An Optimal Set of Indices for Dynamic Combinations of Metric Spaces

Benjamin Bustos*
Center for Web Research
Department of Computer Science
University of Chile
bebustos@dcc.uchile.cl

Nelson Morales†
Instituto Innovacion en Mineria y Metalurgia
nmorales@dim.uchile.cl

ABSTRACT

A recent trend to improve the effectiveness of similarity queries in multimedia databases is based on dynamic combinations of metric spaces. The efficiency issue when using these dynamic combinations is still an open problem, especially in the case of binary weights. Our solution resorts to the use of a set of indices. We describe a binary linear program that finds the optimal set of indices given space constraints. Because binary linear programming is NP-hard in general, we also develop greedy algorithms that find good set of indices quickly. The solutions returned by the approximation algorithms are very close to the optimal value for the instances where these can be calculated.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content analysis and indexing—*indexing methods*

Keywords

Multimedia databases, content-based retrieval, multi-metric spaces

1. INTRODUCTION

Due to a rapidly growing amount of available multimedia data, the development of multimedia database systems is becoming increasingly important. Multimedia databases have applications in many fields such as molecular biology, medicine, CAD/CAM applications, GIS, etc. The progress in the acquisition, storage, and dissemination of various multimedia formats, makes the application of effective and efficient database management systems indispensable.

One of the most important tasks in multimedia databases is to implement effective and efficient similarity search algorithms. Multimedia objects cannot be meaningfully queried in the classical sense (exact search). Instead, the objects in the multimedia domain are treated as objects in a *metric space*, which can be compared with a metric function appropriately defined. A query in a multimedia database system usually requests the most *similar objects* to a given query object or a manually entered query specification. Note that *vector spaces* (e.g., those obtained by using a feature extraction fuction on the multimedia objects) are a particular case of metric spaces.

An example application of multimedia databases is 3D model retrieval. 3D objects are an important multimedia data type with many application possibilities. They can present complex information, and content-based searching in large 3D object repositories arise in many practical fields. For example, in medicine detecting similar organ deformations is useful for diagnostic purposes. 3D object databases also support CAD tools, which have many applications in industrial design and manufacturing: Reusing standard parts can lead to reduced production costs. The main problem is to assess the similarity between any pair of 3D objects based on a suitable notion of similarity. The existence of such similarity measures is an important precondition for implementing effective search algorithms; it lets you query a repository of 3D objects for specific content and facilitates reusing 3D content. Also, similarity metrics let you organize 3D repositories by representing large object collections with limited cluster prototypes, or visualize the content of large databases by appropriate 2D mappings. A similarity notion supports advanced automatic applications such as classifying shapes in industrial screening.

In the standard approach for defining the similarity notion, only one fixed metric distance (MD) is used within a multimedia database for performing similarity queries. However, it is possible to map each multimedia domain into many different metric spaces. For example, in the case of 3D model retrieval there are more than 30 proposed mapping techniques [3].

It has been shown that by using *query dependent combinations of MDs* it is possible to significantly increase the effectiveness of the similarity search [2]. The idea is to select a fixed number of the available MDs by using a query processor, which identifies the best suited MDs for the given query object, and then linearly combine them to perform the similarity query. A binary weight (0 or 1) is used to define which MDs will be taken into account for the combination.

*Funded by Millennium Nucleus Center for Web Research, Grant P04-067-F, Mideplan, Chile.

†Work done at MASCOTTE project, I3S/INRIA/Universite de Nice Sophia-Antipolis.

WOODSTOCK '97 El Paso, Texas USA

We address the problem of indexing combinations of t MDs from a given set of F metric spaces ($t \leq F$). An ideal solution would be to build an index for each possible combination, but in practice this is not possible due to space constraints (the number of possible combinations is $\binom{F}{t}$). A naïve solution to this problem is to use one of the indices especially developed for multi-metric spaces [4, 5], where the weights are not binary but positive real values, and to index all available metric spaces with it. Unfortunately this solution is not efficient, because the search algorithm needs to read information about all the available metric spaces at query time, including those not relevant for the given query.

Instead, it could be more efficient to use several multi-metric indices with small sizes. It turns out that the most efficient choice is to have one index per combination (cf. Section 3.1). Unfortunately this is unrealistic, because the available space for bulding indices is limited. Furthermore we observed that, in real-world multimedia applications, the query processor selects some MDs more often than others, and therefore each combination has a certain probability of being selected. This raises a natural question: Given a certain amount of available space, which is the set of indices that minimizes the *average search time* over the combinations selected by the query processor?

We tackle the problem of finding the most efficient set of indices (*iSet*), considering the fact that the combinations are selected according to certain (estimated) distribution. Our measure of efficiency is the expected search cost of the *iSet*. We show that this problem can be formulated as a binary linear program (BLP), which is able to find the optimal *iSet*, and use it to provide experimental evidence to validate our approach. Unfortunately, the standard methods for solving BLPs are not efficient (binary linear programming is NP-hard). Thus, we also develop approximated algorithms to find good *iSets* in short time.

2. BASIC CONCEPTS AND RELATED WORK

Multimedia databases can be modeled as a *metric space* to implement similarity queries [7]. There is a universe $\mathbb{X}$ of *objects*, and a nonnegative *distance function* δ defined among them, which satisfies the properties of a metric (strict positiveness, symmetry, and the triangle inequality). The smaller the distance between two objects, the more similar they are. We have a finite *database* $\mathbb{U} \subset \mathbb{X}$, which can be preprocessed to build an index. Later, given a query object $q \in \mathbb{X}$, we must retrieve all similar elements found in $\mathbb{U}$.

