Master in Artificial Intelligence (UPC-URV-UB)

Master of Science Thesis

# GRAPH INDEXING AND RETRIEVAL BASED ON GRAPH PROTOTYPES

Xavier Cortés Llosa

Advisor/s: Dr. Francesc Serratosa Casenelles

22/06/2012

# Agraïments

Desenvolupar una tesi com la present, representa moltes hores d'esforç, sacrifici i preocupacions, moltes hores al laboratori, davant la pantalla de l'ordinador o davant la muntanya de papers aparentment desorganitzats que formen els articles que van ocupant poc a poc teva taula de treball. Cada dia és un repte i no hi ha camins marcats, sorgeixen noves complicacions, problemes i imprevistos que has d'anar resolent i més d'una vegada et poden arribar a treure hores de son.

Però tot això és fa molt més fàcil de suportar, si la gent que t'envolta, el teu grup de recerca i tu acabeu formant un bon equip, si la família, els amics i tota la gent que t'envolta saben entendre't quan estàs més preocupat, i això es el que he de dir que m'ha passat. No podia començar aquesta tesi de cap altra manera que no fos mencionant la gent que l'ha fet possible, perquè m'ha aconsellat, guiat i ajudat sempre que em feia falta o bé perquè m'ha fet costat en tot moment.

Voldria començar agraint a en Francesc Serratosa, tutor d'aquesta tesi i cap del grup de recerca on he estat treballant tot aquest temps, per tenir sempre noves idees a desenvolupar i fer correccions oportunes a la feina feta, a l'Albert Solé, company de laboratori i amb qui més hores he compartit, per la gran quantitat de dubtes que m'ha resolt quan estava encallat en alguna cosa i pels seus algoritmes que han servit de base per a la meva feina i per les hores que hem compartit, a la resta de companys del laboratori, en Raül i el Pau, que tot i que no hem coincidit gaires hores, sempre ha estat agradable tenir-los com a companys.

No voldria deixar-me a la meva família, sobretot els meus pares i la meva germana, per la paciència que han tingut en tot moment.

I finalment els amics, per ajudar-me a relaxar i pensar en altres coses quant no estava ocupat amb la tesi.

Voldria acabar aquestes línies, llençant una petita reflexió sobre aquests moments de crisi i incertesa tan profunds com els que estem vivint, és una bona oportunitat per a què no oblidem que la ciència i la recerca és el motor del progrés de la societat, no únicament tecnològic, sinó també econòmic, mèdic... Que ens ajuda a construir les bases d'un futur millor i que l'estalvi en recerca i educació a curt termini sempre acaba resultant més costós a la llarga. Per això penso que des de la universitat, des de la recerca i des de qualsevol espai on puguem, hem de fer una crida al conjunt de la societat, per a què la ciència i l'educació no siguin unes víctimes més d'una crisi de la qual no en són responsables.

# Resum

El trobar-nos davant una gran quantitat de dades i tenir que fer cerques d'aquestes el més ràpid possible és un problema recurrent en el camp de les ciències de la computació pràcticament des dels seus orígens. A l'existència d'aquest problema, se li ha d'afegir, a més a més, el fet de que actualment les bases de dades emmagatzemen tipus de dades de la naturalesa més diversa i molts cops inesperada possible. Ja no parlem de les bases de dades originaries que únicament contenien números o cadenes caràcters.

Estructures complexes que representen dades multimèdia, informació biomètrica, dades científiques.... Les podem podem trobar avui en dia en molts servidors de bases de dades d'arreu del món. Inclús són utilitzades per aplicacions amb les que ens hem habituat a conviure de manera quotidiana sense pràcticament adonar-nos-en, aplicacions de reconeixement facial o que ens ajuden a trobar la imatge que estem veient o la cançó que estem escoltant i que llencen cerques amb uns temps de resposta fins fa pocs anys impensables per la seva rapidesa.

És cert que l'arquitectura del hardware amb que treballem cada dia és més avançada i amb una capacitat de càlcul major, disminuint enormement el temps de de resposta, però no és menys cert, que no estem parlant d'un problema trivial que sigui abordable únicament amb força bruta a partir dels avenços tecnològics que es produeixin el els dispositius físics, segurament que tan importants han estat i seran els progressos en el terreny de l'algorítmia i la intel·ligència artificial que hi ha al darrere de l'emmagatzemament i la recuperació d'aquestes dades.

El que he volgut en aquest treball i penso que en la mesura del que era possible s'ha aconseguit, és desenvolupar i presentar una metodologia per portar a terme aquest procés. Els Metric Trees de prototips, que es basen en la ja coneguda estratègia d'agrupar les dades que anem guardant a una base de dades de la forma més intel·ligent possible per no haver d'explorar totes les instàncies que tenim quan volem fer una cerca, però a més a més s'ha afegit el concepte de prototip. Una estructura, que agrupa la informació d'un conjunt d'instàncies, utilitzada fins ara per a fer classificació i reconeixement.

Conjugant aquests dos conceptes, el de Metric Tree i el de prototip, plantejem la construcció d'arbres de cerca on els prototips siguin els nodes intermedis, que ens ajudin a decidir quin camí explorar quan volem fer una cerca sobre l'arbre. I utilitzant, aquests tant per a fer classificació com per a buscar totes les instàncies que estiguin una distància  més petita d'una distància màxima. Tot això tenint present, que les

dades amb que treballem són grafs, és a dir que la metodologia presentada, té la versatilitat de poder-se aplicar, a qualsevol tipus d'informació que es pugui representar d'aquesta manera.

I com que a la literatura hi ha definides diferents maneres de construir un prototip estructurat a partir d'un conjunt d'instàncies graf, en el treball que aquí presento no m'he limitat únicament a escollir-ne una i implementar-la, sinó que s'ha fet una avaluació sobre el rendiment i l'eficiència que presenta aquest model amb els prototips més importants.

De manera que el treball, es pot dividir en dos grans blocs, un on es fa un anàlisi sobre la funcionalitat dels prototips per fer classificació i l'altre on s'aplica aquest concepte al camp de les bases de dades i els Metric Trees.

# Abstract

Taking a query from a high number of data stored into a database, as fast as possible, is a recurrent problem in the field of computer sciences practically since its origins. At the existence of this problem, it's necessary to add, moreover, the fact that actually databases contains data types of more diverse and unexpected character possible. Now we are not talking about originating databases which only contained sets of numbers or characters strings.

Currently, there are many database servers with complex structures that represent multimedia data, biometric information, scientific data... Even are used by applications that we have learned to live almost without realizing, facial recognition applications, images and music recognition and with a response time of searches unthinkable until recently for its speed.

It's really true that the hardware is becoming more advanced and with a major computing capacity every day, reducing the computation time, but it's also true, that we are not talking about a problem that only can be focused by brute force from technological advances produced into physical dispositives, probably as important have been the advances into algorithmics and artificial intelligence ground under the storing and data retrieval information into databases.

All that I want to make into the present work and I think that was achieved as far as possible, has been to develop and to present a methodology to carry out this process. The Metric Trees of prototypes are based on a well-known strategy, which is based on grouping the data stored in database at the smartest possible way. But also we has added the concept of a graph prototype. A structure that contains information of a set of instances represented by graphs, used until now for classification and recognition.

In this thesis we have used graphs as representatives of elements that have to be queried in databases. Note that graphs have the capacity to represent complex objects, for this reason the number of graph databases is increasing. Due to in the literature appears different ways to build a prototype, the work presented here shows a comparative study between the main methods.

Combining these two concepts, the Metric Tree and the graph prototype, we propose the construction of metric trees where the graph prototypes are routing nodes to help to decide the way to explore when we make a search in the tree. We have used Metric Trees to make classification and to find all instances that are lower than a maximum distance.

So the work can be divided into two blocks, where the first one is an analysis of the functionality of graph prototypes for classification and where the second one this concept is applied to the field of databases and Metric Trees.

# Contents

# 1. Introduction

Indexing structures are fundamental tools in database technology; they are used to obtain efficient access to large collections of images. Traditional database systems manage global properties of images, such as histograms [1]. Many techniques for indexing one-dimensional data sets have been defined. Since a total order function over a particular attribute domain always exists, this ordering can be used to partition the data and moreover it can be exploited to efficiently support queries. Several multi-dimensional indexes have appeared, such as, colour, texture, shape, with the aim of increasing the efficiency in executing queries on sets of objects characterized by multi-dimensional features. Once again, ordering systems of individual orthogonal dimensions are used for partitioning the search space, so these methods can, in fact, be considered as direct extensions of the one-dimensional case.

Effective access to image databases requires queries addressing the expected appearance of searched images [2] [3]. To this end, it is needed to represent the image as a set of entities and relations between them. The effectiveness of retrieval may be improved by registering images as structural elements rather than global features [4]. In the most practiced approach to content-based image retrieval, the visual appearance of each spatial entity is represented independently by a vector of features. Mutual relationships between entities can be taken into account in this retrieval process through a cascade filter, which evaluates the similarity in the arrangement of entities after these have been retrieved on the basis of their individual features [5]. To overcome these systems, local entities and mutual relationships have to be considered to have the same relevance and to be defined as parts of a global structure that captures mutual dependencies [6]. In this case, the model of content takes the structure of an Attributed Graph. The attributes of the vertices of the Attributed Graphs represent the features of the local entities and the attributes of the arcs of the graphs represent the features of the relationships.

While the distance between two sets of independent features can be computed in polynomial time, the exact distance between two graphs is computed in exponential time respect the number of nodes of the graphs. Some sub-optimal solutions have been presented to compare a pair of graphs, then, the computational complexity is reduced to polynomial cost [7] [8]. For this reason, few contributions of practical interest have been proposed supporting the application of graphs to content-based retrieval from image databases [9] and [10].

Out of the specific context of content-based image retrieval, the problem of comparing an input graph against a large number of model graphs has been addressed in several approaches. In some applications, the classes of objects are represented explicitly by a set of graphs, which means that a huge amount of model graphs must be matched with the input graph and so the conventional error-tolerant graph matching algorithms must be applied to each model-input pair sequentially. As a consequence, the total computational cost is linearly dependent on the number of model graphs and exponential (or polynomial in suboptimal methods) with the size of the graphs. For applications dealing with large databases, this may be prohibitive. To alleviate these problems, some attempts have been designed with the aim of reducing the computational time of matching the unknown input patterns to the whole set of models from the database. Those approaches assume that the graphs that represent a cluster or class are not completely dissimilar in the database and, in this way, only one structural model is defined from the graphs that represent the cluster. These structures are called *Graph Class Prototypes*. In the classification process, only one comparison is needed for each cluster.

