Learning graph prototypes for shape recognition

Romain Raveaux, Sébastien Adam, Pierre Héroux, Éric Trupin

Learning Graph Prototypes for Shape Recognition

Romain Raveaux[a], Sébastien Adam[b,*], Pierre Héroux[b], Éric Trupin[b]

[a] *Université de la Rochelle – L3I EA 2128, BP 12, 17042 La Rochelle cedex 01, FRANCE*
[b] *Université de Rouen – LITIS EA 4108, BP 12, 76801 Saint-Etienne du Rouvray, FRANCE*

Abstract

This paper presents some new approaches for computing graph prototypes in the context of the design of a structural nearest prototype classifier. Four kinds of prototypes are investigated and compared : *set median graphs*, *generalized median graphs*, *set discriminative graphs* and *generalized discriminative graphs*. They differ according to (i) the graph space where they are searched for and (ii) the objective function which is used for their computation. The first criterion allows to distinguish *set prototypes* which are selected in the initial graph training set from *generalized prototypes* which are generated in an infinite set of graphs. The second criterion allows to distinguish *median graphs* which minimize the sum of distances to all input graphs of a given class from *discriminative graphs*, which are computed using classification performance as criterion, taking into account the inter-class distribution. For each kind of prototype, the proposed approach allows to identify one or many prototypes per class, in order to manage the trade-off between the classification accuracy and the classification time.

Each graph prototype generation/selection is performed through a genetic algorithm which can be specialized to each case by setting the appropriate encoding scheme, fitness and genetic operators.

An experimental study performed on several graph databases shows the superiority of the generation approach over the selection one. On the other hand, discriminative prototypes outperform the generative ones. Moreover, we show that the classification rates are improved while the number of proto-

*Tel: (+33)2 32 95 52 10 Fax: (+33)2 32 95 52 10
Email addresses: Romain.Raveaux@univ-lr.fr (Romain Raveaux), Sebastien.Adam@univ-rouen.fr (Sébastien Adam), Pierre.Heroux@univ-rouen.fr (Pierre Héroux), Eric.Trupin@univ-rouen.fr (Éric Trupin)

types increases. Finally, we show that discriminative prototypes give better results than the median graph based classifier.

Keywords: Graph classification, graph prototypes, median graphs, discriminative graphs, genetic algorithm, symbol recognition

---

1. Introduction

Labeled graphs are powerful data structures for the representation of complex entities. In a graph-based representation, vertices and their labels describe objects (or part of objects) while labeled edges represent interrelationships between the objects. Due to the inherent genericity of graph-based representations, and thanks to the improvement of computer capacities, structural representations have become more and more popular in many application domains such as computer vision, image understanding, biology, chemistry, text processing or pattern recognition. As a consequence of the emergence of graph-based representations, new computing issues such as graph mining [1, 2], graph clustering [3, 4] or supervised graph classification [5, 6, 7] provoked a growing interest.

This paper deals with the supervised graph classification problem. In the literature, this problem is generally tackled using two kinds of approaches. The first one consists in using kernel based algorithms such as Support Vector Machines (SVM) or Kernel Principal Component Analysis (KPCA) [8, 9, 10, 11, 12, 13]. Using such methods, the graph is embedded in a feature space composed of label sequences which are obtained through a graph traversal. The kernel values are then computed by measuring the similarity between label sequences. Such approaches have proven to achieve high performance but they are computationally expensive when the dataset is large. The second family consists in using a K-Nearest Neighbors (K-NN) rule in a dissimilarity space, using a given dissimilarity measure. This kind of approach is the most frequently chosen for its simplicity to implement and its good asymptotic behavior. However, it suffers from three major drawbacks: its combinatorial complexity, its large storage requirements and its sensitivity to noisy examples. A classical solution to overcome these problems consists in reducing the learning dataset through an object prototype learning procedure and to use a Nearest Prototype Classifier (NPC). Such a prototype-based strategy is not inherent to the graph classification problem. It has already been tackled for comparing shapes in computer vision

application, e.g. in the approach described in [14] that learns some contour prototypes. It has also been studied for a long time in the context of statistical pattern recognition, using either prototype selection methods (see e.g [15, 16]) or prototype generation methods (see e.g. [17, 18]).

In the field of structural pattern recognition, there also has been some recent efforts dedicated to the learning of prototypes. Among them, one can cite the pioneering approach proposed in [19] which builds prototypes by detecting subgraphs that occur in most graphs. Another approach concerning trees is proposed in [20]. It consists in learning some kinds of tree prototypes through the definition of a superstructure called tree-union that captures the information about the tree training set. In the domain of graphs, the approaches proposed in [21] and [22] aim at creating super-graph representations from the available samples. One can also cite the interesting work of Marini proposed in [23] that generates some *creative prototype* by applying to a seed model a well selected set of editing operation. A last approach which is probably the most frequently used concerns median graphs [24, 25, 26, 27, 28]. In a classification context, median graphs are computed independently in each class through a minimization process of the sum of distances to all input graphs. Two kinds of median graphs are proposed in the literature: the set median graphs (smg) and the generalized median graphs (gmg). The only difference between them lies in the space where the medians are searched for. In the first case, the search space is limited to the initial set of graphs (the problem is thus a graph prototype selection problem) whereas in the second case, medians are searched among an infinite set of graphs built using the labels of the initial set (the problem is thus a graph prototype generation problem). Generalized median graphs approaches have proven to keep the most important information in the classes and reject noisy examples [25]. However, a drawback of median graphs when they are used as learning samples of a classification process, as for the all the approaches mentionned before, is that they do not take into account the inter-classes data distribution. In other words, median graphs are rather generative prototypes than discriminative ones.

In this paper, we overcome this drawback by using a discriminative approach while searching an optimal set of prototypes. Thus, it is the classification performance obtained on a validation dataset which is used as criterion in the prototype optimization process. Hence, we propose to use a graph based Genetic Algorithm in order to learn a set of graph prototypes, called discriminative graphs (dg), which minimize the error rate of a classification

system. Two configurations are successively considered for extracting the discriminative graphs. In the first one, a single prototype is generated for each class of the classification problem, as in the case of median graphs. Then, this concept is extended to the extraction of multiple prototypes for each class in order to obtain a better description of the data. This extension is also considered in the case of median graphs in order to provide a suitable comparison. In both configurations, we show that discriminative graphs, and particularly multiple discriminative graphs, enable to obtain very good classification results while considerably reducing the number of dissimilarity computations in the decision stage.