A particular case of this problem arises when the space is $\mathbb{R}^d$. There are effective methods for this case, such as the kd-tree, R-tree, and X-tree, among others [1]. However, there are many applications where the space cannot be regarded as d-dimensional (e.g., in string similarity problems that appear in computational biology applications). We concentrate in this paper in general metric spaces, although the solutions can be directly applied to d-dimensional spaces.

There are two basic types of similarity queries in multimedia databases: *Range query*, which reports all objects from the database that are within a given tolerance radius to q; and *k nearest-neighbors (k-NN) query*, which reports the k objects from the database closest to q.

2.1 Combinations of metric distances

A recent trend to improve the *effectiveness* of similarity queries (i.e., the quality of the answer returned by the similarity search system) is based on using *combinations of MDs*. The idea is to use not one but several metric spaces, thus obtaining a *set of MDs*. At query time, the MDs are combined by using, for example, a linear combination (weighted sum) of the distances with each MD. If the weights are positive (as in our case of binary weights), the combined metric is also a metric.

It has been observed [2] that a linear combination of all MDs will not provide the optimal results, because if one of the considered MDs has a very bad effectiveness for a given query object, then it will spoil the final result. Recently, dynamic combination methods have been proposed [2], which aim at avoiding this problem by combining only those MDs that are most promising for the given query object. The goodness of a MD is estimated against a training database, selecting the MDs with highest goodness values.

In this paper, we assume that the similarity search engine of the multimedia database implements a *query processor*, which given a query object selects t MDs (from the set of available MDs) that are used to perform the similarity query.

2.2 Related work

An index structure specifically designed for dynamically combined MDs (also known as *multi-metric spaces*) is presented in [5]. The M^3-tree is an adaptation of the original M-tree, and stores partial distances (one for each MD belonging to the combination) to dynamically estimate, for each performed query, the new covering radius of the space regions and the new distances from parent to children nodes. The index is built using a fixed combined metric (the index distance) that is an upper-bounding distance function of the query distance. For this index, the weights are values in $[0, 1]$, thus it needs to store information about all the available metric spaces to perform the queries (in contrast to our problem, where we know that we only use t of the MDs).

Another index structure specially designed for dynamically weighted combinations of MDs is presented in [4]. This index consists of a set of pivot-based indices, one for each MD. These can be used to compute the combined pivot table (i.e., the pivot-based index for the combination of MDs) at query time, when the weights for the dynamic combination are known. For this index, the weights can be any positive real value, thus it also needs to store information of all the available MDs.

3. INDEXING COMBINATIONS OF METRIC DISTANCES

Table 1 shows the notation used through the rest of the paper. Let $\mathbb{F}$ be a set of MDs and let $f \in \mathbb{F}$ denote a specific MD. A *combination of MDs* has the form $c \subseteq \mathbb{F}$. To perform similarity queries, the search system combines t of the MDs ($|c| = t$), thus there are $T = \binom{F}{t}$ possible combinations.

A query processor selects at query time one of the combinations to perform the similarity search. That is, given an object $q \in \mathbb{X}$, the query processor selects combination c_i ($1 \leq i \leq T$) with probability p_{c_i}, where $p_{c_i} \geq 0$ and $\sum_{i=1}^{T} p_{c_i} = 1$. Without loss of generality, in what follows we assume that $p_{c_i} \geq p_{c_{i+1}}$.

The similarity query can be solved by sequentially scanning each of the metric spaces related to the selected combination of MDs. The search cost of this sequental scan is given by the function $LS(t)$ (*linear scan*), which is $O(tn)$.

Table 1: Summary of symbols.

Symbol	Definition
$\mathbb{X}$	Set of valid objects
$\mathbb{U} \subset \mathbb{X}$	Database
$n = \|\mathbb{U}\|$	Database size
$q \in \mathbb{X}$	Query object
$\mathbb{F}$	Set of metric distances (MDs)
$F = \|\mathbb{F}\|$	Number of MDs
$f \in \mathbb{F}$	A single MD
$c \subseteq \mathbb{F}$	A combination of MDs
t	Number of combined MDs
$T = \binom{F}{t}$	Total number of combinations
$\mathbb{C}$	Set of all combinations of t MDs
p_c	Probability of selecting combination c
$\mathbb{I}$	Set of indices (*iSet*)
$idx \in \mathbb{I}$	An index from the *iSet*
S	Available space for building indices
$E(\mathbb{I})$	Expected search cost of *iSet* $\mathbb{I}$

Note that the sequential scan *cannot be efficiently implemented*, because it is not known a priori which MDs will be selected for the combination for a given query q. Thus, the data files cannot be optimally arranged on the secondary storage for a fast linear scan.

If there is enough available space to construct and save indices, then one could take advantage of it to improve the efficiency of the search. The idea is to build indices for the most frequently used combinations of MDs, thus reducing the expected search cost. We define an *index of size* k as an index that stores k MDs. The space cost of an index of size k is given by the function $Space(k)$, and its search cost is given by the function $Search(k)$. Both cost functions depend on the actually used index structure and on the data distribution for each metric space. We assume that the search cost always increases with the index size.

