A Framework for the Comparison of Complex Patterns*

Ilaria Bartolini, Paolo Ciaccia, and Marco Patella

DEIS – IEIIT/BO-CNR, University of Bologna, Italy
{ibartolini,pciaccia,mpatella}@deis.unibo.it

Abstract. Data mining and knowledge discovery techniques are commonly used to extract condensed artifacts representing huge volumes of data. The comparison of such compact and rich in semantics representations (which we call *patterns*) can be useful to avoid the direct comparison of underlying raw data. In this paper, we present a general framework for the assessment of similarity between patterns, by identifying the common features that characterize approaches proposed in the literature for particular applications. We also propose an implementation of the framework using an UML formalism, and discuss efficiency issues that arise when similarity queries are considered, i.e. when a similarity predicate is used to query a collection of pattern.

1 Introduction

The advent of the information age poses serious challenges to the database research community. In particular, the overwhelming amount of data (over 1 exabyte per year, according to a recent survey [14]) produced from a variety of sources as satellites, sensors, etc. calls for an efficient and effective management of the information they carry. Clearly, such volumes of data do not constitute information per se, and *data mining* and/or *knowledge discovery* techniques are applied to extract useful knowledge from raw data. We call the so-obtained compact representation of data *patterns*. So far, patterns have not been treated as persistent objects that can be stored, retrieved and queried by the user, whereas several modern day applications could benefit from the management of patterns as first-class citizens [18].

Among the several interesting operations on patterns (modeling, storage, retrieval), one of the most important is that of comparison, i.e. establishing whether two patterns are similar or not [11]. Such operation could be of valuable use whenever we have to measure differences of models describing evolving data or data extracted from different sources, e.g. to monitor monthly sales of a supermarket or to analyze differences of data characteristics across several sets of data (customers transactions, reactions to chemical/biological substances). If the comparison between patterns does not show substantial differences, there’s no need to perform a thorough (and costly) analysis on actual data. A similarity operator between patterns could be used to express similarity queries over pattern bases, i.e. finding the patterns, in a given collection, which are most similar to a given (query) one [18].

Besides ad-hoc approaches for particular cases, to the best of our knowledge the only attempt of defining a general framework for the comparison of patterns has been proposed in [11], where the authors present `FOCUS`, a framework for the measurement of deviations in patterns (these are called *models* in [11]) representing data characteristics. The basic idea of `FOCUS`, which is also shared by our approach, is that

[*] This work was supported by the PANDA IST Thematic Network.

patterns have a two-component nature: the *structure* component represents each pattern within the space of possible patterns, whereas the *measure* component quantifies the quality of the source data representation achieved by each pattern. For instance, in an association rule, the head and the body represent the structure, whereas measures may include support and confidence. With respect to FOCUS, however, our approach has the advantage of being applicable to a wider variety of interesting cases, since it does not require that a *greatest common refinement* exists between the structures of the patterns being compared (see Section 2). Our framework is also able to model patterns whose structural part is composed of other patterns, thus the case of the comparison of *complex* patterns is also considered. Moreover, we also focus our attention to avoid accessing the underlying raw data when comparing two patterns, whereas this important efficiency issue is not considered in [11].

The rest of the paper is organized as follows: Section 2 introduces the basic definitions of the problem, showing important examples where it arises in practice. Section 3 formally presents our approach, whereas in Section 4 we present an implementation of the framework using an OO paradigm. In Section 5 we present some examples of use of the framework. Section 6 discusses efficiency issues related to the resolution of similarity queries over collections of patterns, and Section 7 presents a brief comparison of our approach with the FOCUS framework. Finally, in Section 8 we conclude, drawing interesting directions for future work.

2 Problem Definition

A *pattern* is a compact and rich in semantics representation of raw data [18]. Patterns that share common characteristics belong to the same *pattern type*. The comparison between two patterns of the same type yields a score s, $s \in [0,1]$, assessing their mutual similarity. A similarity operator *sim*, thus, should be defined for each pattern type, in order to allow the comparison of pairs of patterns of the same type.

In order to provide a general solution, we chose to identify the common characteristics of different approaches proposed for the comparison of patterns. Our framework follows the logical model proposed in [17], thus each pattern type includes a structure schema *ss*, defining the pattern space, and a measure schema *ms*, describing the measures that quantify the quality of the source data representation achieved by each pattern.[1] Each pattern p of a pattern type pt can be obtained by instantiating the structure schema and the measure schema, obtaining the structure $p.s$ and the measure $p.m$ of pattern p. In the basic case, the similarity between patterns is computed by taking into account the similarity between the patterns' structures and measures.

