2\}$ and $\{l_2\}$ as well. The cardinality join filter will request the next set of similarity abstractions if either $g(\{h_1, l_2\}) \geq x$ or $g(\{h_2, l_1\}) \geq x$ is true. At the end of every step, the high and low values will be updated.

Join Value Filter: Suppose there exists a relation of the form $y(v_1, v_2) \geq lb$ where lb is a constant and y is an order preserving function over bags containing two real numbers such that whenever this relation evaluates to false, σ is not satisfied. Then, if both f_1 and f_2 are order preserving relations, replace the above with the following filter join:

$$\sigma(\rho^*_{\tau_1,f_1} \overset{[y,lb]}{\bowtie_{op}} \rho^*_{\tau_2,f_2}).$$

The join value filter is executed similar to the join cardinality filter, where the execution of similarity abstractions is stopped when both $y(\{h_1\}, \{l_2\}) \geq lb$ and $y(\{h_2\}, \{l_1\}) > lb$ evaluate to false.

Intuitively, both of the filters take advantage of the fact that similarity implementations return the answers in the descending order of similarity for the metric that

they are using. The cardinality filter checks if it is possible to obtain a better answer in the join from the unmatched elements in the set. The join value filter checks if the joining tuples will no longer satisfy the selection condition. This is the important motivation for pushing selections down as much as possible. In general, it is very hard to generate filters for arbitrary functions. However, we assume that functions that are not combinations of the average, min and max functions are complex operations that need to be implemented and linked to the algebraic query processor. It is also possible for filters to be defined for these functions in a declarative way. The query processor can then choose these filter definitions and derive combination filters when necessary.

As in any algebraic system, the query optimization process consists of rewriting queries using the available operators such that the query with the lowest estimated cost can be found and executed. As the space of all possible rewritings can be prohibitively large, certain heuristics can be developed to explore and prune the search space. Even though pushing selections down as much as possible is almost always desirable, this operation reduces the amount of common subexpressions that can be combined into one. However, checking equivalence and containment between subgraphs of a graph is a very costly operation. Involved containment checks are more suitable for fixed queries that are optimized at compile time. In general, query optimization can be solved by any directed generate-and-search algorithm with a predetermined heuristic benefit function.

5 Implementation and Experiments

In this section, we will describe the current implementation of the similarity algebra on top of the I.SEE (Integrated SEarch Engine) system that is being built at Rensselaer Polytechnic Institute. I.SEE provides a common interface to query interfaces of multiple search engines. Detailed search queries are posed in a generic language with the help of explicit menu items. These queries are in turn rewritten by relaxation to match the capabilities of the underlying search engines. In addition, I.SEE executes all the translated search queries in parallel and then merges the results with respect to the relevant similarity algebra expression. More detailed information on I.SEE can be found at *www.cs.rpi.edu/research/isee/*.

The implementation supports all similarity algebra operations, as well as *Best*, *RSO* and relational selection operators. Only functions allowed in map_f and *RSO* operators are min, max and average. All the similarity abstractions are assumed to contain order preserving operations. The optimization heuristic performs query compaction only on the individual similarity abstractions. The selections are pushed as much as possible and joins are replaced by filter joins whenever it is possible. Figure 3 shows an example query session from I.SEE. The query posed in this example is the first query given in Example 10.

Below, we give experimental results showing the utility of query optimization in similarity algebra operations. The results corresponds to the three queries given in Example 10. Each query is shown executed for the string,

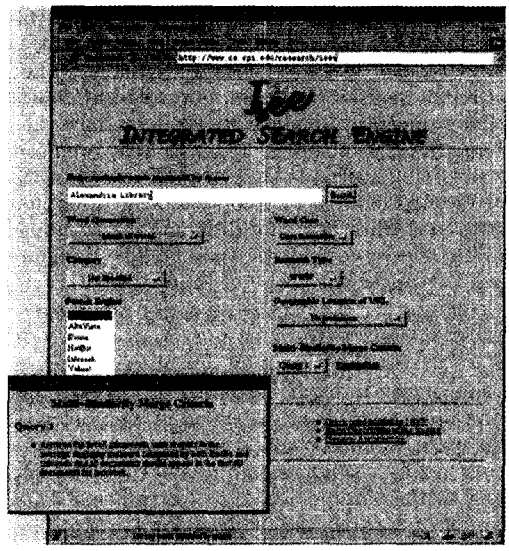

Figure 3: Query 1 in I.SEE

"Alexandria Library". For each query string, two query plans are considered. The first is the results of naive query evaluation without any optimization. The second evaluation is a query plan that makes use of filter joins. For each query, the rewriting is given below. Note that no filtering is possible for q_3 since there is no selection condition on it. However, in this case it is possible to reduce the common subexpressions. We now give the experimental results corresponding to the execution of these two queries.

