Efficiently Managing Multimedia Hierarchical Data with the WINDSURF Library*

Ilaria Bartolini, Marco Patella, and Guido Stromei

DEIS, Università di Bologna, Italy
{i.bartolini,marco.patella,guido.stromei}@unibo.it

Abstract. Complex multimedia data are at the heart of several modern applications, such as image/video retrieval and the comparison of collection of documents. Frequently, such complex data are modeled as hierarchical objects that consist of different components, like videos including shots, images including visually coherent regions, and so on. When such complex objects are to be compared, for example, for assessing their mutual similarity, this is usually done by recursively comparing component elements. However, due to such complexity, it is often hard to efficiently perform a number of tasks, like processing of queries or understanding the impact of different alternatives available for the definition of similarity between objects. In this article, we propose a unified model for the representation of complex multimedia data, introducing the WINDSURF software library, with the goal of allowing a seamless management of such data. The library provides a framework for evaluating the performance of alternative query processing algorithms for efficient retrieval of multimedia data. Important features of the WINDSURF library are its generality, flexibility, and extensibility. These are guaranteed by the appropriate instantiation of the different templates included in the library: in this way, each user can realize her particular retrieval model of need.

1 Introduction

Despite their ubiquitous and prominent role in nowadays life, Multimedia (MM) information still present a variety of challenges for their effective and efficient retrieval. Among these, the extraction of content and its subsequent indexing represent two of the most analyzed areas of research. However, the inherently complex nature of some multimedia data (like videos, images, web pages, and so on) makes it hard to exploit out-of-the-box solutions that were devised for simpler scenarios (e.g., textual documents). Indeed, in many MM cases the classical information retrieval (IR) models cannot be applied without either oversimplifying the type of queries that can be issued by an user or completely giving up efficiency or effectiveness. An example, that arises in several MM scenarios, is that of MM documents that are composed of several component *elements*. Requesting documents that are relevant to a given query document Q entails retrieving elements that are relevant to Q elements, and then somewhat combining the results at the document level. This *hierarchical* structure of documents is general enough to be able to model

* This work was partially supported by the CoOPERARE MIUR Project.

M.S. Obaidat, J.L. Sevillano, and J. Filipe (Eds.): ICETE 2011, CCIS 314, pp. 347–361, 2012.

different MM IR applications, but poses some peculiar challenges due to its very nature: for example, how are document elements compared to query elements? how the relevance of elements is aggregated in order to assess the relevance of whole documents? is indexing of whole documents a possible choice? in case, is it a better choice than indexing elements? Above questions recur whenever the hierarchical model is applied for the retrieval of MM documents; however, answers cannot be given independently from the application at hand, since each particular scenario presents its peculiarities. When enhancing differences among applications, we should however note that several affinities are still present and that solutions proposed for a particular scenario could be applied to other similar scenarios as well, provided that the underlying model is the same.

In this paper, we present the WINDSURF library for management of MM hierarchical data, with the goal of providing a general, flexible, and extensible software framework for analyzing the impact on performance of the different aspects included in its retrieval model. In particular, the library presents an emphasis on query processing techniques, offering different index-based algorithms for the efficient resolution of similarity retrieval queries, where documents are requested whose content is (in some sense) *similar* to that of the query. Indeed, it turns out that algorithms included in the WINDSURF library have a wide range of applicability and can therefore be helpful for a variety of scenarios. We expect the library to be particularly useful to those researchers that have to analyze how different alternatives in the representation/comparison of elements/documents interact in providing different effectiveness/efficiency performances, without the burden of defining ex-novo algorithms for retrieving query results. We also note that processing of similarity queries may not be the main goal of the application at hand, rather it could be just a component of a more complex system: as an example, TRECVID 2011 (`http://trecvid.nist.gov/`) includes several tasks calling for efficient retrieval of similar video shots. For instance, the semantic indexing (SIN) task involves the automatic tagging of video segments in order to perform filtering, categorization, browsing, and search (this is commonly performed by associating the same tags to shots sharing similar visual/audio content [4]); the content-based copy detection (CCD) task, on the other hand, aims to automatically detect copies of video segments, which clearly can be based on the retrieval of similar video content.

We first precisely define the hierarchical retrieval model of WINDSURF (Section 2), by also presenting real-world examples of its use, and provide a general view of the library (Section 3), including its query processing algorithms (Section 4). Then (Section 5), we show how the library can be customized so as to behave according to the requirements of the particular application at hand and we provide examples of use of the library in the Region-Based Image Retrieval (RBIR) scenario (Section 6): this was the original application scenario of the library and also justifies its name (WINDSURF standing for Wavelet-based INDexing of imageS Using Region Fragmentation [1]). Finally, we draw our conclusions, by also highlighting future directions of research (Section 7).