Example 1 (Cluster): An Euclidean cluster in a D-dimensional space can be modeled by specifying the cluster center (a D-dimensional vector) and a radius (a real value): such components form the structure of each *cluster* pattern. Measures for cluster patterns include, for example, the average intra-cluster distance or the cardinality of the data points represented by each cluster. The as-

[1] Other components are included in the logical model of [17]. Since they are not used for the assessment of similarity between patterns, we will not detail them here for lack of space.

sessment of similarity between clusters should take into account both the structure (center + radius) and the measure components of the patterns.

The case of complex patterns, i.e. patterns whose structure schema includes other pattern types, is particularly challenging, because the similarity between structures of complex patterns depends on the similarity between component patterns.

Example 2 (Clustering): A *clustering* pattern is obtained as the composition of *cluster* patterns. In particular, the structure of the clustering pattern consists in a set of clusters. The problem of computing the similarity between two sets of clusters may arise, for example, in the comparison of different clustering algorithms or criteria (in this case, the data source from which the patterns were extracted is the same), or in the comparison of clusters obtained from different data sets (where the clustering algorithm is now kept constant). Examples of applications include the comparison of market or customer segmentations [12], fraud detection, and Region-Based Image Retrieval systems [1],[7], to name a few.

3 A Framework for Evaluation of Similarity between Patterns

In this Section we provide the formal framework for the evaluation of similarity between patterns. We will begin by taking into account simple patterns, then we will step to complex patterns, obtained through composition of other base patterns.

3.1. Similarity between Simple Patterns

The similarity between two patterns of the same pattern type *pt* can be computed by combining, by means of an *aggregation* function f_{aggr}, the similarity between both the structure and the measure components:

$$sim(p_1,p_2) = f_{aggr}(sim_{struct}(p_1.s,\ p_2.s),\ sim_{meas}(p_1.m,\ p_2.m)) \qquad (1)$$

where with *p.s* and *p.m* we indicate the structure and the measure for pattern *p*, respectively. If the two patterns have the same structural component, then $sim_{struct}(p_1.s,p_2.s) = 1$, and the measure of similarity naturally corresponds to a comparison of the patterns' measures, e.g. by aggregating differences between each measure [11]. In the general case, however, the patterns to be compared have different structural components, thus a preliminary step is needed to reconcile the two structures to make them comparable. The aggregation function f_{aggr} can be associated to the pattern type *pt* of the two patterns to be compared or can be specified by the user at query time, by choosing between a set of available functions (for the pattern type *pt*).

The computation of similarity between simple patterns is summarized in Fig. 1, where we show how the similarity is obtained by aggregating, by means of the f_{aggr} function, the similarities between patterns' structures and measures. It should be noted that the sim_{struct} block could encompass the use of an underlying knowledge, e.g. when comparing spatial points over a connectivity network [16], or when matching keywords according to a hierarchical ontology, such as WordNet [15] (see also Example 8). In such cases, the knowledge needed for comparison is included in the pattern type, and is shared by all patterns of the same type.

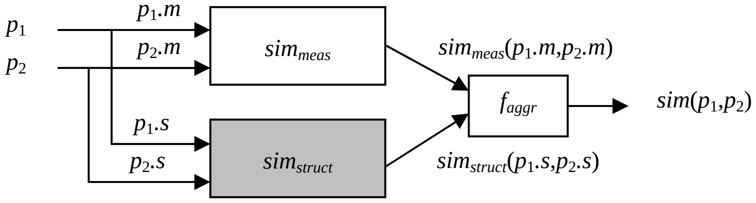


Fig. 1. Assessment of similarity between patterns.

Example 3 (Association rules): An association rule p can be modeled as a simple pattern whose structure consists in a couple ($head \rightarrow body$) and whose measure $conf$ represents the confidence of the rule. A possibility for computing similarity between association rules is the following: the structure similarity can be evaluated as the average of overlap similarities on heads and bodies, $sim_{struct}(p_1.s,p_2.s) = \frac{1}{2}\left(\frac{|p_1.head \cap p_2.head|}{|p_1.head \cup p_2.head|} + \frac{|p_1.body \cap p_2.body|}{|p_1.body \cup p_2.body|}\right)$, whereas similarity between measures is evaluated as the complement of the absolute difference between rules' confidence, $sim_{meas}(p_1.m,p_2.m) = 1 - |p_1.conf - p_2.conf|$, and f_{aggr} is the product. As an example, the similarity score s between rules p_1={(A→BC), 0.6} and p_2={(A→C), 0.7} is obtained as s = 0.675 = 0.5 * (1 + 0.5) * (1 – 0.1).

