# GRAPH RECOGNITION BY SERIATION AND FREQUENT SUBSTRUCTURES MINING

Lorenzo Livi, Guido Del Vescovo and Antonello Rizzi
*Department of Information Engineering, Electronics and Telecommunications,*
*SAPIENZA University, Via Eudossiana 18, 00184 Rome, Italy*

Keywords: Inexact graph matching, Graph seriation, Frequent substructures mining, Embedding, Granular computing, Classification.

Abstract: Many interesting applications of Pattern Recognition techniques can take advantage in dealing with labeled graphs as input patterns. To this aim the most important issue is the definition of a dissimilarity measure between graphs. In this paper we propose a representation technique able to characterize the input graphs as real valued feature vectors, allowing the use of standard classification systems. This procedure consists in two distinct stages. In the first step a labeled graph is transformed into a sequence of its vertices, ordered according to a given criterion. In a second step this sequence is mapped into a real valued vector. To perform the latter stage, we propose a novel Granular Computing procedure searching for frequent substructures, called *GRADIS*. This algorithm is in charge of the inexact substructures identification and of the embedding of the sequenced graphs using the symbolic histogram approach. Tests have been performed by synthetically generating a set of graph classification problem instances with the aim to measure system performances when dealing with different types of graphs, as well when increasing problem hardness.

## 1 INTRODUCTION

Many Pattern Recognition problems have been formulated using labeled graphs as patterns. Despite their relative simplicity and intuitive understanding, their analysis imposes a new type of approach to standard Pattern Recognition techniques, such as classification and clustering. For this purpose, the research field related to Pattern Recognition systems able to deal with structured patterns is growing fast. A key context in this scenario is the one of the *Inexact Graph Matching* measures, aiming to establish a (dis)similarity measure between graphs. Different *graph mining* issues, such as for example *Frequent Subgraphs Mining* (Kuramochi and Karypis, 2002), are of great interest considering both their isolated application, and in conjunction with more complex modeling systems. For example the extrapolation of a set of *significative prototypes* (Riesen and Bunke, 2010) of a given input dataset, permits to establish what is called a *relative* reference framework, tailored to the specific problem at hand.

Our intuition is that this set of prototypes, starting from an input set of graphs, can be determined transforming the original domain of the problem. Following this approach, we have developed a representation technique, able to characterize the input graphs as real valued feature vectors. Once the vector-representation of the graphs is obtained, the inexact graph matching between two graphs can be issued using any inductive modeling system relying on standard tools, fully taking advantage of the algebraic structure usually adopted in $\mathbb{R}^n$.

In Section 2 the scientific context is introduced. In Section 3 the proposed methodology is described. The experimental evaluation of the proposed methodology is described in Section 4. Finally in Section 5 we draw our conclusions.

### 1.1 Preliminary Definitions

In this section we will give some basic preliminary definitions, mainly regarding (labeled) graphs. A labeled graph is a tuple $G = (V, E, \mu, \nu)$ , where $V$ is the (finite) set of vertices (also referred as nodes), $E \subseteq V \times V$ is the set of edges, $\mu : V \to \mathcal{L}_V$ is the vertex labeling function, with $\mathcal{L}_V$ the vertex-labels set and $\nu : E \to \mathcal{L}_E$ is the edge labeling function, with $\mathcal{L}_E$ the edge-labels set. If $\mathcal{L}_V = \mathcal{L}_E = \emptyset$ then $G$ is referred as an *unlabeled* graph. If $\mathcal{L}_E = \mathbb{R}_0$, then $G$ is called



Livi L., Del Vescovo G. and Rizzi A. (2012).
GRAPH RECOGNITION BY SERIATION AND FREQUENT SUBSTRUCTURES MINING.
In *Proceedings of the 1st International Conference on Pattern Recognition Applications and Methods*, pages 186-191
DOI: 10.5220/0003733201860191