2 The WINDSURF Retrieval Model

The retrieval model of WINDSURF is as follows: we have a database $\mathcal{D}$ of N documents, $\mathcal{D} = \{D^1, \ldots, D^N\}$, where each document D is composed of n_D *elements*,

$D = \{R_1, \ldots, R_{n_D}\}$. Each element R is described by way of *features* that represent, in an appropriate way, the content of R. Given a query document $Q = \{Q_1, \ldots, Q_n\}$ composed of n elements, and an element distance function δ, that measures the dissimilarity of a given pair of elements (using their features), we want to determine the set of *best* documents in $\mathcal{D}$ with respect to Q.

The above formulation of the problem is sufficiently general to encompass different retrieval paradigms, each having a different way of specifying which documents are to be considered "best" for the query at hand: this can be demonstrated by applying the WINDSURF retrieval model to some real world examples.

Example 1. Our first example deals with the comparisons of web sites. In this case, each element R is a web page contained in a web site D and we want to discover whether a new web site Q is similar to some existing web sites in our database $\mathcal{D}$. Comparison between web pages is performed by taking into account contained keywords, e.g., by using the vector space model [18], so that features extracted from each page include keywords using $tf \times idf$ values after stopping & stemming (see Figure 1).

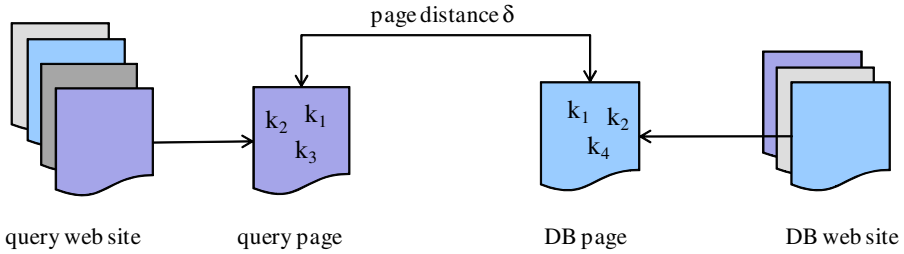


Fig. 1. Comparing web sites

Example 2. In RBIR, the $\mathcal{D}$ database consists in still images that are segmented into regions, where pixels included in a single region R share the same visual content (e.g., color & texture). Image regions are compared according to their visual features and we want to retrieve images that are similar in content to a user-specified query image Q (see Figure 2).

Example 3. As a third example, we consider the comparison of videos based on similarity, where each video D is first segmented into shots, i.e., sequences of video frames that are coherent in their visual content. Then, each shot R is represented by a single key frame (this can be either the first frame of the shot, or the middle one, or the medoid of shot frames), so that shots can be compared by means of a simple image similarity function. Finally, we can compare whole videos by aggregating the similarities between shots (see Figure 3). Note that different applications (like duplicate video detection) might impose different constraints on the "matching" of video shots, e.g., requesting that only shots of similar length can be coupled or that shots that are shown in very different moments cannot be matched; clearly, this has an impact on the computation of similarity between videos, thus a researcher might be interested in investigating the effect of such constraints on the result of a query requesting for, say, the 5 videos most similar to a given query video Q.

