

# PROFI



Project number:	FP6-511572
Project acronym:	PROFI
Title:	Perceptually-relevant Retrieval Of Figurative Images

Deliverable No: D4.2:	Implementation of region matching algorithms
-----------------------	--

**Short description:**

We implemented the randomized algorithm for finding similarities between two shapes  $A$  and  $B$ , and the matching algorithm based on image primitives, that were described in deliverable 4.1. Shapes are modelled by sets of regions. For the transformation based matching the allowed classes of transformations are translations, rigid motions, homotheties and similarities. The default transformation class is similarities.

Due month:	M33
Delivery month:	M33
Lead partner:	Freie Universität Berlin
Partners contributed:	Freie Universität Berlin
Classification:	RE



Project funded by the European Community under the "Information Society Technologies" Programme

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Algorithms as described in Deliverable D4.1</b>	<b>2</b>
2.1	Probabilistic matching . . . . .	2
2.2	Matching based on Image Primitives . . . . .	6
<b>3</b>	<b>Results</b>	<b>10</b>
3.1	Implementation . . . . .	10
3.2	Experiments . . . . .	11
<b>4</b>	<b>Deviations from plan</b>	<b>13</b>
<b>A</b>	<b>Experimental Results</b>	<b>15</b>
<b>B</b>	<b>API – Matching</b>	<b>18</b>
B.1	FigurativeMatcher Class Reference . . . . .	18
B.2	FigurativeMatcher::Result Class Reference . . . . .	20
<b>C</b>	<b>API – Image representation as set of primitives</b>	<b>21</b>
C.1	FigurativeImage Class Reference . . . . .	21
C.2	Figure Class Reference . . . . .	23
C.3	EllipseFigure Class Reference . . . . .	27
C.4	TriangleFigure Class Reference . . . . .	28
C.5	RectangleFigure Class Reference . . . . .	29
C.6	ConvexFigure Class Reference . . . . .	30
C.7	ComplexFigure Class Reference . . . . .	31
C.8	FigureRelation Class Reference . . . . .	32
C.9	XMLValidator Class Reference . . . . .	34

## 1 Introduction

In this work package we develop and implement algorithms for matching two planar shapes. We assume that shapes are modeled by sets of plane simple polygons. Other than in previous work on objectives 3 and 4 the interiors of the polygons are considered as part of the shape, as well, not only their boundaries. As possible classes of transformations we consider translations, rigid motions (i.e., translations and rotations), and similarities (i.e., translations, rotations and scalings). The general situation is that we are given two objects  $A$  and  $B$  and a set  $T$  of allowable transformations and we want to transform  $B$  optimally so that the transformed image of  $B$  is as close to  $A$  as possible.

Working on Objective 1 we developed two algorithms: One of them is a probabilistic algorithm for matching region shapes under translations, rigid motions and similarities with respect to the area of overlap. The second algorithm is motivated by the observation that there are shapes that are perceived to be similar by humans, but for which there is no transformation that maps one shape completely to the other. The idea of matching based on image primitives is to decompose a shape into primitive components and to find best matching between the primitives of two shapes. The similarity of primitive components can be computed using a transformation based algorithm, for example curve based matching described in deliverable D4.3 or region based matching described in deliverable D4.1.

Objective 2 of this work package is implementation of region based algorithms. In this deliverable we provide a C++ and Java implementation of the shape matching algorithm based on image primitives described in deliverable D4.1. This report contains API description of the implemented algorithm and reports on experimental evaluation of the algorithms.

## 2 Algorithms as described in Deliverable D4.1

### 2.1 Probabilistic matching

#### 2.1.1 The similarity measure

We want to compare shapes that are regions in the plane. An established similarity measure for regions is the area of overlap. Two shapes are understood as similar if they overlap much. Psychological studies show that the area of overlap is an essential factor for perceived similarity [17].

The first row of Figure 1 shows two discs, one of them with noise added. Figures extracted from digital images often contain noise that should not affect the similarity of the figures. Since noise is expected to have a very small area, this is not the case if similarity is measured by the area of overlap. So one advantage of the area of overlap as similarity measure is its insensitivity to noise.

Nevertheless, there are examples where the area of overlap does not quite capture perceived similarity. An instance is shown in the second row of Figure 1, where the two shapes are perceived as quite different but the area of overlap is quite large if the blot is moved on top of the star.

The last row of Figure 1 indicates that comparing regions is a different matter than comparing curves. All common similarity measures for curves would assign a very small distance to the disc and the ring but the distance measured by area of overlap is very large.

If no scaling is allowed maximizing the area of overlap is the same as minimizing the symmetric difference which is a metric on the set of all shapes.

#### 2.1.2 The algorithm

Suppose that two regions  $A$  and  $B$  in  $\mathbb{R}^2$  and a set of transformations  $F$  are given. We present a randomized algorithm finding a transformation  $f \in F$  that approximately maximizes the area of intersection of the

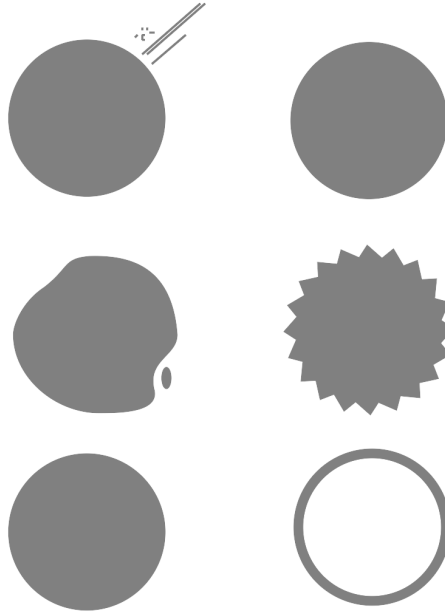


Figure 1: Some examples for the area of overlap as similarity measure.

transformed shape  $f(A)$  and  $B$ . We will investigate the cases where the set of transformations  $F$  consists of translations, rigid motions, and similarities. A rigid motion consists of a rotation angle and a translation vector, a similarity is composed of rotation, translation and scaling.

We define shapes quite generally as Lebesgue measurable sets in  $\mathbb{R}^2$ . Suppose, there are given a fixed, small  $\delta > 0$ , two shapes  $A$  and  $B$ , and a set of transformations  $F$  which is identified with a subspace of some  $\mathbb{R}^k$ . We examine the following algorithmic scheme:

1. For  $i = 1, \dots, n$  choose uniformly distributed a random point sample of suitable size in  $A$  and a random point sample in  $B$ .
2. Give one vote to the unique transformation in  $F$  that maps the sample from  $A$  onto the one from  $B$ .
3. Determine and return the transformation whose  $\delta$ -neighborhood in  $F$  obtained the most votes.

Of how many random points one sample consists depends on the selected set of transformations. Size and form of a sample are chosen in a way that there exists exactly one transformation that maps one sample onto another sample.

The votes approximate the probability distribution in  $F$  resulting from the experiment. The output of the algorithm is a transformation whose  $\delta$ -neighborhood in  $F$  has approximately highest probability. If  $\delta$  is small enough this transformation is one at which the density function of the probability distribution is maximal.

In the following it is shown that for translations and rigid motions the density function of the probability distribution in  $f \in F$  equals the area of overlap of  $f(A)$  and  $B$ . Thus, the algorithm gives a transformation that approximately maximizes the area of overlap.

For similarities we do not have an equally simple characterization. Nevertheless, we will derive a formula of the density function and give an intuitive explanation. Furthermore, it is shown that the induced similarity measure is not symmetric in case of similarities.

### 2.1.3 Translations

In case of translations a random point sample consists of one point in each shape because if we choose a point  $a \in A$  and a point  $b \in B$ , there is exactly one translation that maps  $a$  onto  $b$ , namely  $t = b - a$ . The transformation space  $T$  equals  $\mathbb{R}^2$  where a point  $(t_1, t_2) \in T$  means the translation  $(x, y) \mapsto (x+t_1, y+t_2)$ . Note that  $(-A) \oplus B$  contains exactly those translations that map at least one point of  $A$  onto a point of  $B$ , where  $\oplus$  denotes the Minkowski sum.

For a region  $G \subset \mathbb{R}^2$  we denote its area by  $|G|$ . For a translation  $t \in T$  we denote the  $\delta$ -neighborhood of  $t$  with respect to the maximum norm by  $B_\delta(t)$ .