3.2. Similarity between Complex Patterns

In our framework, the similarity between complex patterns is conceptually evaluated in a bottom-up fashion using two main components, namely:

- The *coupling type*, that is used to establish how component patterns are matched to produce the overall similarity score.
- The *aggregation logic*, which is used to combine the similarity scores obtained for coupled component patterns into a single overall score representing the similarity between complex patterns.

It should be noted that our presentation proceeds bottom-up in order to simplify the exposition of our arguments, identifying the main components of our framework. This does not require that the similarity is actually computed this way: we will discuss efficient evaluation of similarity between patterns in Section 6.

3.2.1 Coupling Type

Since every complex pattern is formed by (a number of) component patterns, in comparing two complex patterns cp_1 and cp_2, we need a way to associate component patterns of cp_1 to component patterns of cp_2; of course, in computing the similarity between cp_1 and cp_2, only similarities between coupled component patterns will be considered. The coupling type, thus, establishes the way component patterns can be associated (*matched*) to produce the overall similarity score s between complex patterns. The overall matching criterion is to couple together patterns that are similar to each other (i.e. whose similarity score is as high as possible).

Formally, assume, without loss of generality, that component patterns can be given an ordinal number, thus each complex pattern *cp.s* can be represented as $(p^1, p^2, \ldots, p^N)$. Each coupling between cp_1 and cp_2 produces a matching set $\mathcal{H} = \{(i,j)$: p_1^i is matched with $p_2^j\}$. The coupling can be represented by a *matching matrix* $\mathbf{X}_{N\times M} = (x_{ij})$, where each $x_{ij} \in [0,1]$ ($i = 1, \ldots, N$; $j = 1, \ldots, M$) represents the matching between p_1^i and p_2^j ($x_{ij} = 1$ if p_1^i is matched with p_2^j). Different coupling types introduce a number of constraints on the x_{ij} coefficients, as the following examples demonstrate:

- *N–M matching*: Here, the coupling is the most general, thus we have no constraints on the **X** matching matrix. In line of principle, every x_{ij} coefficient can assume every value in the range [0,1]. We have to keep in mind, though, that the overall similarity is obtained as the maximum similarity obtained over all possible couplings, thus extreme cases, such as all $x_{ij} = 0$ or all $x_{ij} = 1$, are ruled out dependently on the particular function g_{aggr} used to aggregate basic component scores, provided that g_{aggr} is monotonically non-decreasing (see Section 3.2.2).
- 1–1 *matching*: In this case we have additional constraints on the x_{ij} coefficients, since we accept at most one matching for each component pattern p_1^i or p_2^j. Again, the extreme case were all $x_{ij} = 0$ is not possible, since we are taking the maximum similarity obtained through the aggregation function g_{aggr}. Partial matchings occur whenever it is $N \neq M$. This is the case considered in [2],[4].
- *DTW matching*: The Dynamic Time Warping (DTW) distance has been widely used to compare time sequences [5],[19],[13], but recent applications also benefit from its ability of accommodating elastic deformations of sequences being compared. For example, in [3] the DTW distance has been used for the retrieval of shapes using the Discrete Fourier Transform (DFT). It should be noted that, because of the possibility of stretching the sequences being compared, in the general case the DTW distance is *not* a metric [13]. The matching requested by the DTW distance is a particular case of an *N–M* matching, where additional constraints are introduced to only allow continuous and ordered matching paths.

3.2.2 Aggregation

After the coupling between component patterns has been performed, the overall similarity between complex patterns is computed by aggregating similarity scores obtained for matched component patterns. Formally, each pairing (p_1^i, p_2^j) contributes to the overall score with the similarity between its matched component patterns, $sim(p_1^i, p_2^j)$: $sim_{struct}(cp_1.s, cp_2.s) = \max_{\mathcal{H}}(g_{aggr}((p_1^1, p_1^2, \ldots, p_1^N),(p_2^1, p_2^2, \ldots, p_2^M), \mathcal{H}))$, where in the last formula we explicitly outlined the dependency on the (optimal) matching set $\mathcal{H}$ (see Section 3.2.1).[2]