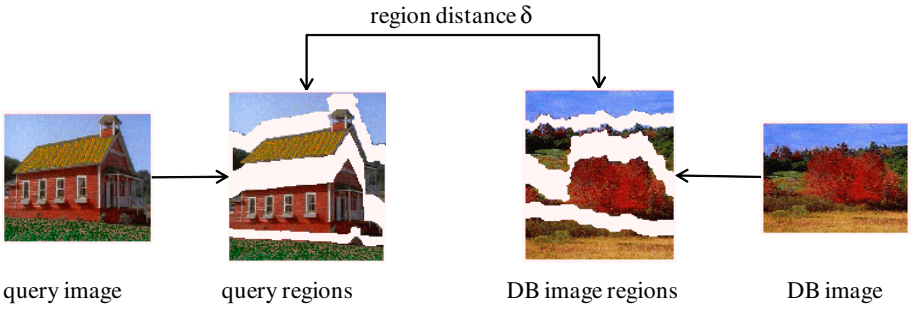


Fig. 2. Comparing segmented images in Region-based Image Retrieval

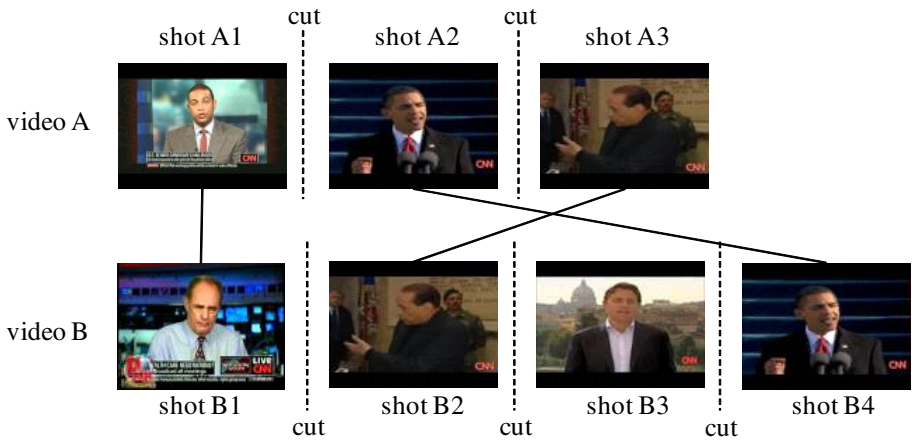


Fig. 3. Comparison of videos based on video shots

For the rest of the paper, we will assume as given the way documents are divided into elements (e.g., the image segmentation algorithm in Example 2, or the shot segmentation of videos in Example 3), the features used to represent such elements, and the (element) distance function δ, being understood that similar elements will have a low δ value: our focus here is to demonstrate how different retrieval models can be enclosed by the WINDSURF model, thus proving its generality.

Another important factor to be considered is the definition of the query result, i.e., how the best documents wrt Q are specified. Indeed, different applications typically have different ways of assessing the similarity between documents, given the similarities between component elements. In WINDSURF, two different retrieval modalities are supported: quantitative (k-NN) and qualitative (Skyline).

– In the k Nearest Neighbor (k-NN) quantitative model [14], similarity between documents is numerically assessed by way of a document distance function d that combines together the single element distances into an overall value. Consequently, document D^a is considered better than D^b for the query Q iff $d\,(Q, D^a) < d\,(Q, D^b)$ holds and the query result consists of the k DB documents closest to the query.

– As an alternative to the quantitative model, the qualitative (Skyline) model does not rely on the specification of a numerical value, according to which DB documents can be sorted for decreasing values of similarity wrt to the query, rather document D^a is considered better than D^b for the query Q iff D^a does no worse than D^b on all query elements and there exists at least one query element on which D^a is strictly better than D^b. This necessarily includes those documents that would be the best alternative according to some specific document distance function [8].

Regarding k-NN queries, it has to be noted that, usually, the computation of the document distance d is obtained by combining three basic ingredients:

1. the element distance δ,
2. the set of constraints that specify how the component elements of the query Q have to be matched to the component elements of another (database) document D, and
3. the aggregation function that combines distance values between matched elements into an overall document distance value (e.g., a simple average of distance values between matched elements).

Often, the overall document distance is computed by aggregating scores of the best possible matching, i.e., the one that minimizes the overall document distance; in this case, the computation of d also includes the resolution of an optimization problem in the space of possible matchings between elements of Q and elements of D. We finally note that the result of any query depends on the combination of all three ingredients, so that changing one of them might lead to completely different results. As we will show later, the characteristics of the overall document distance also determine which algorithms can be used to efficiently solve the k-NN query.

As to the Skyline retrieval model, our definition of domination among documents follows the one described in [3] for the case of segmented images. Intuitively, the concept of domination is defined for tuples, while here we are considering sets of elements; thus, the dominance criterion needs to be properly extended to deal with this additional complexity in the structure of objects to be compared. For this purpose, each document can be defined as the set of possible matchings of its elements with query elements, each matching being a tuple of distance values between a query element Q_i and its matched element of D, R_j. The domination between matchings can be then straightforwardly defined. Finally, domination between documents is built on top of the concept of domination between matchings, stating that a document D^a dominates another document D^b wrt the query Q iff for each matching of D^b there exists a matching of D^a that dominates it.

2.1 Alternative Retrieval Models

Albeit the WINDSURF retrieval model is sufficiently general to encompass the characteristics of several multimedia scenarios, see [10] for a recent example, it is interesting to note its analogies with other different models. For example, the *Bag of Words* (BoW) model for computer vision [7] represents images as sets of *patches* (these are similar to elements in WINDSURF). Then, all patches included in any DB image are converted into *codewords*, where each codeword is representative of several patches. This produces a

codebook and each image can be described as the set of codewords representing its patches. In this way, the retrieval models used for textual documents [18] can be directly applied for images, since the codebook is equivalent to a dictionary. The difficult part here is the generation of the codebook (how many codewords? how to compare patches?).

We also note that our k-NN retrieval model also includes those cases where the image distance d also considers *global* characteristics; for example, this is the case when the particular d to be used for a given query is *learned* by exploiting side information [19,10].

3 Overview of the WINDSURF Library

The WINDSURF library is written in Java and is released under the "QPL" license, being freely available at URI `http://www-db.deis.unibo.it/Windsurf/` for education and research purposes only. It consists of five main packages, each focusing on a section of the main architecture.

Document: the `Document` package includes the definition of classes modelling documents, elements, and features. It also contains the specification of the element distance δ and (possibly) of the document distance d.

FeatureExtractor: the `FeatureExtractor` is the component in charge of extracting the features from a given document. This is performed in two steps: first the document is decomposed into elements (segmentation), then features are computed for each element (extraction).