Four datasets are used in the experimental protocol. The first is a huge synthetic dataset. The others are real-world datasets consisting of graphs built from a graphical symbol recognition benchmark [29] for the second and the third and from character recognition for the fourth. The classification performance obtained using discriminative graphs and median graphs are compared on these four datasets.

The paper is organized as follows. In section 2, the most important concepts and notations concerning median graphs and discriminative graphs are defined. In section 3, the proposed approach for graph prototypes extraction is detailed. Section 4 describes the experimental evaluation of the algorithm and discusses results. Finally, section 5 offers some conclusions and suggests directions for future works.

2. Definitions and notations

In this work, the problem which is considered concerns the supervised classification of directed labeled graphs. Such graphs can be defined as follows:

Definition 1. A directed labeled graph G is a *4-tuple* $G = (V, E, \mu, \xi)$ where:

- V is the set of vertices,
- $E \subseteq V \times V$ is the set of edges,
- $\mu : V \rightarrow L_V$ is a function assigning a label to a vertex,
- $\xi : E \rightarrow L_E$ is a function assigning a label to an edge.

A graph classification algorithm aims at assigning a class to an unknown graph using a mapping function f. This function is usually induced from a learning stage which can be defined as follows:

Definition 2. Let χ be the set of the labeled graphs. Given a graph learning dataset $L = \{\langle g_i, c_i \rangle\}_{i=1}^{M}$, where $g_i \in \chi$ is a labeled graph and $c_i \in C$ is the class of the graph among the N classes. The learning of a graph classifier consists in inducing from L a mapping function $f(g) : \chi \rightarrow C$ which assigns a class to an unknown graph.

In this paper, graph classification is tackled with a Nearest Prototype Classifier (NPC), *i.e.* with a NN rule applied on a reduced set of representative graph prototypes. Hence, the learning stage of the classifier consists in generating these prototypes. The objectives are (i) to overcome the well-known disadvantages of a K-NN procedure, *i.e.* the large storage requirements, the large computational effort and the sensitivity to noisy examples and (ii) to keep classification performance as high as possible.

As mentioned before, median graphs are frequently used as representative in a graph classification context. Two kinds of median graphs may be distinguished: the set median graph smg and the generalized median graph gmg. Both are based on the minimization of the sum of distances (SOD) to all input graphs. Formally, they are defined as follows:

Definition 3. Let $d(.,.)$ be a distance or a dissimilarity function that measures the dissimilarity between two graphs. Let $S = \{g_1, g_2, \ldots, g_n\}$ be a set of graphs. The set median graph (smg) of S is defined by:

$$smg = \arg\min_{g \in S} \sum_{i=1}^{n} d(g, g_i) \tag{1}$$

According to this definition, smg necessarily belongs to the set S. This definition has been extended in [25] to the generalized median graph (gmg) which does not necessarily belong to S:

Definition 4. Let $d(.,.)$ be a distance or a dissimilarity function that measures the dissimilarity between two graphs. Let $S = \{g_1, g_2, \ldots, g_n\}$ be a set of graphs. Let U be the infinite set of graphs that can be built using the labels of S. The generalized median graph (gmg) of the subset S is defined by:

$$gmg = \arg\min_{g \in U} \sum_{i=1}^{n} d(g, g_i) \tag{2}$$

Median graphs, generalized or not, have already been used as class representatives in a classification process, e.g. in [25, 26, 27]. In this case, if N is the number of classes in the learning dataset L, N smg (resp. gmg) are computed independently (one for each class) and the resulting graph set constitutes the learning dataset $SMG = \{smg_i\}_{i=1}^{N}$ (resp. $GMG = \{gmg_i\}_{i=1}^{N}$) of the nearest prototype classifier. It has been shown in [25] that generalized median graphs capture the essential information of a given class. However, such prototypes do not take into account the inter-class distribution of learning samples.

In order to overcome this problem, we propose to use discriminative graphs (dg) as prototypes for graph classification. The main difference between median graphs and discriminative graphs lies in the criterion which is used to generate the prototypes. In the case of dg, rather than optimizing a sum of intra-class distances, prototypes are generated in order to minimize the classification error rate obtained on a validation dataset. Obviously, as in the case of median graphs, these prototypes can be computed in the initial set of graphs, leading to set discriminative graphs (sdg), or in the whole set of graphs, leading to generalized discriminative graphs (gdg). As a consequence, the dg for each class are related to each other and can not be expressed independently. The set SDG of sdg_i can be defined as follows:

Definition 5. Let N be the number of classes in the learning dataset L. Let T be a validation dataset and let $\Delta\left(T, \{g_i\}_{i=1}^{N}\right)$ be a function computing the error rate obtained by a 1-NN classifier on T using the graph prototypes $\{g_i\}_{i=1}^{N} \in L$ as learning samples. Then the set SDG composed of the sdg_i of each class is given by:

$$\begin{aligned} SDG &= \{sdg_1, sdg_2, \ldots, sdg_N\} \\ &= \underset{\{g_i\}_{i=1}^{N} \subset L}{\arg\min}\ \Delta(T, \{g_i\}_{i=1}^{N}) \end{aligned} \tag{3}$$

In the same way, the set GDG of gdg is defined as follows:

Definition 6. Let N be the number of classes in the learning dataset L. Let U be the infinite set of graphs that can be built using labels from L. Let T be a validation dataset and let $\Delta\left(T, \{g_i\}_{i=1}^{N}\right)$ be the error rate obtained by a 1-NN classifier on T using the graph prototypes $\{g_i\}_{i=1}^{N} \in U$ as learning samples. Then the set GDG composed of the gdg of each class is given by:

$$GDG = \{gdg_1, gdg_2, \ldots, gdg_N\}$$

$$= \underset{\{g_i\}_{i=1}^{N} \subset U}{\arg\min} \Delta(T, \{g_i\}_{i=1}^{N}) \quad (4)$$

The concepts presented above involve the generation of a single prototype for each class. In some particular applications, it may be interesting to generate m prototypes for each class in order to obtain a better description of the data. In the following, we give the definition of such prototypes called m-gdg[1].