*weighted* graph. If $\mathcal{L}_E \neq \emptyset \wedge \mathcal{L}_V = \emptyset$ the graph $G$ is referred as *edge-labeled* graph. If $\mathcal{L}_E = \emptyset \wedge \mathcal{L}_V \neq \emptyset$ the graph $G$ is referred as *vertex-labeled* graph. Finally if both sets are non-empty we can refer to $G$ as a *fully-labeled* or simply *labeled* graph. The notations $V(G)$ and $E(G)$ will refer to the set of vertices and edges of the graph $G$. The cardinalities of $V(G)$ and $E(G)$ are referred, respectively, as the *order* and the *size* of the graph. The *adjacency matrix* of $G$ is denoted with $\mathbf{A}^{n\times n}$, where $|V(G)| = n$, and if $G$ is weighted we have the *weighted adjacency matrix* $A_{ij} = \nu(e_{ij})$, sometimes denoted as $\mathbf{W}$. The *transition matrix* is denoted with $\mathbf{T}^{n\times n}$ and is defined as $\mathbf{T} = \mathbf{D}^{-1}\mathbf{A}$, where $\mathbf{D}$ is a diagonal matrix of vertices degree, $D_{ii} = deg(v_i) = \sum_j A_{ij}$. If it is not explicitly defined, in this paper a graph is assumed to be simple and edge-labeled.

## 2 SCIENTIFIC CONTEXT

The recent research on inductive modeling has defined many automatic systems able to deal with classification functions defined on $\mathbb{R}^n$. However, many of the classification problems coming from practical applications deal with *structured patterns*, such as images, audio/video sequences and biological chemical compounds. Usually, in order to take advantage of the existing data driven modeling systems, each pattern of a structured domain $\mathcal{S}$ is reduced to an $\mathbb{R}^m$ vector by adopting a *preprocessing* function $\phi : \mathcal{S} \to \mathbb{R}^m$. The design of these functions is a challenging problem, mainly due to the *semantic and informative gap* between $\mathcal{S}$ and $\mathbb{R}^m$. The key to design an automatic system dealing with these classification problems is *information granulation*. Granular Computing (Bargiela and Pedrycz, 2003) is a novel paradigm concerned with the analysis of complex data, usually characterized by the need of different levels of representation. Granular computing approach aims to group low level information atomic elements into semantically relevant structures. These groups of entities are called *information granules*. Granular modeling consists in finding the *correct* level of information granulation, *i.e.* a way to map a raw data level domain into a higher semantic level, and in defining a proper inductive inference directly into this symbolic domain.

The graph matching problem is an important research field characterized by both theoretical and practical issues. The numerous matching procedures proposed in the technical literature can be classified into two well defined families, those of *exact* and *inexact matching*. The first one is just a *boolean* equality test given two input graphs, while the latter is a more complex and interesting problem where the challenge is in computing *how much* they differ. In the current scientific literature it is possible to distinguish three mainstream approaches: *Graph Edit Distance* based (Neuhaus et al., 2006; Riesen and Bunke, 2009), *Graph Kernels* based (Borgwardt et al., 2005; Vishwanathan et al., 2008) and *Graph Embedding* based (Riesen and Bunke, 2010; Del Vescovo and Rizzi, 2007).

## 3 GRAPH EMBEDDING METHOD

In this section we will explain the proposed methodology to solve the inexact graph matching problem. The basic and intuitive idea is that if two given graphs are similar, they should share many similar *substructures*. To be able to understand the similarity of the two graphs in terms of their common substructures, we first transform each graph into a *sequence* of its vertices, and then we perform a recurrence analysis of the *inexact* substructures of these sequenced graphs. The proposed method is thus a two-stage algorithm, where the first one is dominated by a *graph seriation* procedure, and the last one by a sequence mining and representation of the identified subsequences. The proposed method can be represented as two *mapping* functions. The first one, say $f_1 : \mathcal{G} \to \Sigma$, maps a graph $G \in \mathcal{G}$ to a sequence of vertices identifiers $s \in \Sigma$. The second one, say $f_2 : \Sigma \to \mathcal{E}$, maps each sequence $s \in \Sigma$ to a numeric vector $\underline{\mathbf{h}} \in \mathcal{E}$, where usually $\mathcal{E} \subseteq \mathbb{R}^n$. Therefore, eventually, the problem is posed on a standard domain, such as for example an Euclidean space, where the whole set of known Pattern Recognition methods can be applied directly. Consequently, this methodology should be seen as a mixture of Graph Embedding and Graph Edit Distance approaches.