3.1 An example: Indexing single combinations

Suppose that we are only allowed to build indices of size t (these index only one combination of MDs and could be any metric access method). Let $\mathbb{I}$ be the set of built indices and let $idx \in \mathbb{I}$ denote a specific index. We refer to $\mathbb{I}$ as an *iSet*. Given that there is an amount S of available space to build indices, it follows that $|\mathbb{I}| \leq \lfloor S/Space(t) \rfloor$. The total space cost $R(\mathbb{I})$ is

$$R(\mathbb{I}) = |\mathbb{I}| \cdot Space(t).$$

The expected search cost depends on which combinations of MDs are indexed. If the query processor selects combination c to perform the query, the search engine checks if there exists an index $idx \in \mathbb{I}$ that contains c. If this is the case, idx is used to perform the similarity query. Otherwise, the search engine resorts to a linear scan over the database. Assume that combinations $c_{i_1}, c_{i_2}, c_{i_{|\mathbb{I}|}}$ are indexed, then the expected search cost $E(\mathbb{I})$ of the *iSet* is

$$E(\mathbb{I}) = \sum_{j=1}^{|\mathbb{I}|} p_{c_{i_j}} \cdot Search(t) + \left(1 - \sum_{j \notin \{i_1,\ldots,i_{\mathbb{I}}\}}^{T} p_{c_j}\right) \cdot LS(t).$$

Because $p_i \geq p_{i+1}$, it follows that the optimal choice is $i_j = j, j = 1, \ldots, |\mathbb{I}| = \lfloor S/Space(t) \rfloor$. In particular, if $T \cdot Space(t) \leq S$, then the optimal solution is to build an index for every possible combination of MDs.

3.2 The real problem: Indexing several combinations per index

If the indices of the *iSet* may contain more than t MDs (for example, by using some of the structures mentioned in Section 2.2), it may be possible to do better. An index of size k ($t \leq k \leq F$) contains $\binom{k}{t}$ combinations of MDs. This means that with only one index we can index simultaneously many combinations of MDs. Note, however, that an index of size greater than t *requires more MDs than necessary to be read* to perform the similarity query, thus making the search slower. Thus, there is a trade-off between search time and number of indexed combinations. Note that an index of size F contains all possible combinations of MDs.

The available space S may allow us to build many indices of size t or larger (not necessarily all of the same size). As indices may contain more than one combination of MDs, a given combination could be contained in more than one index. The best choice then is to use the *index with smallest search cost* that contains c to perform the similarity query, which is equivalent to select the smallest index. The function $ms_{\mathbb{I}}(c)$ determines the size of this smallest index:

$$ms_{\mathbb{I}}(c) = \begin{cases} k & \text{if } k \text{ is the size of the smallest index in } \mathbb{I} \text{ that contains } c, \\ \infty & \text{if no index in } \mathbb{I} \text{ contains } c. \end{cases}$$

It follows that $t \leq ms_{\mathbb{I}}(c) \leq F$ if there exists an index in the *iSet* that contains c. Thus, the expected search cost for the *iSet* $\mathbb{I}$ is

$$E(\mathbb{I}) = \sum_{c: ms_{\mathbb{I}}(c) < \infty} p_c \cdot Search(ms_{\mathbb{I}}(c)) + \sum_{c: ms_{\mathbb{I}}(c) = \infty} p_c \cdot LS(t).$$

The space constraint must be respected. If $size(idx_i)$ is the number of MDs that idx_i contains, then

$$R(\mathbb{I}) = \sum_{i=1}^{|\mathbb{I}|} Space(size(idx_i)) \leq S.$$

Once the functions $Search(k)$ and $Space(k)$ are appropriately defined (see Section 7 for an example), the question is, what is the optimal *iSet* to build, given the probabilities of selecting each combination of MDs and the space constraints? To find the optimal solution, we have to take into account that: (a) There is a trade-off between index size and search cost. An index that contains more MDs indexes more combinations, but its search time will be *longer* than the search time of a smaller index; (b) A combination may be indexed by many indices, but the search system must use the one with smallest search cost; (c) We have a limited space S available.

The problem can be formalized as the next optimization problem: *Given a set of combinations of MDs, their probabilities of being selected, the search and space cost functions for the index structures, and the available space for building indices, find the optimal* iSet *so that the expected search cost is minimized.*

3.3 This is NOT a classical packing problem

Notice that the problem of minimizing the expected search time does not correspond to a classical packing problem [8]. Indeed, in a packing problem we have objects and boxes.

Objects use some space (if packed) and produce certain benefit (again if packed). Boxes provide only a limited capacity. The goal in a packing problem is to maximize the benefit of the packed objects. Notice that (a) the benefit of an object depends only on the decision to pack it or not, and (b) the sizes of the boxes are given in advance.

In the case of minimizing the expected search cost, we may think of objects (either MDs or combinations) and boxes (indices), but the similarities stop there. In regards to (b) we observe that in our problem only the total amount of space is global while we can choose the size of our boxes to match the size of the contained objects (there is no wasted space per box). Respecting (a), we can observe that the benefit obtained when packing a MD depends on the fact that a given MD (or combination) is packed or not, and on which index we put the MD (or combination). Roughly, the larger the index, the higher the expected search cost.

