Complexity vs. Performance in Granular Embedding Spaces for Graph Classification

Luca Baldini[a], Alessio Martino[b] and Antonello Rizzi[c]
Department of Information Engineering, Electronics and Telecommunications, University of Rome "La Sapienza", via Eudossiana 18, 00184 Rome, Italy

Keywords: Structural Pattern Recognition, Supervised Learning, Embedding Spaces, Granular Computing, Graph Edit Distances.

Abstract: The most distinctive trait in structural pattern recognition in graph domain is the ability to deal with the organization and relations between the constituent entities of the pattern. Even if this can be convenient and/or necessary in many contexts, most of the state-of the art classification techniques can not be deployed directly in the graph domain without first embedding graph patterns towards a metric space. Granular Computing is a powerful information processing paradigm that can be employed in order to drive the synthesis of automatic embedding spaces from structured domains. In this paper we investigate several classification techniques starting from Granular Computing-based embedding procedures and provide a thorough overview in terms of model complexity, embedding space complexity and performances on several open-access datasets for graph classification. We witness that certain classification techniques perform poorly both from the point of view of complexity and learning performances as the case of non-linear SVM, suggesting that high dimensionality of the synthesized embedding space can negatively affect the effectiveness of these approaches. On the other hand, linear support vector machines, neuro-fuzzy networks and nearest neighbour classifiers have comparable performances in terms of accuracy, with second being the most competitive in terms of structural complexity and the latter being the most competitive in terms of embedding space dimensionality.

1 INTRODUCTION

The possibility of solving pattern recognition problems in the graph domain challenged computer scientists and machine learning engineers alike for more than two decades. That is because graphs are able to encode both topological information (namely, relationship between entities) and semantic information (whether nodes and/or edges are equipped with suitable attributes). In turn, this high level of abstraction and customization made graphs suitable mathematical objects for modelling several real-world systems in fields such as biology, social networks analysis, computer vision and image analysis. The drawback when dealing with graph-based pattern recognition relies on the computational complexity required in order to measure the (dis)similarity between two graphs, which exponentially grows with respect to their size (Bunke, 2003). This inevitably results in an heavy computational burden when it comes to perform pattern recognition in the graph domain, especially when also node and/or edge attributes have to be taken into account.

One of the mainstream approaches when dealing with graph-based pattern recognition relies on embedding spaces: in short, the pattern recognition problem is cast from the structured input domain towards the Euclidean space in which classification is performed. Nonetheless, building the embedding space is a delicate issue that must fill the informative and semantic gap between the two domains (Baldini et al., 2019a).

This paper follows two previous works (Baldini et al., 2019a; Baldini et al., 2019b) on the definition of embedding spaces for graph classification thanks to the Granular Computing paradigm (Bargiela and Pedrycz, 2006; Bargiela and Pedrycz, 2003; Pedrycz, 2001). According to the latter, structured domains can be analyzed by means of suitable *information granules*, namely small data aggregates that endow similar functional and/or structural characteristics. Specifically, we investigate several classification sys-

[a] https://orcid.org/0000-0003-4391-2598
[b] https://orcid.org/0000-0003-1730-5436
[c] https://orcid.org/0000-0001-8244-0015

Baldini, L., Martino, A. and Rizzi, A.
Complexity vs. Performance in Granular Embedding Spaces for Graph Classification.
DOI: 10.5220/0010109503380349
In *Proceedings of the 12th International Joint Conference on Computational Intelligence (IJCCI 2020)*, pages 338-349
ISBN: 978-989-758-475-6

tems working on a properly-synthesized embedding space. In fact, in our previous works (Baldini et al., 2019a; Baldini et al., 2019b) we used a plain nearest neighbours decision rule for the sake of simplicity. Notwithstanding that, a nearest neighbours decision rule suffers from several drawbacks, including high computational burden, sensitivity to the classes distribution and the number of classes. Hence, in this work, we consider five different classifiers operating in the embedding space: two support vector machines variants (with linear and radial basis kernels), two neuro-fuzzy approaches and K-nearest neighbours. We provide a general three-fold overview in terms of performances, embedding space complexity and structural complexity of the adopted classification system.

The remainder of this paper is structured as follows: in Section 2 we provide an overview of possible strategies for solving pattern recognition problems in structured domains, with a major emphasis on Granular Computing-based systems; in Section 3 we introduce GRALG, the Granular Computing-based graph classification system at the basis of this work; in Section 4 we introduce the datasets used for analysis, along with the proper computational results; finally, Section 5 concludes the paper.