In this paper, we show an indexing scheme, modelled by an m-tree, in which the cluster knowledge embedded in each node of the m-tree is represented by one of the six Graph Class Prototypes presented in the literature. The different representations of Graph Class Prototypes are: 1) Set Median Graph; 2) Generalise Median Graph [11] [12] [13] [14] synthesised through a hierarchical method [15], synthesised through a genetic algorithm [16] or synthesised through an extension of the Graduated Assignment algorithm [17]; 3) First-Order Random Graphs [18]; 4) Function-Described Graphs [19] [20]; 5) Second-Order Random Graphs [21]; 6) Closure Graphs [22]. Moreover, we show two different graph queries; the ones that the user imposes the number of graphs to be queried and the ones that the user imposes the maximum distance between the query graph and the returned graphs. It is not the aim of this paper to explain the structural representation of each graph prototype.

The rest of the paper is organised as follows. In the next section, we introduce the basic definitions of Database Indexing. In chapter 3, we comment the few approaches that have been presented for indexing Attributed Graphs. In section 4, we comment the methods used to synthesise the graph prototypes. In section 5, we experimentally evaluate the graph prototypes as routing elements of m-trees. We finish the paper drawing some conclusions.

From last decade, structural methods for pattern recognition have been deprecated in favor of embedding methods, this embedding methods [23], have become more common into the literature. One of the purposes of this work has been to try to recover the major structured methods and demonstrate that they can be competitive to classification and integration with m-trees.

Finally, note that all work presented here uses data structures at any time based on graphs, given its potential, graphs are a very flexible representation of data capable of representing a large sort of problems related to pattern recognition. Examples could be found in image databases, video analysis, biomedical and biological applications and so on. In some of these applications, it is usual the need of finding a structure that represents a set of graphs. This structure is used as a representative of the set. The first step to generate this structure is to find a Common Labeling between the vertices of all the graphs such that a general cost is minimized. It is crucial to find a good common labeling to generate a good representative.

# 2. Attributed Graph and Distance between Attributed Graphs

## 2.1 Attributed Graphs.

Given *($\Delta_v$ and $\Delta_e$)* is defined by a tuple $G = (\Sigma_v, \Sigma_e, \gamma_v, \gamma_e)$, where $\Sigma_v = \{v_k \mid k = 1, \ldots, R\}$ is the set of vertices, $\Sigma_e = \{e_{ij} \mid i, j \in \{1, \ldots, R\}, i \neq j\}$ is the set of arcs and $\gamma_v : \Sigma_v \rightarrow \Delta_v$and $\gamma_e : \Sigma_e \rightarrow \Delta_e$ assign attribute values to vertices and arcs respectively. In case it is required, any Attributed Graph can be extended with null nodes arc arcs, which have special attribute $\emptyset \in \Delta_v$ and $\emptyset \in \Delta_e$.

## 2.2 Distance between Attributed Graphs.

Let $f^{pq}$ be the isomorphism $f^{q,p} : \sum_v^p \rightarrow \sum_v^q$ that assigns each vertex from $G^p$ to a vertex of $G^q$. The cost of this isomorphism, $C^G$ ($G^p$, $G^q$, $f^{p,q}$) is a function that represents how similar are the Ags and how correct is the isomorphism. We consider this cost to be:

$$\boldsymbol{C^G\ (G^p, G^q, f^{p,q}) = \sum_{a=1}^{R} \sum_{b=1}^{R} \sum_{i=1}^{R} \sum_{j=1}^{R} F^{p,q}[a,i] \cdot F^{p,q}[b,j] \cdot C_{ai,bj}^{p,q}} \qquad \textbf{Eq. 1}$$

Where $F^{p,q}[a,i]$ is a permutation matrix wich values are 1 if $f^{pq}(v_a^p) = v_i^q$ and $c_{ai,bj}^{p,q}$ represents the cost of matching nodes $v_a^p$ to $v_i^q$ and $v_b^p$ to $v_j^q$ plus the cost of matching the corresponding edge $e_{ab}^p$ to $e_{ij}^q$.

Usually, $C^G = 0$ represents that both Ags are identical and that the isomorphism miniom cost of all possible isomorphism $f^{p,q}$. That is, $D(G^p, G^q) = min_{f^{p,q} \in T}\, C^G(G^p, G^q, f^{p,q})$. We say that the isomorphism $f^{p,q}$ is optimal if it is the one used to compute the distance.

.

# 3. Graph Class Prototypes

This section presents the most important models used to represent a set of Attributed Graphs. The first three models are non-probabilistic and the last three models are probabilistic. In each model, we briefly comment how to obtain the prototype given a computed Common Labelling and a set of Attributed Graphs.

The Common Labelling is a function that assigns all the graph nodes of a set of Attributed Graphs to a Virtual Structure that represents the different nodes. That is, nodes in the Attributed Graphs that represent the same local part of the object or image have to be represented by only one node in the virtual structure.

## 3.1 Common Labelling

Given a set of graphs $\Gamma = \{G^1, G^2, \ldots, G^N\}$, we define a common labelling $\psi = \{\, h^1, h^2, \ldots\, , h^n\}$ over the graphs in $\Gamma$ as a set bijective mappings from nodes of graphs in $\Gamma$ to a virtual vertex set $L_v$ , $h^i = \Sigma_v^i \rightarrow L_v$ .

## 3.2 Common Labeling Algorithms

In this section, we present two different theoretical approaches to the CL problem.

### 3.2.1 Common Labeling based on Graduated  Assignment Algorithm

Common Labeling based on Graduated Assignment have been based on two steps. In the first step, they compute all the labeling between the graphs and then, in the second step, they obtain a Common Labeling. With regard to these methodologies, the first method published was presented in [24]. The method applies the Graduated Assignment algorithm to compute all possible pairwise labeling between the given set of Attributed Graphs. Next, using certain consistency rules, they generate what they refer to as a Super-graph. Once the Super-graph is obtained, the Common Labeling is trivially deduced from it. Another method of this type was published in [25] . Presenting two extensions of the Graduated Assignment algorithm. In the first extension, the Graduated Assignment probability matrix is extended to a hyper-cube that each cell represents the probability of the labeling between all the graphs. In the second extension, authors present a method whereby using all the pairwise labeling between graphs, they compute an approximation of the probabilistic hyper-cube. The main drawback of these methods is that they rely on pairwise labeling that could induce poor sub-optimal solutions.

In the second methods, the global information of the whole set of graphs is taken into consideration from the first steps to built directly the common labeling. It has the advantage that, despite obtaining a sub-optimal solution, the final solutions tends to be nearer to the optimal one than the first type of methods. The three main methods are based on a genetic algorithm, an extension of the graduated assignment algorithm and a probabilistic framework [26].

### 3.2.2 Genetic Common Labeling

Genetic search techniques are motivated by concepts from the theory of biological evolution. They are general-purpose optimization methods that have been successfully applied to difficult tasks in search, optimization, machine learning, etc. In particular, it has been shown that genetic algorithms are useful for finding approximate solutions to general NP-complete problems [27]. Applications of genetic algorithms have also been reported to solve graph matching problems [28], [29].

Basic to genetic search is the idea of maintaining a population of chromosomes representing possible solutions to the discrete optimization problem at hand [30]. A cost function, termed the **fitness**, is associated with the solution candidates. Given an initial population, genetic algorithms use genetic operators to alter chromosomes in the population and create a new generation.

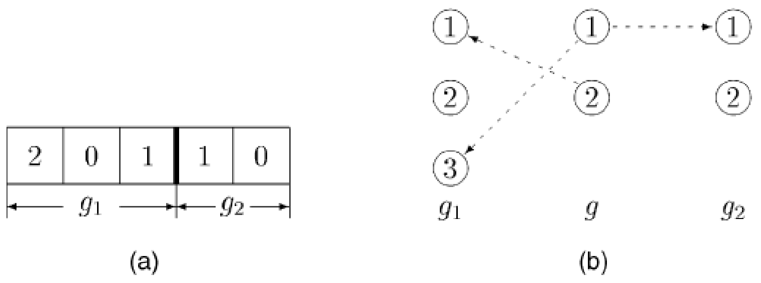

**Fig. 1**: A candidate generalized median graph G represented by (a) a chromosome and (b) its graphical illustration.

The genetic operator **crossover** involves selecting pairs of solution candidates randomly from the current population and interchanging the solutions at selected configuration sites. **Mutation** aims at introducing new information into the population by randomly altering the component symbols of individual solution candidates. The process of population generation is repeated **until a stop condition is satisfied**. The reader is referred to [31] for more details on genetic algorithms in general.

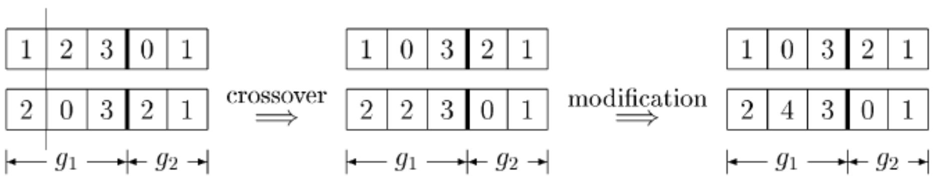

**Fig. 2**. Modification procedure for the crossover operator.

Genetic algorithms have a number of advantages against other discrete optimization methods. One of the unique features of genetic search is given by the crossover operator. It effectively provides a means of combining locally consistent subsolutions to generate a globally consistent solution. In addition, although discrete gradient-ascent optimization with multiple random starts shares the idea of maintaining a population of alternative solutions, it is the genetic operators that ensure a higher likelihood of global convergence. For these reasons, resort to a genetic search strategy in this work.

In our case, the genetic algorithm, each chromosome represents a specific common labeling and is encoded as an integer array. The length of the array is equal to the total number of nodes in the different input graphs. Each node has a corresponding position in the array. The number at a specific position in a chromosome tells us how the corresponding node is generated by transforming the candidate graph. A number n $\geq$ 1 means that the node is substituted from the candidate graph node n, while the number 0 denotes an insertion of the corresponding node into the candidate graph.

In the extension of the Graduated Assignment algorithm, a similar procedure as the Graduated Assignment methodology is performed but, it is applied to solve the Common Labeling problem. The aim of the algorithm is to reduce an energy related to the Common Labeling in an iterative manner. This energy depends on the probability that nodes of the graphs are labeled to nodes of the virtual vertex set $L_v$. When the algorithm reaches a minimum value, a discretisation process has to be applied. Note that, in [32] the proposed algorithms deal with probabilities of labeling a pair of nodes of two graphs instead of labeling from nodes of graphs to nodes of the virtual vertex set.