### 3.1 Graph Seriation

Given a graph $G$, the aim is to establish an order on the set of vertices $V(G)$, with $|V(G)| = n$, such that the derived sequence of vertices $s = (v_{i_1}, v_{i_2}, ..., v_{i_n})$ respects a given property of the graph. For example, an interesting approach is the one that analyzes the *spectrum* (*i.e.* the set of its eigenvalues/eigenvectors) of the matrix representation of the graph (Robles-Kelly and Hancock, 2005). The leading eigenvector $\phi$ of the adjacency matrix contains the information about the structural connectivity of each vertex of the graph. Similarly, analyzing the (symmetric) transition matrix $\mathbf{T}^{n\times n}$, it is possible to obtain information about the a priori probability of a given vertex in a random walk

scenario. In general this approach can be used for unweighted and weighted graphs. However, we observe that it is possible to extend this method also to the graphs with real vectors as weights on the edges, taking into account the (Euclidean) norm $W_{ij} = \| \nu(e_{ij}) \|$ of the vector, assuming that the highest is the norm, the stronger is the relationship.

In this paper, we will make use of this seriation method (Robles-Kelly and Hancock, 2005) for three types of edge-labeled graphs: unweighted, weighted with a scalar in $[0,1]$ and with a vector in $[0,1]^n$. Note that the vertices can be arbitrarily labeled, but they are irrelevant in this particular phase.

## 3.2 Sequence Mining and Embedding

In order to represent a seriated graph as a real valued vector, we propose a method called *GRADIS* (GRanular Approach for DIscrete Sequences), aimed at performing two crucial tasks: the inexact substructures identification, called the *alphabet* of symbols, and the embedding of the sequences using the *symbolic histogram* approach (Del Vescovo and Rizzi, 2007). With a little bit of formalism, the proposed methodology, seen from an high level of abstraction, can be characterized using a *mapping function* $f : \Sigma \to \mathcal{E}$ that maps each sequence to a numeric vector, with $\mathcal{E} \subseteq \mathbb{R}^n$. This way, the problem is mapped into the Euclidean domain.

Given an input dataset of sequences (*i.e.* sequences of characters) $\mathcal{S} = \{s_1, ..., s_q\}$, generated from a finite alphabet $\Omega$, the first task is to identify a set $\mathcal{A} = \{a_1, ..., a_d\}$ of *symbols*. These symbols are pattern substructures that are recurrent in the input dataset. The identification of this set is performed using an inexact matching strategy in conjunction with a clustering procedure. Given a lower $l$ and upper $L$ limit for the length of the subsequences, a variable length $n$-gram analysis is performed on each input sequence of the dataset. An $n$-gram of a given sequence $s$ is defined as a contiguous subsequence of $s$ of length $n$. For instance, if an input sequence is $s = (A,B,C,D)$, with $l = 2, L = 3$, we obtain the set of $n$-grams $\mathcal{N}_s = \{(A,B),(A,B,C),(B,C),(B,C,D),(C,D)\}$. The cardinality of this set is then $|\mathcal{N}_s| = \sum_{i=l}^{L}(n-i)+1 = O(n^2)$, where $L \leq n = len(s)$ ($len(s)$ is the number of elements into the sequence $s$). However, normally $L \ll n$ and thus $|\mathcal{N}_s| \simeq c \cdot n$, where $c > 1$ is a constant factor. If $M$ is the maximum length of a sequence of the input dataset $\mathcal{S}$, the cardinality of the whole set of $n$-grams $\mathcal{N} = \{n_1, ..., n_{|\mathcal{N}|}\}$ of $\mathcal{S}$ is upper bounded by $|\mathcal{N}| \leq |\mathcal{S}| \cdot \sum_{i=l}^{L}(M-i)+1 \simeq |\mathcal{S}| \cdot c \cdot M$.