Definition 7. Let N be the number of classes in the learning dataset L. Let U be the infinite set of graphs that can be built using labels from L. Let m be the number of prototypes to be computed in each class. Let T be a validation dataset and let $\Delta\left(T, \{g_{ik}\}_{i=1,k=1}^{N,m}\right)$ be the error rate obtained by a 1-NN classifier[2] on T using the graph prototypes $\{g_{ik}\}_{i=1,k=1}^{N,m} \in U$ as learning samples. Then the set $mGDG$ composed of the m-gdg of each class is given by:

$$\begin{aligned} mGDG &= \{gdg_{11}, ..., gdg_{1m}, ..., gdg_{N1}, ..., gdg_{Nm}\} \\ &= \underset{\{g_{ik}\}_{i=1,k=1}^{N,m} \subset U}{\arg\min} \Delta\left(T, \{g_{ik}\}_{i=1,k=1}^{N,m}\right) \end{aligned} \quad (5)$$

In order to provide some fair comparisons in the experimental protocol, we also extend the median graph concept to multiple prototypes. In this case, the m-gmg (as well the m-smg) are defined independently for each class :

Definition 8. Let $d(.,.)$ be a distance or a dissimilarity function that measures the dissimilarity between two graphs. Let n be the number of samples in the considered class. Let m be the number of prototypes, gp_k be the prototypes and g_i be the graphs of the considered class. Then, the set $mGMG$ composed of the m-gmg for the considered class is given by :

$$\begin{aligned} mGMG &= \{gmg_1, \ldots, gmg_m\} \\ &= \underset{\{gp_k\}_{k=1}^{m} \subset U}{\arg\min} \sum_{i=1}^{n} \min_{k \in \{1,m\}} d(gp_k, g_i) \end{aligned} \quad (6)$$

[1]the definition of m-sdg is easily obtained through the change of the search space from U to S.

[2]In this case, a k-NN procedure with $k > 1$ will be considered in future works, for example to allow the system to reject some patterns

The algorithms involved in the computation of the different kinds of representative prototypes are presented in the following section.

3. Genetic algorithms for Graph Prototypes Generation

In section 2, the graph prototype search problem has been defined as an optimization process. Two kinds of prototypes have been distinguished: (i) set prototypes and (ii) generalized prototypes.

(i) The set prototype search problem consists in selecting the m prototypes per class which optimize an objective function. A combinatorial exploration of the solution space would result in evaluating the criterion for each of the potential solutions. If we consider that each of the N classes contains n_i elements, there are

$$\left(\begin{array}{c} m \\ n_1 \end{array} \right) \times \left(\begin{array}{c} m \\ n_2 \end{array} \right) \times \cdots \times \left(\begin{array}{c} m \\ n_N \end{array} \right) \tag{7}$$

combinations for selecting m prototypes to represent each class. For a quite simple problem with 2 classes and 100 graphs in each class, the search for 5 prototypes per class would result in more than 75×10^6 evaluations of the criterion. Hence, a complete exploration of the solution space rapidly becomes intractable. Many heuristic methods such as multistart, genetic algorithms or tabu search [18] have been used to tackle the problem of set prototype search when dealing with vectorial data. Among them, genetic based methods have shown good performance [30, 18].

(ii) The generalized prototype search problem can also be stated as an optimization problem. However, it cannot be solved with a combinatorial approach since the set U in which the solutions are searched for is unbounded (only a subset S of U is known). In [24], the authors use genetic algorithms to approximate the generalized median graph of a set of graphs. In the context of computing a single generative prototype, they report that the solution reached by a genetic approach is often the optimal solution. In this paper, we also propose to use genetic algorithms but to solve both the set/generalized median/discriminative prototype extraction problem. The next subsections precisely describe our approach.

3.1. Genetic Algorithm

Genetic Algorithms (GA) are evolutionary optimization techniques with a wide scope of applications [31]. They have been used to solve many combinatorial problems [32]. An individual of a GA corresponds to a possible

solution of an optimization problem. The relationship between this individual and the corresponding solution is given by an appropriate encoding. The quality of each individual is evaluated thanks to a score function which enables to quantify the quality of the corresponding solution. In order to converge to the optimal solution, individuals from a size-limited population are randomly selected at each generation according to a fitness value which is computed using the scores of all the individuals of the population. New individuals are then generated from those selected individuals thanks to genetic operators such as crossover or mutation. From a general point of view, the crossover operator aims at promoting the exchange of *good* genetic material between individuals of the previous generation. The mutation operator is used to promote genetic diversity and to explore the solution space. Given these general principles, solving a specific optimization problem using GA requires the definition of :

- an appropriate encoding of the solutions;
- a function which evaluates the score of the individual;
- a selection strategy ;
- some dedicated genetic operators (mutation and crossover operators)

The following paragraphs tackle each of these points for both graph prototype selection and generation, and describe the proposed genetic algorithm.

3.2. Individual encoding

The encoding aims at giving a one-to-one relationship between the individuals manipulated by the GA and the solutions of the optimization problem. As defined before, the prototype selection/generation problem aims at providing m prototypes for each of the N classes. So, we adopt a general scheme where an individual contains $m \times N$ genes, and each gene encode a graph prototype. An example is given in Fig. 1. In this example, the individual encodes 2 prototypes for each class in a 3 classes problem and $g_{i,j}$ is the i^{th} graph prototype describing class j. Obviously, this encoding is specialized for each problem.

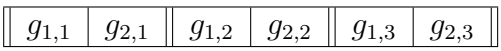

Figure 1: General encoding scheme for the m prototypes problem. Each individual contains $m \times N$ genes. Each one corresponds to a graph prototype.

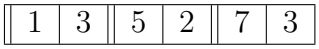

Figure 2: Set prototype encoding scheme for the m prototypes problem. Each individual contains $m \times N$ genes. Each gene is the index of the graph in the considered class of the learning dataset.

3.2.1. Set prototype problem encoding

As stated in section 2, the possible solutions of a set prototype problem are the combinations of m elements selected from each class in the initial graph set. For this kind of problem, an individual can be defined by a list of $N \times m$ integers which is structured as a sequence of N m-sets. Each m-set describes one of the N classes and contains the m indices of the elements from the initial set which are selected as prototype. The exemple in Fig. 2 presents the encoding of an individual for a 3-class problem where 2 prototypes are selected to describe each class. This individual indicates that class 1 is described with elements 1 and 3 of a learning subset composed of the graphs of the first class, that class 2 is described with elements 5 and 2 of the class, and that class 3 is described with graphs the indices of which are 7 and 3 in the third class subset.

3.2.2. Generalized prototype problem encoding

The index model used in the set prototype problem can not be used for the solution encoding of the generalized prototype problem since the definition of generalized (median and discriminative) graphs implies that prototypes may be outside of the initial set of graphs. As a consequence, each gene of an individual can not be a *simple* index and has to encode all the information contained in the corresponding graph. We have chosen to represent each graph with its adjacency matrix. Hence, an individual can be defined by a list of $N \times m$ adjacency matrices, structured as a sequence of N m-sets. Fig. 3 illustrates such an encoding where only one of the 6 genes is represented.