## 3.3 Median Graph

A **Median Graph** is an Attributed Graph that minimizes the sum of distances between it and all the graphs in the set of Attributed Graphs. It has exactly the same information and domain than the Attributed Graphs that it represents. Notice that it is usually not a member of the set, and in general, more than one Median Graph may exist for a given set of graphs. Formally, Median Graph. Given a set of graphs $\Gamma = \{G^1, G^2, \dots, G^N\}$ in a set U and a distance between Attributed Graphs $d(G^i, G^j)$, the *Median Graph* is defined as follows.

$$\bar{g} = \begin{matrix} argmin \\ g \in \mathrm{U} \end{matrix} \sum_{G^i \in \Gamma} d\left(g, G^i\right) \quad \textbf{Eq. 2}$$

That is, the Median Graph is the graph $g \in$ U that minimizes the sum of distances to all graphs in $\Gamma$. The set U represents the universe of Attributed Graphs in a specific domain.

The computation of a Median Graph is a NP-complete problem. Nevertheless, several suboptimal methods to obtain approximate solutions for the Median Graph, in reasonable time. These methods apply some heuristic functions in order to reduce the complexity of the graph distance computation and the size of the search space.

An alternative to Median Graphs, which is less computationally demanding, is the Set Median Graph. The difference between the two models consists in the search space where the Median Graph is looked for. The search space for the Median Graph is the whole universe of Attributed Graphs *U*.

## 3.4 Set Median Graph

In contrast, the search space for the **Set Median Graph** is simply the set of graphs that represents Γ. Formally, Set Median Graph. Given a set of graphs $\Gamma = \{G^1, G^2, \dots, G^N\}$ and a distance between Attributed Graphs $d(G^i, G^j)$, the *Set Median Graph* is defined as follows.

$$\bar{g} = \arg\min_{g\in\Gamma} \sum_{G^i\in\Gamma} d(g, G^i) \quad \textbf{Eq. 3}$$

The computation of the Set Median Graph is exponential in the size of the graphs, due to the complexity of graph edit distance, but quadratic with respect to the number of graphs in the set. Nevertheless, Set Median Graphs have less ability to capture the information of the cluster than Median Graphs due to the reduction of the search space.

The advantages of using Median Graphs as routing elements in an m-tree are manifold. The main effect of using them is the reduction of the overlap between sub-clusters, due to the radius of the covering region, $d^h$, can be more tightly adjusted. In fact, if we use the Median Graphs as a routing element, the radius of the covering region has to be equal or lower than the radius of the covering region represented by a Set Median Graph of the cluster.

Figure 3 and 4 shows an example of representing the cluster by a Set Median Graphs (figure 3) and by a Median Graph (figure 4). Both figures show the same 6 elements in two sub-clusters and the radius of their covering regions. Suppose a hypothetical query graph Q with a query range represented by the outer doted circle. The execution of the search behaves very different on both representations. In the Set Median Graph approach, neither entry p nor q holds for equations 2 and 3, so the *sub*$^q$ and *sub*$^p$ must be explored. However, in the Median Graph, equation 2 holds for both tree node entries p and q. Consequently, it can be assumed that none of the entries contain any desired graph. Thus, they can be discarded and not explored.

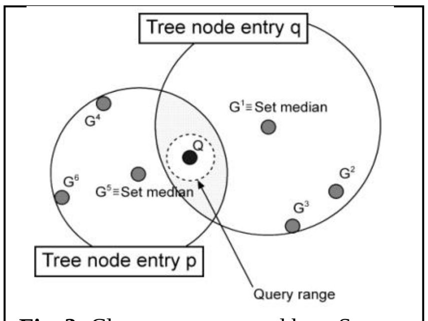

**Fig. 3**: Clusters represented by a Set Median Graph.

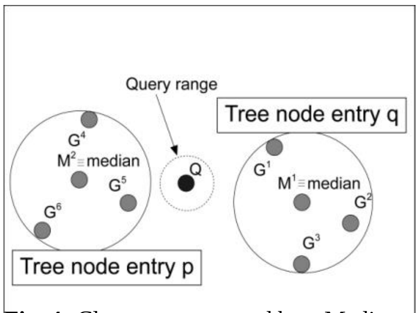

**Fig. 4:** Clusters represented by a Median Graph.

The representative of the cluster inserted in the tree node was one of the graphs in the cluster. For this reason, although it was not commented in that paper, the representative was a Set Median Graph.

## 3.5 Closure Graph

A **Closure Graphs** is another model graph that the structure of the attributes is different to the Attributed Graphs that represent. The structurally similar nodes of the set of Attributed Graphs that have different attribute values are represented in the Closure Graph with only one node but with more than one attribute. Closure Graphs need the attributes at nodes of the graphs to be discrete, if it is not the case; an extra discretization process has to be performed. Closure Graphs needs few more physical space than the Median Graphs.

Formally, a Closure Graph $\mathrm{M} = \left(\widehat{\Sigma_v}, \widehat{\Sigma_e}, \widehat{\gamma_v}, \widehat{\gamma_e}\right)$ is a graph where the node and arc attribute domains are a set of domains of the nodes and arcs of the Attribute Graphs $\{\Delta_v, \Delta_v, ..., \Delta_v\}$ and $\{\Delta_e, \Delta_e, ..., \Delta_e\}$. The specific value are $\{a_1, a_2, ..., a_t\}$ and $\{b_1, b_2, ..., b_h\}$. We synthesise a Closure Graph from a set of Attributed Graphs $\Gamma$ and a Common Labelling $\psi$ as follows:

$$\widehat{\gamma_v}(\widehat{v_j}) = \left\{\cup\, a_k \middle| \exists a_k = \gamma_v\left(h^{i^{-1}}(\widehat{v_j})\right), 1 \leq i \leq |\Gamma|, \gamma_v\left(h^{i^{-1}}(\widehat{v_j})\right) \neq \phi\right\}$$

$$\widehat{\gamma_e}(\widehat{e_{qj}}) = \{\cup\, b_k | \exists b_k = \varepsilon, \varepsilon \neq \phi\}$$

**Eq. 4**

being$\varepsilon$ the attribute value of the arc $e_{h^{i^{-1}}(\widehat{v_q}), h^{i^{-1}}(\widehat{v_j})}$in $G^i$.The main idea of Eq. 4 is that nodes or arcs of the Closure Graph can take all values that nodes or arcs of the Attributed Graphs have taken. In the case that there does not exist a node or arc in the Attributed Graph, its value is not considered.

Practical evaluations show that Median Graphs and Closure Graphs tend to generalize to much the set that they represent; allowing graphs that are distant from the ones that have not been used to synthesize them. To alleviate this weakness, the following probabilistic models have been defined.

## 3.6 First-Order Random Graph (FORG)

A **First-Order Random Graph** (FORG) is a model graph that contains first-order probabilities on nodes and arcs to describe a set of Attributed Graphs. To deal with the first-order probabilities, there is a random variable associated with each vertex or arc, which represents the attribute information of the corresponding graph nodes and arcs in the set of Attributed Graphs. This random variable has a one-dimensional probability density function defined over the same attribute domain of the Attributed Graphs, including a null value that denotes the non-instantiation of a FORG graph node or arc in an Attributed Graphs.

This model was the first probabilistic one that appeared in the literature to represent a set of Attributed Graphs. It assumes that the Attributed Graphs in a set or cluster had similar local parts. Nevertheless, in practical applications some graphs can be quite different despite of belonging to the same class. For this reason, representing a set of graphs with only first order probabilities seems to be too restrictive.

A First Order Random Graph $M = (\widehat{\Sigma_v}, \widehat{\Sigma_e}, P)$ is a graph where the node and arc attribute domains are random variables with values in $\Delta_v$ and $\Delta_e$. Probabilities at the nodes and arcs of the FORG are related to the number of times the values have appeared at the nodes or arcs of the Attributed Graphs related to this node or arc. In the case of the nodes,

$$p_j(\mathrm{a}) = \frac{\# G^i : \gamma_v\left(h^{i^{-1}}(v_j)\right) = \mathrm{a}}{z} \qquad \textbf{Eq. 5}$$

Where $z$ is the number of Attributed Graphs in the set. In the case of the arcs,

$$p_{qj}(\mathrm{b}) = \frac{\# G^i : \gamma_v\left(h^{i^{-1}}(v_q)\right) \neq \Phi \wedge \gamma_v\left(h^{i^{-1}}(v_j)\right) \neq \Phi \wedge \varepsilon = \mathrm{b}}{u_{qj}} \qquad \textbf{Eq. 6}$$

being$\varepsilon$ the attribute value of the arc that links node $h^{i^{-1}}(v_q)$with node $h^{i^{-1}}(v_j)$ in $G^i$ and $u_{qj}$ is the number of Attributed Graphs that contain both non-null vertices. That is,

$$u_j = \# G^i : \gamma_v\left(h^{i^{-1}}(v_q)\right) \neq \Phi \wedge \gamma_v\left(h^{i^{-1}}(v_j)\right) \neq \Phi \qquad \textbf{Eq. 7}$$

## 3.7 Function-Described Graph (FDG)

A **Function-Described Graph** (FDG) [33] is a model graph that appeared with the aim of overcoming the representational power of FORGs. It contains first-order probabilities of attributes and second-order structural information to describe a set of Attributed Graphs. The first order information was represented in the same way than FORGs trough probability density functions. The second-order structural information is a qualitative information of the second-order joint probability of each pair of vertices or arcs. This information is represented by binary relations called *Antagonisms*, *Occurrences* and *Existences* between nodes and arcs. FDGs increased the representational power at the cost of increasing also the required physical space.

Two nodes or arcs are *antagonistic* if they have never taken place together in any graph used to synthesise the FDG although these two nodes or arcs are included in the FDG as different elementary parts. There is an *occurrence relation* between two nodes or arcs of the FDG if always that one of related nodes or arcs in the graph has appeared; also the other node or arc of the same graph has appeared. Finally, there is an *existence* relation between two nodes or arcs if all the graphs in the class described by the FDG have at least one of the two nodes or arcs.

A Function-Described Graph $M = (\widehat{\Sigma_v}, \widehat{\Sigma_e}, P, R)$is a graph where$\widehat{\Sigma_v}, \widehat{\Sigma_e}, P$ is defined exactly in the same way than FORGs; including, in addition a set of binary relations R. See [7] for the construction of these binary relations given a Common Labelling.

## 3.8 Second-Order Random Graph (SORG)

A **Second-Order Random Graph** (SORG) is a probabilistic model closely related to FDGs. The main difference lies in the fact that the second-order structural information is not defined as binary relations but with the specific information of the second-order joint probability. Thus, the physical space needed to represent SORGs is much higher than FDGs but also its ability to represent the set of Attributed Graphs increases. In [6,7] it is shown how to convert a SORG to an FDGs simply by analysing the second-order probabilities and deciding if the binary relations hold.