We are interested in the area of the intersection of  $A$  translated by  $t$  with the shape  $B$  denoted by

$$\mu(t) = |(A + t) \cap B|.$$

Let  $S$  be a set of  $n$  translations resulting from the experiment. The number of votes for the  $\delta$ -neighborhood of  $t$  is given by  $|S \cap B_\delta(t)|$ . Our estimate of the probability that a translation resulting from one random experiment lies in the  $\delta$ -neighbourhood of  $t$  is the ratio of the number of votes for the  $\delta$ -neighborhood of  $t$  to the number of all votes, namely  $\frac{|S \cap B_\delta(t)|}{|S|}$ . We need to normalize this value with the sizes of the shapes and the size of the neighborhood of  $t$  to get an estimate  $e_S(t)$  of  $\mu(t)$ :

$$e_S(t) = \frac{|S \cap B_\delta(t)| \cdot |A| \cdot |B|}{|S| \cdot |B_\delta(t)|}.$$

It can be shown that the error of the estimate becomes small if  $n$  is large enough. More precisely, it can be shown that for each  $\varepsilon > 0$ ,  $p < 1$  and each  $\delta > 0$  there is a  $n \in \mathbb{N}$  such that the error  $|\mu(t) - e_S(t)|$  is less than  $\varepsilon$  with probability at least  $p$ . We achieve this by proving the following lemma:

**Lemma 2.1.** *For all shapes  $A$  and  $B$ ,  $\delta > 0$  and  $t \in T$  with probability at least  $1 - 2e^{-\frac{\varepsilon^2 \cdot n}{2}}$  it holds*

$$|\mu(t) - e_S(t)| \leq \frac{|A| \cdot |B| \cdot \varepsilon}{4\delta^2} + \sqrt{2}\Delta\delta$$

where  $\Delta$  denotes the maximum of the lengths of the boundaries of  $A$  and  $B$ .

From this lemma it follows that in case of translations the algorithm returns a translation that approximately maximizes the area of overlap of a translated copy of  $A$  with the shape  $B$  for all shapes  $A$  and  $B$ . More formally: Let  $t^*$  be the map that maximizes the estimate  $e_S(t)$  and let  $t_{\text{opt}}$  be an optimal solution. If the approximation error  $|\mu(t) - e_S(t)|$  is less than  $\varepsilon$  then the difference between approximation and optimum is at most  $2\varepsilon$ :

$$\mu(t^*) \geq e_S(t^*) - \varepsilon \geq e_S(t_{\text{opt}}) - \varepsilon \geq \mu(t_{\text{opt}}) - 2\varepsilon.$$

**2.1.3.1 A bound for the required number of random samples in case of translations.** For translations we can deduce a concrete bound for the required number of random samples from Lemma 2.1.

**Corollary 2.2.** *Let  $\tilde{\varepsilon} > 0$  and  $p \in [0, 1]$ . Let  $t^*$  be the map that maximizes the estimate  $e_S(t)$  and let  $t_{\text{opt}}$  be an optimal solution. If*

$$n \geq \frac{|A|^2 |B|^2 (-\log(\frac{1-p}{2}))}{8\delta^4 (|B|\tilde{\varepsilon} - \sqrt{2}\Delta\delta)^2}$$

the approximation error  $|\mu(t_{\text{opt}}) - e_S(t^*)|$  is less than  $2\tilde{\varepsilon}$  with probability at least  $1 - p$ . The bound is minimal for  $\delta = \frac{\sqrt{2}|B|\tilde{\varepsilon}}{3\Delta}$ .

The theoretical bound for  $n$  seems to be quite pessimistic. Our test implementation shows that the real convergence is much faster than the bound indicates. In fact, the results that our preliminary implementation shows for translations are very promising.

### 2.1.4 Rigid Motions

The space of rigid motions is given by  $R = [0, 2\pi) \times T \subseteq \mathbb{R}^3$  the first coordinate defining the anti-clockwise rotation angle. A point  $(\alpha, (t_1, t_2)) \in R$  denotes the map defined by  $(x, y) \mapsto M_\alpha \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_1 \\ t_2 \end{pmatrix}$ .

Observe that there exists exactly one translation that maps a point  $p \in A$  onto a point  $q \in B$  while there exist infinitely many rigid motions that map  $p$  onto  $q$ . To be more precise, for each  $\alpha \in [0, 2\pi)$  there is exactly one rigid motion whose anti-clockwise rotation angle is given by  $\alpha$  and that maps  $p$  onto  $q$ . This rigid motion is given by  $r = (\alpha, q - M_\alpha p)$  where  $M_\alpha$  is the rotation matrix  $\begin{pmatrix} \cos \alpha & \sin \alpha \\ -\sin \alpha & \cos \alpha \end{pmatrix}$ .

If we choose two random points  $p$  and  $p'$  in  $A$  and two random points  $q$  and  $q'$  in  $B$  there is exactly one similarity that maps  $p$  onto  $q$  and  $p'$  onto  $q'$ . (To determine one affine transformation we need 3 points from each shape.) If we want to determine exactly one rigid motion there is no size of the random sample that works. Therefore we pick uniformly distributed a random angle  $\alpha$  in  $[0, 2\pi)$ , one point  $a$  in  $A$  and one point  $b$  in  $B$ . We give one vote to the unique rigid motion with anti-clockwise rotation angle  $\alpha$  that maps  $a$  onto  $b$ .

If we do this experiment very often we get an approximation of a probability distribution on  $R$ . The algorithm returns a rigid motion whose  $\delta$ -neighborhood in  $R$  got the most votes and thus has approximately highest probability. If  $\delta$  is small enough the rigid motion whose  $\delta$ -neighborhood in  $R$  has highest probability is the rigid motion at which the density function of the probability distribution is maximal.

**Lemma 2.3.** *The density function on  $R$  is given by  $f(r) = \frac{|r(A) \cap B|}{2\pi \cdot |A| \cdot |B|}$ .*

Because of this lemma as in case of translations it holds that also in case of rigid motions the output of the algorithm is a rigid motion that approximately maximizes the area of overlap. For matching under rigid motions we can derive similar approximation error bounds as in the case of translations.

### 2.1.5 Similarities

For matching under similarities, our experiment consists of choosing two random points  $a$  and  $a'$  in  $A$  and two random points  $b$  and  $b'$  in  $B$ . Then there is exactly one similarity that maps  $a$  onto  $b$  and  $a'$  onto  $b'$ . We parameterize the space of similarities  $S \subseteq \mathbb{R}^4$  in the following way:  $(s_1, s_2, s_3, s_4) \in S$  denotes the similarity  $\begin{pmatrix} x \\ y \end{pmatrix} \mapsto \begin{pmatrix} s_1 & s_2 \\ -s_2 & s_1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} s_3 \\ s_4 \end{pmatrix}$ , so  $s_1$  is the product of scaling factor and cosine of the anti-clockwise rotation angle,  $s_2$  is the product of scaling factor and sine of the rotation angle and  $\begin{pmatrix} s_3 \\ s_4 \end{pmatrix}$  is the translation vector. Having chosen  $a, a', b, b'$  the scaling factor  $\lambda$  is given by  $\frac{\|b'-b\|}{\|a'-a\|}$ , the cosine of the rotation angle  $\alpha$  is  $\frac{\langle a'-a, b'-b \rangle}{\|a'-a\| \cdot \|b'-b\|}$ , the sine is  $\frac{\det \begin{pmatrix} a'-a \\ b'-b \end{pmatrix}}{\|a'-a\| \cdot \|b'-b\|}$  (where  $(a' - a)$  and  $(b' - b)$  are written as row vectors) and the translation vector is given by  $b - \lambda M_\alpha a$ . The following formula for the density function on  $S$  can be shown:

**Lemma 2.4.** *The density function  $f$  of the probability distribution resulting from the experiment on  $S$  is given by*

$$f(s) = \frac{1}{|A|^2 \cdot |B|^2} \int_{(A \cap s^{-1}(B))^2} \|a' - a\|^2 d(a', a).$$

Intuitively, the density function is large and thus the probability of a small neighborhood is large as well if essential parts of  $A$  have a large distance from each other. The density function is large if  $A$  has a elongated shape in contrast to a circular shape.

Since for all  $a, a' \in C$  the squared distance  $\|a' - a\|^2$  is less than the squared diameter  $\text{diam}(C)^2$  the following holds

**Proposition 2.5.** For shapes  $A, B \subseteq \mathbb{R}^2$  and a similarity  $s \in S$  it holds

$$f(s) \leq \frac{\text{diam}(A \cap s^{-1}(B))^2 \cdot |A \cap s^{-1}(B)|^2}{|A|^2 \cdot |B|^2}.$$

For similarities the algorithm is not symmetric as the example in Figure 2 shows. If  $A$  is a dumbbell and  $B$  is a disc the algorithm returns a similarity that shrinks  $A$  into  $B$  because the density is maximal for this similarity. If  $A$  is a disc and  $B$  is a dumbbell the two similarities with highest density are the translations that put  $A$  on top of one of the discs of the dumbbell.

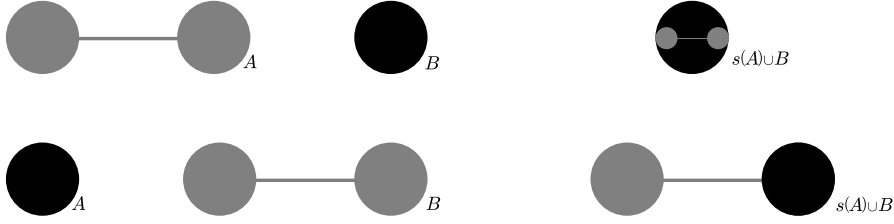


Figure 2: Matching  $A$  with  $B$  under similarities gives a different result than matching  $B$  with  $A$ .

## 2.2 Matching based on Image Primitives

### 2.2.1 Motivation

For the comparison of figurative images represented by sets of (polygonal) curves we developed algorithms (objective 3) that were based on finding similarity transformations that match the two images. This works well for images whose parts lie close together and whose parts do not differ much in perceived significance.