We require that the aggregation function g_{aggr} is monotone: it is reasonable to only consider functions that are non-decreasing in the similarity between matched patterns. More precisely, if we consider two different matching sets $\mathcal{H}$ and $\mathcal{H}'$, that only differ in a matching pair $(i,j) \neq (i',j')$, $(i,j) \in \mathcal{H}$, $(i',j') \in \mathcal{H}'$, and $sim(p_1^i, p_2^j) \geq sim(p_1^{i'}, p_2^{j'})$,

[2] It should be noted that here, for ease of presentation, we suppose that the structure part of complex patterns only consists of complex types including component patters. In the general case, other base types can be part of the structure of the complex pattern, and sim_{struct} should also take into account the similarity between such components. Our framework can be easily extended to deal with such cases, thus we will ignore them in the following.

then $g_{aggr}(cp_1.s, cp_2.s, \mathcal{H}) \geq g_{aggr}(cp_1.s, cp_2.s, \mathcal{H}')$, i.e. best-matched pairings can only increase the overall similarity score.

Example 4 (Aggregate Time Series): An aggregate time series is obtained by combining together a number of (simple) time series. For example, the price of a stock during an entire year can be obtained by aggregating the daily stock prices grouped on a monthly basis: in this case, the aggregate time series is a list of monthly sequences. If we want to compare the annual trends of two different stocks, we have to aggregate the similarities between monthly sequences (note that in this case an obvious 1–1 matching is performed). Usually, such aggregation is obtained by averaging the similarities between matched subsequences.

3.3. Overall View of the Framework

The process of computing the structure similarity between complex patterns in our framework is summarized in Fig. 2, showing how the aggregation function is used to assess the similarity between pattern structures (gray box in Fig. 1). In particular, the “max” block should loop through all the possible couplings produced by the “Matcher” block, i.e. those allowed by the available constraints; for all such couplings, the “sim” block should provide the similarity scores between matched patterns, that will be aggregated by the “g_{aggr}” block. In case of multi-level aggregations, the similarity block (gray box in Fig. 2) might encompass the recursive computation of similarity between complex patterns.

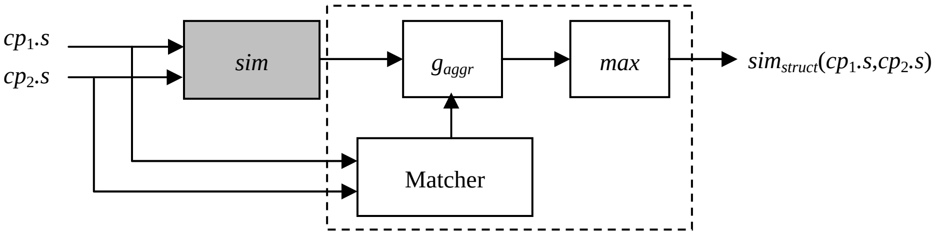


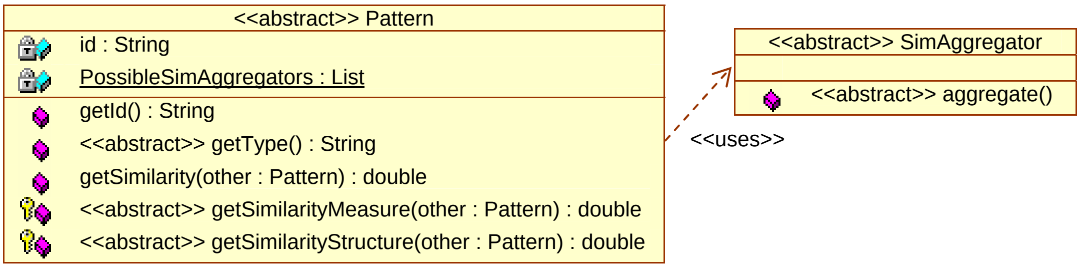
Fig. 2. How the similarity between complex patterns’ structures is evaluated.

Example 5 (Sets of Association Rules): Consider the comparison of two sets of association rules extracted from a data set, $P_1 = \{p_1^1=\{(A\rightarrow BC), 0.6\}, p_1^2=\{(AC\rightarrow B), 0.5\}, p_1^3=\{(AB\rightarrow C), 0.4\}\}$, and $P_2 = \{p_2^1=\{(A\rightarrow B), 0.8\}, p_2^2=\{(A\rightarrow C), 0.7\}\}$. Since the complex patterns have no measure, their similarity is assessed directly as in Fig. 2, e.g. using a simple average as g_{aggr}. If the similarity between rules is evaluated as in Example 3, the overall score s is equal to 0.6 and corresponds to matching rule p_1^1 with rule p_2^2 (with score s = 0.675 = 0.75 * (1–0.1)) and rule p_1^2 with rule p_2^1 (s = 0.525 = 0.75 * (1–0.3)).