Since the cardinality of $\mathcal{N}$ can become large, it is convenient to perform this analysis on a subset of $\mathcal{N}$, say $\mathcal{N}^*$. For this purpose, a probabilistic selection of each $n$-gram can be performed. With a user-defined probability $p$ an $n$-gram is selected, conversely with probability $1-p$ it is not selected. So, given a set of $m$ $n$-grams and a selection probability $p$, the chosen $n$-grams can be described by a Binomial distribution. In fact the expected number of selected $n$-grams in $\mathcal{N}^*$ is $|\mathcal{N}| \cdot p$, with variance $|\mathcal{N}| \cdot p \cdot (1-p)$.

The set $\mathcal{N}^*$ is subjected to a clustering procedure based on the *BSAS* algorithm. The Levenshtein distance is used as the dissimilarity measure between $n$-grams and clusters are represented by the *MinSOD* (Minimum Sum Of Distances) element. The clustering procedure aims to identify a list of different partitions of $\mathcal{N}^*$, say $L = (\mathcal{P}_1, ..., \mathcal{P}_z)$. In fact the outcome of the *BSAS* algorithm is known to strongly depend on a clustering parameter called $\Theta$. The optimal value of $\Theta$ is automatically determined using a logarithmic search algorithm. The $\Theta$ search interval (usually $[0,1]$) is recursively split in halves, stopping each time that two successive values of the parameter yield the same partition. The recursive search stops anyway when the distance between two successive $\Theta$ values falls under a given threshold. Each partition $\mathcal{P}_i = \{\mathcal{C}_1, ..., \mathcal{C}_{u_i}\}, i = 1 \to z$, is composed by $u_i$ clusters of $n$-grams. Successively, these clusters are subjected to a validation analysis, that takes into account the *quality* of the cluster. For this purpose, for each cluster $\mathcal{C}_j \in \mathcal{P}_i$ a measure of cluster compactness cost $K(\mathcal{C}_j)$ and size cost $S(\mathcal{C}_j)$ is evaluated. Starting from these measures, the following convex combination is defined as the total cluster cost

$$\Gamma(\mathcal{C}_j) = (1-\mu) \cdot K(\mathcal{C}_j) + \mu \cdot S(\mathcal{C}_j) \qquad (1)$$

where $K(\mathcal{C}_j) = \frac{1}{n-1}\sum_{i=1}^{n-1} d(n_i, n_{\mathcal{C}_j}^{SOD})$, $n_{\mathcal{C}_j}^{SOD}$ is the MinSOD element of the cluster $\mathcal{C}_j$ and $d(\cdot,\cdot)$ is the Levensthein distance. The size cost is defined as $S(\mathcal{C}_j) = 1 - \frac{|\mathcal{C}_j|}{|\mathcal{N}^*|}$. If the cost $\Gamma(\mathcal{C}_j)$ is lower than a given threshold $\tau$, the cluster is retained, conversely is rejected. Each (accepted) cluster $\mathcal{C}_j$ is modeled by a representative element, that is an $n$-gram $n_{\mathcal{C}_j}^{SOD}$ that minimizes the sum of distances between all the other elements in the cluster. This representative element, say $a_j = n_{\mathcal{C}_j}^{SOD}$, is considered as a symbol of the input dataset and it is added to the alphabet $\mathcal{A}$. At the end of this clustering procedure, the alphabet $\mathcal{A}$ of the input dataset is determined. Actually, each symbol $a_j$ is defined as a triple $(n_{\mathcal{C}_j}^{SOD}, K(\mathcal{C}_j) \cdot \varepsilon, 1 - \Gamma(\mathcal{C}_j))$, where $K(\mathcal{C}_j) \cdot \varepsilon$ is a factor used in the subsequent embedding phase ($\varepsilon$ is again a user-defined tolerance) and $1 - \Gamma(\mathcal{C}_j)$ is the ranking score (quality value of the cluster $\mathcal{C}_j$) of the symbol $a_j$ in the alphabet $\mathcal{A}$. Our

analysis aims to aggregating as much as possible similar subsequences of the input dataset. For this reason, each selected cluster should be seen as an information granule of $n$-grams.