Although one of the laws invented by gestalt theory states that configurations cannot be analyzed into parts and relations [11], for multi-component images the comparison based on the individual image components is more effective than a comparison based on the whole image [8].

With regard to the ground truth provided by professional trademark examiners, some observations can be made which are formulated as follows:

- People look for figures in the image that can easily be memorized. These figures may be abstract figures such as squares, circles, and triangles or figures of everyday life such as letters, digits, and stylized eyes or paperclips. If such figures exist within the image, their concrete proportions and positions play a minor role.  
This is supported by the facts that:
  - a small number of common shape elements can form a basis for humans to discriminate between a wide variety of images [6] (cited in [9]).
  - "there is an unconscious effort to simplify what is perceived into what the viewer can understand". [10] (cited in [4])
- If the image consists of spatially independent parts, the size of the gaps in between plays a minor role.
- If an essential part of the image is framed, the shape of the frame and even the existence of the frame play a minor role. In [13] experiments on the way humans decompose figurative images were made. 5 of the images had a frame, for 3 of them all subjects completely ignored the frame and for 1 image only the second least significant decomposition (out of 9) contained the frame.

- Looking at a figurative image, the number of essential parts that are perceived is typically very small. For example in a regular pattern of little circles, one does normally not discriminate between the different circles, but group them together to a '*pattern of circles*'. Moreover when comparing such patterns it plays only a minor role whether 16 circles form a  $4 \times 4$  grid or whether 25 circles form a  $5 \times 5$  grid.

Our approach for comparison of figurative images is based on a very simple idea: try to characterize a figurative image the same way humans would do. If there is a circle in a triangle, characterize it as '*a circle in a triangle*', if there is something never seen before, characterize it as '*something never seen before*' and describe it by what is known about it — in our case the region as given by the bounding polygonal curve. Many patent offices use such a characterization based on the so called Vienna classification [1]. The

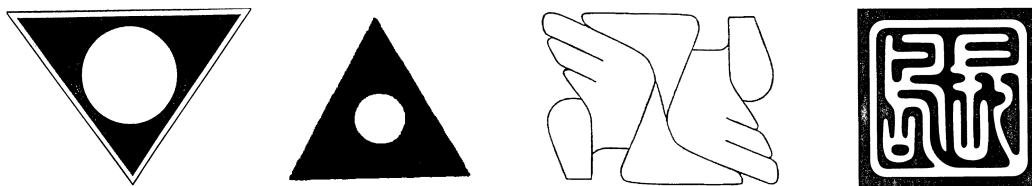


Figure 3: actual trademark images — some easy to describe by geometric primitives and some not.

codes for the examples given in fig. 3 would possibly be '*26.3.10 Triangles containing one or more circles, ellipses or polygons*' and '*26.13.25 Other geometrical figures, indefinable designs*' respectively.

Following this idea in our approach, an image is divided into a set of (not necessarily spatially independent) parts — preferably simple and salient geometric figures. These parts are classified, weighted, and related. The relationships are weighted as well. Comparing two images is accomplished by searching for subsets of the parts and their relations that match well.

The comparison of the parts is done independently, leaving aside their relative sizes and positions. It can be done using a similarity measure that works well for shapes whose parts lie close together whereas the resulting measure can handle arbitrarily composed shapes.

In [14] a similar approach of dividing the images into geometric primitives and finding a match between these primitives is proposed. Its main drawbacks are 1.) that the comparison of the primitives does not discard their concrete positions and 2.) that the similarity between primitives belonging to different categories is defined as being zero, which is contrary to human perception e.g. when comparing a circle and a regular 12-gon.

### 2.2.2 Technique

It is assumed that figurative images are given as a set  $S$  of polygonal regions  $s_1 \dots s_m$ . Based on these polygonal regions a set  $P$  of figures  $p_1 \dots p_n$  is extracted and “relations”  $R = r_{1,2} \dots r_{n,n-1}$  among them are computed.

The process of figure detection is not described in detail here, but the decomposition is assumed to be part of the input.

**Figures** The figures can either be simple geometric objects (*image primitives*) or more complex objects. The primitives considered in our implementation are:

- ellipses (as a generalization of circles)
- rectangles (as a generalization of squares)



- triangles

The parts of the image that cannot be represented by these primitives are categorized as

- convex polygons
- arbitrary sets of polygons and polylines

Analogously to concentric circles, ‘concentric’ ellipses, rectangles, triangles, and convex polygons are combined to a single figure with multiple layers.

**Relations** For a pair  $(p_i, p_j) \in P \times P, i \neq j$  of figures the relation  $r_{i,j}$  consists of numerical values reflecting

- the size of  $p_j$  relative to the size of  $p_i$  (The size of a figure is defined to be the perimeter of the bounding box that maximizes the aspect ratio.)
- the relative distance of  $p_j$  to  $p_i$  (The distance of the centers relative to the size of  $p_i$ .)
- the qualitative relation, i.e., whether  $p_i$  and  $p_j$  are very similar, just mirrored, or just rotated etc.

**Comparison of two images** For the comparison of two images  $I^1$  and  $I^2$  the *relevance*  $w_P$  of the figures and the *relevance*  $w_R$  of the relations is preset such that  $w_P + w_R = 1$  — for images consisting only of one type of figures, e.g., only squares, the relations between these figures are of greater importance than for images consisting of totally different figures. The figures and relations get weights  $w(p_i)$  and  $w(r_{i,j})$  such that for each image all weights sum up to 1, namely:  $\sum_{p \in P} w(p) = w_P$  and  $\sum_{r \in R} w(r) = w_R$ .

For every pair  $(p_i^1, p_k^2) \in P^1 \times P^2$  of figures and every pair  $(r_{i,j}^1, r_{k,l}^2) \in R^1 \times R^2$  of relations a value of similarity  $s \in [0, 1]$  is computed, using simple measures of similarity as described later.

Let  $\mathcal{M}$  be the set of all one-to-one matchings between figures of image  $I^1$  and image  $I^2$ . The value of similarity of the two images is then defined as the weighted sum of the similarities of the matched figures, plus the weighted sum of the similarities of the (implicitly) matched relations:

$$s(I^1, I^2) = \max_{M \in \mathcal{M}} \left\{ \sum_{(p^1, p^2) \in M} s(p^1, p^2) \cdot \frac{w(p^1) + w(p^2)}{2} + \sum_{\substack{(p_i^1, p_k^2) \in M \\ (p_j^1, p_l^2) \in M}} s(r_{i,j}^1, r_{k,l}^2) \cdot \frac{w(r_{i,j}^1) + w(r_{k,l}^2)}{2} \right\}$$

The problem of determining whether  $s(I^1, I^2) \geq \theta$  for a given threshold  $0 < \theta \leq 1$  is an extension of the *quadratic assignment problem* (see e.g. [16]) and therefore is NP-complete<sup>1</sup>. Since the number of essential parts that are perceived is typically very small, the admissible number of figures that represent an image can be bounded by a small constant. Thus, the value of similarity  $s(I^1, I^2)$  may even be computed using methods having a high asymptotic runtime which would not be acceptable in a different context:

1. using a weighted association graph  $G_{1,2} = (V, E)$ .

The vertex set  $V = P^1 \times P^2 \cup R^1 \times R^2$  consists of all possible pairs of figures of  $I^1$  and  $I^2$  and of all pairs of relations of  $I^1$  and  $I^2$ . The weight  $w(v^1, v^2)$  of a vertex  $v = (v^1, v^2)$  is the product of the similarity and the arithmetic mean of the weights of the two figures or relations respectively:

<sup>1</sup>The NP-completeness can easily be seen by reduction of the problem whether a graph contains a Hamiltonian cycle.

$$w(v^1, v^2) = s(v^1, v^2) \cdot 0.5 \cdot [w(v^1) + w(v^2)]$$

Between any two vertices  $v_1, v_2$  an edge is inserted into the graph  $G_{1,2}$ , if and only if choosing the two underlying pairs would lead to a contradiction (e.g. if a figure  $p_a^1$  of  $I^1$  is matched to a figure  $p_b^2$  of  $I^2$  it cannot be matched to a figure  $p_c^2$  at the same time).

The value  $s(I^1, I^2)$  equals the weight of the maximum (weight) independent set of vertices in  $G_{1,2}$  which can be approximated using the greedy algorithm described in [15].

2. using a branch and bound algorithm for enumeration of the promising matches.

**Weights** Every figure  $p_i$  gets an absolute weight  $w_a(p_i)$  which equals the square root of the figure's size (perimeter of the figure's bounding box that maximizes the aspect ratio). Every relation  $r_{i,j}$  gets an absolute weight  $w_a(r_{i,j})$  based on the weights of the figures  $p_i$  and  $p_j$ . Before the comparison the weights are normalized such that  $\sum w(p) = w_P$  and  $\sum w(r) = w_R$ . If we compare two images  $I^1$  and  $I^2$  with different numbers  $n^1, n^2$  of figures, then, of course, we can match at most as many figures as the smaller of two images has. Therefore, only the relations for  $n_{min} = \min(n^1, n^2)$  figures may be selected in the matching. In this case, the weights of the relations of the image consisting of more figures are adjusted, such that the maximum over all  $n_{min}$ -subsets of the figures of the sum of the relation weights between the figures in a subset equals  $w_R$ .