4 Implementation of the Proposed Framework

Here we provide a possible implementation of the framework using an UML formalism. We present an overview of the foundation of classes, demonstrating the generality of the proposed framework in accommodating different aggregation logics and similarity measurements for specific pattern types.

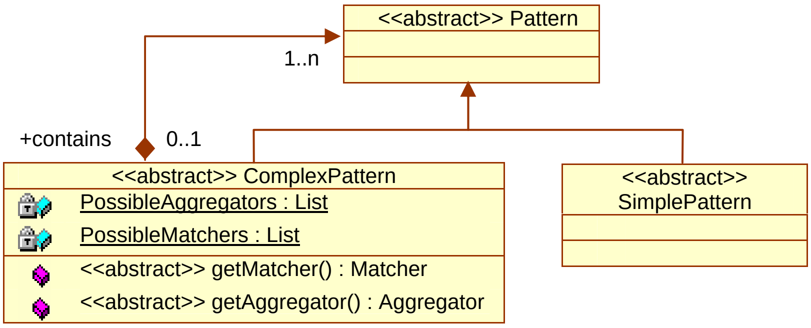
The core of the framework is, obviously, the abstract Pattern class (Fig. 3). Each pattern has an identifier id and refers to a type through the getType() method (which uses reflection to return the name of the class of each pattern instance). The getSimilarity() method computes the overall similarity score by aggregating scores obtained through the abstract methods getSimilarityMeasure() and getSimilarityStructure() (these are defined in each concrete extension of the Pattern class). Such scores are combined by the getSimilarity() method into the overall score *s* through an object of the abstract class SimAggregator representing the aggregation function f_{aggr} (we have implemented several simple aggregation functions that are not detailed in Fig. 3). The different aggregation functions available for each pattern type (see Section 3.1) are stored in the static member PossibleSimAggregators (static members are underlined in the following figures), containing references to SimAggregator objects.



<<abstract>> Pattern
id : String
PossibleSimAggregators : List
getId() : String
<<abstract>> getType() : String
getSimilarity(other : Pattern) : double
<<abstract>> getSimilarityMeasure(other : Pattern) : double
<<abstract>> getSimilarityStructure(other : Pattern) : double
<<uses>>
<<abstract>> SimAggregator
<<abstract>> aggregate()

Fig. 3. The Pattern class.

The abstract class pattern has two different subclasses (Fig. 4): ComplexPattern and SimplePattern. The abstract ComplexPattern class contains, through aggregation, other instances of Pattern, that can be themselves simple or complex patterns, thus achieving a multi-level aggregation hierarchy. The static members PossibleAggregators and PossibleMatchers are used to hold references to aggregation functions (g_{aggr}) and matchers that can be possibly used for a given pattern type and/or chosen at query time.



<<abstract>> Pattern
1..n
+contains
0..1
<<abstract>> ComplexPattern
PossibleAggregators : List
PossibleMatchers : List
<<abstract>> getMatcher() : Matcher
<<abstract>> getAggregator() : Aggregator
<<abstract>>
SimplePattern

Fig. 4. The Pattern class hierarchy.

The relations existing among complex patterns, matchers and aggregation functions are depicted in Fig. 5, where we highlight the fact that each pattern type uses its own Matcher and Aggregator objects, chosen from the PossibleMatchers and PossibleAggregators lists. Each Matcher, in computing the similarity score between two complex objects, uses their relative aggregation function, obtained through the getAggregator() method, to compute the overall score for each match (see Section 3.2.2).