QueryProcessor: the `QueryProcessor` (QP) is the component that solves queries over document features. It contains algorithms for the efficient resolution of both k-NN and Skyline queries, by exploiting the presence of indices built on document features. In case indices are not available, the package also incorporates sequential algorithms for solving queries.

FeatureManager: the `FeatureManager` (FM) is the component in charge of storing/retrieving the document features from the DB, providing an abstraction from the underlying used DBMS. In order to achieve an efficient management of features, these can be saved into a relational DBMS (in particular, the WINDSURF library includes code for using the MySQL[1] RDBMS).

IndexManager: the `IndexManager` (IM) package contains classes managing the feature indices. These can be exploited by the QP for the efficient resolution of queries over the features (see Section 4). WINDSURF supports indices built on top of both elements and documents: as we will see in the following, this allows the definition of alternative query processing algorithms. In particular, an implementation of the M-tree index [6] is included.[2]

Figure 4 provides an abstract view of how packages of the library cooperate during the insertion and the retrieval phase. When a new document is to be added to the document

[1] http://www.mysql.com/

[2] For efficiency reasons, the implementation of M-tree is written in C++.

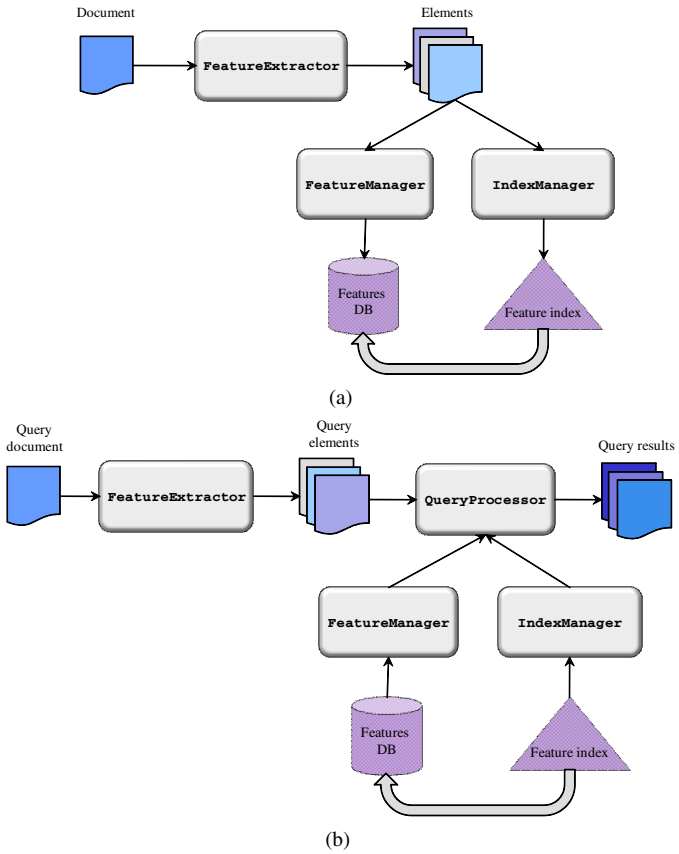


Fig. 4. Data flow in the WINDSURF library: (a) insertion phase, (b) retrieval phase

database (Figure 4 (a)), it is first processed by the `FeatureExtractor` package which breaks it into component elements and extracts elements' features. These are then forwarded to the FM and IM components that store the features in the features DB and the features index, respectively. On the other hand, at query time (Figure 4 (b)) features extracted by the `FeatureExtractor` are fed into the QP component, whose algorithms exploit the Feature and Index managers in order to pick query results out.

4 Query Processing Algorithms

Our main goal in designing the WINDSURF library was the performance comparison of different algorithms for the retrieval of complex documents, in terms of both efficiency and effectiveness. In this view, the core of the library consists of the QP component, that presents alternative algorithms for the resolution of queries. Regarding efficiency, QP algorithms might exploit indices built on features in order to avoid a full sequential evaluation, a non viable solution for large document DBs. Our arguments will be developed independently of the specific index; rather, we will refer to a generic *distance-based*

index, i.e., any index that relies on the computation of distances to return back objects. Distance-based indices include both multi-dimensional [9] and metric [5] indices, relevant examples of which are the R-tree [11] and the M-tree [6], respectively. To be useful for our purposes, distance-based indices should also provide a sorted access interface, i.e., to output data in increasing order of distance with respect to the object with which the index is queried: this is quite common, thanks also to the existence of algorithms of general applicability [12,13]. Depending on the used algorithm, indices in the WINDSURF library might be built on either elements (for which the element distance δ is used for indexing purposes) or whole documents (where indexing is based on the document distance d).