**Frames** For every figure the likeliness of being a frame is rated based on the following propositions:

- frames are convex and symmetric
- frames contain at least one complex figure or two primitive figures
- frames are not too small compared with surrounding frames
- frames are not surrounded by something that is not a frame

Based on this likeliness the weight of a frame figure is decreased by a factor  $\in [1.0, 2.0]$ .

**Repetitions** If a logo contains groups of identical figures, the concrete number of these identical figures plays only a minor role in comparison and some trademark images even contain miscellaneous variants of the actual logo. Therefore the weights of such copies are reduced.

**Underlying measures of similarity** For the underlying measures of similarity between figures or relations respectively, values between 0 and 1 are required so that the resulting value will range from 0 to 1. In [2] such a normalized measure of similarity is described which works respectably well for figurative images whose parts lie close together. The basic idea behind this approach is to find a (similarity) transformation  $t : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  that maps parts of the one figure  $p^1$  into the proximity of corresponding parts of the other figure  $p^2$  and the similarity is rated based on proximity and parallelism of  $t(p^1)$  and  $p^2$ . For the comparison of image primitives (ellipses, rectangles, triangles) the values of similarity may be predefined, for the comparison of primitives with complex figures the values may be precomputed so that only the values for the comparison of complex figures have to be computed online.

The similarity of two relations  $r_{i,j}^1$  and  $r_{k,l}^2$  is computed by a formula based on the difference in relative distances, the difference in relative sizes, and the qualitative relations i.e. whether  $p_i$  and  $p_j$  are very similar, just mirrored, or just rotated etc.

**Dealing with different representations** Whenever such a measure of similarity depends on the way the images are decomposed, there is the risk of underestimating the similarity just because two images get decomposed in different ways (e.g. two triangles forming a square vs. a square plus its diagonal). To diminish that effect, the images are also compared using the simple similarity measure that is used for the figures. The maximum of the two values is taken as the similarity of the images.

## 3 Results

The similarity evaluation is based on a preprocessed description of the shapes in the images. This pre-processing may directly use the shapes as given by the segmentation (as is desired with the high level York-segmentation) or it may rearrange and group the shapes given by the segmentation (as is necessary with the low level FU-segmentation).

### 3.1 Implementation

The Algorithms for the comparison based on image primitives as described in deliverable D4.1 have been implemented in C++ as well as in Java. Input and output is via filesystem. In addition to the similarity based on image primitives the curve based similarity is also computed (it yields better results than region based similarity – see section 3.2).

**Java** The Java archive *Comparator.jar* contains a stand alone application that allows using the algorithms via a command-line interface. It offers the following options:

**segment <inputFile> <outputFile>** For performing a low level segmentation of an image (png/gif) and storing the closed polygonal chains of the shapes.

**prepare-fu <inputFile> <outputFile>** For reading in segmented shapes from an xml-file and generating the image description with rearrangement and grouping of polylines.

**prepare-york <inputFile> <outputFile>** For reading in segmented shapes from an xml-file and generating the image description without rearrangement and grouping of polylines.

**segmentandprepare <inputFile> <outputFile>** For performing a low level segmentation of an image (png/gif) and generating the image description with rearrangement and grouping of polylines. The result of a call

```
segmentandprepare <inputFile> <outputFile> equals the two consecutive calls
segment <inputFile> <intermediateFile> and
prepare-fu <intermediateFile> <outputFile>.
```

**compare <file1> <file2> <outputFile> [<ccm>] [<cpm>] [<ppm>]** For comparing two image descriptions. The optional parameters 'ccm', 'cpm', 'ppm' specify whether a complete-complete-matching (ccm), whether a complete-partial-matching (cpm), and whether a partial-partial-matching is to be computed. They may be true, false or 1, 0 respectively. The default is true true false.

**C++** The C++ library *libshapelib.so* contains the algorithms for comparing shapes and images. Via the class *FigurativeMatcher* it provides direct access to the comparison methods to other applications.

**static void setSeed(int seed)** To have reproducible behavior of the probabilistic parts of the algorithms, the pseudo-random-generator may be seeded.

**static void compare(char\* inputFile1, char\* inputFile2, char\* outputFile, ...)** To compare two images based on their descriptions.

**static bool validate(char\* inputFile, char\* dtd)** To check the validity of an image description file. The input file is checked regarding the validity of the xml format according to the given dtd, and regarding the consistency of the data. This part is separated from the comparison of images to have the possibility of having collections of valid files that can be compared without extra costs of re-checking.

## 3.2 Experiments

To get comparable results, the retrieval performance of the present implementations as well as of preliminary implementations of additional algorithms were tested with the same set of trademark images and the same 24 reference queries that were used to test the ARTISAN system [7].

For a query all images were compared to the query images and ranked according to the computed value of similarity. These rankings were rated based on the following evaluation indicators as defined in [7, 3]: Let  $N$  be the total number of items and  $n$  be the number of relevant items. Let furthermore  $r_i$  be the rank of the  $i^{\text{th}}$ -best-ranked relevant item and  $r_l$  the rank of the least-best-ranked relevant item.

**Normalized Recall  $R_n$**  Value in the range from 0 (worst case) to 1 (perfect retrieval). The normalized recall gives a higher weight to success in retrieving the first few items.

$$R_n = 1 - \frac{\sum_{i=1}^n r_i - \sum_{i=1}^n i}{n(N - n)}$$

**Normalized Precision  $P_n$**  Value in the range from 0 (worst case) to 1 (perfect retrieval). The normalized precision gives equal weight to all retrievals.

$$P_n = 1 - \frac{\sum_{i=1}^n \log(r_i) - \sum_{i=1}^n \log(i)}{\log\left(\frac{N!}{(N-n)! \cdot n!}\right)}$$

**Normalized Last-Place-Ranking  $L_n$**  Value in the range from 0 (worst case) to 1 (perfect retrieval). The normalized last-place-ranking indicates the number of retrieved items a user has to search in order to have reasonable expectation of finding all relevant items.

$$L_n = 1 - \frac{r_l - n}{N - n}$$

**Average Precision  $P_a$**  Value in the range from 0 (worst case) to 1 (perfect retrieval). The average precision is the mean of the precision scores obtained after each relevant item is retrieved.

$$P_a = \frac{1}{n} \sum_{i=1}^n \frac{i}{r_i}$$

**Number of Retrieved Images  $n_{0.01}$**  The number of relevant images ranked within the top 1 percent of the entire collection.

### 3.2.1 Probabilistic Region-based Shape Matching

We used a preliminary version of the implementation for the experiments, which is written in Java. The algorithm works on the black-and-white pixel images. No satisfactory segmentation of the images is available, but fortunately, none is required for this algorithm.

We compared each of the 24 query images to each of the 10745 trademark images. First, a transformation that matches the query image best to the other input image is computed by our algorithm. Second, a certain distance value of the transformed query image and the other image is measured. These distance values determine the ranking of the images. The distance value is chosen as the number of pixels of the transformed query images that differ from the other input image. This value is kind of similar to the symmetric difference of the two shapes, which cannot be computed from the pixel images easily.

**Matching under translations** For reader's convenience, we summarize the algorithm briefly.

**Algorithm 1**

Given: shapes  $A$  and  $B$ , integer  $n$ , positive rational  $\delta$ .

1. Do the following experiment  $n$  times:
  - (a) Choose uniformly distributed a random point  $a$  in  $A$  and a random point  $b$  in  $B$ .
  - (b) Give one vote to the unique translation that maps  $a$  onto  $b$  that is  $b - a$ .
2. Determine and return one of the transformations whose  $\delta$ -neighborhood obtained the most votes.

Our analysis shows that this algorithm computes with high probability a translation that approximately maximizes the area of overlap of the input shapes if  $n$  is large enough.

During the comparison of two images random experiments are performed. The number of experiments was set to 100,000.

From the theoretical point of view matching under translations is not appropriate to solve the general trademark retrieval task because of its obvious flaw that a shape and its scaled copy are not recognized to be the same. The achieved values are:

**Normalized Recall**  $R_n$ : 0.69 on average

**Normalized Precision**  $P_n$ : 0.41 on average

**Normalized Last-Place-Ranking**  $L_n$ : 0.24 on average

**Average Precision**  $P_a$ : 0.22 on average

**Number of Retrieved Images**  $n_{0.01}$ : 95 in total

The detailed results can be seen in table 1.

**Matching under similarities** The number of experiments was set to 1,000,000. Unfortunately, the results were worse than for translations.

**Conclusion** In general, region-based shape matching is not well suited for trademark retrieval because the area of overlap reflects perceived similarity not well enough. There are some issues that cannot be solved with this approach in principle. Two examples are shown below.

**Inverted images** Inverted images are perceived as similar in some cases, but there is no overlap between the shapes if one is put exactly onto the other. So region-based methods fail to detect similarity in these cases.



**Different spacing** Affine transformations are not capable of mapping shapes with different spacing onto each other.



### 3.2.2 Probabilistic Curve-based Shape Matching

The preliminary experiments were performed based on a naive segmentation: Every boundary (not processed so far) between a white and a black pixel was followed to form a closed contour. These closed contours were clustered. Negligibly small clusters were classified as noise and the closed contours were deleted. Clusters with a sufficiently large expanse and a sufficiently large number of contained contours were classified as texture. The closed contours in these clusters were replaced by the boundary of the clusters. The remaining closed contours for which every vertex corresponds to a pixel, were then simplified using the Douglas-Peucker algorithm [5] (cited in [12]).