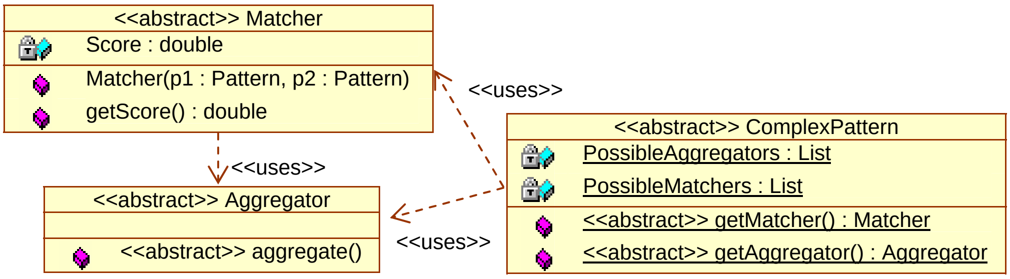

<<abstract>> Matcher
Score : double
Matcher(p1 : Pattern, p2 : Pattern)
getScore() : double
<<uses>>
<<uses>>
<<abstract>> ComplexPattern
PossibleAggregators : List
PossibleMatchers : List
<<abstract>> getMatcher() : Matcher
<<abstract>> getAggregator() : Aggregator
<<abstract>> Aggregator
<<abstract>> aggregate()
<<uses>>

Fig. 5. Complex patterns, matchers and aggregation functions.

5 Examples of Use of the Framework

In this Section, we will present some relevant cases where the problem of comparison between complex patterns arises in practice. For each case, we also provide hints about the implementation of the abstract classes presented in Section 4.

Example 6 (Region-Based Image Retrieval Systems): The goal of content-based image retrieval (CBIR) systems is to define a set of properties (*features*) able to effectively characterize the content of images and then to use such features during retrieval in order to provide effective and efficient access to image databases based on content. To increase the effectiveness of image retrieval, in recent times a number of *region-based* image retrieval systems has been presented [7],[1], which “fragment” each image into regions, i.e. sets of pixels sharing common visual characteristics, like color and texture. Similarity between images is then assessed by computing similarity between pairs of regions and combining the results at the image level.

Conceptually, each image is represented as a set of component regions. Thus, we can represent each region as a simple pattern and the overall image as a set of region patterns. This way, the problem of finding the k images that most resemble a given query one (Top-k query) can be modeled as a best-matches query over the space of image patterns. In particular, following the model presented in Section 3, we have:

- The criterion used to assess the similarity between two regions varies from system to system. For example, the WINDSURF system [1] segments images into sets of pixels that are homogeneous for color and texture by using the Discrete Wavelet Transform (DWT) and a fuzzy c-means algorithm. Each region is then represented as an elliptical cluster in the HSV space and the similarity between regions is computed by taking into account both differences in the color and texture descriptors (the pattern structure) by way of the Bhattacharyya distance and in their relative size (the pattern measure). For more details, see [1].

– Each image region could be matched to at most one region of the image being compared, thus an 1–1 matching is required with possible partial matches (see Section 3.2.1).
– The most used aggregation function g_{aggr} is the average (e.g. this is the case for Blobworld [7] and Windsurf [1] systems).

The Hungarian algorithm [2] is used to solve the optimal matching problem at the image level, thus we provide an implementation of such algorithm in the HungarianMatcher sub-class of the Matcher class.

Example 7 (Dynamic Time Warping): In its essence, the DTW is used to compare sequences, performing an *N*–*M* matching that defines a connected warping path over the space of possible couplings (see Section 3.2.1). The optimal warping path can be computed in quadratic time by dynamic programming [5]. The TimeWarpingMatcher class we implemented can also take into account other types of constraints on the warping path, e.g. to limit its deviation from the diagonal [13].

Example 8 (Web sites): We finish with a complex example, including a two-levels hierarchy of pattern composition, in order to demonstrate the expressive power of our framework. Suppose we want to compare two web sites, represented as graphs of HTML pages connected by hyper-links. The pages are complex patterns whose structure is represented by a set of keywords (simple patterns). The similarity between keywords can be assessed using a similarity function that exploits a hierarchical structure, like WordNet [15] (see Section 3.1); for example, it is common to take into account the difference in height between the keywords and their lowest common ancestor in the hypernymy/hyponymy tree [10]. Similarity between pages is then computed using an 1–1 matching between keywords (see Section 3.2.1), maximizing the aggregation function g_{aggr} (e.g. the average similarity between matched keywords). Finally, the overall similarity between web sites (graphs of pages) can be computed by way of a measure of similarity between graphs [6], where the score between matched nodes is obtained as indicated before.

6 Efficient Evaluation of Similarity Queries

Besides the problem of the efficient comparison between two complex patterns, that, as shown in Section 5, requires the matcher to be aware of the available constraints and of the g_{aggr} function used to aggregate scores in order to implement the appropriate algorithm to solve the assignment problem at hand, performance issues arise when considering similarity queries, i.e. queries whose predicates are based on a similarity operator. In particular, we are interested in the following query types, which can be applied to a pattern collection (*class*) $\mathcal{C}$:

Top-*k* queries: Here the user requests for the best k matching patterns with respect to a given (query) pattern qp.