In any case, it turns out that minimizing the expected search cost does not correspond to a packing problem or, at least, if any correspondence exists, it is not direct.

4. A BINARY LINEAR PROGRAM FOR THE OPTIMIZATION PROBLEM

We model the optimization problem as a *binary linear program* (BLP), which allows us to find the optimal *iSet* using an integer linear programming optimization package (e.g., GLPK or CPLEX).

Let $\mathbb{C}$ be the set of combinations of t MDs. Let us define the set of all possible indices

$$(K,L) = \left\{ (k,\ell) : t \le k \le F, 1 \le \ell \le \min\left[\left\lfloor \frac{S}{Space(k)} \right\rfloor, \binom{F}{k} \right] \right\}$$

so (k,ℓ) is the ℓ^{th} index of size k. Figure 1 illustrates an example of a set (K,L) with $F = 6$, $t = 2$, $S = 9$, and $Space(k) = k$. Note that (K,L) enumerates all the possible indices of different sizes that fit in the available space. We still need to decide which MDs will be inserted on the actually constructed indices.

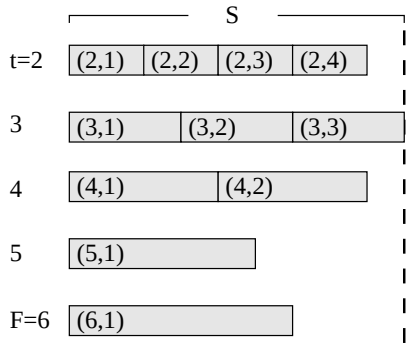

Figure 1: Example of set (K,L).

We extend (K,L) with a special index (∞,ℓ) that contains non-indexed combinations of MDs (i.e., if $k = \infty$ then $Search(\infty) = LS(t)$) and uses no space. We also introduce some binary variables (see Figure 2). Variable $x_{f,k,\ell}$ associates MDs and indices, i.e., this variable equals 1 if we insert MD f into index (k,ℓ). Variable $y_{c,k,\ell}$ indicates which of the constructed indices is the cheapest one to perform a similarity query for a given combination c. Variable $z_{k,l}$ indicates which indices from (K,L) are constructed.

$$x_{f,k,\ell} = \begin{cases} 1 & \text{if } f \text{ belongs to } (k,\ell), \\ 0 & \text{if not.} \end{cases}$$

$$y_{c,k,\ell} = \begin{cases} k & \text{if } (k,\ell) \text{ is the cheapest index s.t. } c \in (k,\ell) \\ 0 & \text{if not.} \end{cases}$$

$$z_{k,\ell} = \begin{cases} 1 & \text{if index } (k,\ell) \text{ exists,} \\ 0 & \text{if not.} \end{cases}$$

Figure 2: Binary variables for the model.

Given these variables and parameters, we describe the constraints of the problem. Firstly, the space constraint:

$$\sum_{k,l} Space(k) \cdot z_{k,l} \le S. \tag{1}$$

We ask that every combination is indexed (notice that, in the worst case, this happens in the special index (∞,ℓ)):

$$(\forall c) \sum_{k,\ell} y_{c,k,\ell} = 1. \tag{2}$$

Note that this only guarantees that exactly one index is considered in the search cost of the combination, but not that this index has the minimum search cost. We will show that the actual cost will be associated to the cheapest index.

Now, we set the relation between a combination and MDs. A combination belongs to an index if all the MDs in the combination are in the index:

$$(\forall c,k,\ell)\ y_{c,k,\ell} \le \frac{1}{t} \sum_{f \in c} x_{f,k,\ell}. \tag{3}$$

The right-hand size of the equation is strictly smaller than 1 (thus it forces $y_{c,k,\ell} = 0$) unless the t MDs in combination c are present in the index (k,ℓ).

We need to fix the capacity of an index in number of MDs. If a feasible solution contains an index of size k, then this index contains at most k MDs:

$$(\forall k,\ell) \sum_{f} x_{f,k,\ell} \le k \cdot z_{k,\ell}. \tag{4}$$

Finally, we write the target function (to be minimized), which is the expected search cost:

$$E = \sum_{c,k,\ell} p_c \cdot Search(k) \cdot y_{c,k,\ell}. \tag{5}$$

At this point we can show that, as long as the search cost increases with the size of the indices, the cost of a combination c is counted only in the cheapest index. Indeed, if the MDs of a combination c are present in two indices $(k,\ell),(k',\ell')$ such that $k < k'$ and $y_{c,k',\ell'} = 1$, then setting $y_{c,k,\ell} = 1, y_{c,k',\ell'} = 0$ is a feasible solution with a strictly smaller cost, i.e., in the optimum, for any fixed c, the variable $y_{c,k,\ell}$ equals 1 if and only if index (k,l) contains combination c and $Search(k)$ is minimum.

We observe that the size of the above described BLP is polynomial in the input of the problem. The number of variables is $|\mathbb{F}||(K,L)|+|\mathbb{C}||(K,L)|+|(K,L)|$ and the number of constraints is $1+|\mathbb{C}|+|\mathbb{C}||(K,L)|+|(K,L)|$. Our model shows that finding the optimum *iSet* is *NP*. However, BLPs are *NP-Hard* [8, 10] to be solved in the general case, so having this formulation does not provide an efficient way to find the optimum. Nevertheless, standard methods such as the *Branch and Bound* [9] can be used for small instances or to obtain an approximation of the optimal solution.