The achieved values are:

**Normalized Recall  $R_n$ :** 0.93 on average

**Normalized Precision  $P_n$ :** 0.71 on average

**Normalized Last-Place-Ranking  $L_n$ :** 0.68 on average

**Average Precision  $P_a$ :** 0.47 on average

**Number of Retrieved Images  $n_{0.01}$ :** 191 in total

The detailed results can be seen in table 2.

### 3.2.3 Figure-based Shape Matching

The preliminary experiments were performed based on a naive segmentation: Every boundary (not processed so far) between a white and a black pixel was followed to form a closed contour. These closed contours were clustered. Negligibly small clusters were classified as noise and the closed contours were deleted. Clusters with a sufficiently large expanse and a sufficiently large number of contained contours were classified as texture. The closed contours in these clusters were replaced by the boundary of the clusters. The remaining closed contours for which every vertex corresponds to a pixel, were then simplified using the Douglas-Peucker algorithm [5] (cited in [12]).

The achieved values are:

**Normalized Recall  $R_n$ :** 0.96 on average

**Normalized Precision  $P_n$ :** 0.79 on average

**Normalized Last-Place-Ranking  $L_n$ :** 0.78 on average

**Average Precision  $P_a$ :** 0.54 on average

**Number of Retrieved Images  $n_{0.01}$ :** 230 in total

The detailed results can be seen in table 3.

## 4 Deviations from plan

There were no deviations from plan.

## References

- [1] International classification of the figurative elements of marks (vienna classification) fifth edition. WORLD INTELLECTUAL PROPERTY ORGANIZATION, 2002. ISBN 92-805-1054-7. 7
- [2] Helmut Alt, Ludmila Scharf, and Sven Scholz. Profi deliverable 4.3: Algorithms to match sets of curves, 2006. 9
- [3] Chris Buckley and Ellen M. Voorhees. Evaluating evaluation measure stability. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 33–40, Athens, Greece, 2000. 11
- [4] Dempsey Chang, Laurence Dooley, and Juhani E. Tuovinen. Gestalt theory in visual screen design: a new look at an old subject. In *CRPIT '02: Proceedings of the seventh world conference on computers in education: Australian topics*, pages 5–12, Darlinghurst, Australia, 2002. Australian Computer Society, Inc. 6
- [5] D. Douglas and T. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. In *The Canadian Cartographer*, volume 10, pages 112–122, 1973. 13
- [6] Mary C. Dyson, Hilary Box, and Michael Twyman. The perception of symbols on screen and methods of retrieval from database, 1994. 6
- [7] John P. Eakins, Jago M. Boardman, and Margaret E. Graham. Similarity retrieval of trademark images. *IEEE MultiMedia*, 5(2):53–63, 1998. 11
- [8] John P. Eakins, K. Jonathan Riley, and Jonathan D. Edwards. Shape feature matching for trademark image retrieval. In *CIVR*, pages 28–38, 2003. 6
- [9] John P. Eakins, Kevin Shields, and Jago Boardman. ARTISAN – a shape retrieval system based on boundary family indexing. In *Storage and Retrieval for Still Image and Video Databases IV. Proceedings SPIE 2670*, pages 17–28, 1996. 6
- [10] Mercedes M. Fisher and Karen Smith-Gratto. Gestalt theory: A foundation for instructional screen design. *Journal of Educational Technology Systems*, 27(4), 1998-1999. 6
- [11] H. Helson. The fundamental propositions of gestalt psychology. *Psychological Review*, 40(1):13–32, 1933. 6
- [12] John Hershberger and Jack Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. In *Proceedings of the 5th International Symposium on Spatial Data Handling*, volume 1, pages 134–143, Charleston, South Carolina, 1992. 13
- [13] Victoria J. Hodge, Garry Hollier, John P. Eakins, and Jim Austin. Eliciting perceptual ground truth for image segmentation. In *CIVR*, pages 320–329, 2006. 6
- [14] Hui Jiang, Chong-Wah Ngo, and Hung-Khoon Tan. Gestalt-based feature similarity measure in trademark database. *Pattern Recognition*, 39(5):988–1001, 2006. 7
- [15] Akihisa Kako, Takao Ono, Tomio Hirata, and Magnús M. Halldórsson. Approximation algorithms for the weighted independent set problem. In *WG*, pages 341–350, 2005. 9
- [16] Eugene L. Lawler. The quadratic assignment problem. *anagement Science*, 9:586–599, 1963. 8
- [17] Eleanor Rosch, Carolyn B. Mervis, Wayne D. Gray, David M. Johnson, and Penny Boyes-Braem. Basic objects in natural categories. *Cognitive Psychology*, 8(3):382–439, 1976. 2

## A Experimental Results

Table 1: Probabilistic Region-based Matching (Translations) – 24 query images plus values for normalized recall, normalized precision, normalized last place ranking, average precision, and number of relevant images ranked within the top 1 percent of the entire collection

























	query	relevant images	$R_n$	$P_n$	$L_n$	$aP$	$n_{0.01}$
1.		26	0.95	0.71	0.75	0.4	13
2.		16	0.74	0.67	0.06	0.54	10
3.		12	0.58	0.35	0.19	0.25	3
4.		10	0.73	0.73	0.06	0.7	7
5.		10	0.94	0.60	0.80	0.29	4
6.		18	0.53	0.27	0.11	0.08	4
7.		11	0.59	0.28	0.04	0.1	2
8.		20	0.59	0.42	0.01	0.3	6
9.		25	0.83	0.57	0.27	0.26	12
10.		11	0.61	0.23	0.08	0.09	1
11.		10	0.76	0.45	0.26	0.3	3
12.		4	0.95	0.53	0.84	0.03	2
13.		16	0.77	0.35	0.23	0.13	2
14.		6	0.55	0.29	0.22	0.17	1
15.		13	0.72	0.49	0.03	0.39	5
16.		13	0.31	0.14	0.00	0.08	1
17.		17	0.55	0.20	0.03	0.06	1
18.		12	0.52	0.21	0.03	0.08	1
19.		21	0.71	0.28	0.04	0.05	1
20.		8	0.58	0.28	0.04	0.13	1
21.		8	0.72	0.38	0.07	0.13	1
22.		10	0.81	0.33	0.63	0.1	1
23.		23	0.56	0.23	0.06	0.01	2
24.		13	0.98	0.88	0.83	0.68	11
	mean / sum	333	0.69	0.41	0.24	0.22	95
	standard deviation		0.17	0.20	0.29	0.20	



Table 2: Curve-Based Matching – 24 query images plus values for normalized recall, normalized precision, normalized last place ranking, average precision, and number of relevant images ranked within the top 1 percent of the entire collection

















































	query	relevant images	$R_n$	$P_n$	$L_n$	$aP$	$n_{0.01}$
1.		26	0.97	0.84	0.83	0.69	19
2.		16	0.94	0.86	0.52	0.69	13
3.		12	0.92	0.71	0.66	0.56	7
4.		10	0.97	0.83	0.73	0.71	7
5.		10	0.94	0.60	0.83	0.31	4
6.		18	0.95	0.91	0.45	0.85	16
7.		11	0.97	0.68	0.84	0.32	5
8.		20	0.95	0.71	0.57	0.36	9
9.		25	1.00	0.99	1.00	0.96	25
10.		11	0.99	0.68	0.95	0.16	8
11.		10	0.99	0.88	0.95	0.68	9
12.		4	1.00	0.92	0.99	0.77	4
13.		16	0.89	0.48	0.55	0.16	3
14.		6	0.90	0.52	0.82	0.34	2
15.		13	0.94	0.72	0.64	0.46	8
16.		13	0.97	0.78	0.87	0.54	9
17.		17	0.89	0.50	0.57	0.13	4
18.		12	0.96	0.54	0.91	0.11	4
19.		21	0.57	0.22	0.08	0.05	1
20.		8	0.92	0.69	0.43	0.32	4
21.		8	0.95	0.70	0.84	0.51	4
22.		10	0.94	0.63	0.59	0.31	3
23.		23	0.95	0.78	0.15	0.46	13
24.		13	0.94	0.85	0.51	0.77	10
	mean / sum	333	0.93	0.71	0.68	0.47	191
	standard deviation		0.08	0.18	0.25	0.26	

Table 3: Figure-Based Matching – 24 query images plus values for normalized recall, normalized precision, normalized last place ranking, average precision, and number of relevant images ranked within the top 1 percent of the entire collection

query	relevant images	$R_n$	$P_n$	$L_n$	$aP$	$n_{0.01}$
1. 	26	0.98	0.86	0.92	0.67	19
2. 	16	0.99	0.88	0.90	0.62	13
3. 	12	0.97	0.89	0.62	0.81	10
4. 	10	0.93	0.81	0.42	0.70	7
5. 	10	0.99	0.68	0.97	0.32	3
6. 	18	0.94	0.81	0.34	0.55	10
7. 	11	0.97	0.71	0.87	0.43	6
8. 	20	0.98	0.85	0.75	0.56	16
9. 	25	1.00	1.00	1.00	0.98	25
10. 	11	0.93	0.58	0.76	0.16	5
11. 	10	1.00	0.91	0.98	0.73	8
12. 	4	1.00	0.99	1.00	0.95	4
13. 	16	0.96	0.62	0.65	0.13	7
14. 	6	0.95	0.75	0.70	0.43	4
15. 	13	0.99	0.87	0.87	0.66	11
16. 	13	1.00	0.98	0.99	0.90	13
17. 	17	0.92	0.59	0.71	0.27	7
18. 	12	0.97	0.61	0.85	0.13	7
19. 	21	0.68	0.30	0.12	0.05	1
20. 	8	0.97	0.80	0.84	0.52	6
21. 	8	1.00	0.91	0.98	0.70	7
22. 	10	0.99	0.74	0.91	0.34	8
23. 	23	0.99	0.88	0.94	0.57	20
24. 	13	0.95	0.86	0.54	0.77	10
mean / sum	333	0.96	0.79	0.78	0.54	230
standard deviation		0.06	0.16	0.23	0.27	

## B API – Matching

### B.1 FigurativeMatcher Class Reference

```
#include <figurative_matcher.h>
```

#### B.1.1 Detailed Description

This class implements the matching based on image primitives as described in deliverable D4.1 section 2.2.