3.3. Fitness function

A fitness function aims at evaluating how the solution encoded by an individual is good for the optimization problem with respect to the entire

Figure 3: Generalized prototype encoding scheme for the m prototypes problem. Each individual contains $m \times N$ genes. Each gene is an adjacency matrix describing the corresponding graph. Only $g_{1,2}$ is represented here. In the adjacency matrix, the digits state for vertex identifiers. a, b, and c are vertices labels, they appear in the last column of the matrix. W, X and Y are edge labels, they appear in the adjacency matrix at the line (resp. column) corresponding to the source (resp. target) vertex.

population. The computation of a fitness value relies on two steps. First, the score of the individual has to be evaluated. It corresponds to the value of the objective function to be optimized. Then, this value is normalized with respect to the scores of all the individuals of the population. As mentioned in section 2, objectives are different for the median prototype problem and for the discriminative prototype problem. As a consequence, score functions differ for each problem.

3.3.1. Score function for median prototypes

As defined in section 2, the score function in the median prototype problem is given by :

$$S_\alpha = \sum_{i=1}^{N} \left(\sum_{j=1}^{n_i} \min_{k \in [1,m]} d\left(L_{ij}, smg_{ik}\right) \right) \tag{8}$$

where N is the number of classes, n_i is the number of elements of class i in the learning dataset, m is the number of prototypes per class, L_{ij} is the j^{th} sample of class i, and smg_{ik} is the k^{th} prototype of class i in the individual α.

3.3.2. Score function for discriminative prototypes

The score value of an individual in the discriminative prototype problem is a function which is directly linked to the error rate of the Nearest Prototype Classifier evaluated on a validation dataset T using the prototypes encoded in the individual. It is given by :

$$S_\alpha = \Delta\left(T, \{g_{ik}\}_{i=1,k=1}^{N,m}\right) \tag{9}$$

where T is the validation dataset, N is the number of classes, m is the number of prototypes per class, g_{ik} is the k^{th} prototype of class i in the individual and $\Delta\left(T, \{g_{ik}\}_{i=1,k=1}^{N,m}\right)$ is the error rate obtained by a 1-NN classifier on T using the graph prototypes $\{g_{ik}\}_{i=1,k=1}^{N,m}$ as learning samples.

The computation of both the Δ value of eq. 9 and the S_α value of eq. 8 makes use of graph distance computation. The following paragraph discusses our choice for this distance definition.

3.3.3. Distance computation

Any kind of distance can be used in the proposed framework (graph edit distance [33, 34] or its approximations [35], distance based on the maximum common subgraph [36], distance based on graph union [37]...). In the experiments proposed in section 4, the graph comparison computation is performed using a dissimilarity measure proposed by Lopresti and Wilfong [38]. This measure is based on graph probing which has been proved to be a lower bound for the reference graph edit distance within a factor of 4.

Let g be a directed attributed graph with edges labeled from a finite set $L_E = \{l_1, \ldots, l_a\}$. A given vertex of g can be represented with its edge structure as a $2a$-tuple of non-negative integers $\{x_1, \ldots, x_a, y_1, \ldots, y_a\}$ such that the vertex has exactly x_i incoming edges labeled l_i and y_j outgoing edges labeled l_j.

In this context, two types of probes are defined in [38]:

- $P_1(g)$: a vector which gathers the counts of vertices sharing the same edge structure for all encountered edge structures ;
- $P_2(g)$: a vector which gathers the number of vertices for each vertex label.

Based on these probes and on the L_1-norm, the graph probing distance between two graphs g_1 and g_2 is given by :

$$\begin{aligned} gpd\left(g_1, g_2\right) &= L_1\left(P_1\left(g_1\right), P_1\left(g_2\right)\right) \\ &\quad + L_1\left(P_2\left(g_1\right), P_2\left(g_2\right)\right) \end{aligned} \tag{10}$$

The graph probing distance respects the non-negativity, symmetry, and triangle inequality properties of a metric, but it does not respect the uniqueness property. In other words, *gpd* is a pseudo-metric and two non-isomorphic graphs can have the same probes.

However, the main advantage of graph probing in this study is its low computational cost (linear function of the vertex number). Due to the intensive use of distance computations during the genetic algorithm, this property makes the graph probing distance a good candidate. Nevertheless, it is important to note that any kind of dissimilarity measure may be used in the proposed framework.

3.3.4. Fitness computation

Once the score value of an individual has been computed, a second step of individual evaluation consists in computing its fitness, through a normalization of the score value with respect to all the individuals of the population. We use the following classical fitness assignment procedure in this scope:

$$F_\alpha = \frac{S_\alpha}{\sum_{i=1}^{\rho} S_i} \tag{11}$$

3.4. Selection strategy

The selection operator aims at selecting a proportion of the existing population to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions (as measured by the fitness function defined in eq. 11) are typically more likely to be selected. We use the well known roulette wheel strategy [31] in which the probability of an individual to be selected is proportional to its fitness value. In the whole reproduction process, an elitism mechanism is coupled with this selection strategy such that the μ best individuals from the previous generation are ensured to be in the next generation.

3.5. Crossover

As mentioned before, the crossover operator is designed to generate offsprings from selected individuals. The exchange of genetic material aims at generating offsprings sharing *good* genes from their parents.

In our case, the crossover is performed by a random exchange of prototypes between the parent for each class. Fig. 4 illustrates the crossover operation. The operation is the same for all the kinds of prototypes. In the case of set prototypes, where the graphs prototypes are designated by indices, only indices are permuted whereas the complete graph descriptions are exchanged when dealing with the generalized prototype problem.

3.6. Mutation

Mutations are used to promote genetic diversity and allow the exploration of regions of the solution space which can not be reached only with crossover. As the solution space is different for set prototype and generalized prototype problems, the mutation operator has to be specialized for each case.

$g_{1,1}$	$g_{2,1}$	$g_{1,2}$	$g_{2,2}$	$g_{1,3}$	$g_{2,3}$
$g'_{1,1}$	$g'_{2,1}$	$g'_{1,2}$	$g'_{2,2}$	$g'_{1,3}$	$g'_{2,3}$

(a) Pair of individuals selected for the crossover operation : the parents

$\mathbf{g'_{1,1}}$	$g_{2,1}$	$\mathbf{g'_{1,2}}$	$\mathbf{g'_{2,2}}$	$g_{1,3}$	$g_{2,3}$
$\mathbf{g_{1,1}}$	$g'_{2,1}$	$\mathbf{g_{1,2}}$	$\mathbf{g_{2,2}}$	$g'_{1,3}$	$g'_{2,3}$

(b) Pair of children generated by the crossover operation.