Also note that our model is general in the sense that it can be used with any index structure that supports multi-metric spaces. One only needs to define the *Search* and *Space* cost functions appropriately, depending on which index structure is used. It is even possible to use different index structures for different index sizes: The BLP ensures that the optimal solution will be found, given the set of parameters (cost functions, available space, and probabilities for the combinations). Thus, if more efficient index structures for multi-metric spaces are created, our proposed model can always take advantage of them, finding the optimal use of the available space for building indices.

5. BOUNDS FOR THE OPTIMAL SOLUTION

Now we analyze upper and lower bounds for the minimum expected search cost. Let $P(S) = \sum_{i=1}^{\lfloor S/Space(t)\rfloor} p_i$ and $M(t,S) = \min\{Search(t+1), LS(t)\}$.

THEOREM 1. *Assume that $Search(t) < LS(t)$, $Search(t+1) > Search(t)$. Let $OPT(S)$ be the minimum expected search cost when there is space S for indexing. Then, upper and lower bounds for the minimum expected search cost are given by*

$$OPT(S) \leq Search(t)\cdot P(S) + LS(t)\cdot(1-P(S)), \quad (6)$$

$$OPT(S) \geq Search(t)\cdot P(S) + M(t,S)\cdot(1-P(S)). \quad (7)$$

PROOF. To prove inequality (6), consider the *iSet* $\mathbb{I}$ consisting of $\lfloor S/Space(t)\rfloor$ indices of size t indexing the combinations with highest probabilities, and that does not index any other combination. Then $E(\mathbb{I}) = Search(t)\cdot P(S) + LS(t)\cdot(1-P(S))$, but $OPT(S) \leq E(\mathbb{I})$, so we are done.

For inequality (7), we distinguish two cases: (i) Either $Search(t+1) \geq LS(t)$, in which case no index of size $t+1$ is created, so we find ourselves in the case of Section 3.1 (one can build only indices of size t) and the result follows; or (ii) $Search(t+1) < LS(t)$, and therefore the optimum solution may use indices of different sizes.

Let $A^* = \{c_i : c_i \in idx,\ size(idx) = t$ in the optimal solution$\}$, so $|A^*| \leq \lfloor S/Space(t)\rfloor$, and let $B = \mathbb{C} - A^*$. Recall that $ms_{\mathbb{I}}(c_i)$ is the size of the smallest index that contains combination c_i. If $p(A^*) = \sum_{c_i\in A^*} p_{c_i}$, then

$$\begin{aligned} OPT(S) &= \sum_{c_i\in C} Search(ms_{\mathbb{I}}(c_i))\cdot p_i \\ &= \sum_{c_i\in A^*} Search(t)\cdot p_{c_i} + \sum_{c_i\in B} Search(ms_{\mathbb{I}}(c_i))\cdot p_{c_i} \\ &\geq \sum_{c_i\in A^*} Search(t)\cdot p_{c_i} + \sum_{c_i\in B} Search(t+1)\cdot p_{c_i} \\ &= Search(t)\cdot p(A^*) + Search(t+1)\cdot(1-p(A^*)) \end{aligned}$$

Because $Search(t+1) > Search(t)$, the right side is minimized when $p(A^*)$ is maximum. But $|A^*| \leq \lfloor S/Space(t)\rfloor$, therefore the maximum value of $p(A^*)$ is attained when $|A^*| = \lfloor S/Space(t)\rfloor$, i.e., $p(A^*) = P(S)$, which concludes the proof. □

6. ALGORITHMS FOR SOLVING THE OPTIMIZATION PROBLEM

We propose three algorithms to find good solutions for the optimization problem. In Section 7, we show that our algorithms find solutions close to the optimum.

Algorithm A.

It starts with $\mathbb{I} = \emptyset$. On each iteration, the algorithm decides the best to do between: (a) Adding a new index of size t (if there is enough available space), and (b) adding a MD to one of the indices already in $\mathbb{I}$. From both possible actions, the algorithm selects the one that minimizes the expected search cost. The algorithm iterates until there is no more available space or none of the actions improves the expected search cost.

Algorithm B.

It starts with an *iSet* that contains indices of size t for all combinations of t MDs. Then, the algorithm searches two indices to merge, such that the ratio of the increasement on the expected search cost and the amount of saved space is minimal. The merge operation frees some of the space used by the indices. The algorithm iterates until the used space for the *iSet* is equal or smaller than S. Note that this algorithm only works if $S \geq Space(F)$. When $S = Space(F)$, it simply returns one index that contains all MDs. Thus, it is not possible to save space by merging indices as soon as this solution is reached.

Algorithm C.

This is a slight modification of Alg. A. Instead of starting with an empty *iSet*, Alg. C starts with an *iSet* that contains $\lfloor S/Space(t)\rfloor$ indices of size t for the most frequently used combinations. Then, while there is available space, it tries to expand the indices if this further decreases the expected search cost. Recall that $P(S) = \sum_{i=1}^{\lfloor S/Space(t)\rfloor} p_i$ and $M(t,S) = \min\{Search(t+1), LS(t)\}$.

THEOREM 2. *Algorithm C finds an* iSet *whose expected search cost is at most*

$$\frac{Search(t)\cdot P(S) + LS(t)\cdot(1-P(S))}{Search(t)\cdot P(S) + M(t,S)\cdot(1-P(S))}$$

times the minimum expected search cost.