2 CURRENT APPROACHES FOR PATTERN RECOGNITION ON THE GRAPH DOMAIN

In the literature, several mainstream strategies can be found in order to solve pattern recognition problems in structured domains. According to the macro-taxonomy presented in (Martino et al., 2018a), five main strategies can be found. A first strategy consists in engineering numerical features to be drawn from the structured data at hand, to be concatenated in a vector form. Examples of feature engineering techniques involve entropy measures (Han et al., 2011; Ye et al., 2014; Bai et al., 2012), centrality measures (Mizui et al., 2017; Martino et al., 2018b; Leone Sciabolazza and Riccetti, 2020; Martino et al., 2020a), heat trace (Xiao and Hancock, 2005; Xiao et al., 2009) and modularity (Li, 2013). Whilst this approach is straightforward and allows to move the pattern recognition problem towards the Euclidean space in which any pattern recognition algorithm can be used without alterations, designing the mapping function (i.e., enumerating the set of numerical features to be extracted) requires a deep knowledge of both the problem and the data at hand: indeed, the input spaces being equal, specific subsets of features allow to solve different problems.

A dual strategy consists in defining ad hoc dissimilarity measures tailored to the input space under analysis. In this manner, the pattern recognition problem can directly be solved in the input space without any explicit cast towards the Euclidean space. On the plus side, the input space might not be metric altogether, yet this is not a prerogative; on the negative side, this limits the range of pattern recognition algorithms that can be employed. Indeed, the possibility of the input space being non-metric reflects the non-metric peculiarity of the dissimilarity measure itself: as such, the pattern recognition algorithm must not rely on any algebraic structures (e.g., mean, median, inner product) and leverage pairwise dissimilarities only. Examples of custom dissimilarity measures include the so-called edit distances, defined both on graph (Baldini et al., 2019a; Baldini et al., 2019b) and sequence (Levenshtein, 1966; Cinti et al., 2020) domains.

A further family is composed by embedding techniques, which can either be explicit or implicit. Amongst the implicit embedding techniques, kernel methods emerge (Schölkopf and Smola, 2002; Shawe-Taylor and Cristianini, 2004): kernel methods exploit the so-called kernel trick (i.e., the inner product in a reproducing kernel Hilbert space) in order to measure similarity between patterns. In the literature, have been proposed several graph kernels (Kondor and Lafferty, 2002; Borgwardt and Kriegel, 2005; Shervashidze et al., 2009; Shervashidze and Borgwardt, 2009; Vishwanathan et al., 2010; Shervashidze et al., 2011; Livi and Rizzi, 2013; Neumann et al., 2016; Ghosh et al., 2018; Bacciu et al., 2018) that, for example, consider substructures such as (random) walks, trees, paths, cycles in order to measure similarity or exploit propagation/diffusion schemes. Conversely, as explicit embedding techniques are concerned, dissimilarity spaces are one of the main approaches (Pękalska and Duin, 2005). A dissimilarity space can be built by first defining an ad-hoc dissimilarity measure and then evaluating the pairwise dissimilarity matrix which can be considered as embedded in a Euclidean space: in other words, each original (structured) pattern is represented by a real-valued vector containing the distances with respect to all other patterns (Pękalska and Duin, 2005) or with respect to a properly selected subset of prototypes (Pękalska et al., 2006). Alongside embedding methods, deep learning approaches are emerging as effective frameworks for different tasks like graph classification, link prediction and node classification (Zhang et al., 2019; Wu et al., 2020). In this context, the neural architectures typically implement a convolution layer resembling the well-know convolutional neural

networks, with the scope to learn nodes representation taking into account neighbours features. Two distinct approaches are typically employed: spectral-based methods perform convolution operation in the spectral domain by means of Graph Fourier Transform (Kipf and Welling, 2016; Bianchi et al., 2019), whilst spatial-based approaches used to aggregates nodes attributes by directly exchanging information amongst neighbours (Niepert et al., 2016; Hamilton et al., 2017).