Figure 4: Illustration of the crossover operator : two selected parents (a) generate two offsprings (b). Genes 1,3 and 4 have been swapped during the operation

1	3	5	2	7	3

(a) individual selected for mutation

1	**4**	**6**	2	7	**5**

(b) individual resulting from the mutation operation

Figure 5: Illustration of the mutation operator for set prototypes : genes 2,3 and 6 have mutated

3.6.1. Mutation for set prototype problem

In the set prototype problem, the solution space is defined by the combinations allowing the selection of m prototypes for each class. An elementary modification of an individual would consist in replacing a prototype by an element from the same class that is not already selected in the individual. Hence, considering the index model used to represent graphs, a simple way to perform a mutation is to arbitrarily substitute an index values by a random integer. Fig. 5 illustrates the mutation process. In this example, we can observe that element 3 has been replaced by element 4 in the mutated version of the description of class 1. In the same way, instance 5 has been replaced by instance 6 in the description of class 2. Finally, the mutated version describes class 3 using the element 5 instead of element 3.

3.6.2. Mutation for the generalized prototype problem

In the generalized prototype problem, the solution space is not restricted to the combinations of elements selected in L. Graphs that are not element of L can be generated as prototypes. As a consequence, the mutation operation can not be restricted to an index modification. It must be able to produce new graphs. To do this, a random edit operation is applied to the graph prototypes that are included in the individual. For each graph of a given

individual, a first random choice according to a mutation probability enables to decide if a mutation is applied or not. Then, one of the six following possible operations illustrated on Fig. 6 is chosen randomly :

- Vertex deletion : delete a randomly chosen vertex and all its connected edges. This operation corresponds to the deletion of a row and a column in the adjacency matrix (see Fig. 6(b)).
- Edge deletion : delete a randomly chosen edge. This operation corresponds to the deletion of an edge value in the adjacency matrix (see Fig. 6(c)).
- Vertex insertion : insert a new vertex in the graph with a randomly chosen label among the vertex label dictionary. This operation corresponds to the addition of a new row and a new column in the adjacency matrix. The label column is also updated using the randomly chosen label (see Fig. 6(d)).
- Edge insertion : insert a new edge between two random vertices with a randomly chosen label among the edge label dictionary. This operation corresponds to the addition of a randomly labeled edge in the adjacency matrix (see Fig. 6(e)).
- Vertex substitution : substitute the label of a randomly chosen vertex using the vertex label dictionary. This operation corresponds to the modification of the label column for the randomly chosen vertex(see Fig. 6(f)).
- Edge substitution : substitute the label of a randomly chosen edge using the edge label dictionary. This operation corresponds to the modification of the label for the randomly chosen edge (see Fig. 6(g)).

3.7. Proposed algorithm

Alg. 1 gives the generic structure of the GA used for the graph prototype generation/selection problems. This algorithm complies with the principles defined in section 3.1 and is specialized by setting the adapted encoding, fitness function and genetic operators presented previously.

First, an initialization procedure aims at building the initial population where each individual corresponds to a possible solution of the optimization

	1	2	3	4	
1		X		Z	a
2			Y		b
3	Y				c
4		W			a

(a) Initial graph

	1	2	3	
1		X		a
2			Y	b
3	Y			c

(b) Vertex deletion

	1	2	3	4	
1		X			a
2			Y		b
3	Y				c
4		W			a

(c) Edge deletion

	1	2	3	4	5	
1		X		Z		a
2			Y			b
3	Y					c
4		W				a
5						c

(d) Vertex insertion

	1	2	3	4	
1		X		Z	a
2			Y		b
3	Y			Z	c
4		W			a

(e) Edge insertion

	1	2	3	4	
1		X		Z	a
2			Y		b
3	Y				c
4		W			b

(f) Vertex substitution

	1	2	3	4	
1		X		Z	a
2			Y		b
3	Z				c
4		W			a

(g) Edge substitution

Figure 6: Illustration of the mutation operators on both generalized graphs and the corresponding adjacency matrices

problem. In the case of set prototypes, distinct indices are randomly chosen for each individual in order to represent the N classes with $N \times m$ graphs. For generalized prototypes, we have chosen to initialize the individuals with randomly chosen graphs from the learning dataset, since it has been shown in [24] that it is a better solution than a complete random procedure.

Then, the GA iterates over the generations, building new size-limited populations from the previous ones. Each new generation is composed of:

- the μ best individuals from the previous one. Such an elitist strategy ensures the convergence of the algorithm.
- mutated or crossed version of individuals that have been selected from the previous generation.

Finally, the algorithm provides the best individual from the last generation as the best solution of the optimization procedure.

4. Experimental results and analysis

This section is devoted to the experimental evaluation of the proposed approach. First, both the datasets and the experimental protocol are described before investigating and discussing the merits of the proposed approach.

4.1. Dataset description

The experiments described in this section have been carried out on four databases. The first one is composed of synthetic data allowing (i) an evaluation in a general context on a huge dataset and (ii) an evaluation with respect to the number of classes. The others sets are domain specific, they are related to pattern recognition issues where graphs are meaningful. Each dataset has been split into three subsets respectively called training subset, validation subset and test subset. The content of each database is summarized in table 1. For each dataset, this table gives : the number of classes (Classes), the total number of data (Samples), the sizes of learning/validation/test datasets and the mean properties of the graphs.

Synthetic dataset: Base A

This dataset contains over 28,000 graphs, roughly identically distributed in 50 classes (about 560 graphs per class). The graphs are directed with edges and vertices labeled from two distinct alphabets. They are built using a modified version of the generic framework used to construct random graphs

Algorithm 1 Genetic algorithm

Require: L: the training set
Require: T: the validation set
Require: m: number of prototypes per class
Require: populationSize
Require: generationNumber
Require: mutationRate
Require: μ: elitism value
Ensure: A set of $N \times m$ prototypes

```
Pop[0][] ← popInit(L,T,m,populationSize) [1]
popEval(Pop[0],L,T)
fitnessEval(Pop[0])
for i = 1 to generationNumber do
    Pop[i][1 : μ] ← μ best individuals in Pop[i − 1]
    j ← μ + 1
    while j ≤ populationSize do
        op ← choice between mutation and crossover [2]
        if op = mutation then
            ind ← select an individual in Pop[i − 1] [3]
            Pop[i][j] ← mutation(ind)
            j ← j + 1
        else
            ind_1 ← select an individual in Pop[i − 1] [3]
            ind_2 ← select an individual in Pop[i − 1] [3]
            (newInd_1, newInd_2) ← crossover(ind_1, ind_2)
            Pop[i][j] ← ind_1
            Pop[i][j + 1] ← ind_2
            j ← j + 2
        end if
    end while
    popEval(Pop[i],L,T)
    fitnessEval(Pop[i])
end for
return the best individual from the last generation
```

[1] T is not used for the initialization in the case of discriminative graphs
[2] This choice is made according to *mutationRate*
[3] Selection is done using a roulette wheel according to fitness values

Table 1: Properties of the four datasets (A,B,C,D) used in the experiments : number of graphs, distribution of the graphs in the learning/validation/test subsets and properties of the graphs in the dataset.