PROOF. It follows from Theorem 1 and the fact that if it is not convenient to expand any of the indices of size t, the algorithm will return as solution $\lfloor S/Space(t)\rfloor$ indices of size t for the most probable combinations. This solution has an expected search cost equal to the upper bound (eq. (6)) of the optimal solution. Thus, any solution returned by Alg. C which includes an index of size greater than t must have an expected search cost lower than the upper bound. □

Note that if $M(t,s) = LS(t)$ then Alg. C finds the optimal solution, and that Theorem 2 holds for an algorithm that

returns the *iSet* that has minimum expected search cost value between those returned by Alg. A, B, and C.

7. EXPERIMENTAL EVALUATION

We used a real dataset to compare the *iSets* obtained by the BLP and the proposed algorithms. The database used for our experiments consists of 1,838 3D objects that we collected from the Internet[1]. This set of 3D objects represents an heterogeneous mix of models, including several kinds of animals, planes, cars, plants, chairs, human bodies, and so on. From this set, 472 objects were manually classified by shape similarity into 55 different model classes. The rest of the objects were left as unclassified. Each classified object of each model class was used as a query object. The objects belonging to the same model class, excluding the query, were taken as the relevant objects.

We implemented 16 different types of descriptors to perform experiments [3], which includes: statistical descriptors (3D moments), geometry based descriptors (principal curvature, shape distribution, ray-based, ray-based with spherical harmonics, shading, complex valued shading, cords-based, segment volume occupation, voxel based, 3DDFT, rotation invariant spherical harmonics), image based descriptors (depth buffer, silhouette), and other approaches (rotation invariant point cloud descriptor).

We pre-processed the implemented descriptors before computing the combinations. First, we applied a PCA-based dimensionality reduction that all descriptors have the same dimensionality, keeping the 32 principal axes of each descriptor. Then, we normalized the coordinate values of all descriptors in the range $[0, 1]$. As MDs, we used the Manhattan distance on the spaces defined by each descriptor.

To compute the probabilities of using a given combination of MDs, we used the *entropy impurity method* [2]. This method uses a reference dataset, classified in object classes, to compute the entropy impurity of a MD given a query object. For each MD, a similarity query is performed on the reference dataset. Then, the entropy impurity is computed looking at the model classes of the retrieved objects: The entropy impurity is zero if all objects belong to the same model class, and it has a maximum value if each object belongs to a different model class. The t MDs with smallest entropy impurity values are selected for the combination. We run our set of queries and let the query processor select the best combination, storing which combination was selected for each query. We used the frequency of selection of each combination as its probability of being selected.

We used a VA-File [12] as index structure. Its associated space (in bits) and search cost are $Space(k) = knb$ and $Search(k) = O(knb)$, where b is the number of bits used for each dimension. For simplicity, we used $Search(F) = 1$ (which implies that $Search(k) = \frac{k}{F}$ if the same number of bits per dimension is used, as the cost of the VA-File is linearly dependent on the number of indexed descriptors), $Space(F) = F$ (which also implies that $Space(k) = k$), and $LS(t) = K \cdot Search(t)$, i.e., to search using a sequential scan is K times slower than using an index of size t for the selected combination of MDs. For our computations, we used $K = 10$ [12, 6, 11]. These selections were only done to facilitate the computation of the search cost, and other values can be used without affecting the behavior of the *iSet* (though, the optimal solution may be different).

[1]Konstanz 3D model search engine. `http://-merkur01.inf.uni-konstanz.de/CCCC/`

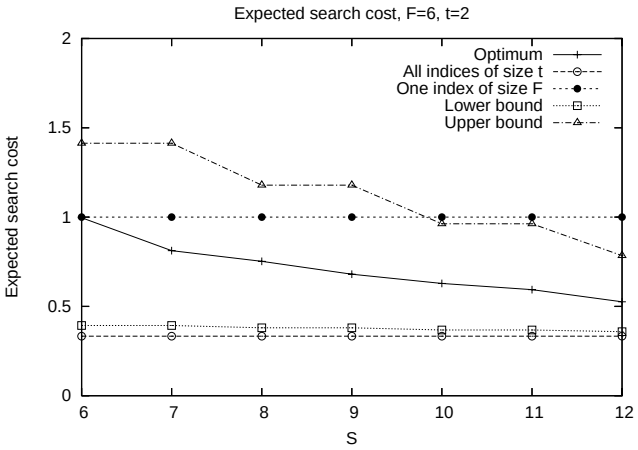

Figure 3: Expected search cost of the optimal *iSet*, $F = 6,\ t = 2$.

We considered two sets of MDs ($F = 6$ and $F = 16$) and two combination sizes ($t = 2$ and $t = 3$). For all the values of F and t, we run our algorithms, calculated the bounds of Theorem 1, and solved the BLP over different values of S. Finally, we normalized the results by the search cost of a single index of size F. To solve the BLP, we used CPLEX linear optimization solver v. 7.5. Our machine had two Pentium IV 3.7 Ghz processors running Fedora Linux 4, with 1Gb of RAM.