A Second-Order Random Graph $\mathrm{M} = \left(\widehat{\Sigma_v}, \widehat{\Sigma_e}, \mathrm{P}\right)$is a graph where$\widehat{\Sigma_v}, \widehat{\Sigma_e}, \mathrm{P}$ is defined similarly to FORGs but in P there are first order and also second order probability densities. See [34] for the construction of these second-order probability densities given a Common Labelling.

## 3.9 First Order Gaussian Graph (FOGG)

A **First Order Gaussian Graphs** (FOGG) [35] is a probabilistic model based on Gaussian approximation, it proposes a random graph model in which vertices and edges are associated with discrete random variables taking values over the attribute domain of the graph.

Given an Atributed Graph AG over L = $(L_v, L_e)$, G = $(V_G, E_G, m_v, m_e)$ where V is a set of vertices, E is a set of edges, $m_v\colon V \rightarrow L_v$, , $m_e\colon E \rightarrow L_e$ and FOGG R = $(V_R, E_R, \mu_v, \mu_e)$, the task is to compute the probability that G is outcome of R. To simplify our notation we let $p_{vi}$ denote the probability density function of $\mu_v(v_i)$ and $p_{e_{ij}}$ the density of $\mu_e(e_{ij})$. Furthermore, let $v_i = m_v(\emptyset(v_i))$ and $e_{ij} = m_e(\emptyset(e_{ij}))$ denote the observed attributes for vertex $v_i$ and edge $e_{ij}$ respectively under isomorphism $\emptyset$.

We define the probability that *R* generates *G* in terms of a vertex factor:

$$\boldsymbol{V_R(G,\emptyset) = \prod_{v_i \in V_R} p_{v_i}(v_i)} \quad \textbf{Eq. 8}$$

and an edge factor:

$$\boldsymbol{E_R(G,\emptyset) = \prod_{e_{ij} \in E_R} p_{e_{ij}}\left(e_{ij} \middle| v_i, v_j\right)} \quad \textbf{Eq. 9}$$

The probability that *G* is an outcome graph of *R* is then given by:

$$P_R(G,\emptyset) = \prod_{e_{ij} \in E_R} \frac{p_{e_{ij}}(e_{ij}|v_i,v_j) p_{e_{ij}}(e_{ij}|)}{p_{v_i}(v_i) p_{v_j}(v_j)} \quad \textbf{Eq. 10}$$

# 4. Database Indexing based on Metric Trees

A metric-tree (m-tree) [36] is a method to partition a database in a hierarchical set of clusters, collecting similar objects. Each cluster has a routing object and a radius providing an upper bound for the maximum distance between the reference object and any other object in the cluster. Triangle inequality can be used during the access to the database to prune clusters that are bound out of an assigned range from the query.

More formally, a metric-tree, is a tree of nodes, each containing a fixed maximum number of *m* entries, *< node > := {< entry >}*$^m$. In turn, each entry is constituted by a routing element *M*; a reference to the father $r^H$ of a sub-index containing the element in the so-called covering region of *M*; and a radius $d^M$ providing an upper bound for the distance between *M* and any element in its covering region, *< entry > := {M,* $r^M$*,* $d^M$*}*. During retrieval, triangular inequality is used to support efficient processing of range queries. That is, queries seeking for all the elements in the database which are within a given range of distance from a query element *Q*. To this end, the distance between *Q* and any element in the covering region of a routing element *M* can be max-bounded using the radius $d^M$ and the distance between *Q* and *M*.

To perform range queries in Metric Trees, the tree is analyzed in a top down fashion. Specifically, if $d_{max}$ is the range of the query and *Q* is the query graph, the following conditions are employed, at each node of the tree, to check whether all the elements in the covering region of *M*, $sub^M$, can be discarded, accepted or need more exploration. The conditions are based on the evaluation of the distance between the routing element and the query element *d(Q,M)*.

If the following condition holds, we reject all elements deeper from the routing element.

$$d(Q,M) \geq d_{\max} + d^M \quad \Rightarrow \textit{No element in } sub^M \textit{ is acceptable} \qquad \textbf{Eq. 11}$$

In a similar manner, the following condition checks whether all the elements in the covering region of *M*, $sub^M$, fall within the range of the query. In this case, all the elements in the region can be accepted:

$$d(Q,M) \leq d_{\max} - d^M \quad \Rightarrow \textit{Every element in } sub^M \textit{ is acceptable} \qquad \textbf{Eq. 12}$$

In the critical case that neither of the two inequalities holds, the covering region of *M*, $sub^M$, may contain both acceptable and no acceptable elements, and the search must be repeated on the sub index $sub^M$.

There is another type of queries performed on m-trees called *K*-Nearest Neighbour Queries. In those queries, instead of the input parameter $d_{max}$, the parameter *K* is introduced into the system. The system returns the *K* graphs in the database that have the lowest distance between the query and them. Moreover, the knowledge in the tree nodes is not the Median Graph but a Graph Prototype.

The m-tree can be constructed using different schemes for the insertion of a new element and the selection of the routing element. Following a static scheme, such as that of the special kind of m-tree called mvp-tree [22], routing elements are selected when the entire database is determined. In this case, the m-tree is constructed in a top-down manner, by repeatedly partitioning the database through the selection of routing elements which yield a balanced split. Following an alternative approach, in the m-tree in [20], the index is constructed dynamically by inserting new elements from the bottom layer and by splitting nodes and promoting routing elements when insertion overflow occurs.

In this paper, we use a general construction methodology from which we are able to construct a metric tree independently of the type of the routing element. That is, the structure of the m-tree and the clusters that generates in the database are independent of the kind of representation [11,12,13] or [14] we use in the node tree. Given an Attributed Graph set, it is crucial to obtain the same structure of the m-tree for all the types of routing elements, since we want to compare its representational power in similar conditions. We use a non-balanced tree constructed through a hierarchical clustering algorithm and complete linkage clustering or [23]. In this way, given a set of graphs, the distance matrix over the whole set is computed and then a dendogram is constructed. Using the dendogram and some horizontal cuts, a set of partitions is obtained that clusters the graphs in the database. With these partitions the m-tree is generated. Finally, the information on the routing elements in the m-tree is inserted, $M$ and $d^M$.

In our case, $M$ is a Median Graph and $d^M$ is the maximum distance between the Median Graph and any of the graphs in the covering region $sub^M$. Figure 5 shows an example of a dendogram. The elements $G^i$ are placed on the leaves of the dendogram and the routing elements $M^j$ are placed on the junctions between the cuts and the horizontal lines of the dendograms. We have defined 4 different partitions. Figure 6 shows the obtained m-tree. Note that in some tree nodes, there are Median Graphs ($M^i$) together with original graphs ($G^j$).

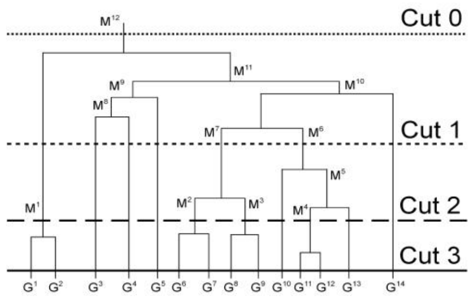

**Fig. 5:** Example of a dendogram.

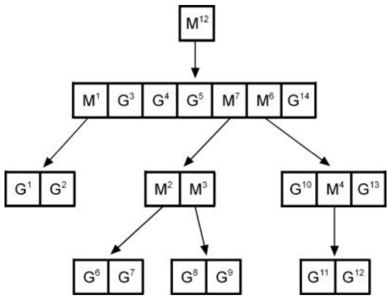

**Fig. 6**: The obtained m-tree.

# 5. Indexing Databases of Graphs

Some indexing techniques have been developed for graph queries. We divide these techniques in two categories. In the first ones, the index is based on several tables and filters or [30,27] In the second ones, the index structure is based on metric-trees [9,28,29].

In the first group of techniques, the ones that are not based on trees, we emphasize the method developed by Shasha *et. al.* [30] called GraphGrep. GraphGrep is based on a table in which each row stands for a path inside the graph (up to a threshold length) and each column stands for a graph. Each entry in the table is the number of occurrences of the path in the graph. Queries are processed in two phases. The filtering phase generates a set of candidate graphs for which the count of each path is at least that of the query. The verification phase verifies each candidate graph by subgraph isomorphism and returns the answer set. More recently, Yan *et. al* [27] proposed GIndex that uses frequent patterns as index features. These frequent patterns reduce the index space as well as improve the filtering rate. The main drawback of these models is that the construction of the indices requires an exhaustive enumeration of the paths or fragments which increases the memory and time requirements. Moreover, since paths or fragments carry little information about a graph, the lost of information at the filtering step seems to be unavoidable.

Considering the second group, the first time that metric trees were applied to graph databases was done by Berretti *et. al.* [9]. Attributed graphs were clustered hierarchically according to their mutual distances and indexed by m-trees [20] . Queries are processed in a top-down manner by routing the query along the index tree. Each node of the index tree represents a cluster and it has one of the graphs of the cluster as a representative. The graph matching problem, in the tree construction and at query time, was solved by an extension of the A* algorithm that uses a look-ahead strategy plus a stopping threshold. A drawback of this method is that the computational cost is exponential respect the number of nodes in the graphs. Lee *et. al.* [28] used this technique to model graphical representations of foreground and background scenes in videos. The resulting graphs were clustered using the edit-distance metric, and similarity queries were answered using a multi-level index structure.

More recently, He and Singh [29] proposed what they called a Closure-tree. It uses a similar structure than the one presented by Berretti [9] but, the representative of the cluster was not one of the graphs but a graph prototype (called closure graph) that could be seen as the union of the Attributed Graphs that compose the cluster. Figure 7 shows the closure of 3 graphs. The structurally similar nodes that have different attributes in the graphs are represented in the Closure graph with only one node but with more than one attribute. Closure trees have two main drawbacks. First, they can only represent discrete attributes at nodes of the attributed graphs. Second, they tend to generalize too much the set that represent, allowing graphs that have not been used to synthesize the closure graph.