	A	B	C	D
$\mid Classes \mid (N)$	50	10	32	15
$\mid Samples \mid$	28229	200	12800	6750
$\mid Training \mid$	10596	88	7200	3796
$\mid Validation \mid$	14101	56	3200	1688
$\mid Test \mid$	3532	56	2400	1266
$\mid vertices \mid_{mean}$	12.03	5.56	8.84	4.7
$\mid edges \mid_{mean}$	9.86	11.71	10.15	3.6
$\mid degree \mid_{mean}$	1.63	4.21	1.15	1.3

proposed in [39]. Since this framework does not aim at depicting classes, in the sense of similar graphs, we add a second step to the data generation process in order to create classes of graphs. In the initial step a number N (where N is the desired number of classes) of graphs are constructed using the Erdös-Rényi model [39]. This model takes as input the number of vertices of the graph to be generated, and the probability of generating an edge between two vertices. A low probability for edges leads to sparse graphs, that typically occur in proximity based graph representations found in pattern recognition. In the second step, each of the generated graphs are modified using two processes. In a first stage edges and vertices are randomly deleted or relabeled according to a given probability. Then, a second stage of modifications is applied by selecting a vertex from a graph and replacing it with a random subgraph. The whole process leads to graph classes which have an intra-class similarity greater than the inter-class similarity. Numerical details concerning this dataset are presented in table 1. The large size of this dataset is a key point to measure up our approach to the scalability problem.

Symbol recognition related dataset: Base B

This second dataset contains graphs which are generated from a corpus of 200 noisy symbol images, corresponding to 10 ideal models (classes) proposed in a symbol recognition contest [29] (GREC workshop). The class distribution is given in table 2. In a first step, considering the symbol binary image, both black and white connected components are extracted. These connected

Table 2: Class sizes of the dababase B

Class	Samples
1	25
2	13
3	17
4	13
5	20
6	39
7	22
8	17
9	17
10	17

components are then automatically labeled with a partitional clustering algorithm [40] using Zernike moments as features[41]. Using these labeled items, a graph is built. Each connected component correspond to an attributed vertex in this graph. Then, edges are built using the following rule: two vertices are linked with an undirected and unlabeled edge if one of the vertices is a neighbor of the other vertex in the corresponding image. This neighborhood is decided according to the distance between the centroids of each connected components with respect to a predefined threshold (see [42] for more details). An example of the association between two symbol images and the corresponding graphs is illustrated in figure 7. Numerical details concerning this dataset are presented in table 1.

Figure 7: From symbols to graphs through connected component analysis. At the top : a model symbol. At the bottom : a distorded symbol. In both graphs, the vertex a denotes the black connected component whereas the others denote white connected components. In the bottom graph (distorded version), the label e has replaced the label b of the initial

Ferrer dataset: Base C

This third dataset is also related to the symbol recognition problem. It is derived from the GREC database [29]. It is composed of 12,800 graphs identically distributed among 32 classes (examples of symbols are given on figure 8). These graphs are built using a slightly modified version of the approach proposed in [26]. Using Ferrer′s approach, a symbol is represented

Figure 8: Examples of symbols used to build the graphs of the Ferrer dataset [29] - base C.

Figure 9: From symbols to graphs using a 2D mesh. On the left, a vectorized symbol. One the bottom right, the two graphs G_x and G_y obtained using Ferrer's approach. The vertices correspond to the Terminal Points (TPs) and the Junction Points (JPs) of the vectorial representation, labeled with either their x coordinates (on the left) or their y coordinates (on the right). The edges correspond to the segments which connect those points in the image. On the top right, the graphs used to evaluate the proposed approach where the vertices label are obtained through a discretization of $\mathbb{R}^2$.

as an undirected labeled graph which stems from a vectorial representation of the symbol image. In this graph, the vertices correspond to the Terminal Points (TPs) and the Junction Points (JPs) of the vectorial representation and the edges correspond to the segments which connect those points in the image. The information associated to vertices or edges are their cartesian coordinates (x,y). Due to the graph spectral theory limitation, Ferrer′s graphs have to be labeled using real positive or null values and can not handle complex objects. This restriction leads to the construction of two graphs for a single symbol: a graph Gx labeled with x coordinates and a graph Gy with y coordinates, as shown on figure 9. In our case, the chosen graph signature imposes the use of nominal labels. Consequently, a 2-Dimensional mesh is applied to achieve the JP and TP discretisation (see the top right of figure 9). An experimental study which is not presented in this paper has been used in order to choose mesh granularity.

In order to prove the robustness of such a graph representation against noise, 4 different levels of distortion were introduced in [26]. These distortions are generated by moving each TP or JP randomly within a circle of radius r (given as a parameter for each level) centered at original coordinates of the point. If a JP is randomly moved, all the segments connected to it are also moved. With such distortions, gaps in line segments, missing line segments and wrong line segments are not allowed. Moreover, the number of vertices of each symbol is not changed.

Letter database: Base D

This last database consists of graphs representing distorted letter drawings. It is a slightly modified version of the letter dataset proposed in the

IAM graph database repository [43] [3] where LOW, HIGH and MED parts of the dataset have been merged. It considers the 15 capital letters of the Roman alphabet that consists of straight lines only (A, E, F, ...). For each class, a prototype line drawing is manually constructed. To obtain arbitrarily large sample sets of drawings with strong distortions, arbitrarily distortion operators are applied to the prototype line drawings. This results in randomly shifted, removed, and added lines. These drawings are then converted into graphs in a simple manner by representing lines by edges and ending points of lines by vertices. Each vertex is labeled with a two-dimensional attribute giving its position. Since our approach only focuses on nominal attributes, a quantification is performed by the use of a mesh, as in the case of database C. This dataset contains 12800 graphs, identically distributed among the 15 classes. More information concerning those data are given in table 1.

4.2. Experimental protocol

The experiments proposed in this section aim at comparing the classification performance which can be reached using the different graph prototypes defined in section 2. To achieve such a goal, the following protocol has been applied.

First, each dataset has been split into three subsets respectively called training subset (Tr), validation subset (Tv) and test subset (Ts). These subsets are used differently according to the prototypes which are involved.