$$q_1 \equiv \Pi_{\$1,\$3}(RSO_{avg}(Best^{avg}_{5,\{E,I\}}(\rho^*_{\{E\},x/100}(EDB,O,all) \underset{(avg,|n|)}{\bowtie_\cup} \rho^*_{\{I\},x/100}(IDB,O,60)),\{E,I\}),3)).$$

$$q_2 \equiv \Pi_{\$1,\$3}(RSO_{min}(\sigma^m_{\$4\geq 0.8} in(\rho^*_{\{E\},x/100}(EDB,O,all) \underset{[min,0.6]}{\bowtie_\cup} \rho^*_{\{I\},x/100}(IDB,O,all)),\{E,I\})))$$

$$q_3 \equiv \Pi_{\$1,\$3}(RSO_{avg}((SV_1 \bowtie_\cup SV_2),\{E,H\}) \cup RSO_{x/2}((SV_1 \setminus SV_2),\{E\}) \cup RSO_{x/2}((SV_2 \setminus SV_1),\{H\}))$$

$$SV_1 = \rho^*_{\{E\},x/100}(IDB,O,20)$$

$$SV_2 = \rho^*_{\{H\},x/100}(EDB,O,20).$$

Query String	# Network Connections	# Tuples Processed	Execution Time
"Alexandria Library"			
Query 1, Naive	106	63000	254.7s
Query 1, Rewritten	47	30240	107.8s
Query 2, Naive	106	63000	252.8s
Query 2, Rewritten	40	8303	83.2s
Query 3, Naive	12	1118	7.3s
Query 3, Rewritten	4	1118	7.2s

The table above illustrates the following points: First, using our query rewriting strategies, the number of requested network connections decreases substantially, as does the number of tuples processed. The total time required to process our queries does decrease substantially as a result of query rewriting. However, query rewriting is not always effective. For example, in the case of query q_3, the similarity union-join $\bowtie_\cup$ was empty, leaving little room for optimization. In general, we have noticed that the expected number of objects after a join operation was

a significant factor in optimization. This figure was also very hard to estimate for arbitrary query strings, such as "Postmodern Culture Netzine". Nevertheless, we could cut the time required to process queries in half using our query optimization methods in most cases.

6 Conclusions and Related Work

Most existing similarity retrieval algorithms assume that the notion of similarity is fixed, and that user queries are processed against this fixed similarity metric. In practice, however, there are many applications of similarity based retrieval (e.g.document retrieval through Internet search engines are the best known example) where there are many different indexes and algorithms for similarity based retrieval. This is also true (to a slightly lesser extent) in image retrieval domains where multiple image processing algorithms often retrieve data in multiple ways according to multiple notions of similarity. Our aim in this paper was two-fold. First, to provide an algebraic framework within which users can access not just one, but several similarity implementations, and "mix and match" the results of such accesses within a single algebra. To date, there has been little work on developing a version of the relational algebra for images/text. Our architecture applies to not just one, but many such implementations and so does our algebra. In addition, our algebra *extends* the relational algebra rather than start from scratch, thus making it present a relational interface to non-relational structures – thus, it is consistent with prevailing trends in industry to add "datablades", "extenders", and "cartridges" to relational DBMS systems to extend their capabilities. Our second goal was to develop ways of optimizing queries over such similarity based retrieval engines. We accomplished this by proving a set of equivalences between queries (which in turn lead to query rewrite rules) in the $r\mathcal{MSA}$. Later we defined query trees and showed how query trees may be transformed through specialized operators such as the tree compaction operator, similarity abstraction filter based optimizations, join cardinality filters, and join value filters. By transforming query trees using these operations, we may rewrite a query tree into an equivalent one and use our cost model to choose the best. Last, but not least, we have developed a prototype application of our algebra to develop an Integrated Web Search Engine (I.SEE) based upon which we have validated the experimental benefits of our approach.