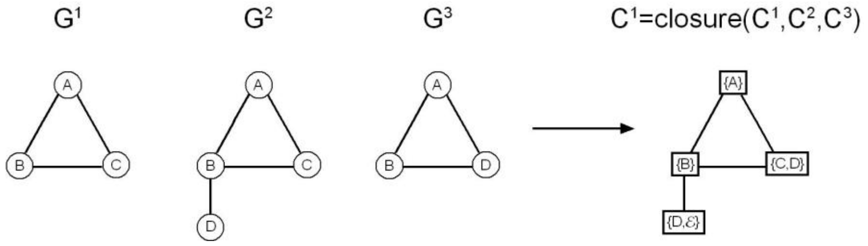

**Fig. 7**: Example of a Closure obtained by 3 Attributed Graphs.

Finally, Median Graphs have been used as a new prototype to represent Attributed Graphs in [17,18]. More specifically, in [17], they defined queries in which a maximum distance between the query and the graphs was considered. And in [31], they performed k-nearest neighbour queries.

Our proposal is to use Median Graphs as a representative of the sub-clusters in the routing nodes of the metric trees instead of an Attributed Graph representative [9] or a closure graph [29]. We aim to show that Median Graphs are good representatives of a cluster and that they can be used in conjunction with m-trees.

# 6. Nearest Neighbours on Graph Databases

In this section, we first explain Nearest Neighbour queries applied on Metric Trees as tools to retrieve elements of any type of database. Then, we explain how the techniques to index graph databases have evolved until arriving at the method we present in this paper.

## 6.1 Nearest Neighbour Queries on Metric Trees

In pattern recognition, the k-nearest neighbour algorithm (*k*-NN) is a method for classifying objects based on closest training examples in the feature space [4]. To define "closest", a distance measure has to be established between the elements of the set. *k*-NN is a type of instance-based learning where the function is only approximated locally and all computation is deferred until classification. The k-nearest neighbour algorithm is one of the simplest of all machine learning algorithms: an object is classified by a majority vote of its neighbours, with the object being assigned to the most class common among its *K* nearest neighbours. Being the nearest depends on a previously specified distance measure. *k* is a small positive integer. If $k = 1$, then the object is simply assigned to the class of its nearest neighbour. The neighbours are taken from the training set of objects for which the correct classification is known.

Another method used to classify objects is to group several elements of a cluster in only one representative or prototype of the class. Then the 1-nearest neighbours is applied not to the elements but to the representative. The advantage of using a representative is that the amount of comparisons needed to decide the class of an object can be reduced since only depends on the number of representatives but does not depend on the number of elements in the database. Two types of representatives are defined. In the first ones, the representative is one of the elements of the set. Usually, it is the element that has the minimum distance between them and the rest of elements in the cluster. The main advantage of this type of representative is that its computation is straightforward and fast. However, it is statistically difficult to find a good representative if the number of elements in the set is small. In the second ones, a new element is defined. If the elements are characterised by vectors, an easy way to define the representative is simply by the mean vector. Nevertheless, when the format of the data is a graph, it is more difficult to model a new graph prototype and more computationally expensive. Nevertheless, this new prototype offers a higher quality representation of the set, which should improve performance with respect to use one of the elements of the set.

The aim of the Nearest Neighbour Queries is to retrieve the *k* elements in a database that have minimum distance between them and the query element. It is assumed that, at least, there are *k* elements in the database. There is another type of queries performed on m-trees called Similarity Queries. In those queries, instead of the input parameter *K*, there is a distance value $d_{max}$ and the system returns all the elements in the database that have a lower distance between the query and them. Moreover, in [17] the knowledge in the tree nodes is restricted to Median Graphs.
The most common method to perform Nearest-Neighbour queries [4] uses a branch-and-bound algorithm, which utilises two global structures: A priory queue PR that stores the possibly fruitful tree nodes to be explored and an array NN that stores the best *k* elements found until the moment. PR is a queue of pointers to active sub-trees, i.e. sub-trees where objects that may be near to the

query element can be found. The $k$ values of NN are initialised to a null element. PR is initialised with one element which is the root of the metric tree. Note that PR does not have a maximum number of elements.

Let $Q$ be a query element and $d_{max}^{NN}(Q)$ the maximum distance from $Q$ to any element in NN. The distance $d_{max}^{NN}(Q)$ is initialised to infinitive. In each iteration of the search algorithm, the tree node in PR with lower distance to $Q$ is selected, let this node be named $N$ and its children be $N^1, \ldots, N^t$. The distances between $Q$ and all the children of $N$ must be computed.

If son $N^i$ is a **routing node**, this node is inserted in PR if $d(N^i, Q) - d^{N^i} \leq d_{max}^{NN}(Q)$. This insertion is done due to it is possible to find an element with lower distance than the ones already found. On the contrary, it is not possible that any of their descendants have a distance lower than $d_{max}^{NN}(Q)$ and so, none of the nodes and leaves of the branch are explored.

If son $N^i$ is a **leave**, that is, a database element, and $d(N^i, Q) \leq d_{max}^{NN}(Q)$, array NN and $d_{max}^{NN}(Q)$ are updated to consider this element in the following way. There are two possibilities:

If all NN has its k positions full with leaves, then the element with higher distance is discarded. Moreover, the distance $d_{max}^{NN}(Q)$ is updated to be the maximum distance from $Q$ to any element in NN. This new value has to be lower than the previous value since $d_{max}^{NN}(Q)$ acts as a dynamic maximum search distance of range queries.

If NN does not have its $k$ positions full with leaves, then, the new tree leave $N^i$ (that is, an element of the database) is inserted in an empty position of NN. Related to the distance $d_{max}^{NN}(Q)$, there are again two cases. If NN continues to be non full, then, the distance keeps with its initial value, which is infinitive. But, if the new situation of NN is that all the elements are used, $d_{max}^{NN}(Q)$ takes the maximum value of the distances between the leaves in NN and $Q$. For a detailed description of k-nn queries the reader is referred to the original article.

The routing element $M$ can be defined as one of the elements of the sub-cluster or a new element that represents the elements of the sub-cluster. The main effect of using a new prototype might be the reduction of the overlap between sub-clusters, due to the radius of the covering region can be more tightly adjusted.

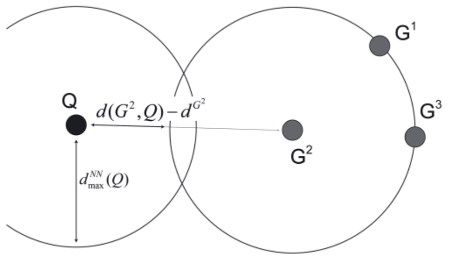

**Fig. 8** Clusters represented by an element.

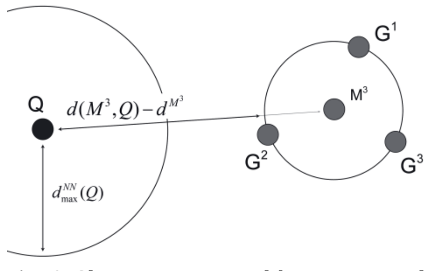

**Fig. 9** Clusters represented by a computed prototype.

Fig. 8 and 9 show an example where the same query is performed using the two types of routing nodes. In the example, Q is compared to a cluster that represents elements $\mathrm{G}^1$, $\mathrm{G}^2$ and $\mathrm{G}^3$. In Fig. 6 the cluster is represented by one of the elements in the cluster, $\mathrm{G}^2$. In Fig. 7 the cluster is represented by a new prototype. In the given example the cluster represented in Fig. 6 must be explored due to $\mathrm{d}(\mathrm{G}^2, \mathrm{Q}) - \mathrm{d}^{\mathrm{G}^2} \leq \mathrm{d}_{\mathrm{max}}^{\mathrm{NN}}(\mathrm{Q})$. However, in Fig. 7 the cluster radius is better adjusted and we can ensure that it will not have any desired element due to $\mathrm{d}(\mathrm{M}^3, \mathrm{Q}) - \mathrm{d}^{\mathrm{M}^3} > \mathrm{d}_{\mathrm{max}}^{\mathrm{NN}}(\mathrm{Q})$. Note that, in metric trees, the smaller the radius of clusters are, the lower the number of comparisons that we must perform to find the solution.

# 7. Synthesis of Graph Prototypes related to Metric Trees

There are two methods to generate Graph Prototypes when they are the knowledge of a tree node of a metric-tree. We consider that the structure of the metric-tree has been computed (section 2) and we have to compute the Graph Prototype and the radius of the cluster. The first method is based on a hierarchical synthesis. The other one is based on a global synthesis based on a Common Labelling [30,31,32].

In the Hierarchical method, each Graph Prototype is computed only using two Graph Prototypes or Attributed Graphs at a time. Therefore, a Common Labelling is not needed and Graph Prototypes are computed as pairwise consecutive computations of other Graph Prototypes obtained in lower levels of the tree.

## 7.1 Synthesis of Median Graphs

We explain two methods to generate Median Graphs. The first one is based on a hierarchical synthesis. The other is based on a global synthesis based on a common labelling.

### 7.1.1 Hierarchical Synthesis of Median Graphs

In this method, each Median Graph is computed only using two Median Graphs or Attributed Graphs. The algorithm to obtain the new Median Graph is explained in [32]. The main idea is to compute the edit path that transforms one Attributed Graph to the other and then return the Attributed Graphs located on the path and with the same distance between it and both Attributed Graphs. Therefore, Median Graphs are computed as pairwise consecutive computations of other Median Graphs obtained in lower levels of the tree. For instance, to compute *M*$^7$, which appears at Figure 1.1, we only use *M*$^2$ and *M*$^3$ Median Graphs. That is, we assume that:

$$M^7 \cong \overline{(M^2, M^3)} \cong \overline{(\overline{(G^6,G^7)},\overline{(G^8,G^9)})} \qquad \textbf{Eq. 13}$$

The covering region radius *d*$^p$ of the Median Graph *M*$^p$ is computed applying three rules, depending whether the type of the descendant of *M*$^p$ in the dendogram is another Median Graph (that is, a routing node of the m-tree) or an Attributed Graph (that is, a leaf of the m-tree):

- When both descendants are graphs (*G*$^a$ and *G*$^b$):

$$d^p = Max(d(M^p, G^a), d(M^p, G^b)) \qquad \textbf{Eq. 14}$$

- When a descendant is a Median Graph (*$M^a$*) and the other is a graph (*$G^b$*):

$$d^p = Max(d(M^p, M^a) + d^a, d(M^p, G^b)) \qquad \textbf{Eq. 15}$$

- When both descendants are Median Graphs (*$M^a$* and *$M^b$*):

$$d^p = Max(d(M^p, M^a) + d^a, d(M^p, M^b) + d^b) \qquad \textbf{Eq. 16}$$

Fig. 10 and 11 illustrate the second and third rule, respectively. In the first case, $d(M^5,M^4) + d^{M4}$ is greater than $d(M^5,G^6)$, and in the second case $d(M^7,M^3) + d^{M3}$ is greater than $d(M^7,M^2)+d^{M2}$.