In the case of using discriminative graphs as prototypes, the training set is used to generate the initial population of the GA, as explained in 3.7. Hence, individuals of the first generation are composed of graphs of Tr. The validation set Tv is involved in the evaluation of the individuals using the 1-NPC classifier during the GA. Finally, the test set is used for evaluating the quality of the best individual (i.e. the best classifier) found at the end of the algorithm. Using such a split, the final performance of the proposed approach is evaluated on a set that has not been considered in the graph prototype learning stage.

In the case of using median graphs as prototypes, the learning process does not involve a classification stage. Consequently, the training and the validation subsets are merged together for medians computation and the test set is used for evaluating the final performance.

[3] Available at `http://www.greyc.ensicaen.fr/iapr-tc15/`

Table 3: Parameters used for the Genetic Algorithm in the proposed experiments

	Acronym	Value
Population Size	ρ	200
Mutation rate	σ	0.3
♯ of generations	G	100
♯ of runs	W	10

Concerning the number m of prototypes to be computed for each class, different values have been tested in the protocol. These values have been chosen with respect to the properties of the dataset.

Furthermore, since GA's are stochastic algorithms, it is necessary to estimate the variability of the results in order to assess the statistical significance of the performance. This was done by running W times the GA and then calculating the conventional couple average and standard deviation $< \overline{Rec}, \sigma >$ at the end of the W runs.

Algorithm 2 gives an overview of the whole protocol. The entire experimental session was performed according to the setting described in Tab. 3, these latter parameters have been chosen experimentaly.

From this stage, our experiments are organized in a five step methodology. First, a study on set median graph computation is carried out to prove the good convergence of the proposed genetic algorithm. Second, an evaluation of the classification performance that can be reached using smg, gmg, sdg and gdg ($m = 1$) as prototypes is performed. Third, we have investigated the influence of m value on the obtained results when multiple prototypes are used for each class. These results are compared to those obtained by a 1-NN classifier trained on the whole learning base ($Tr \cup Tv$), without reduction. Fourth, a closer look is given to the number of classes impact. Finally, the time complexity is benchmarked though different points of view, the prototype nature and the number of classes.

4.3. Algorithm Convergence

In the particular case of computing a single set median graph smg for a given class, the problem is computionally feasible and reachable in $O(N^2)$ where N is the number of elements in the given class. Therefore, it is interesting to compare the set median graphs when they are calculated in a

Algorithm 2 Experimental protocol

Require: Tr: the training dataset
Require: Tv: the validation dataset
Require: Ts: the test dataset
Require: W: the number of runs
Require: $m[m_{max}]$: the m_{max} values of m to be tested [1]
Require: ga_{param} : GA parameters [2]
Ensure: $m_{smg}[m_{max}]$,$\sigma_{smg}[m_{max}]$
Ensure: $m_{gmg}[m_{max}]$,$\sigma_{gmg}[m_{max}]$
Ensure: $m_{sdg}[m_{max}]$,$\sigma_{sdg}[m_{max}]$
Ensure: $m_{gdg}[m_{max}]$,$\sigma_{gdg}[m_{max}]$
  for $j = 1$ to m_{max} **do**
    for $i = 1$ to W **do**
      smg[i][1:j] $\leftarrow$ GA(Tr,Tv,$m[j]$,ga_{param}) [3]
      gmg[i][1:j] $\leftarrow$ GA(Tr,Tv,$m[j]$,ga_{param}) [3]
      sdg[i][1:j] $\leftarrow$ GA(Tr,Tv,$m[j]$,ga_{param}) [3]
      gdg[i][1:j] $\leftarrow$ GA(Tr,Tv,$m[j]$,ga_{param}) [3]
      err_{smg}[i] $\leftarrow$ err1ppv(Ts,smg[i][1:j])
      err_{gmg}[i] $\leftarrow$ err1ppv(Ts,gmg[i][1:j])
      err_{sdg}[i] $\leftarrow$ err1ppv(Ts,sdg[i][1:j])
      err_{gdg}[i] $\leftarrow$ err1ppv(Ts,gdg[i][1:j])
    end for
    $m_{smg}[j] \leftarrow$ mean(err_{smg}[i])
    $\sigma_{smg}[j] \leftarrow$ std(err_{smg}[i])
    $m_{gmg}[j] \leftarrow$ mean(err_{gmg}[i])
    $\sigma_{gmg}[j] \leftarrow$ std(err_{gmg}[i])
    $m_{sdg}[j] \leftarrow$ mean(err_{sdg}[i])
    $\sigma_{sdg}[j] \leftarrow$ std(err_{sdg}[i])
    $m_{gdg}[j] \leftarrow$ mean(err_{gdg}[i])
    $\sigma_{gdg}[j] \leftarrow$ std(err_{gdg}[i])
  end for

[1] m values differ according to the considered dataset
[2] include populationSize, generationNumber, mutationRate and μ
[3] each GA is specialized to the kind of prototypes to be computed

Figure 10: Evolution of the sum of SOD with respect to the generation number obtained using the proposed genetic algorithm for the computation of *smg* (blue curve) and *gmg* (gray curve) on the four datasets. The red line states for the sum of SOD obtained using a combinatorial approach.

Figure 11: Recognition rates obtained using a 1-NN rule applied on Ts and using *gdg*, *sdg*, *gmg* and *smg* as learning prototypes for the four datasets.

computional way and by GA. This test is illustrated in figure 10 which reports the sum of the SOD for all classes when the computation is done (i) in a deterministic way (red line) and (ii) when using GA (blue curve for *smg* and gray curve for *gmg*). Results highlight that our algorithm always reaches the global optimum and moreover that few generations are needed to obtain this good performance. In addition, over the four databases, the lowest SODs are achieved by the generalized median graphs. Such a result shows the capacity of our algorithm to build efficient generalized graphs.

4.4. Classification performance with a single prototype

The first classification experiments which have been performed aim at comparing the performance in graph classification obtained on datasets A, B, C, D using an 1-NPC when choosing a single representative per class. The obtained classification rates are reported in table 4 and illustrated in figure 11. Such results lead to several remarks. First of all, regarding all the databases, results obtained by *gmg* are better than those results obtained by *smg*. This latter observation corroborates the idea that *gmg* have a better modeling behaviour than *smg*. This observation relies on a straightforward explanation, *gmg* belong to a more complete graph space while *smg* are limited to elements constituting the training dababase. Secondly, another remark states the case that the discriminative approaches outperform the generative ones. This statement relies on the comparisons between (*sdg vs smg*) and (*gdg vs gmg*). In both cases, the discriminative graph performance exceed median graph results in a significant way. These important improvements justify to choose *gdg* in order to synthetize a given graph set in a classification context.