The second stage of the procedure is aimed at producing an explicit embedding of each input string $s_i$ into a relative dissimilarity space (Pękalska and Duin, 2005). This is done by counting the number of (inexact) occurrences of each symbol $a_j \in \mathcal{A}$ into the input string $s_i$. If $|\mathcal{A}| = d$, where $d \leq \sum_{i=1}^{z} u_i$ is the number of selected clusters representatives, the embedding space $\mathcal{E}$ is a $d$-dimensional dissimilarity space, and each input sequence $s_i$ is represented as a numerical vector $\underline{\mathbf{h}}_i$. Therefore, the set of symbols $\mathcal{A}$ should be seen as a set of *prototypes*, used to explicitly model the input data. The inexact matching is again performed using the Levensthein distance. The counting of the number of occurrences of $a_j$, extracted from cluster $\mathcal{C}_j$, into $s_i$ is performed as follows. Let $\nu = len(s_i) \cdot \lambda$ be the length tolerance adopted in order to find matches in $s_i$, where $\lambda$ is a user-defined parameter in $[0,1]$. The selected set of $n$-grams $\mathcal{N}_{s_i}$ of $s_i$, of variable length $l$ between $len(s_i) - \nu \leq l \leq len(s_i) + \nu$, is extracted and the inexact matching is performed against $a_j$ and each $o \in \mathcal{N}_{s_i}$. If the alignment cost is lower or equal to $K(\mathcal{C}_j) \cdot \Theta$, the counting is incremented by 1. The value $K(\mathcal{C}_j) \cdot \Theta$ is used to automatically enable a cluster-dependent thresholding mechanism. In fact, the more the cluster is compact, the higher the matching score should be in order to associate that particular $n$-gram with the particular cluster. Conversely, if the cluster is not so compact (*i.e.* a noisy cluster), a larger matching tolerance should be used in order to recognize the occurrence. For instance, a cluster of perfectly identical elements would yield $K(\mathcal{C}_j) = 0$, requiring an exact matching to recognize the occurrence of the symbol.

*GRADIS* should be seen as a data driven modeling system aimed at producing the alphabet of symbols and an explicit representation model of the given input dataset $\mathcal{S}$. This model can be used in different Pattern Recognition problems, such as for example the ones of classification and clustering. In summary, in order to perform *GRADIS* on an input set of graphs $\mathcal{G} = \{G_1, ..., G_m\}$, we firstly seriate each graph $G_i$ producing a sequence of its vertices $s_i = (v_{i_0}, v_{i_1}, ..., v_{i_n})$. Then we apply the described procedure to finally embed $s_i$ into the dissimilarity space $\mathcal{E}$, obtaining the symbolic histogram representation $\underline{\mathbf{h}}_i$. On the dissimilarity space $(\mathcal{E}, d(\cdot,\cdot))$ any given Pattern Recognition problem can be defined directly, using any (metric) distance $d : \mathcal{E} \times \mathcal{E} \to \mathbb{R}_0^+$. Usually, the embedding space $\mathcal{E}$ is a large dimensional space, and an automatic dimensionality reduction technique using a feature selection scheme is normally a good choice.

# 4 TESTS AND RESULTS

The performance tests are performed by facing classification problems using different synthetically generated datasets of graphs $\mathcal{G}$. As we have said before, three kinds of edge-labeled graphs are taken into account: unweighted, with a scalar weight in $[0,1]$ and weighted with a vector in $[0,1]^n$, taking $n = 10$. The general classification scheme is composed by a training phase with a validation set ($\mathcal{S}_v$), where feature selection is performed over the symbolic histogram representation of the input patterns. The subset of the features is selected optimizing a fitness function using a genetic algorithm. If $\underline{\mathbf{c}} \in \{0,1\}^d$, with $d = |\mathcal{A}|$, is the *bit-mask* of the features that better performs on the validation set, and $\underline{\mathbf{h}}$ is the symbolic histogram representation of an input pattern, the final synthesized model will be fed directly by the reduced vector of features $\hat{\underline{\mathbf{h}}} = \underline{\mathbf{h}} \perp \underline{\mathbf{c}}$. Note that the symbol $\perp$ means that we project the original vector $\underline{\mathbf{h}}$ into the subspace induced by selecting only the features denoted by the bit-mask $\underline{\mathbf{c}}$. The fitness function adopted for the feature selection is defined as $f(\hat{\underline{\mathbf{h}}}) = 1 - (\alpha Err_{\mathcal{S}_v} + (1-\alpha)FC)$, where $FC$ stands for the fraction of selected features using the bit-mask $\underline{\mathbf{c}}$. It is a linear convex combination of the classification error rate $Err_{\mathcal{S}_v}$ on the validation set (as an estimate of the generalization capability when selecting a given feature subset $\hat{\underline{\mathbf{h}}}$) and of a complexity measure of $\hat{\underline{\mathbf{h}}}$. The overall performance measure on the test set ($\mathcal{S}_{ts}$) is defined as the complement of the classification error rate $Err_{\mathcal{S}_{ts}}$, that is as $f(\hat{\underline{\mathbf{h}}}) = 1 - Err_{\mathcal{S}_{ts}}$.