7.1 Experimental results

Figure 3 shows the results for the BLP in the case $F = 6, t = 2$ ($S = 6, \ldots, 12$), as well as the search cost of a single index containing all MDs, the general lower and upper bound, and the search cost when each combination is indexed in its own index of size t (feasible only if $S \geq \binom{6}{2} \cdot Space(t) = 30$). The reduction in the search cost was noticeable: two times by having twice the space of the naïve solution (one index of size 6).

Figures 4 and 5 show the solutions obtained by the algorithms for the case $F = 6$, $t = 2$ and $t = 3$, and $S = 6, \ldots, 12$, as well as the optimum value (from the BLP) and the bounds of Theorem 1. Even though Alg. C did not find good *iSets*, it paid off well to have indices of size larger than t. Indeed, this the reason of the drastic reductions in the search cost obtained by Alg. C when S is not divisible by t. The results of the Alg. A for the case $t = 3$ were optimal in the whole range of S values.

For the case $F = 16$, neither the BLP nor Alg. B (for the case $t = 3$) output a solution within reasonable time, so we present results only regarding Alg. A, B (for $t = 2$), and C, as well as the theoretical bounds. Figures 6 and 7 show the results for $t = 2$ and $t = 3$ when $S = 12, \ldots, 24$. Here it becomes clear that in some important cases the only practical way to get a good solution is by resorting to approximation algorithms.

Finally, Table 2 presents the run time of the algorithms and the BLP for $F = 6$ and $t = 2$. The first column shows the available space S. The second, third, and fourth columns show the run time for Alg. A, B, and C, respectively. The

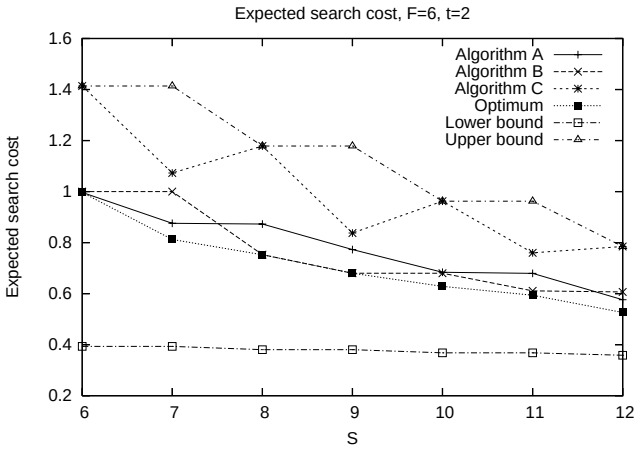

Figure 4: Expected search cost of the *iSets* returned by the algorithms, $F = 6,\ t = 2$.

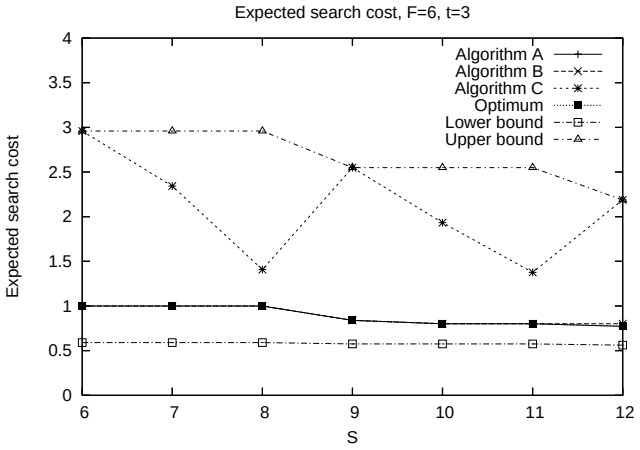

Figure 5: Expected search cost of the *iSets* returned by the algorithms, $F = 6,\ t = 3$.

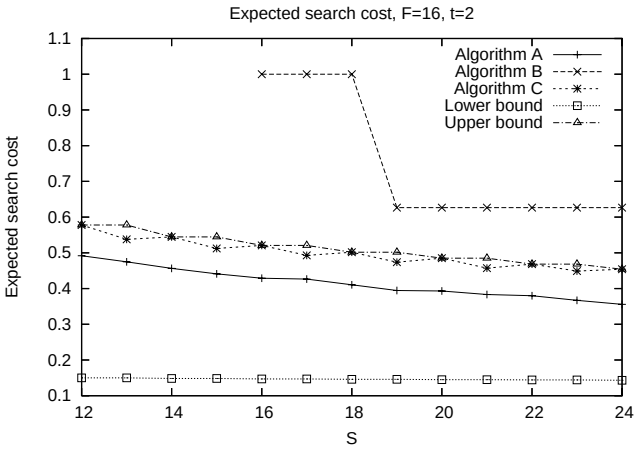

Figure 6: Expected search cost of the *iSets* returned by the algorithms, $F = 16,\ t = 2$.

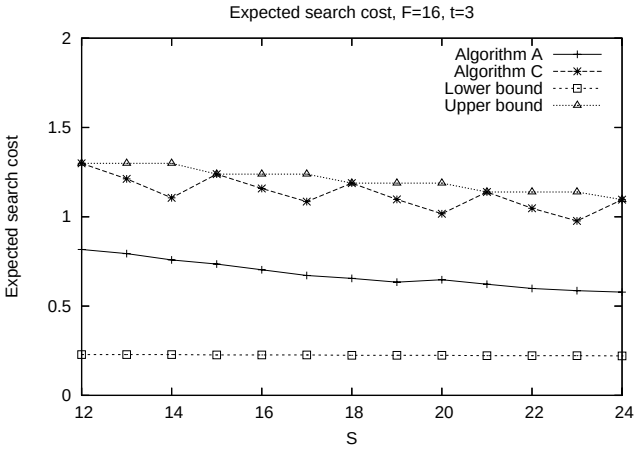

Figure 7: Expected search cost of the *iSets* returned by the algorithms, $F = 16,\ t = 3$.

fifth column shows the run time of the BLP. It follows that the timed needed by the BLP increased steeply with S, and that the algorithms are many orders of magnitude faster than the BLP.