#### Public Member Functions

- void [setPartial](#) (bool partial)
- void [setImages](#) ([FigurativeImage](#) \*f1, [FigurativeImage](#) \*f2)
- void [clear](#) ()
- void [run](#) ()
- void [run](#) (bool ccm, bool cpm, bool ppm)
- void [getFigureSimilarities](#) (vector< vector< double > > &fSims, vector< [Figure](#) \* > &thisFigures, vector< [Figure](#) \* > &otherFigures)
- void [getRelationSimilarities](#) (vector< vector< double > > &rSims, vector< [Figure](#) \* > &thisFigures, vector< [Figure](#) \* > &otherFigures)
- double [getBestWeight](#) () const
- const [Result](#) & [result](#) () const
- void [writeResults](#) (const char \*fileName=NULL)

#### Static Public Member Functions

- static void [setSeed](#) (int seed)
- static void [compare](#) (char \*inputFile1, char \*inputFile2, char \*outputFile, bool ccm=true, bool cpm=true, bool ppm=false)
- static bool [validate](#) (char \*inputFile, char \*dtd)

#### Protected Types

- enum { [COMPLETE](#), [PARTIAL](#) }

#### Classes

- class [Result](#)

*This class represents a matching result.*

#### B.1.2 Member Function Documentation

##### B.1.2.1 static void [FigurativeMatcher::setSeed](#) (int *seed*) [[inline](#), [static](#)]

To have reproducible behaviour of the probabilistic parts of the algorithms, the pseudo-random-generator may be seeded.

**Parameters:**

*seed* The seed for seeding.

**B.1.2.2 void FigurativeMatcher::compare (char \* *inputFile1*, char \* *inputFile2*, char \* *outputFile*, bool *ccm* = true, bool *cpm* = true, bool *ppm* = false) [static]**

Compares two FigurativeImages.

Input is read from file system, output is written to file system. If any one of the input files do not exist, no output is written. If any one of the input files is corrupted, the behaviour is unspecified.

**Parameters:**

*inputFile1* Pathname of first image description

*inputFile2* Pathname of second image description

*outputFile* Pathname of result

*ccm* Whether complete-complete comparison shall be performed

*cpm* Whether complete-partial comparison shall be performed

*ppm* Whether partial-partial comparison shall be performed

**B.1.2.3 bool FigurativeMatcher::validate (char \* *inputFile*, char \* *dtd*) [static]**

Checks the validity of an input file.

The input file is checked regarding the validity of the xml format according to the given dtd, and regarding the consistency of the data. Input is read from file system, dtd is read from file system. The dtd has to comply with the present program version.

**Parameters:**

*inputFile* Pathname of image description

*dtd* Pathname of dtd

**Returns:**

true if input file is a valid image description, false otherwise

**B.1.2.4 void FigurativeMatcher::setPartial (bool *partial*) [inline]**

Indicates whether partial or complete matching is to be performed.

**Parameters:**

*partial* If set true partial matching is performed.

**B.1.2.5 void FigurativeMatcher::setImages (FigurativeImage \* *f1*, FigurativeImage \* *f2*)**

Assings the images that are to be compared by the algorithm.

Images have to be of type [FigurativeImage](#), that is, they have to be already decomposed into set of primitives, e.g., by using [Figurator](#) class for a PolygonalShape.

**B.1.2.6 void FigurativeMatcher::clear () [inline]**

Resets the matching result if one has already been computed.

**B.1.2.7 void FigurativeMatcher::run ()**

Performs matching based on image primitives for the images assigned by setImages.

Additionally images are matched using probabilistic alignment algorithm. Matching results of both methods are recorded in a [Result](#) object and can be accessed through the [result\(\)](#) function or written to a file by [writeResult\(\)](#) function.

**B.1.2.8 double FigurativeMatcher::getBestWeight () const [inline]**

Returns the best similarity value found by the matching algorithms.

**B.1.2.9 const Result& FigurativeMatcher::result () const [inline]**

Returns [Result](#) object which contains full matching information.

**B.1.2.10 void FigurativeMatcher::writeResults (const char \*fileName = NULL)**

Writes matching results to a file.

The information contains similarity values and matching assignments of the figures of both images for matching based on image primitives, and again similarity value, transformation that leads to the best similarity and the parts matched by that transformation for probabilistic matching.

**Parameters:**

*fileName* Name of the file for writing matching results, if omitted the name is constructed from file names of the images.

**B.2 FigurativeMatcher::Result Class Reference**

```
#include <figurative_matcher.h>
```

**B.2.1 Detailed Description**

This class represents a matching result.

It contains the similarity values computed by figurative matching algorithms and by probabilistic alignment algorithm. Additionally, it stores the following information:

- file names of the image files,
- matching assignment of the figures of two images that leads to the best similarity value for the figurative matching,
- transformation that gives the best matchig results for probabilistic alignment algorithm,
- information about the parts of two images matched by that transformation.

### Public Member Functions

- double [weight](#) () const
- void [print](#) () const
- vector< pair< int, int > > [getMatching](#) () const
- bool [hasInfo](#) () const

### Friends

- class [FigurativeMatcher](#)

## B.2.2 Member Function Documentation

### B.2.2.1 double [FigurativeMatcher::Result::weight](#) () const [inline]

Returns the best similarity value of those computed by figurative matching and probabilistic alignment algorithms.

### B.2.2.2 void [FigurativeMatcher::Result::print](#) () const

Prints matching results to standard output device.

### B.2.2.3 vector<pair<int, int>> [FigurativeMatcher::Result::getMatching](#) () const [inline]

Returns assignments of the figures of two images that gives the best similarity value.

#### Returns:

Vector containing the pairs of integer values in the vector are indices of the figures that are assigned to each other by the best matching.

### B.2.2.4 bool [FigurativeMatcher::Result::hasInfo](#) () const [inline]

Indicates whether the results of matching are computed.

## C API – Image representation as set of primitives

### C.1 FigurativeImage Class Reference

```
#include <figurative_image.h>
```

#### C.1.1 Detailed Description

Image representation as a set of primitives of type [Figure](#), with all intern relations between the primitives, represented by [FigureRelation](#).

### Public Member Functions

- void `clear ()`
- bool `isEmpty () const`
- int `size () const`
- bool `hasFrames (double frameThreshold) const`
- void `getFigures (vector< Figure * > &ret, int start, int end)`
- PolygonalShape \* `createShapeRep (double frameThreshold=1.0)`
- Figure \* `getFigure (unsigned int index)`
- FigureRelation \* `getFigureRelation (unsigned int index)`
- PolygonalShape \* `getOriginalShape () const`
- void `normalizeWeights (unsigned int k)`
- const string & `name () const`

### Static Public Member Functions

- static FigurativeImage \* `readFromXML (char *fileName)`

### Public Attributes

- string `m_name`

## C.1.2 Member Function Documentation

### C.1.2.1 void FigurativeImage::clear () [inline]

Clear all data the object holds.

### C.1.2.2 bool FigurativeImage::isEmpty () const [inline]

Indicates whether the object contains any figures.

### C.1.2.3 int FigurativeImage::size () const [inline]

Returns the number of Figures the image consists of.

### C.1.2.4 bool FigurativeImage::hasFrames (double *frameThreshold*) const

Indicates whether the image contains any figures classified as frames.

#### Parameters:

*frameThreshold* Certainty threshold for classifying frames. Figures that were rated as frames with certainty at least `frameThreshold` will be marked as such.

### C.1.2.5 FigurativeImage \* FigurativeImage::readFromXML (char \* *fileName*) [static]

Read a `FigurativeImage` from an XML-File.

**C.1.2.6 PolygonalShape\* FigurativeImage::createShapeRep (double *frameThreshold* = 1.0)**

Returns a PolygonalShape to be drawn on the screen that represents the outline of the Image.

If a threshold for the likeliness of a figure to be a frame is given, these figures will be excluded in the returned shape.

**C.1.2.7 Figure\* FigurativeImage::getFigure (unsigned int *index*) [inline]**

Returns a pointer to the Figure specified by the index in the vector of Figures.