Software implementation is based on the C++ *SPARE* library (Del Vescovo et al., 2011). The tests are executed on a machine with an Intel Core 2 Quad CPU Q6600 2.40GHz and 4 Gb of RAM over a Linux OS.

## 4.1 Problem Definition and Datasets

Each dataset of graphs $\mathcal{G} = \{G_1, G_2, ..., G_m\}$ is generated in a stochastic fashion based on *Markov Chains*. Assuming to deal with graphs of the same order, that is $|V(G_i)| = |V(G_j)| = n, \forall i, j$, the Markov generation process is entirely described by its transition matrix $\mathbf{T}^{n \times n}$. The size of each graph $G$ is chosen with uniform probability in the range $n - 1 \leq |E(G)| \leq \binom{n}{2}$. Firstly, the $n$ vertices of the graphs are generated, selecting the first vertex of the random walk, say $v_{j_0}$, with uniform probability. Then, the $i$-th vertex in the

random walk, $v_{j_i}$, is selected using the transition probabilities of the given transition matrix $\mathbf{T}$, that is with probability $T_{j_{i-1},j_i}$. During the random walk the edges are created, together with the chosen labels that characterize the type of graphs. This Markov graphs generation strategy will characterize the graphs with peculiar walks, that are indeed the distinctive feature in the recognition problem. In this way, we have considered two distinct *stationary* Markov chains that we see as *classes* of graphs. Our classification problem is then defined as a two class problem, where each class is labeled with one of the two Markov chains. Clearly the objective is to recognize graphs that were generated from the same Markov chain.

## 4.2 Results for Different Graph Types

The first test is carried out over the three types of graphs, and for each type the test is repeated five times with different random seeds. This choice is motivated by the fact that different parts of the proposed algorithm are characterized by a random behavior, so we need to estimate the overall performance measure relying on different runs and computing the average and variance of the results. For this purpose, we have generated a random two-class set of datasets (in total $5 \cdot 3 = 15$ datasets), with 1000, 500 and 500 graphs, of order 100, for the training, validation and test set, respectively. As explained in Section 4.1, the size of the graphs is randomly determined. The graphs are equally distributed between the two classes. In Table 1 are shown the classification results achieved using *SVM* and two training algorithms, namely *ARC* and *PARC* (Rizzi et al., 2002), used to adopt a neuro-fuzzy Min-Max model. We report the averaged classification accuracy and, in the column indexed with the symbol $\sigma$, its standard deviation over the different test sets. For what concerns both *SVM* and *ARC*, it is possible to note a great accuracy in the classification of these randomly generated graphs, with a low deviation of the results. These tests show that the proposed embedding procedure is effective when feeding even very different classification systems, based on distinct modeling strategies, as the considered ones.