7.2 Analysis of the results

Among Alg. A, B, and C, the best overall was Alg. A. It was fast to compute and returned nearly optimal solutions: About 4% in average from the optimum in the cases where we could calculate the optimum using the BLP. Alg. B also returned good *iSets* (often better than A), but it became too slow when the total number of combinations T became *large* (when $F = 16, t = 3$, then $T = \binom{16}{3} = 560$). This is because the initial solution for Alg. B contains exactly one index per combination and it is $O(T^2)$ on each iteration. Also, Alg. B does not work when $S < Space(F)$. Alg. C was the fastest, but it produced the worst results compared to the other algorithms. Its only advantage is that we can prove a guarantee (Theorem 2) on the relative error of its output. A good compromise could be to use the *iSet* with minimum expected search cost between the outputs of Alg. A and C.

Finally, notice that in Figure 7 Alg. A produced a better solution for $S = 19$ than for $S = 20$. This may happen when the algorithm does not have more space for building a new index of size t and therefore it expands an existing index ($S = 19$). When more available space is allowed ($S = 20$), the greedy algorithm may decide to create a new index, missing a chance to expand afterwards.

8. CONCLUSIONS

We presented methods for finding the set of indices (*iSet*) that minimizes the expected search cost of similarity queries that use dynamic combinations of metric distances. We modeled the problem as a binary linear program, which provides us with a tool to find the optimal *iSet* for small instances of the problem. We also proposed fast algorithms that are able to find good sets of indices.

The applicability of the proposed model is not restricted to the particular cases that we presented in this paper. Our approach is very flexible in the sense that is not restricted to a particular index structure or to specific cost functions. The

Table 2: Time (in seconds) needed for the algorithms and the BLP to find the solution ($F = 6$).

S	A	B	C	BLP
6	0.037	0.379	0.052	0.13
7	0.038	0.344	0.028	1.41
8	0.042	0.346	0.025	3.15
9	0.046	0.349	0.030	16.51
10	0.054	0.350	0.025	76.83
11	0.060	0.344	0.032	451.70
12	0.066	0.347	0.025	765.95

model can be used to evaluate different indexing schemas: It suffices to define the cost functions, available space, and probabilities of using the combinations, and the model will return the optimal solution for that setup. Should *LS* be replaced by more efficient techniques, our model can still be applied to find the best indexing. It suffices to change the corresponding parameter in the model. The same applies if more efficient index structures become available.

In our future work, we plan to study the complexity of finding the optimal *iSet*. We have shown that the problem is NPO, but its complexity is still unknown. We presume that finding the optimal *iSet* is NPO-hard, but we still need to prove it formally. We also want to improve the proposed algorithms, by analyzing their weaknesses, and to improve the presented lower bound for the optimal solution.

9. REFERENCES

[1] C. Böhm, S. Berchtold, and D. Keim. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, 2001.

[2] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. Using entropy impurity for improved 3D object similarity search. In *Proc. IEEE Intl. Conf. on Multimedia and Expo (ICME'04)*, pages 1303–1306. IEEE, 2004.

[3] B. Bustos, D. Keim, D. Saupe, T. Schreck, and D. Vranić. Feature-based similarity search on 3D object databases. *ACM Computing Surveys*, 37(4):345–387, 2005.

[4] B. Bustos, D. Keim, and T. Schreck. A pivot-based index structure for combination of feature vectors. In *Proc. 20th Annual ACM Symp. on Applied Computing, Multimedia and Visualization Track (SAC-MV'05)*, pages 1180–1184. ACM Press, 2005.

[5] B. Bustos and T. Skopal. Dynamic similarity search in multi-metric spaces. In *Proc. 8th ACM Workshop on Multimedia Information Retrieval (MIR'06)*, pages 137–146. ACM Press, 2006.

[6] K. Chakrabarti and S. Mehrotra. The hybrid tree: An index structure for high dimensional feature spaces. In *Proc. 15th Intl. Conf. on Data Engineering (ICDE'99)*, pages 440–447. IEEE Computer Society, 1999.

[7] E. Chávez, G. Navarro, R. Baeza-Yates, and J. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

[8] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[9] R. Garfinkel and G. Nemhauser. *Integer Programming*. John Wiley & Sons, 1972.

[10] J. Hromkovic. *Algorithms for Hard Problems*. Springer, 2001.

[11] G. Qian, Q. Zhu, Q. Xue, and S. Pramanik. The ND-tree: A dynamic indexing technique for multidimensional non-ordered discrete data spaces. In *Proc. 29th Intl. Conf. on Very Large Databases (VLDB'03)*, pages 620–631. Morgan Kaufmann, 2003.

[12] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Intl. Conf. on Very Large Databases (VLDB'98)*, pages 194–205. Morgan Kaufmann, 1998.