Most existing research on similarity based retrieval to date falls into three classes. The first class aims at defining what constitutes similarity, and has been studied by many researchers in the context of text/document data [6, 17, 1, 1], image data [3, 9, 11, 14], and audio data [5]. Santini and Jain [15] provide a good overview of different approaches to similarity. The second class of work involves ways to index media data to support similarity-based retrieval – these include data structures such as those in [4, 9, 13]. The third class includes providing abstract models of similarity as elegantly developed by Jagadish et. al. [12]. Chaudhuri and Gravano [8] were the first to suggest ways of optimizing multimedia queries using a cost based model of such queries. Carey and Kossmann [7] introduced an SQL operator analogous to "Best" named "stop after" for relational queries that aims to reduce the cardinality of input streams. Our work, in contrast, provides an algebra for multiple similarity engines to be integrated ([10] also makes a proposal for this in the context of Internet Search Engines, but provides no algebra or equivalence results) and provides provably correct query rewriting methods, and query optimization techniques.

References

[1] S. Adali, C. Bufi and Y. Temtanapat. "Integrated Search Engine", to appear in the Proc. of 1997 IEEE Knowledge and Data Engineering Exchange Workshop, KDEX'97.

[2] S. Adali, K.S. Candan, Y. Papakonstantinou and V.S. Subrahmanian. "Query Caching and Optimization in Distributed Mediator Systems", Proc. of the 1996 Sigmod Conference on Management of Data, pp. 137 - 148.

[3] M. Arya, W. Cody, C. Faloutsos, J. Richardson and A. Toga. (1995) "Design and Implementation of QBISM, a 3D Medical Image Database System", in: (V.S. Subrahmanian and S. Jajodia, eds.) "Multimedia Database Systems: Issues and Research Directions", Springer 1995.

[4] S. Berchtold, D. A. Keim, and Hans-Peter Kriegel. (1996) "The X-tree : An Index Structure for High-Dimensional Data", Proc. 1996 Intl. Conf. on Very Large Databases, Bombay, India, pps 28-39.

[5] T. Blum, D. Keislar, J. Wheaton and E. Wold. (1995) "Audio Databases with Content-based Retrieval", Proc. 1995 IJCAI workshop on Intelligent Multimedia Information Retrieval, Montreal, Canada.

[6] E.W. Brown, J.P. Callan and W. B. Crof. (1994) "Fast Incremental Indexing for Full-Text Information retrieval", Proc. 1994 Intl. Conf. on Very Large databases, Santiago, Chile, pps, 192–202.

[7] M. J. Carey and D. Kossmann (1997) "On Saying "Enough Already!" in SQL." Proceedings of the 1997 Sigmod Conference on Management of Data, pp. 219-230.

[8] S. Chaudhuri and L. Gravano. (1996) "Optimizing Queries over Multimedia Repositories", Proc. 1996 ACM SIGMOD Conf. on Management of Data, pps 91–102.

[9] C. Faloutsos. (1996) "Searching Multimedia Databases by Content", Kluwer Academic Publishers.

[10] L. Gravano, C. Chang, H. Garcia-Molina and A. Paepcke. "STARTS: Stanford Proposal for Internet Meta-Searching", SIGMOD 1997, pp. 207–218.

[11] V.N. Gudivada and V.V. Raghavan. (1993) "Design and Evaluation of Algorithms for Image Retrieval by Spatial Similarity", ACM Transactions on Information Systems, 13,1, pps 115-144.

[12] H. V. Jagadish, A. Mendelzon and T. Milo. "Similarity-Based Queries", Proc. of ACM PODS 1995, pp. 36–45.

[13] K.-I. Lin, H.V. Jagadish and C. Faloutsos. (1994) "The TV-Tree: An Index Structure for High-dimensional Data", VLDB Journal, 3, pps 517–542.

[14] W. Niblack, et. al. (1993) "The QBIC Project: Querying Images by Content Using Color, Texture and Shape", IBM Research Report, Feb. 1993.

[15] S. Santini and R. Jain. (1996) "Similarity Matching", to appear in: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[16] X. Qian. "Query folding." In Proceedings of the Twelfth International Conference on Data Enginnering, 1996.

[17] A. Tomasic, H. Garcia-Molina and K. Shoens. (1994) "Incremental Updates of Inverted Lists for Text Retrieval", Proc. 1994 ACM SIGMOD Conf. on Management of Data, Minneapolis, pps 289–300.