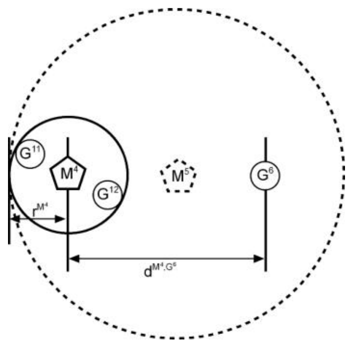

**Fig. 10**: Second radius computation rule.

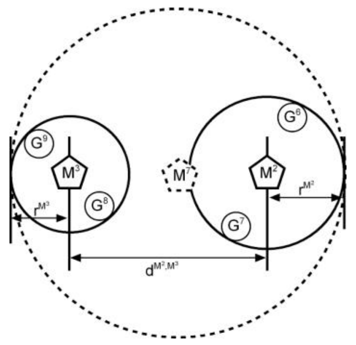

**Fig. 11**: Third radius computation rule.

## 7.1.2 Global Synthesis of Median Graphs using a Common Labelling

In this method, each Median Graph is computed using the whole set of Attributed Graphs in the cluster that the m-tree node represents, independently of whether the m-tree node has other nodes as descendants in the tree. The first step of this method computes a Common Labelling from the Attributed Graphs of the sub-cluster and the second step obtains the Median Graph. In this paper, we use two different methods to obtain the Common Labelling commented in section 4. The first one is based on a genetic algorithm [18] and the other one is based on the Graduated Assignment algorithm [19].

Once the Common Labelling $\psi$ is obtained, we compute the Median Graph from a set of Attributed Graphs $\Gamma$ under this Common Labelling $\psi = \{h^1, h^2, \dots, h^n\}$ as another Attributed Graph where attributes on nodes and arcs are:

$$\gamma_v(v_j) = \frac{1}{M}\sum_{\forall G^i \in \Gamma |\, \gamma_v\left(h^{i^{-1}}(v_j)\right) \neq \phi} \gamma_v\left(h^{i^{-1}}(v_j)\right) \text{ and } \gamma_e(e_{kj}) = \frac{1}{N}\sum_{\forall G^i \in \Gamma |\, \varepsilon \neq \phi} \varepsilon \qquad \textbf{Eq. 17}$$

being $M$ the number of nodes such that $\forall G^i \in \Gamma |\, \gamma_v\left(h^{i^{-1}}(v_j)\right) \neq \phi$, N the number of arcs such that $\varepsilon \neq \phi$ and being $\varepsilon$ the attribute value of the arc $e_{h^{i^{-1}}(v_k), h^{i^{-1}}(v_j)}$ in $G^i$. The main idea of eq. (13) is that the attribute values of the Median Graph is the mean of the values of all the nodes or arcs of the

Attributed Graphs that their nodes or arcs have been matched to this node or arc of the Median Graph. Moreover, in the case that there does not exist a node or arc in the Attributed Graph, its value is not considered to compute the mean.

In the Global Synthesis, each Graph Prototype is computed using the whole set of Attributed Graphs in the cluster that the m-tree node represents, independently of whether the m-tree node has other nodes as descendants in the tree. The first step of this method computes a Common Labelling from the Attributed Graphs of the sub-cluster and the second step obtains the Graph Prototype. In this paper, we have used two different Common Labelling algorithms, which have the main feature that are independent of the prototype graph to be synthesised. The first one is based on the Graduated Assignment algorithm [17] and the second one is based on a genetic algorithm [16].

Note that the Set Median is a special prototype since it does not need to be synthesised. The Set Median is one of the graphs of the cluster that has the minimum distance between it and the other graphs.

# 8. Practical Evaluation

## 8.1 Datasets

The datasets are Letter HIGH and LOW, COIL-RAG, Fingerprint and GREC (presented in [37] ).

### 8.1.1 Letter

The **Letter HIGH / LOW** database is composed of 15 capital letters (classes) of the Roman alphabet composed of straight lines. Each class is composed by 150 graphs. The straight lines are represented by edges and the terminal points of the lines are represented by the nodes. The complete dataset is split into a training set of 15x50 graphs and a test set of 15x100 graphs.

### 8.1.2 GREC

**GREC**: The GREC database is composed of a set of 150 symbols from architecture, electronics and other technical fields. We have used a subset of 22 different symbols (classes), those which are composed only of straight lines. These images are converted into graphs by assigning a node to each junction or terminal point and an edge to each line. For each of the 22 symbols in the dataset there are 50 distorted instances. So, totally we have 1100 images. The complete dataset is split into a training set of 440 (20 graphs for each class) and a test set of 660 elements (30 graphs per class).

### 8.1.3 COIL-RAG

**COIL-RAG**: The COIL-100 database consists of images of 100 different objects (100 classes). Images of the objects are taken at pose intervals of 5 degrees, therefore, there are 72 images per object. Images have been segmented into regions of homogeneous colour using a mean shift algorithm. Segmented images are transformed into region adjacency graphs by representing regions by nodes, labelled with a two-dimensional attribute giving its position. Undirected edges are inserted to link nodes that their regions are connected. Edges do not have attributes. Images taken at angles 0, 10, 20,... represent the test set and angles taken at angles 5, 15, 25,... represent the reference set.

### 8.1.4 Fingerprint

**Fingerprint**: Image fingerprints are converted into graphs by filtering the images, extracting regions and obtaining the ridges. Ending points and bifurcation points of the ridges are represented by nodes. Undirected edges are inserted to link nodes that are directly connected through a ridge in the skeleton. Each node is labelled with a two-dimensional attribute giving its position. Edges do not have attributes. The original fingerprint database is based on the NIST-4 reference database of fingerprints . It consists of 2,800 fingerprint images totally out of the 4 classes arch, left, right, and whorl from the Galton-Henry classification system. Each of the four classes has been equally divided into the test set and the reference set.

**Table 1.** Summary of graph data set characteristics, viz. the size of the training (*tr*), the validation (*va*) and the test set (*te*), the number of classes (|Ω|), the label alphabet of both nodes and edges, the average number of nodes and edges (mean nodes/edges).

| *Database* | *Size (*tr,va, te*)* | \|Ω\| | *Node labels* | *Edge labels* | *mean nodes* | *mean edges* |
|---|---|---|---|---|---|---|
| Letter LOW | 750, 750, 750 | 15 | x, y coordinates | None | 4.7 | 4.5 |
| Letter HIGH | 750, 750, 750 | 15 | x, y coordinates | None | 4.7 | 4.5 |
| GREC | 20, 0, 30 | 22 | x, y coordinates | None | 11.5 | 11.9 |
| COIL | 2400, 2400, 2400 | 400 | x, y coordinates | None | 12.8 | 27.1 |
| Fingerprint | 500, 300, 2000 | 4 | x, y coordinates | None | 5.4 | 4.4 |

## 8.2 Graph Prototypes for Classification

**Table2.** This table presents classification results obtained by each prototype for Letters and GREC databases, taking a comparative between the classification results obtained by embedding methods presented in [23,38] . Highlighted in bold and underline appears the best result of structural methods and only in bold the second one.

| | | ***Letter LOW*** | ***Letter HIGH*** | ***GREC*** |
|---|---|---|---|---|
| **Ref. systems** | ***k*-NN** ① | 91.07 | 61.60 | 86.06 |
| | ***k*-NN** ② | 89.10 | - | - |
| | ***k*-NN** | 94.26 | 82.24 | 94.99 |
| **Embedding methods** | **Emb. Kernel** ① | 91.80 | 74.27 | 89.17 |
| | **sps-c** ② | 92.30 | - | - |
| | **bps-c** ② | 92.90 | - | - |
| | **k-cps-c** ② | 92.00 | - | - |
| **Structural methods** * | **Set Median** | **96.40** ❶ | 74.53 ❶ | 80.45 ❶ |
| | | **96.27** ❷ | 68.93 ❷ | 85.30 ❷ |
| | **Median Graph** | 90.27 ❶ | 70.00 ❶ | **90.91** ❶ |
| | | 89.20 ❷ | 70.93 ❷ | **90.76** ❷ |
| | **Closure Tree** | 93.60 ❶ | 49.07 ❶ | 56.97 ❶ |
| | | 69.20 ❷ | 22.13 ❷ | 56.21 ❷ |
| | **FORG** | 93.87 ❶ | 80.13 ❶ | 85.76 ❶ |
| | | 92.27 ❷ | 79.20 ❷ | 83.03 ❷ |
| | **FDG** | 93.87 ❶ | **81.60** ❶ | 85.76 ❶ |
| | | 92.27 ❷ | **81.73** ❷ | 83.03 ❷ |
| | **SORG** | **94.00** ❶ | **80.93** ❶ | **91.21** ❶ |
| | | **92.80** ❷ | **79.87** ❷ | **89.55** ❷ |
| | **FOGG** | 06.67 ❶ | 06.67 ❶ | 04.54 ❶ |
| | | 06.67 ❷ | 06.67 ❷ | 04.54 ❷ |

① Presented results in [38]

② Presented results in [23]
❶ Results with KNN of N prototypes per class grouped by 10, with K = 3
❷ Only one prototype of 50 elements per class for the letters, and 20 for the GREC.

## 8.3 M-Trees Querys Results

### 8.3.1 M-Trees Querys Analizys

We aim to compare four m-trees that have been synthesised with different types of routing information. In each test, only one m-tree is constructed with some graphs of the reference set. The parameters used to construct this m-tree are:

- Type of Database: COIL-RAG, Letter, Fingerprint and GREC.
- Number of graphs used to construct the m-tree: 50 for COIL, Letter and GREC and 25 for Fingerprint. Graphs have been randomly extracted from the reference set.
- Type of routing element: Set Median (section 5.1), Hierarchical Median (section 5.2.1), Genetic Median (section 5.2.2) , Graduated Assignment Median (section 5.2.2), First Order Random Graph, Function-Descrived Graph (FDG) and Second Order Random Graph.
- Number of dendogram partitions: 7. The partitions are the number of cuts used to generate the m-tree (section 2). And also, it is the number of levels of the m-tree, due to the method used to generate the m-tree. The distances to set the cuts are the following (see figure 1.1); distance of cut 0 = $D_{max}$, distance of cut 1 = $D_{max}\cdot 6/7$, distance cut 2 = $D_{max}\cdot 5/7$, ... distance of cut 6 = $D_{max}/7$. Where $D_{max}$ is the maximum distance of any two graphs of the m-tree.

The parameters for DMX query are:

- Graph query. The graph has been extracted from the test set.
- Range of the query: $\mathbf{d_{max}} = 0.3\cdot D_{max}$.
- Metric-tree.

The parameters for KNN query are:

- Graph query. The graph has been extracted from the test set.
- Metric-tree.