Threshold queries: Given a query pattern qp and a similarity threshold ϑ, a threshold query requests for patterns having a similarity score s with respect to qp not lower than the threshold, $s \geq \vartheta$.

The naïve (sequential) evaluation of both kind of queries entails comparing the query pattern qp to all the patterns in $\mathcal{C}$, thus its complexity is linear in the cardinality of $\mathcal{C}$. When the number of patterns in $\mathcal{C}$ is significant, however, sequential evaluation becomes impractical, and we have to recur to index structures to prune out from the search a (large) subset of $\mathcal{C}$.

The use of index structures for efficient processing of similarity queries requires the use of upper bounds on the similarity between complex patterns. In fact, if we are able to quickly compute an upper bound $sim_{UB}(qp, p_i)$ on the similarity between qp and a pattern p_i, then it is possible to avoid computing $sim(qp, p_i)$ if $sim_{UB}(qp, p_i) < \vartheta$, in the case of a threshold query, or if $sim_{UB}(qp, p_i)$ is not higher than the similarity between qp and its worst (k-th) matching computed so far, for Top-k queries.

The bound $sim_{UB}(qp, p_i)$ can be computed in two different ways, depending on the particular matching criterion:

1. **Using constraints and monotonicity of g_{aggr}:** In this case, we use upper bounds on the similarity between component patterns to derive, using matching constraints and the monotonicity of g_{aggr}, the upper bound $sim_{UB}(qp, p_i)$. A typical case is when component patterns are accessed through an index that returns matches in decreasing order of similarity with respect to component patterns of qp [2],[4].
2. **Using triangle inequality:** When the similarity criterion is based on a distance function δ, $sim(qp, p_i) = f(\delta(qp, p_i))$, lower bounds on δ can be obtained by way of the triangle inequality, in cases where δ is a metric. This is, indeed, the basic rationale used by metric access methods (see Section 6.1) to prune away from the search subsets of the data space.

It is clear that, in the first case, the index structure should be tightly coupled with the matching block of Fig. 2, in that the matching block should be able to compute the upper bound $sim_{UB}(qp, p_i)$ using only the knowledge obtained so far by the index structure. In the other case, the index structure can simply use the similarity block of Fig. 1 as a black box, thus completely ignoring its internal logics.

6.1. Index Structures

Efficient resolution of similarity queries on patterns requires the use of appropriate index structures. As pointed out before, optimization of queries requires computing upper bounds on the similarity between complex and/or component patterns. Access methods for query processors, therefore, should provide facilities for accessing patterns in decreasing order of similarity with respect to a query pattern qp, and/or for computing upper bounds on the similarity between complex patterns by means of an efficient access to component patterns.

The efficient processing of similarity queries through indexing is usually based on an indirect evaluation of similarity scores: in this case, what is actually measured is the distance between patterns, being understood that high scores correspond to low distances and low scores to high distances. If the distance δ used to compute the similarity satisfies the metric postulates, then metric access methods, like the M-tree [9], provide the basic functionalities needed by the query processor. In particular, metric access methods are able to:

- Efficiently solve range queries, returning all the patterns whose distance δ with respect to the query pattern *qp* is lower than a user-specified threshold value.
- Efficiently solve *k*-NN queries, where the user requests for the *k* patterns which are closest, according to δ, to the query pattern *qp*.
- Perform a sorted access, returning all the indexed patterns in increasing order of δ with respect to the query pattern *qp*.

Such properties allow us to use metric access methods to index complex and/or component patterns, depending on the functions used to compute similarity at each level. In the case where δ is not a metric function, it is often possible to find a metric function δ_{LB} which is a lower bound on δ, $\delta_{LB} \leq \delta$, and to use δ_{LB} in place of δ to prune the pattern space [8].

7 Comparison with the FOCUS framework

The first attempt to present a general approach for the comparison of compact information mined from raw data was proposed in [11], where the FOCUS framework for measuring changes in data characteristics is presented. In FOCUS, patterns are described through structure and measure components.

The main limit of FOCUS is the fact that, in order to compare two patterns, it is supposed that a *greatest common refinement* (GCR), i.e. a common structure that is a refinement of structures of both patterns, can be found. Even if this is the case for all patterns considered in [11], this is not true in the general case (e.g. consider the case of patterns extracted from different data sets). Moreover, recurring to a GCR does not allow matchings that take into account the patterns' measures, like the DTW matching of Example 7, since matching components are selected only by considering the structure of the GCR. Our framework, on the other hand, does not rely on such restrictive requirements, but only needs that a similarity operator can be applied to component patterns, thus it is applicable to a much broader variety of cases.