In order to evaluate the efficiency of each query processing algorithm, all classes provide statistics about relevant operations, including:

Document distances: the number of distance evaluations among documents (only relevant for k-NN queries); this is considered a costly operation, since it typically involves comparing several component elements and combining them in order to produce the overall score (as said, the latter might also require solving an optimization problem).

Element distances: the number of distance evaluations among elements; depending on the number of features and on the element distance function δ, this too might be a costly operation.

Sorted accesses: the number of accesses to the underlying element index; as we will show, some algorithms exploit an index built on document elements, that is used to sort DB elements in order of increasing distance values with respect to query elements. A sorted access returns a single DB element and requires the index to perform some computations.

Document dominations: the number of comparisons among documents in order to see whether a document dominates another one (Skyline queries only); again, this is a costly operation since it might require comparing several matchings.

Time: the overall time needed to solve a single query; this can be also detailed by considering the time needed for retrieving features from the DB, accessing the underlying indices, computing document distances, or comparing documents for domination.

The QP includes efficient algorithms for the efficient resolution of both k-NN and Skyline queries [3]. Each algorithm will be described here in general terms, by specifying under which hypotheses it is able to correctly solve a query.

SEQ. This sequential k-NN algorithm (`QueryProcessor.SF.QuerySFSequential` class) retrieves all documents in $\mathcal{D}$ and compares them with Q, by using the document distance d. Only the k best documents, i.e., the ones having the lowest d values, are kept and returned as the query result. No specific requirement on d or δ is needed, since the algorithm simply follows the definition of k-NN query.

***k*-NN-set.** This index-based k-NN algorithm (`QueryProcessor.SF.kNNset.kNNset` class) exploits an element index $\mathcal{T}_R$ to reduce the number of document and element

distances to be computed [3]. The k-NN-set algorithm iteratively alternates sorted accesses to the index $\mathcal{T}_R$ to retrieve DB elements with random accesses that compute a document distance $d(Q, D)$ between the query and the document whose element has been retrieved by the last sorted access. In this case, document distances are computed only during the random access phase, while element distances can be computed within the index and during each random access (since distances between all elements of both Q and of D might be required to compute $d(Q, D)$).

The algorithm applies to any document distance function d that can be bounded from below, i.e., for those d such that if, for document $D = \{R_1, \ldots, R_{n_D}\}$ and query $Q = \{Q_1, \ldots, Q_n\}$, it is $\delta(Qi, Rj) \geq \theta_i, \forall i, j$, then a function T exists such that $d(Q, D) \geq T(\theta_i)$. This is required to guarantee correctness of the provided result: it means that, for a document D^a whose all elements are "closer" to query elements than all those of another document D^b, it is also $d(Q, D^a) \leq d\left(Q, D^b\right)$. Indeed, since the underlying index $\mathcal{T}_R$ provides DB elements in order of increasing distance to query elements (sorted access), the algorithm cannot terminate until it is guaranteed that no document yet to be seen in a sorted access is closer to Q than the best k documents seen so far.

***k*-NN-imgIdx.** This k-NN algorithm (`QueryProcessor.SF.ImgIdx.QuerySFIndex` class) exploits a document index $\mathcal{T}_D$. Since, for hypothesis, $\mathcal{T}_D$ supports sorted accesses, the k-NN-imgIdx algorithm simply performs k of such accesses to return the query result. We note here that multi-dimensional access methods cannot be used to index whole documents, because a document is a set (and not a vector) of elements, thus metric indices are needed for this purpose. It then follows that the distance d used to compare documents should be a metric.

Sky-set. This is the only index-based Skyline algorithm included in the WINDSURF library (`QueryProcessor.Skyline.Skyset.Skyset` class) and uses an element index $\mathcal{T}_R$ [3] (the Skyline retrieval model cannot be supported by document indices, because a document distance function is not defined in this case). Similar to the k-NN-set algorithm, Sky-set resorts to sorted and random accesses; the main difference with k-NN-set is that, after each sorted access, no document distance is computed, rather the newly accessed document D is compared for domination with documents in the current solution, possibly leading to drop some current results or D itself. The correctness of Sky-set follows from the very definition of domination among documents and the use of a threshold tuple $\underline{\theta}$. In fact, unseen documents will only contain elements whose distance values are higher than those included in $\underline{\theta}$: it follows that any document D which is not dominated by $\underline{\theta}$ cannot be dominated by any unseen document, thus it can be output as a Skyline result. We finally note that, although our definition of the result of a Skyline query only include undominated documents, Sky-set is able to iteratively return results in layers [2]: according to this definition, documents in a layer are not dominated by any document, except by documents in previous layers (for each document D in layer i and for all $j < i$, it exists at least a document D' in layer j that dominates D).

5 Customizing the Library

The WINDSURF library includes abstract and general classes able to represent any application following the retrieval model described in Section 2. As stated in the introduction, one of the basic features of the library is its generality and ability of being customized to cover a broad range of application scenarios. In this section we first detail how a user of the WINDSURF library can instantiate classes so as to implement her specific needs, then describe some possible customizations.