We used three indices to evaluate and compare the performance of database models: Access ratio, Precision, Recall and F-measure.

#### 8.3.1.1 Access Ratio

This index evaluates the capacity of the m-tree to properly route the queries. Given a query graph *Q*, this index is the number of accessed nodes and leaves of the m-tree or the number of matching performed, *R*. Finally, it is normalized by the number of graphs used to generate the m-tree, *N*. If the Access ratio is higher than 1, then it is faster not to use an m-tree since without the m-tree, the system would perform less comparisons.

$$AccessRatio(Q,K,T) = \frac{R}{N} \qquad \textbf{Eq. 18}$$

### 8.3.1.2 Precision

Precision is the number of items correctly labeled as belonging to the class divided by the total number of elements labeled as belonging to the positive class for KNN and under acceptable distance for DMAX. We obtain this metrics as follows, from the *K* graphs returned by the m-tree with minimum distance between them and *Q* for DMAX, and the number of elements of the same class as *Q* for KNN, and the number of true positives *Tp*.

$$Precision(\, Q,K,T) = \frac{T_p}{K} \qquad \textbf{Eq. 19}$$

### 8.3.1.3 Recall

Recall is the number of items correctly labeled as belonging to the class divided by the total number of elements that actually belong to the class for KNN and under acceptable distance for DMAX. We obtain this metrics as follows, from the *C* graphs returned by all the reference set with minimum distance between them and *Q* for DMAX, and the number of elements of the same class as *Q* for KNN, and the number of true positives *Tp*.

$$Recall(\, Q,K,T) = \frac{T_p}{C} \qquad \textbf{Eq. 20}$$

### 8.3.1.4 F-Mesure

The F-measure is a measure of a test's accuracy computed through the Precision and Recall.

$$f - mesure = \frac{2\, Precision\; Recall}{Precision + Recall} = \frac{2\, Tp}{C+K} \qquad \textbf{Eq. 21}$$

## 8.3.2 DMAX Retrieval on Graphs Datbases

Results of graph retrieval given a max distance, it must return all the graph from the training set found at a distance less or equal than the max distance.

**Table 3.** Table presenting acces ratio results on DMAX queries. Highlighted in bold and underline appears the best expected result and only in bold the second one.

| | *Synthesis* | *COIL* | *Letter LOW* | *Letter HIGH* | *GREC* | *Fingerprint* |
|---|---|---|---|---|---|---|
| Median | Set Median | 0.99 | 0.82 | 1.37 | <u>**0.07**</u> | <u>**0.04**</u> |
| | Hierarchical | 0.99 | <u>**0.56**</u> | <u>**1.09**</u> | 1.07 | 0.25 |
| | Genetic | **0.58** | 1.34 | 1.44 | **0.79** | 0.44 |
| | Grad. Assign. | <u>**0.24**</u> | **0.71** | <u>**1.09**</u> | 0.95 | <u>**0.04**</u> |
| Non-prob. | Closure | - | - | - | - | - |
| Probabilistic | FORG | - | - | - | - | - |
| | SORG | - | - | - | - | - |

|  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|
|  | FDG | - | - | - | - | - |

**Table 4.** Table presenting precision results on DMAX queries. Highlighted in bold and underline appears the best expected result and only in bold the second one.

|  | *Synthesis* | *COIL* | *Letter LOW* | *Letter HIGH* | *GREC* | *Fingerprint* |
|---|---|---|---|---|---|---|
| Median | Set Median | **0.99** | **1** | **1** | 0.79 | 0.91 |
|  | Hierarchical | **0.99** | **1** | **1** | **0.98** | **0.93** |
|  | Genetic | 0.75 | **1** | **1** | 0.92 | **0.99** |
|  | Grad. Assign. | 0.75 | **1** | **1** | **0.98** | 0.91 |
| Non-prob. | Closure | - | - | - | - | - |
| Probabilistic | FORG | - | - | - | - | - |
|  | SORG | - | - | - | - | - |
|  | FDG | - | - | - | - | - |

**Table 5.** Table presenting recall results on DMAX queries. Highlighted in bold and underline appears the best expected result and only in bold the second one.

|  | *Synthesis* | *COIL* | *Letter LOW* | *Letter HIGH* | *GREC* | *Fingerprint* |
|---|---|---|---|---|---|---|
| Median | Set Median | **0.99** | **1** | **1** | **0.99** | **1** |
|  | Hierarchical | **0.99** | **1** | **1** | 0.91 | **1** |
|  | Genetic | 0.96 | 0.97 | **1** | **0.96** | 0.94 |
|  | Grad. Assign. | **1** | **1** | **1** | 0.91 | **1** |
| Non-prob. | Closure | - | - | - | - | - |
| Probabilistic | FORG | - | - | - | - | - |
|  | SORG | - | - | - | - | - |
|  | FDG | - | - | - | - | - |

**Table 6.** Table presenting f-mesure results on DMAX queries. Highlighted in bold and underline appears the best expected result and only in bold the second one.

|  | *Synthesis* | *COIL* | *Letter LOW* | *Letter HIGH* | *GREC* | *Fingerprint* |
|---|---|---|---|---|---|---|
| Median | Set Median | **0.99** | **1** | **1** | 0.88 | 0.95 |
|  | Hierarchical | **0.99** | **1** | **1** | **0.94** | **0.97** |
|  | Genetic | 0.84 | 0.99 | **1** | **0.95** | **0.97** |
|  | Grad. Assign. | 0.86 | **1** | **1** | **0.94** | 0.95 |
| Non-prob. | Closure | - | - | - | - | - |
| Probabilistic | FORG | - | - | - | - | - |
|  | SORG | - | - | - | - | - |
|  | FDG | - | - | - | - | - |

## 8.3.3 KNN Retrieval on Graphs Databases

The k-nearest neighbor algorithm (*k*-NN) is a method for classifying objects based on closest training examples in the feature space. To define "closest", a distance measure has to be defined between the elements of the set.

*k*-NN is a type of instance-based learning where the function is only approximated locally and all computation is deferred until classification. The k-nearest neighbor algorithm is one of the simplest of all machine learning algorithms: an object is classified by a majority vote of its neighbors', with the object being assigned to the class most common among its *k* nearest neighbors. Being the nearest depends on a previously specified distance measure. *k* is a small positive integer. If *k* = 1, then the object is simply assigned to the class of its nearest neighbor. The neighbors are taken from the training set of objects for which the correct classification is known.

**Table 7.** Table presenting acces ratio results on KNN queries. Highlighted in bold and underline appears the best expected result and only in bold the second one.

| | *Synthesis* | *COIL* | *Letter LOW* | *Letter HIGH* | *GREC* |
|---|---|---|---|---|---|
| Median | Set Median | 0.54 | 0.34 | **0.35** | 0.45 |
| | Hierarchical | **0.39** | **<u>0.31</u>** | **<u>0.32</u>** | **<u>0.37</u>** |
| | Genetic | 0.40 | 0.39 | 0.36 | 0.57 |
| | Grad. Assign. | 0.44 | 0.34 | 0.37 | 0.40 |
| Non-prob. | Closure | 0.64 | **0.33** | 0.57 | 0.80 |
| Probabilistic | FORG | 0.42 | 0.35 | 0.42 | 0.47 |
| | SORG | **<u>0.35</u>** | **0.33** | 0.36 | **0.38** |
| | FDG | 0.45 | 0.36 | 0.45 | 0.56 |

**Table 8.** Table presenting precision results on KNN queries. Highlighted in bold and underline appears the best expected result and only in bold the second one.

| | *Synthesis* | *COIL* | *Letter LOW* | *Letter HIGH* | *GREC* |
|---|---|---|---|---|---|
| Median | Set Median | 0.47 | 0.76 | 0.43 | 0.49 |
| | Hierarchical | 0.37 | 0.63 | 0.35 | 0.22 |
| | Genetic | 0.11 | 0.16 | 0.23 | 0.34 |
| | Grad. Assign. | 0.32 | **0.95** | **0.47** | 0.33 |
| Non-prob. | Closure | **0.63** | **<u>0.99</u>** | 0.39 | **<u>0.59</u>** |
| Probabilistic | FORG | 0.59 | 0.87 | 0.43 | **0.57** |
| | SORG | **<u>0.65</u>** | 0.81 | **<u>0.51</u>** | 0.37 |
| | FDG | 0.48 | 0.84 | 0.45 | 0.53 |

**Table 9.** Table presenting recall results on KNN queries. Highlighted in bold and underline appears

the best expected result and only in bold the second one.

| | *Synthesis* | *COIL* | *Letter LOW* | *Letter HIGH* | *GREC* |
|---|---|---|---|---|---|
| Median | Set Median | 0.26 | 0.58 | 0.32 | 0.50 |
| | Hierarchical | 0.24 | 0.48 | 0.26 | 0.22 |
| | Genetic | 0.06 | 0.12 | 0.18 | 0.34 |
| | Grad. Assign. | 0.18 | **0.72** | **0.36** | 0.34 |
| Non-prob. | Closure | **0.38** | **<u>0.76</u>** | 0.30 | **<u>0.60</u>** |
| Probabilistic | FORG | **0.38** | 0.66 | 0.32 | **0.58** |
| | SORG | **<u>0.40</u>** | 0.62 | **<u>0.38</u>** | 0.38 |
| | FDG | 0.30 | 0.64 | 0.32 | 0.54 |

**Table 10.** Table presenting f-mesure results on KNN queries. Highlighted in bold and underline appears the best expected result and only in bold the second one.

| | *Synthesis* | *COIL* | *Letter LOW* | *Letter HIGH* | *GREC* |
|---|---|---|---|---|---|
| Median | Set Median | 0.33 | 0.66 | 0.37 | 0.49 |
| | Hierarchical | 0.29 | 0.54 | 0.30 | 0.22 |
| | Genetic | 0.08 | 0.14 | 0.20 | 0.34 |
| | Grad. Assign. | 0.23 | **0.82** | **0.41** | 0.33 |
| Non-prob. | Closure | **0.47** | **<u>0.86</u>** | 0.34 | **<u>0.60</u>** |
| Probabilistic | FORG | 0.46 | 0.75 | 0.37 | **0.58** |
| | SORG | **<u>0.50</u>** | 0.70 | **<u>0.43</u>** | 0.38 |
| | FDG | 0.37 | 0.73 | 0.36 | 0.54 |

# 9. Conclusions

At this point, it’s time to make some reflections and evaluations about the obtained experimentally results.

First of all, it has been demonstrated that structured class prototypes continue to be usable and competitive even if we take in consideration embedding methods, note that embedding methods have an associated cost because they have to make comparisons element by element to determine the distances.