Table 4: A single prototype per class, a comparison

%	*smg*		*gmg*		*sdg*		*gdg*	
	$\overline{Rec}$	σ	$\overline{Rec}$	σ	$\overline{Rec}$	σ	$\overline{Rec}$	σ
Base A	33.75	0.0	36.00	1.52	66.10	0.981	66.67	1.59
Base B	62.5	0.0	75	0.0	71.42	2.5	83.39	2.5
Base C	86.92	0.0	85.48	2.05	86.58	0.596	90.70	0.59
Base D	69.61	0.0	69.14	0.34	69.67	0.67	71.24	1.47

Figure 12: Recognition rate evolution according to m for each kind of prototypes and on the four datasets

4.5. Classification performance with regard to the number of prototypes

This second part of experiments aims at investigating the influence of the number m of prototypes on classification results. The results illustrated in figure 12 clearly show that the classification rate is improved when increasing the number of representatives for both median and discriminative graphs. This fact shows that a larger number of prototypes tends to better describe the difficult problems of classification. Also we noticed that the use of a very restricted representative set (i.e. $m = 1$) leads to a lower recognition rate in comparison to the results obtained by a 1-NN classifier trained on the whole learning dataset ($Tr \cup Tv$). However, the time and memory complexities are considerably reduced since there are only N distances to be calculated. Nevertheless, when increasing the number of prototypes, performance match and even outperform the quality of the 1-NN classifier (see table 5) while maintaining the reduction rate quite high. This trade-off to be made between CPU resources and accuracy gives a solution to tackle the scalability problem and consequently to face large data sets taking fast decisions in the classification stage.

4.6. Impact of the number of classes

Thanks to our synthetic graph generator, the number of classes can be tuned to evaluate the algorithm behaviour according to this criterion. In ad-

Table 5: Reduction rate and performance comparisons between *gdg* and a 1-NN classifier using the entire learning set $Tr \cup Tv$. Reduction rate stands for $100 - \frac{m \times N}{|Tr \cup Tv|}$

	BaseA		*BaseB*		*BaseC*		*BaseD*	
	gdg	1-NN	*gdg*	1-NN	*gdg*	1-NN	*gdg*	1-NN
Reduc. rate (%)	92.92	0	50.71	0	86.67	0	76.3	0
Rec (%)	86.34	85.16	97.14	96.43	99.71	99.47	91.04	90.16

Figure 13: Performance comparison between the different kinds of prototypes with respect to the number of classes on different subsets of the database A.

dition, the scalability problem can be addressed reaching a number of classes up to 50. This comparison is presented in figure 13. Implicitly, a higher number of classes will lead to a more complicated issue, in such a way that the recognition rate will be deteriorated. When increasing the number of classes, the gap in term of accuracy between modelizing and discriminative graphs is more important. This difference of accuracy starts from 3.68% in the 5-classes problem to reach 21.3% when the number of classes is 50. The higher is the number of classes, the larger is the gap between modelizing and discriminative graphs. This advantage makes discriminative graphs suitable for difficult classification problems. Independently from the number of classes, it is interesting to report the following statements. This test strengthened our prior observations. The *gmg* better modelizes classes than *smg* and *gdg* outperform all the others prototypes over the four subsets.

4.7. Time complexity analysis

As a matter of fact, learning algorithms are performed off-line. In such a configuration, it seems reasonable to mention that time complexity is not a crucial issue. It is much more significant to be fast at the decision stage. However, a way to compare the computational cost of the different types of prototypes was to undertake an empirical study. The algorithm complexity is directly linked to the number of classes, the influence of the dataset size is depicted by the figure 14. A first comment illustrates the strong impact of the class number on the computational cost when producing a discriminative graph. Moreover a comparison of the runtime execution according to the kind of prototypes on the largest database has been led. The complexity

Figure 14: Run-time evolution with respect to the number of classes on different subsets of the database A.

of the median graph search came out from this test. The SOD criterion is less demanding in term of distance computation, therefore, it is less time consuming. At worst case, in our experiments on the largest database, the median graph computation was 15 times faster. However, this over-load does not discourage the use of discriminative graphs since the gain they imply is really significant. It is a commonplace in machine learning to state the case that training algorithms require much time and many computations to assimilate the data variability.

5. Conclusion and future works

This paper has presented several approaches for the construction of prototype-based structural classifiers. These approaches have been experimentally compared according to several criteria on both synthetic and real databases.

The experimental results first confirm that the generalized median graph approximated using a genetic algorithm has a better modeling ability than the set median graph. Moreover, the results show that prototypes which take into account the whole classification problem (discriminative approach) offer better results than the class centered median graph approach.

Furthermore, the proposed GA framework allows to synthetize m graph prototypes per class. The experimental results illustrate that, when m increases, the classification problem is better described and the performance improves and converges rapidly towards the classification rate of a 1-NN classifier applied on the whole learning dataset.

Finally, the assessments carried out on four datasets expressed that gdg and $m - gdg$ obtain better or comparable results, in terms of accuracy, than the state-of-the-art prototypes schemes for structural data on multi-class graph classification problem. Our contribution gives the proof for the following key points: (i) genetic algorithms are well suited to deal with graph structures and (ii) the recognition rate on a validation dataset is a better criterion of the optimization process than a classical SOD in a classification context. Also, the scalability to large graph datasets has been assessed on a synthetic database with success. This observation illustrates that a prototype-based classifier is well suited to manage masses of structural data.

Short-term, we intend to investigate the ability of setting a different number of prototypes for each class. This strategy would allow to distribute a global number of prototypes among the classes and then to automatically fit the difficulty of the classification problem. This modification impacts on the algorithm and requires a redefinition of the genetic algorithm (problem coding and genetic operators).

We also intend to investigate the ability to propose several prototype sets for different values for m. These sets would correspond to different trade-offs between the concurrent objectives that are the recognition rate and the reduction of the training set which allows to reduce the classification time and spatial complexity. A multi-objective procedure [44] could be used to optimize these non commensurable criterions. Finally, a human operator would *a posteriori* make the final decision according to the use case.

Finally, the reject of elements which do not belong to any known class is a feature which is often required when classifiers are faced with actual data. When dealing with Nearest Neighbor rule, it is generally implemented throught the definition of threshold values. In the same time, the reject of an element is often prefered to a misclassification. This kind of feature can be undertaken with k nearest neighbors rules with values of k greater than 1. Future works should be dedicated to include reject consideration as an additional criterion to be optimized while maintaining the classification rate as high as possible. In this case again, a multi-objective procedure could be useful.

References