In order to correctly exploit the library, a user has to follow five basic steps:

1. Extending the `Document` and `Element` classes within the `Document` package. For this, the user has to specify the format of features that represents documents and document elements. In particular, the element distance δ is modelled by the `distance` method in the `Element` class, while the document distance d is (possibly) implemented by the `distance` method in the `Document` class.
2. Implementing classes in the `FeatureExtractor` package for analyzing documents, in order to break them into their component elements and extract their features.
3. Writing classes in the `FeatureManager` and `IndexManager` packages for storing/retrieving document/element features to/from the underlying DBMS and indices.
4. Building the DB and the indices containing documents and elements. This is performed by way of the `insert` method within the `FeatureManager` and `IndexManager` classes, that save features of a single `Document` within the DB/index, according to the insertion logic depicted in Figure 4 (a).
5. Querying the DB (possibly exploiting indices) by creating an instance of the `Query` class within the `QueryProcessor` package. Such object (which is built using a single `Document`) could be used in conjunction with any of the algorithms listed in Section 4, see Figure 4 (b).

Although the previously listed steps are the only ones required for the basic use of the library, advanced users may require additional, more sophisticated, customizations. Most commonly, these will affect classes in the following packages.

`FeatureManager` and `IndexManager` packages: The library already includes generic code for using the MySQL DBMS and the M-tree [6] index (a template-based C++ library itself), but other implementations of the generic abstract classes for features management are possible. It is worth noting that, as stated in Section 4, separate index structures should be provided for the management of documents *and* elements, and that such indices should support the sorted access interface: this is required by the k-NN-set and the Sky-set algorithms, but also allows the retrieval of documents/elements using k-NN or range queries [21].

`QueryProcessor` package: This package contains the implementations of algorithms described in Section 4, but also allows the specification of other aspects of document retrieval using either the k-NN or the Skyline model. Particularly important is the `QueryProcessor.SF` sub-package, containing the implementation of several

alternatives for the computation of the document distance d via the use of *scoring functions*. The library already implements four of such functions, that will be detailed in the following.

Earth's Mover Distance (EMD): using the EMD scoring function [17], elements of the documents to be compared are matched in a many-to-many modality. The "amount" of matching of any element is limited to the "size" of such element (for example, in the case of image regions, this equals the fraction of image pixels included in the region at hand); the average of best-matched elements is used as the aggregation function, thus defining an optimization problem that corresponds to the well-known transportation problem, which can be solved in $O(n^3 \log n)$ time. It is easily proved that a document distance d defined in this way is a metric and can be bounded from below, thus it could be exploited by algorithms described in Section 4.

IRM: the IRM scoring function used by the SIMPLIcity RBIR system [20] is based on a greedy algorithm (with complexity $O(n^2 \log n)$) that obeys the same constraints and uses the same aggregation function (i.e., the average) as EMD. Consequently, the document distance computed by IRM is never lower than the one of EMD: this also implies that IRM can be also bounded from below (although with a looser bound wrt the one for EMD) but it does not satisfy the metric postulates.

$1-1$ **Assignment:** in this case, which is the one originally exploited by the WINDSURF RBIR system [1], each element of a document can be only matched to at most one element of the other document, and vice versa. Then a "biased" average is used to aggregate distance values of matched elements, so as to appropriately penalize documents that do not match all the query elements. This defines an assignment problem, which can be solved using the Hungarian Algorithm in $O(n^3)$ time [16]. Again, it is easy to see that this document distance can be bounded from below but is not a metric.

Greedy $1-1$**:** this last scoring function is computed by way of a greedy algorithm (whose complexity is $O(n^2)$) for the assignment problem. The corresponding document distance is thus never lower than the one computed using the previous function, is also bounded from below, but is not a metric.

In case the number of document elements, n, is high, above algorithms would be limited by their super-linear complexity. In such cases, it is likely that the user would specify alternative (approximate) algorithms, e.g., the pyramid match algorithm detailed in [10].

6 Use Cases

In this section, we demonstrate how the use of the WINDSURF library classes can be helpful in performing complex tasks over documents that comply with the WINDSURF model. The case study we consider here is that of a researcher investigating the impact of the different alternatives offered by the WINDSURF RBIR system (see Example 2). In particular, she is interested in the efficiency and the effectiveness of the query models available in the library as applied to the WINDSURF image features, which are detailed

in [1]. Following the five steps enumerated in Section 5, the user has to first implement classes in the following packages (note that the library already includes such code):

`Document` package: features for each image region (element) include color/texture characteristics that are represented by way of a 36-dimensional vector; the region distance δ implements the Bhattacharyya metric distance [15], while the image distance d implements all the alternatives included in Section 5, see [3].

`FeatureExtractor` package: a Haar-Wavelet filter is applied to each image (document) and pixels of the filtered image are then clustered together using a K-means algorithm; so-obtained clusters correspond to image region, whose features are extracted from visual characteristics of included pixels.