The first part of this work, the results of which are shown in Table 2, shows a comparative table between structured methods and most current embedding methods to make classification, comparing the prototypes of each class to the input query graph that we are seeking, looking for the nearest prototype. At this point, we can see that there are significant differences, between results presented by each prototype and that’s presents a little behavior variations between different databases. Like as we expected, the probabilistic methods, presents the best performance, because they store probabilistic information rather than compress it. We can conclude that SORG’s presents the best performance of all prototypes.

It is worth mentioning the case of FOGG prototype, which have not give the expected results, thats because at time to make the gaussian approximation, if the sampling values of training set don't approximate a Gaussian distribution, the covariance matrix is not defined as positive, and it's impossible to calculate the probability of the density function.

The second part of the work, refers to performance of m-trees of the prototypes, which we can deduce some things.

For max distance queries, it made no sense to put the results for m-trees of prototypes that uses as intermediate nodes, models different that graphs. That’s because, the input dmax of the query and the routing algorithm for it, is based on distances between two graphs structures.

Then, taking into account only the appropriate models, we can see that Generalized Median Graphs and Hierarchical Median Graphs works better than the Set Median and Genetic Median into the tables from 3 to 6. Then, when we make these assessments, so important is to maximize the precision and the recall, as minimizing the ratio shortcuts. Remember that a lower access ratio implies a fewer number of comparisons.

The KNN results presented into the tables from 7 to 10, we can see that Hierarchical method, presents the better access ratio results, that’s because the radius of the routing nodes is smaller than the other models, and Clousures give better results with precision and recall. But if we combine, access ratio results and precision and recall, we can conclude, that Second Order Random Graphs, give better performance than any other model.

Finally analyzing the common labeling algorithms, the Graduated Assignment for Common Labeling, gives a very good performance, in fact, part of the improvement of the results for classification, is due to this method.

On the other hand, genetic algorithms present an interesting idea to resolve complex problems, the drawback is that they need a large number of generations, with associated computation cost, to find some acceptable results. The other problem is that, they can fall into local minimums when they try to find optimal solutions. To improve genetic algorithms, I propose to start from a suboptimal result at the first generation.

# 10. List of Publications

From this work, we have generated the following publications:

- Francesc Serratosa, Albert Solé-Ribalta, Xavier Cortés: K-nn Queries in Graph Databases Using M-Trees. CAIP (1) 2011: 202-210
- Francesc Serratosa, Albert Solé-Ribalta, Xavier Cortés: Automatic Learning of Edit Costs Based on Interactive and Adaptive Graph Recognition. GbRPR 2011: 152-163
- F. Serratosa, X. Cortés, A. Solé, "Interactive Graph Matching by means of Imposing the Pairwise Costs", International Conference on Pattern Recognition, **ICPR2012**, Tsukuba, Japan, Accepted for publication, Volume , pp: -, 2012.

# 11. References

1 K. Konstantinidis, K., Gasteratos, A. & Andreadis, I.: Image retrieval based on fuzzy colour histogram processing. In: Optics Communications 248, pp: 375–386, (2005).

2 Gudivada, V.N., Raghavan, V.V.: Special issue on Content Based Image Retrieval Systems. In: Computer, vol. 28, no. 9, (1995).

3 Datta, R., Joshi, D. Li, J. & Wang, J.Z.: Image Retrieval: Ideas, Influences, and Trends of the New Age. In: ACM Computing Surveys, 40 (2), Article 5, (2008).

4 Tao, Y., Grosky, W.I.: Spatial Colour Indexing: A Novel approach for Content-Based Image Retrieval. In: Proc. IEEE international conference Multimedia computing and Systems, (1999).

5 Smith, J.R., Samet, H.: VisualSEEk: A Fully Automated Content-Based Image Query System. In: Proc. ACM Multimedia, pp. 87-98, (1996),

6 Le Saux, B., Bunke, H.: Feature selection for graph-based image classifiers. In: IbPRIA 2005. LNCS, vol. 3523, pp. 147–154 (2005).

7 Gold, S. & Rangarajan, A.: A Graduated Assignment Algorithm for Graph Matching. In: Transactions on Pattern Analysis and Machine Intelligence, 18(4), pp: 377 - 388, (1996).

8 Neuhaus, M., Riesen, K., Bunke, H.: Fast Suboptimal Algorithms for the Computation of Graph Edit Distance. In: In: Structural and Syntactic Pattern Recognition and Statistical Techniques in Pattern Recognition, LNCS 4109, pp. 163-172, (2006).

9 Berretti, S., Del Bimbo, A., Vicario, E.: Efficient Matching and Indexing of Graph Models in Content-Based Retrieval. In: IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 23, no. 10, pp. 1089-1105, (2001).

10 Zhao, J.L., Cheng, H.K.: Graph Indexing for Spatial Data Traversal in Road Map Databases. In: Computers & Operations Research, vol. 28, pp: 223-241, (2001).

11 Serratosa, F., Alquézar, R., Sanfeliu, A.: Function-described graphs for modeling objects represented by attributed graphs. In: Pattern Recognition, vol. 36 no. 3, pp. 781-798, (2003).

12 Serratosa, F., Alquézar, R., Sanfeliu, A.: Synthesis of Function-Described Graphs and clustering of Attributed Graphs. In: International Journal of Pattern Recognition and Artificial Intelligence, vol. 16, no. 6, pp. 621-655, (2002).

13 Sanfeliu, A., Serratosa, F., Alquézar, R.: Second-Order Random Graphs for modeling sets of Attributed Graphs and their application to object learning and recognition. In: International Journal of Pattern Recognition and Artificial Intelligence, vol. 18, no. 3, pp. 375-396, (2004).

14 Jiang, X., Münger, A., Bunke, H.: On median graphs: Properties, algorithms and applications. In: IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 23, no. 10, pp: 1144-1151, (2001).

15 Ferrer, M., Valveny, E., Serratosa, F., Riesen, K., Bunke, H.: Generalized Median Graph Computation by Means of Graph Embedding in Vector Spaces. In: Pattern Recognition, vol. 43, no. 4, pp. 1642-1655, (2010).

16 Ferrer, M., Valveny, E., Serratosa, F.: Median graphs: A genetic approach based on new theoretical properties. In: Pattern Recognition, vol. 42, no. 9, pp. 2003-2012, (2009).

17 Serratosa, F., Solé, A. & Vidiella, E.: Graph Indexing and Retrieval based on Median Graphs, In: Mexican Conference on Pattern Recognition, MCPR2010, Puebla, Mexico, LNCS 6256, pp: 311-321, (2010).

18 Bunke, H., Munger, A, Jiang, X: Combinatorial search versus genetic algorithms: A case study based on the generalized median graph problem. In: Pattern Recognition Letter, 20, pp: 1271-1277, (1999).

19 Solé, A. & Serratosa, F.: Graduated Assignment Algorithm for Finding the Common Labelling of a set of Graphs. In: Syntactic and Structural Pattern Recognition, SSPR2010, Izmir, Turkey, LNCS 6218, pp: 180-190, (2010).

20 Ciaccia, P., Patella, M., Zezula, P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In: Proc. 23rd VLDB Conference, pp. 426-435, (1997).

21 Cortés, X., Serratosa, F. & Solé, A.: K-nn Queries based on Metric-trees and Graph Prototyping, Submitted to Pattern Recognition Letters, 2011.

22 Bozcaya, T & Ozsoyoglu, M: Indexing Large Metric Spaces for Similarity Search Queries. In: ACM transactions on Database Systems, 24 (3), pp: 361-404, (1999).

23 Horst Bunke, Kaspar Riesen: Towards the unification of structural and statistical pattern recognition, Pattern Recognition Letters 33 (2012) 811–825

24 Lee, S.Y., Hsu, F.: Spatial Reasoning and Similarity Retrieval of Images using 2D C-Strings Knowledge Representation. In: Pattern Recognition, vol. 25 no. 3, pp. 305-318, (1992).

25 He, H., Singh, A.K.: Closure-Tree: An Index Structure for Graph Queries. In: Proc. International Conference on Data Engineering, pp. 38, (2006).

26 Shasha ,D., Wang, J.T.L., Giugno, R.: Algorithmics and applications of tree and graph searching. In: ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 39-52, (2002).

27 K.A. DeJong and W.M. Spears, ªUsing Genetic Algorithms to Solve NPComplete Problems,º Proc. Third Int'l Conf. Genetic Algorithms, pp. 124-132, 1989.

28 A.D.J. Cross, R.C. Wilson, and E.R. Hancock, ªInexact Graph Matching Using Genetic Searchº Pattern Recognition, vol. 30, no. 6, pp. 953-970, 1997.

29 Y.-K. Wang, K.-C. Fan, and J.-T. Horng, ªGenetic-Based Search for Error- Correcting Graph Isomorphism,º IEEE Trans. Systems, Man, and Cybernetics, vol. 27, no. 4, pp. 588-597, 1997.

30 Xiaoyi Jiang, Andreas MuÈnger, and Horst Bunke, "On Median Graphs: Properties, Algorithms, and Applications", IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 23, NO. 10, OCTOBER 2001

31 M. Mitchell, An Introduction to Genetic Algorithms. MIT Press, 1996.

32 He, H., Singh, A.K.: Closure-Tree: An Index Structure for Graph Queries. In: Proc. International Conference on Data Engineering, pp. 38, (2006).

33 Serratosa Casanelles, Francesc, Alquézar Mancho, Renato, Sanfeliu Cortés, Alberto, "Function-described graphs for modelling objects represented by attributed graphs", Pattern Recognition 36(3): 781-798 (2003)

34 Serratosa Casanelles, Francesc, Alquézar Mancho, Renato, Sanfeliu Cortés, Alberto, "Second-order random graph for modeling sets of attributed graphs and their application to object learning and recognition", International Journal of Pattern Recognition and Artificial Intelligence 18(3): 375-396 (2004)

35 Andrew D. Bagdanov, Marcel Worring, "First order Gaussian graphs for efficient structure classification", Pattern Recognition Volume 36, Issue 6, June 2003, Pages 1311–1324

36 Ciaccia, P., Patella, M., Zezula, P.: M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In: Proc. 23rd VLDB Conference, pp. 426-435, (1997).

37 Riesen, K. & Bunke, H.: IAM graph database repository for graph based pattern recognition and machine learning. In N. Structural, Syntactic, and Statistical Pattern Recognition, SSPR2008, LNCS 5342, pp: 287 – 297, (2008).

[38] Kaspar Riesen, and Horst Bunke, Graph Classification Based on Vector Space Embedding., IJPRAI, Vol. 23, Nr. 6 (2009) , p. 1053-1081.