## 4.3 Results with Increasing Difficulties

The aim of the second test is to progressively increase the hardness of the test, reducing the differences between the transition matrices that generate the input graphs $\mathcal{G}$, that is, reducing the characteristic traits inside the generated graphs. For this test, we focus on an equally distributed two-class classification problem only for weighted graphs. The order of the graphs

Table 1: Classification Accuracies Using *SVM*, *ARC*, *PARC*.

| Algorithm | Graph Type | % Accuracy | $\sigma$ |
|---|---|---|---|
| **SVM** | Unweighted | 100 | 0.0000 |
| | Weighted | 97.92 | 0.2683 |
| | Vector-Weighted | 98.64 | 0.3847 |
| **ARC** | Unweighted | 99.40 | 0.2607 |
| | Weighted | 98.70 | 0.3551 |
| | Vector-Weighted | 99.20 | 0.1525 |
| **PARC** | Unweighted | 96.08 | 1.6769 |
| | Weighted | 95.56 | 1.3069 |
| | Vector-Weighted | 96.12 | 0.6418 |

is 100 and, as usual, their size is randomly determined, with 1000, 500 and 500 graphs for the training, validation and test sets, respectively. To control the hardness of the classification problem, we produce a sequence of generating transition matrices with different level of randomness. A transition matrix is said to be fully random if the transition probabilities are uniform. Following this fact, for each classification problem instance, we generate the two transition matrices (one for each class of graphs) introducing two real parameters, $\alpha, \beta \in [0,1]$, controlling the similarity of the transition matrices. Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be two different permutation matrices, and let $\mathbf{U}$ be the uniform transition matrix, with a zero diagonal. We firstly generate two intermediate matrices, say $\mathbf{A}$ and $\mathbf{B}$ as follows

$$\begin{aligned} \mathbf{A} &= \alpha\mathbf{U} + (1-\alpha)\mathbf{P}_1 \\ \mathbf{B} &= \alpha\mathbf{U} + (1-\alpha)\mathbf{P}_2 \end{aligned} \tag{2}$$

Finally we obtain the two transition matrices, characterized by a desired similarity, as follows

$$\begin{aligned} \mathbf{T}_1 &= (\beta/2)\mathbf{A} + (1-(\beta/2))\mathbf{B} \\ \mathbf{T}_2 &= (\beta/2)\mathbf{B} + (1-(\beta/2))\mathbf{A} \end{aligned} \tag{3}$$

In the following tests different combinations of $\alpha$ and $\beta$, as shown in Table 2, are used, generating 118 graph classification problem instances. Only the *SVM* classifier is used. In Table 2 is shown a sampling of the results, together with the total computing time for the classification model training, with the genetic feature selection optimization, and finally the number of selected features per single test. The average classification error is 13.75 %. The classification error increases considerably, when $0 < \alpha < 1$, only for $\beta = 1$, that is the recognition procedure exhibits a very robust behavior. The unperformant results obtained for the first 11 runs, that is for $\alpha = 0$ and $\beta = 0 \rightarrow 1$, are well explainable. Considering Equation 2, for $\alpha = 0$ we obtain two intermediate matrices that are equal and are again permutation matrices. For any considered value of $\beta$ in Equation 3, we obtain again the same permutation matrices. Thus, also in this case the com-

plexity of recognition task is high, and the obtained results confirm this fact.

Table 2: Classification Results.

| α | β | # Feature | Time (sec.) | % Error |
|---|---|---|---|---|
| 0 | 0.0 | 3850 | 3820 | 0.0 |
| | 0.2 | 3891 | 6733 | 50.8 |
| | 0.4 | 3951 | 7021 | 47.4 |
| | 0.6 | 3925 | 7228 | 43.2 |
| | 0.8 | 3877 | 6982 | 40.0 |
| | 1 | 3964 | 6759 | 58.6 |
| 0.2 | 0.0 | 3103 | 2624 | 4.8 |
| | 0.2 | 3145 | 3142 | 3.6 |
| | 0.4 | 3145 | 3052 | 2.2 |
| | 0.6 | 3231 | 3130 | 3.6 |
| | 0.8 | 3175 | 3218 | 2.0 |
| | 1 | 3638 | 2260 | 59.8 |
| 0.5 | 0.0 | 3774 | 1908 | 2.2 |
| | 0.2 | 3748 | 1863 | 2.2 |
| | 0.4 | 3789 | 1919 | 0.8 |
| | 0.6 | 3699 | 1910 | 2.2 |
| | 0.8 | 3716 | 1913 | 1.0 |
| | 1 | 3891 | 1218 | 40.4 |
| 0.7 | 0.0 | 3774 | 2497 | 2.4 |
| | 0.2 | 3697 | 2462 | 2.4 |
| | 0.4 | 3692 | 2346 | 1.4 |
| | 0.6 | 3754 | 2381 | 1.2 |
| | 0.8 | 3817 | 2496 | 0.8 |
| | 1 | 3811 | 1606 | 40.4 |
| 1 | 0.0 | 3854 | 1632 | 40.0 |
| | 0.2 | 3848 | 1520 | 40.0 |
| | 0.4 | 3823 | 1546 | 39.8 |
| | 0.6 | 3801 | 1473 | 39.8 |
| | 0.8 | 3792 | 1613 | 39.2 |
| | 1 | 3756 | 1468 | 40.0 |