`FeatureManager` and `IndexManager` packages: classes are included for storing/retrieving image/region features to/from the MySQL DBMS and the M-tree index.

We include here the results of some experiments performed on a real image dataset consisting of about 15,000 color images (corresponding to about 63,000 regions) extracted from the IMSI collection (`http://www.imsisoft.com`).

As a first demonstration of use of the library, we compare the effectiveness of the Bhattacharyya region distance with respect to a simpler Euclidean (L_2) distance for establishing the similarity between region features: this is easily done by simply redefining the δ distance within the `Document` package. Figure 5 shows that the use of the Bhattacharyya distance is justified by its far superior accuracy with respect to the Euclidean distance, in spite of its higher cost (almost doubling the time needed to compute the L_2 metric). Although we only present here results for k-NN queries, experiments for Skyline queries (not included here for the sake of brevity) confirm the trend exhibited by Figure 5. Again, we note that this result can be obtained by simply redefining the `distance` method of the `Element` class within the `Document` package.

As another proof of usability of the library, we compared the effectiveness of the document distances described in Section 5. To this end, the k-NN-set algorithm was repeatedly executed with the different d distances. We obtained the results shown in Figure 6. It can be seen that all image distances behave almost the same, with the remarkable exception of the Greedy $1-1$ alternative, whose accuracy is very low for the first retrieved results. This result, which has been obtained with no cost, since all alternatives are already available within the library, may suggest that a choice between the first three alternatives should be based on efficiency considerations only.

Finally, we show a result of the performance comparison for the three index-based algorithms described in Section 4: Figure 7 compares the efficiency of k-NN-set (using both the EMD and the $1-1$ document distance), k-NN-imgIdx (using EMD), and Sky-set according to 4 different performance metrics, as described in Section 4. It is worth noting that k-NN-imgIdx performs the worst among considered algorithms: this might sound strange at first, since only k sorted accesses to the document index are needed and no computation is done outside of the index itself, but this is not enough to compensate

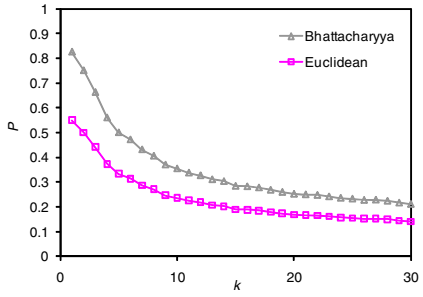
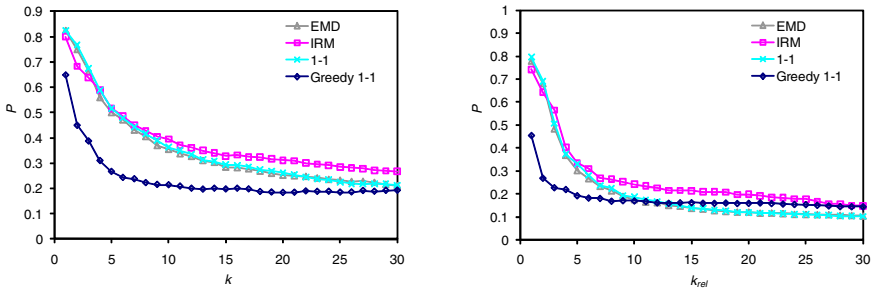


Fig. 5. Effectiveness of different element distance functions for the RBIR case: Precision (P) as a function of the number of retrieved documents (k)



(a) (b)

Fig. 6. Effectiveness of different document distance functions for the RBIR case: Precision (P) as a function of the number of (a) retrieved documents (k) and (b) relevant retrieved documents (k_{rel})

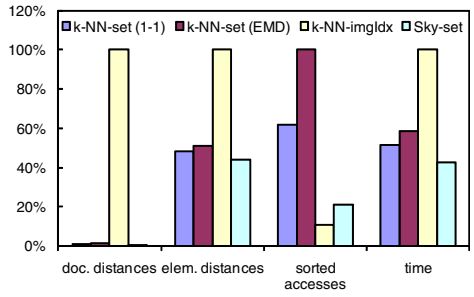


Fig. 7. Efficiency of the query processing index-based algorithms: k-NN-set using the EMD and the $1-1$ document distances, k-NN-imgIdx using EMD and Sky-set (graphs are normalized to the maximum values so as to emphasize relative performance)

for the very high number of document distances that are computed within the index.[3] Again, the library classes already contain the code for obtaining this important result, demonstrating that, when dealing with complex documents, a simplistic approach is not always the best one, and several alternatives should be taken into account to find out the best combination of efficiency and effectiveness.