As to efficiency issues, the FOCUS framework requires the computation of the GCR of patterns being compared. This entails accessing raw data to compute measures of patterns in the GCR (even if, for some pattern types, upper bounds can be computed without scanning the data). The goal of our framework, on the other hand, is to avoid at all accessing the underlying raw data when comparing two patterns, thus it only exploits information already present in the patterns.

8 Conclusions

In this paper we have presented a framework for the comparison of patterns that can be used to quickly compare characteristics of large data sets without accessing raw data. Assessing the similarity between patterns entails the comparison of structures and measures, and our framework is also able to capture the case of complex patterns, i.e. patterns whose structure consists of other patterns, obtaining a *part-of* hierarchy. Our work extends the FOCUS framework presented in [11], enlarging its generality and taking into account the important subject of efficient computation of similarity, an issue raised by the resolution of similarity queries over collections of patterns.

A series of interesting working issues are still left open, which we plan to tackle in the future. In our opinion, the most important ones concern the efficient processing of similarity queries over collection (classes) of patterns. Even if the solution of the general problem appears as a formidable task, some solutions that have been proposed for particular cases [2],[13] seem to be applicable to a much broader class of problems.

9 References

[1] S. Ardizzoni, I. Bartolini, and M. Patella. Windsurf: Region-based image retrieval using wavelets. IWOSS'99, pp. 167-173, 1999.

[2] I. Bartolini, P. Ciaccia, and M. Patella. A sound algorithm for region-based image retrieval using an index. QPMIDS 2000, pp. 930-934, 2000.

[3] I. Bartolini, P. Ciaccia, and M. Patella. Using the time warping distance for Fourier-based shape retrieval. IEIIT-BO-03-02 Technical Report, 2002.

[4] I. Bartolini, P. Ciaccia, and M. Patella. Correct algorithms for the comparison of complex patterns. PANDA Workshop on Pattern-Base Management Systems, pp. 55-62, 2003.

[5] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. AAAI Workshop on Advances in Knowledge Discovery in Databases, pp. 359-370, 1994.

[6] H. Bunke: Recent developments in graph matching. ICPR 2000, pp. 2117-2124, 2000.

[7] C. Carson, M. Thomas, S. Belongie, J. M. Hellerstein, and J. Malik. Blobworld: A system for region-based image indexing and retrieval. VISUAL'99, pp. 509-516, 1999.

[8] P. Ciaccia and M. Patella. Searching in metric spaces with user-defined and approximate distances. ACM Transactions on Database Systems, 27(4), pp. 398-437, 2002.

[9] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. VLDB '97, pp. 426-435, 1997.

[10] P. Ganesan, H. Garcia-Molina, J. Widom: Exploiting hierarchical domain structure to compute similarity. ACM Transaction on Information Systems, 21(1), pp. 64-93, 2003.

[11] V. Ganti, J. Gehrke, R. Ramakrishnan, and W.-Y. Loh. A framework for measuring changes in data characteristics. PODS'99, pp. 126-137, 1999.

[12] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. ACM Computing Surveys, 31 (3), pp. 264-323, 1999.

[13] E. J. Keogh. Exact indexing of dynamic time warping. VLDB 2002, pp. 406-417, 2002.

[14] P. Lyman and H. R. Varian. How much information. Available at URL (valid as in March 2004) http://www.sims.berkeley.edu/how-much-info.

[15] G. A. Miller: WordNet: A lexical database for English. Communications of the ACM, 38(11), pp. 39-41, 1995.

[16] D. Papadias, J. Zhang, N. Mamoulis, Y. Tao. Query processing in spatial network databases. VLDB 2003, pp. 802-813, 2003.

[17] S. Rizzi, E. Bertino, B. Catania, M. Golfarelli, M. Halkidi, M. Terrovitis, P. Vassiliadis, M. Vazirgiannis, and E. Vrachnos. Towards a logical model for patterns. ER 2003, pp. 77-90, 2003.

[18] Y. Theodoridis, M. Vazirgiannis, P. Vassiliadis, B. Catania, and S. Rizzi. A manifesto for pattern bases. PANDA Technical Report TR-2003-03, 2003.

[19] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. ICDE 1998, pp. 201-208, 1998.