**C.1.2.8 FigureRelation\* FigurativeImage::getFigureRelation (unsigned int *index*) [inline]**

Returns a pointer to the FigureRelation specified by the index in the vector of FigureRelations.

**C.1.2.9 PolygonalShape\* FigurativeImage::getOriginalShape () const [inline]**

Returns a copy of the original shape.

**C.1.2.10 void FigurativeImage::normalizeWeights (unsigned int *k*)**

Normalizes the figure and relation weights according to the number of figures of the other image.

**Parameters:**

*k* number of figures of the other image

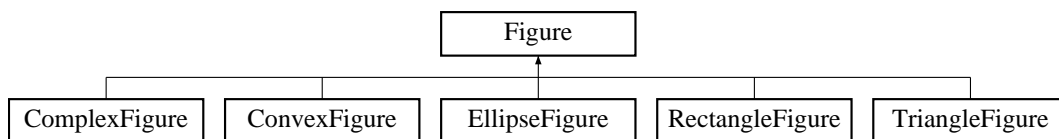
**C.1.2.11 const string& FigurativeImage::name () const [inline]**

Returns the name of xml-file containing the image description.

**C.2 Figure Class Reference**

```
#include <figure.h>
```

Inheritance diagram for Figure::

**C.2.1 Detailed Description**

Abstract class for image primitives, or Figures.

An image (FigurativeImage) consists of a set of Figures.

**Public Types**

- enum `Type` { `ELLIPSE`, `RECTANGLE`, `TRIANGLE`, `PRIMITIVES` }
- enum { `CONVEX` = `PRIMITIVES`, `COMPLEX` }



**Public Member Functions**

- [Figure](#) (unsigned int type)
- unsigned int [type](#) () const
- double [size](#) () const
- double [deformation](#) () const
- double [lowering](#) () const
- double [weight](#) () const
- void [weight](#) (double w)
- double [absoluteWeight](#) () const
- void [normalizeWeight](#) (double factor)
- int [index](#) () const
- const Point & [center](#) () const
- bool [isSymmetric](#) () const
- bool [isRotating](#) () const
- double [frame](#) () const
- double [computeSimilarity](#) ([Figure](#) \*other, unsigned int transfType=Transformation::SIMILARITY)
- [FigureRelation](#) \* [getRelation](#) (int index) const
- virtual double [rate](#) ([Figure](#) \*other)
- void [setRelation](#) (int index, [FigureRelation](#) \*rel)
- unsigned int [getNrLayers](#) () const
- virtual const PolygonalShape \* [getShapeRep](#) () const
- virtual [Figure](#) \* [buildFromXML](#) (const xmlpp::Node \*node)
- bool [selfCheck](#) ()
- virtual [Figure](#) \* [copy](#) ()=0

**Static Public Member Functions**

- static double [getStdRating](#) (const [Figure](#) &f1, const [Figure](#) &f2)
- static [Figure](#) \* [sbuildFromXML](#) (const xmlpp::Node \*node)

**C.2.2 Member Enumeration Documentation****C.2.2.1 enum [Figure::Type](#)**

Primitive figure types.

**C.2.2.2 anonymous enum**

Other figure types.

**C.2.3 Constructor & Destructor Documentation****C.2.3.1 [Figure::Figure](#) (unsigned int *type*) [inline]**

[Figure](#) is initialised with the specified type.

**C.2.4 Member Function Documentation****C.2.4.1 unsigned int [Figure::type](#) () const [inline]**

Returns type of this figure.

**C.2.4.2 double Figure::size () const** [inline]

The size of the figure, which is defined to be the perimeter of enclosing rectangle minimizing the aspect ratio.

**C.2.4.3 double Figure::deformation () const** [inline]

The aspect ratio.

**C.2.4.4 double Figure::lowering () const** [inline]

Factor for lowering the figure's weight because of being a frame or a duplicate.

**C.2.4.5 double Figure::weight () const** [inline]

Returns relative weight within the image if this [Figure](#).

**C.2.4.6 void Figure::weight (double *w*)** [inline]

Assigns the [Figure](#) its relative weight within the image.

**C.2.4.7 double Figure::absoluteWeight () const** [inline]

The weight of the figure based on its size and salience.

**C.2.4.8 void Figure::normalizeWeight (double *factor*)** [inline]

Computes the relative weight as absoluteWeight/factor.

**C.2.4.9 int Figure::index () const** [inline]

Returns index of this [Figure](#) within the image representation.

**C.2.4.10 const Point& Figure::center () const** [inline]

Returns center of the figure which is defined to be the center of the enclosing rectangle minimizing the aspect ratio.

**C.2.4.11 bool Figure::isSymmetric () const** [inline]

Indicates whether this [Figure](#) has a reflection symmetry.

**C.2.4.12 bool Figure::isRotating () const** [inline]

Indicates whether this [Figure](#) has a rotational symmetry.

**C.2.4.13 double Figure::frame () const** [inline]

Returns the likeliness of this [Figure](#) being a frame.

**C.2.4.14** `double Figure::computeSimilarity (Figure * other, unsigned int transfType = Transformation::SIMILARITY)`

Computes and returns the similarity value of this [Figure](#) with the given other [Figure](#).

**Parameters:**

*other* Other figure to be compared to.

*transfType* Specifies the class of transformations one of the Figures is allowed to undergo. If none is specified the type defaults to similarity maps.

**C.2.4.15** `FigureRelation* Figure::getRelation (int index) const [inline]`

Returns a pointer to a [FigureRelation](#) object that describes the inner relation of this [Figure](#) and the one specified by the index.

**Parameters:**

*index* Index of a [Figure](#) within the same image.

**C.2.4.16** `double Figure::rate (Figure * other) [virtual]`

Returns a rating in [0,1] to indicate a similarity between this and the other figure.

Reimplemented in [ComplexFigure](#), and [ConvexFigure](#).

**C.2.4.17** `void Figure::setRelation (int index, FigureRelation * rel) [inline]`

Adds a [FigureRelation](#) to the Figures list of relations.

**Parameters:**

*index* Index of the other [Figure](#).

*rel* Pointer to [FigureRelation](#) object corresponding to the relation between this [Figure](#) and the one indicated by index.

**C.2.4.18** `double Figure::getStdRating (const Figure & f1, const Figure & f2) [static]`

Returns the similarity rating for two given figures if they are both of a 'primitive' type, that is, ELLIPSE, RECTANGLE or TRIANGLE.

Ratings of two primitives are constant and can be held in a static array. Exceptions are Ellipse-Ellipse and Rectangle-Rectangle ratings, which are computed dynamically.

**C.2.4.19** `unsigned int Figure::getNrLayers () const [inline]`

Returns the number of layers the [Figure](#) has.

**C.2.4.20** `virtual const PolygonalShape* Figure::getShapeRep () const [inline, virtual]`

Returns representation of this shape as a polygonal curve or a set of polygonal curves.

**C.2.4.21** `Figure * Figure::buildFromXML (const xmlpp::Node * node)` [static]

Constructs representation of a figure from an xml-notation.

**C.2.4.22** `Figure * Figure::buildFromXML (const xmlpp::Node * node)` [virtual]

Constructs representation of a figure from an xml-notation.

Reimplemented in [ComplexFigure](#), [ConvexFigure](#), [EllipseFigure](#), [RectangleFigure](#), and [TriangleFigure](#).

**C.2.4.23** `bool Figure::selfCheck ()`

Tests the consistency the representation.

**C.2.4.24** `virtual Figure* Figure::copy ()` [pure virtual]

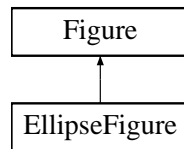
Returns a copy this [Figure](#).

Implemented in [ComplexFigure](#), [ConvexFigure](#), [EllipseFigure](#), [RectangleFigure](#), and [TriangleFigure](#).

**C.3 EllipseFigure Class Reference**

```
#include <ellipse_figure.h>
```

Inheritance diagram for EllipseFigure::

**C.3.1 Detailed Description**

[Figure](#) that represents an ellipse.

**Public Member Functions**

- [EllipseFigure](#) (Point &center, double radius, double ratio=1.0, double angle=0.0)
- [Figure \\* buildFromXML](#) (const xmlpp::Node \*node)
- [Figure \\* copy](#) ()

**Static Public Member Functions**

- static [Figure \\* createStdFigure](#) (double ratio=1.0)

**C.3.2 Constructor & Destructor Documentation****C.3.2.1** `EllipseFigure::EllipseFigure (Point & center, double radius, double ratio = 1.0, double angle = 0.0)`

Create an ellipse with given parameters.

**Parameters:**

- center* the center of the ellipse
- radius* half the main axis
- ratio* aspect ratio
- angle* angle between x-axis and main axis

**C.3.3 Member Function Documentation****C.3.3.1** `Figure * EllipseFigure::createStdFigure (double ratio = 1.0) [static]`

Creates an ellipse with the given axes ratio.

Default axes ratio is 1, which creates a circle.

**C.3.3.2** `Figure * EllipseFigure::buildFromXML (const xmlpp::Node * node) [virtual]`

Constructs representation of a figure from an xml-notation.

Reimplemented from [Figure](#).

**C.3.3.3** `Figure* EllipseFigure::copy () [inline, virtual]`

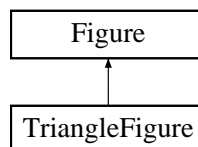
Returns a copy this [Figure](#).