7 Conclusions

We have presented the WINDSURF library for the management of complex (hierarchical) multimedia data, with the goal of providing tools for their efficient retrieval. The library was designed with the aim of generality and extensibility, so as to be applicable to a wide range of multimedia scenarios that fit its similarity-based retrieval model. Due to the inherent complexity of multimedia data, we designed the WINDSURF retrieval model to include all the different facets introduced by the hierarchical nature of the data (for example, how documents are characterized, how they are split into component elements, how elements are to be compared, how similarities at the element level are to be aggregated, and so on). Such facets can be instantiated in several alternative ways (each choice possibly giving different results) and an user may want to compare the performance of such alternatives in the scenario at her hand: we believe that the use of the WINDSURF library could help in abstracting away the details of generic query processing algorithms, since the above-mentioned facets can be realized by simply implementing abstract classes of the library. We are currently working in extending the library with new query processing algorithms and to incorporate other scenarios (e.g., videos [4]) as instances of the library available for downloading. Moreover, a current limitation of the WINDSURF retrieval model is that elements of a document are all of a same type: we plan to extend the model to consider elements of different types, so that only elements of the same type can be compared. For example, if we consider a multimedia document composed of textual sections and images, it makes sense to only compare text with text and images with images. Another important application of this concept is the use of cross-domain information to improve the retrieval of a given type of content, for example, exploiting surrounding text and/or links existing to other documents (à la PageRank) to boost image/video retrieval.

References

1. Ardizzoni, S., Bartolini, I., Patella, M.: Windsurf: Region-based image retrieval using wavelets. In: IWOSS 1999, Florence, Italy, pp. 167–173 (September 1999)
2. Bartolini, I., Ciaccia, P., Oria, V., Özsu, T.: Flexible integration of multimedia sub-queries with qualitative preferences. Multimedia Tools and Applications 33(3), 275–300 (2007)

[3] We note here that k-NN-set computes document distances outside of the index, only for those documents that are retrieved under sorted access. On the other hand, Sky-set does not compute *any* document distance, but has nonetheless to compare documents for domination: in Figure 7 each of such comparisons is computed as a document distance, in order to compare algorithms on a fair basis.

3. Bartolini, I., Ciaccia, P., Patella, M.: Query processing issues in region-based image databases. Knowledge and Information Systems 25(2), 389–420 (2010)
4. Bartolini, I., Patella, M., Romani, C.: SHIATSU: Semantic-Based Hierarchical Automatic Tagging of Videos by Segmentation using Cuts. In: AIEMPro 2010, Florence, Italy (September 2010)
5. Chávez, E., Navarro, G., Baeza-Yates, R., Marroquín, J.L.: Proximity searching in metric spaces. ACM Computing Surveys 33(3), 273–321 (2001)
6. Ciaccia, P., Patella, M., Zezula, P.: M-tree: An efficient access method for similarity search in metric spaces. In: VLDB 1997, Athens, Greece, pp. 426–435 (August 1997)
7. Fei-Fei, L., Fergus, R., Torralba, A.: Recognizing and learning object categories. In: CVPR 2007 Short Course, Minneapolis, MN (June 2007)
8. Fishburn, P.: Preference structures and their numerical representations. Theoretical Computer Science 217(2), 359–383 (1999)
9. Gaede, V., Günther, O.: Multidimensional access methods. ACM Computing Surveys 30(2), 170–231 (1998)
10. Grauman, K.: Efficiently searching for similar images. Communications of the ACM 53(6), 84–94 (2010)
11. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: SIGMOD 1984, Boston, MA, pp. 47–57 (June 1984)
12. Hjaltason, G.R., Samet, H.: Distance browsing in spatial databases. ACM TODS 24(2), 265–318 (1999)
13. Hjaltason, G.R., Samet, H.: Index-driven similarity search in metric spaces. ACM TODS 28(4), 517–580 (2003)
14. Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-k query processing techniques in relational database systems. ACM Computing Surveys 40(4) (October 2008)
15. Kailath, T.: The divergence and Bhattacharyya distance measures in signal selection. IEEE Transactions on Communication Technology 15(1), 52–60 (1967)
16. Kuhn, H.W.: The hungarian method for the assignment problem. Naval Research Logistic Quarterly 2, 83–97 (1955)
17. Rubner, Y., Tomasi, C.: Perceptual Metrics for Image Database Navigation. Kluwer, Boston (2000)
18. Salton, G.: Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer. Addison-Wesley, Reading (1989)
19. Wu, L., Hoi, S.C.H., Jin, R., Zhu, J., Yu., N.: Distance metric learning from uncertain side information with application to automated photo tagging. In: ACM MM 2009, Vancouver, Canada, pp. 135–144 (October 2009)
20. Wang, J.Z., Li, J., Wiederhold, G.: SIMPLIcity: Semantics-sensitive Integrated Matching for Picture LIbraries. IEEE TPAMI 23(9), 947–963 (2001)
21. Zezula, P., Amato, G., Dohnal, V., Batko, M.: Similarity Search - The Metric Space Approach, Advances in Database Systems, vol. 32. Springer (2006)