## 5 CONCLUSIONS

The obtained results, for both tests, show that this methodology is valid, robust and accurate, showing a controllable and predictable behavior. The two transformations, that is the seriation of a graph $G_i$ into a sequence of its vertices identifiers, say $s_i$, and the final mapping into the dissimilarity space $\mathcal{E}$ aimed at obtaining a histogram vector $\underline{\mathbf{h}}_i$, seems to be able to capture and use the key information of the input dataset $\mathcal{G}$, generating a representation model that can be used for different analysis. It is important to underline that the true engine of this representation technique, that is the *GRADIS* procedure, can be employed as the core of a wider inductive modeling system, as well on its own to solve data mining and knowledge discovery problems. Moreover, in this paper we have described *GRADIS* as a procedure able to deal with sequences of characters. The same scheme can be adopted on a set of more complex patterns, such as fully labeled graphs or any other complex discrete object, once a definition of substructure, a dissimilarity measure between two substructures and a way to represent a cluster of substructures are provided. Future directions will deal also with such a generalization and a substantial amount of effort will be devoted to the development of a faster algorithm for the alphabet identification step.

## REFERENCES


Bargiela, A. and Pedrycz, W. (2003). *Granular computing: an introduction*. Number v. 2002 in Kluwer international series in engineering and computer science. Kluwer Academic Publishers.

Borgwardt, K. M., Ong, C. S., Schönauer, S., Vishwanathan, S. V. N., Smola, A. J., and Kriegel, H.-P. (2005). Protein function prediction via graph kernels. *Bioinformatics*, 21:47–56.

Del Vescovo, G., Livi, L., Rizzi, A., and Frattale Mascioli, F. M. (2011). Clustering structured data with the SPARE library. In *Proceeding of 2011 4th IEEE Int. Conf. on Computer Science and Information Technology*, volume 9, pages 413–417.

Del Vescovo, G. and Rizzi, A. (2007). Automatic classification of graphs by symbolic histograms. In *Proceedings of the 2007 IEEE International Conference on Granular Computing*, GRC '07, pages 410–416, San Jose, CA, USA. IEEE Computer Society.

Kuramochi, M. and Karypis, G. (2002). An efficient algorithm for discovering frequent subgraphs. Technical report, IEEE Transactions on Knowledge and Data Engineering.

Neuhaus, M., Riesen, K., and Bunke, H. (2006). Fast suboptimal algorithms for the computation of graph edit distance. In *Structural, Syntactic, and Statistical Pattern Recognition. LNCS*, pages 163–172. Springer.

Pękalska, E. and Duin, R. (2005). *The dissimilarity representation for pattern recognition: foundations and applications*. Series in machine perception and artificial intelligence. World Scientific.

Riesen, K. and Bunke, H. (2009). Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Comput.*, 27:950–959.

Riesen, K. and Bunke, H. (2010). *Graph Classification and Clustering Based on Vector Space Embedding*. Series in Machine Perception and Artificial Intelligence. World Scientific Pub Co Inc.

Rizzi, A., Panella, M., and Frattale Mascioli, F. M. (2002). Adaptive resolution min-max classifiers. *IEEE Transactions on Neural Networks*, 13:402–414.

Robles-Kelly, A. and Hancock, E. R. (2005). Graph edit distance from spectral seriation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27:365–378.

Vishwanathan, S. V. N., Borgwardt, K. M., Kondor, R. I., and Schraudolph, N. N. (2008). Graph kernels. *CoRR*, abs/0807.0093.