Implements [Figure](#).

**C.4 TriangleFigure Class Reference**

```
#include <triangle_figure.h>
```

Inheritance diagram for TriangleFigure::

**C.4.1 Detailed Description**

[Figure](#) that represents a triangle.

**Public Member Functions**

- [TriangleFigure](#) (const Point &p1, const Point &p2, const Point &p3)
- [Figure \\* buildFromXML](#) (const xmlpp::Node \*node)
- [Figure \\* copy](#) ()

**Static Public Member Functions**

- static [Figure \\* createStdFigure](#) ()

## C.4.2 Constructor & Destructor Documentation

### C.4.2.1 TriangleFigure::TriangleFigure (const Point & *p1*, const Point & *p2*, const Point & *p3*)

Creates a triangle with given corner points.

## C.4.3 Member Function Documentation

### C.4.3.1 Figure \* TriangleFigure::createStdFigure () [static]

Creates an equilateral triangle.

### C.4.3.2 Figure \* TriangleFigure::buildFromXML (const xmlpp::Node \* *node*) [virtual]

Constructs representation of a figure from an xml-notation.

Reimplemented from [Figure](#).

### C.4.3.3 Figure\* TriangleFigure::copy () [inline, virtual]

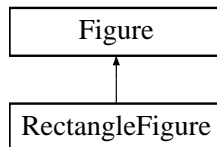
Returns a copy this [Figure](#).

Implements [Figure](#).

## C.5 RectangleFigure Class Reference

```
#include <rectangle_figure.h>
```

Inheritance diagram for RectangleFigure::



### C.5.1 Detailed Description

[Figure](#) that represents a rectangle.

#### Public Member Functions

- [RectangleFigure](#) (Point &center, double radius, double ratio=1.0, double angle=0.0)
- [Figure \\* buildFromXML](#) (const xmlpp::Node \*node)
- [Figure \\* copy](#) ()

#### Static Public Member Functions

- static [Figure \\* createStdFigure](#) (double ratio=1.0)

## C.5.2 Constructor & Destructor Documentation

### C.5.2.1 RectangleFigure::RectangleFigure (Point & *center*, double *radius*, double *ratio* = 1.0, double *angle* = 0.0)

Create a rectangle with given parameters.

#### Parameters:

- center* the center of the rectangle
- radius* half the longer side
- ratio* aspect ratio
- angle* angle between x-axis and longer side

## C.5.3 Member Function Documentation

### C.5.3.1 Figure \* RectangleFigure::createStdFigure (double *ratio* = 1.0) [static]

Creates axes aligned rectangle with a given aspect ratio, centered at origin.

Default aspect ratio is 1, that is a square is created.

### C.5.3.2 Figure \* RectangleFigure::buildFromXML (const xmlpp::Node \* *node*) [virtual]

Constructs representation of a figure from an xml-notation.

Reimplemented from [Figure](#).

### C.5.3.3 Figure\* RectangleFigure::copy () [inline, virtual]

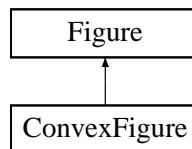
Returns a copy this [Figure](#).

Implements [Figure](#).

## C.6 ConvexFigure Class Reference

```
#include <convex_figure.h>
```

Inheritance diagram for ConvexFigure::



### C.6.1 Detailed Description

[Figure](#) that represents a convex polygon.

**Public Member Functions**

- [ConvexFigure](#) (PolygonalCurve &p)
- double [rate](#) ([Figure](#) \*other)
- [Figure](#) \* [buildFromXML](#) (const xmlpp::Node \*node)
- [Figure](#) \* [copy](#) ()

**C.6.2 Constructor & Destructor Documentation****C.6.2.1 ConvexFigure::ConvexFigure (PolygonalCurve & p) [inline]**

Creates a convex [Figure](#) from a given *convex* polygonal curve.

**C.6.3 Member Function Documentation****C.6.3.1 double ConvexFigure::rate (Figure \* other) [virtual]**

Returns a rating in [0,1] to indicate a similarity between this and the other figure.

Reimplemented from [Figure](#).

**C.6.3.2 Figure \* ConvexFigure::buildFromXML (const xmlpp::Node \* node) [virtual]**

Constructs representation of a figure from an xml-notation.

Reimplemented from [Figure](#).

**C.6.3.3 Figure\* ConvexFigure::copy () [inline, virtual]**

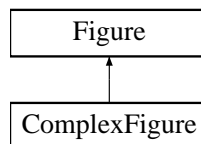
Returns a copy this [Figure](#).

Implements [Figure](#).

**C.7 ComplexFigure Class Reference**

```
#include <complex_figure.h>
```

Inheritance diagram for ComplexFigure::

**C.7.1 Detailed Description**

[Figure](#) that represents a polygonal shape, whichever kind.

**Public Member Functions**

- [ComplexFigure](#) (const PolygonalShape &shape)



- double [rate](#) ([Figure](#) \*other)
- [Figure](#) \* [buildFromXML](#) (const `xmlpp::Node` \*node)
- [Figure](#) \* [copy](#) ()

## C.7.2 Constructor & Destructor Documentation

### C.7.2.1 `ComplexFigure::ComplexFigure (const PolygonalShape & shape)` [inline]

Creates a [Figure](#) object from a given PolygonalShape.

The shape can be a single polygon curve or a set of such curves.

## C.7.3 Member Function Documentation

### C.7.3.1 `double ComplexFigure::rate (Figure * other)` [virtual]

Returns a rating in [0,1] to indicate a similarity between this and the other figure.

Reimplemented from [Figure](#).

### C.7.3.2 `Figure * ComplexFigure::buildFromXML (const xmlpp::Node * node)` [virtual]

Constructs representation of a figure from an xml-notation.

Reimplemented from [Figure](#).

### C.7.3.3 `Figure* ComplexFigure::copy ()` [inline, virtual]

Returns a copy this [Figure](#).

Implements [Figure](#).

## C.8 FigureRelation Class Reference

```
#include <figure_relation.h>
```

### C.8.1 Detailed Description

This class stores information about a relation between two Figures of one image.

A [FigureRelation](#) object contains information about

- the distance between the figures relative to image size,
- whether two figures are (nearly) equal, or whether one of them is a rotated/mirrored copy of the other,
- size ratio of two figures.

Each relation between two figures also has a weight.

### Public Member Functions

- [FigureRelation](#) (int index)
- double [absoluteWeight](#) ()
- double [weight](#) ()
- void [normalizeWeight](#) (double w)
- int [index](#) ()
- double [rate](#) ([FigureRelation](#) \*other)
- [FigureRelation](#) \* [buildFromXML](#) (const xmlpp::Node \*node)
- void [clear](#) ()

### Static Public Member Functions

- static void [relate](#) ([Figure](#) \*f1, [Figure](#) \*f2, int imageSize)

### Static Public Attributes

- static int [unique\\_id](#) = 0

## C.8.2 Constructor & Destructor Documentation

### C.8.2.1 [FigureRelation::FigureRelation](#) (int *index*)

Creates an empty relation with a given index.

## C.8.3 Member Function Documentation

### C.8.3.1 double [FigureRelation::absoluteWeight](#) () [inline]

Weight of the relation based on the weight of the two figures.

### C.8.3.2 double [FigureRelation::weight](#) () [inline]

Returns relative weight if this relation.

### C.8.3.3 void [FigureRelation::normalizeWeight](#) (double *w*) [inline]

Computes the relative weight as `absoluteWeight/w`.

### C.8.3.4 int [FigureRelation::index](#) () [inline]

Returns index of this relation within the image.

### C.8.3.5 double [FigureRelation::rate](#) ([FigureRelation](#) \* *other*)

Returns the value of similarity between this relation and a relation between two figures in another image.

### C.8.3.6 void [FigureRelation::relate](#) ([Figure](#) \* *f1*, [Figure](#) \* *f2*, int *imageSize*) [static]

Computes and stores relation between two given figures of one image.

### C.8.3.7 FigureRelation \* FigureRelation::buildFromXML (const xmlpp::Node \* node)

Creates a [FigureRelation](#) from xml-notation.

## C.9 XMLValidator Class Reference

```
#include <xml_validator.h>
```

### C.9.1 Detailed Description

An object of this class can check the validity of an input file.

The input file is checked regarding the validity of the xml format according to a given dtd, and regarding the consistency of the data.

#### Public Member Functions

- [XMLValidator](#) (char \*dtd=NULL)
- bool [validate](#) (char \*xmlFile, bool check=true)

### C.9.2 Constructor & Destructor Documentation

#### C.9.2.1 XMLValidator::XMLValidator (char \* dtd = NULL)

Creates an object for validating based on a given dtd.

The dtd is read from file system, it has to comply with the present program version.

#### Parameters:

*dtd* Pathname of dtd

### C.9.3 Member Function Documentation

#### C.9.3.1 bool XMLValidator::validate (char \* xmlFile, bool check = true)

Checks the validity of an input file.

The input file is checked regarding the validity of the xml format according to the previously given dtd, and (if wanted) regarding the consistency of the data. Input is read from file system.

#### Parameters:

*inputFile* Pathname of image description

*check* whether consistency of data shall also be checked

#### Returns:

true if input file is a valid image description, false otherwise