As anticipated in Section 1, Granular Computing is a powerful information processing paradigm for building explicit embedding spaces and has been successfully applied for synthesizing effective and interpretable advanced pattern recognition systems for structured data (see e.g. (Rizzi et al., 2012; Baldini et al., 2019a; Baldini et al., 2019b; Martino et al., 2019a; Martino et al., 2020b; Bianchi et al., 2014a; Bianchi et al., 2014b; Bianchi et al., 2016) and references therein). In short, Granular Computing is often described as a human-centered information processing paradigm based on formal mathematical entities known as *information granules* (Bargiela and Pedrycz, 2006). The human-centered computational concept in soft computing and computational intelligence was initially developed by Lotfi A. Zadeh through fuzzy sets (Zadeh, 1979) that exploits human-inspired approaches to deal with uncertainties and complexities in data. The process of 'granulation', intended as the extraction of meaningful aggregated data, mimics the human mechanism needed to organize complex data from the surrounding environment in order to support decision making activities and to describe the world around (Pedrycz, 2016). For this reason, Granular Computing can be described as a framework for analyzing data in complex systems aiming to provide human interpretable results (Martino et al., 2020b). The importance of information granules resides in the ability to underline properties and relationships between data aggregates. Specifically, their synthesis can be achieved by following the *indistinguishability rule*, according to which elements that show enough similarity, proximity or functionality shall be grouped together (Zadeh, 1997; Pedrycz and Homenda, 2013). With this approach, each granule is able to show homogeneous semantic information from the problem at hand. Furthermore, data at hand can be represented using different levels of 'granularity' and thus different peculiarities of the considered system can emerge. Depending on this resolution, an unknown computational process to be modelled may exhibit different properties and different atomic units that show different representations of the system as a whole. Clearly, an efficient and automatic procedure to select the most suitable level of abstraction according to both the problem at hand and the data description is of utmost importance. Due to the tight link between 'information granules' and 'groups of similar data', the most straightforward approach in order to synthesize a possibly meaningful set of information granules can be found in data clustering (Frattale Mascioli et al., 2000; Pedrycz, 2005; Ding et al., 2015). Since the clustering procedure is usually employed to discover groups of similar data aggregates, it operates in the input (structured) domain: to this end, not only the (dis)similarity measure, but also the cluster representative shall be tailored accordingly. In order to represent clusters in structured domains, the medoid (also called MinSOD) is usually employed (Martino et al., 2019b), due to the fact that its evaluation relies only on pairwise dissimilarities between patterns belonging to the cluster itself, without any algebraic structures that can not be defined in non-geometric spaces. The cluster representatives from the resulting partition can be considered as symbols belonging to an *alphabet* $\mathcal{A} = \{s_1, ..., s_n\}$: these symbols are the pivotal granules on the top of which the embedding space can be built thanks to the *symbolic histograms* paradigm. According to the latter, each pattern can be described as an n-length integer-valued vector which counts in position i the number of occurrences of the i^{th} symbol drawn from the alphabet within the patter itself. The embedding space can finally be equipped with algebraic structures such as the Euclidean distance or the dot product and standard classification systems can be used without alterations.

3 GRALG

The classification system considered in this work is called GRALG (GRanular computing Approach for Labeled Graphs) originally proposed in (Bianchi et al., 2014a). GRALG aims to be a general purpose classifier for labelled graphs embracing the Granular Computing paradigm. The main core of the classifier is a graph embedding procedure via symbolic histograms, the geometric representation of graphs built upon the symbols in an alphabet $\mathcal{A}$ which, in turn, are synthesized in an unsupervised way through an ensemble of clustering algorithms. These two stages rely on a suitable dissimilarity measure, which allows the pattern comparison in the graph domain. In this section, we first introduce the dissimilarity measure involved in the atomic operations between graphs (Section 3.1) and then we provide a brief description of the building blocks that make up the stages needed

to perform the explicit graph mapping towards a suitable embedding space (Sections 3.2–3.5), along the way in which they cooperate for training (Section 3.6) and testing (Section 3.7) the overall system.

3.1 Graph Edit Distance

As anticipated in Section 2, the *Graph Edit Distance* (GED) is a common approach to evaluate the dissimilarity between graphs. GEDs define the distance between two graph G_1 and G_2 as the minimum cost path needed to turn G_1 into G_2, by applying a series of atomic operations defined on both vertices and edges: substitution, insertion and deletion. Besides the costs, the edit operations may also be associated to some weights in order to reflect the importance of the aforementioned operations. Generally speaking, the dissimilarity measure $d : \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ between two graphs can be cast as the following minimization problem:

$$d(G_1, G_2) = \min_{(\varepsilon_1,\ldots,\varepsilon_k)\in O(G_1,G_2)} \sum_{i=1}^{k} c(\varepsilon_i) \quad (1)$$

where $c(\varepsilon_i)$ is the cost associated to the generic[1] edit operation ε_i and $O(G_1, G_2)$ is the set of prospective operations that gives an isomorphism between the two graphs. The main drawback when using a GED is the unfeasible computational complexity needed to compute an exact solution of Eq. (1) and, for this reason, many heuristics were proposed in order to solve the graph matching problem in a sub-optimal manner, yet with reasonable computational burden. Given these aspects, in GRALG we employ a greedy heuristic called *node Best Match First* (nBMF) (Bianchi et al., 2016) in order to measure the dissimilarity between graphs. Formally, let $G_1 = (\mathcal{V}_1, \mathcal{E}_1, \mathcal{L}_v, \mathcal{L}_e)$ and $G_2 = (\mathcal{V}_2, \mathcal{E}_2, \mathcal{L}_v, \mathcal{L}_e)$ be two graphs labelled on both edges and vertices. Furthermore, let $d_v^{\pi_v} : \mathcal{L}_v \times \mathcal{L}_v \to \mathbb{R}$ and $d_e^{\pi_e} : \mathcal{L}_e \times \mathcal{L}_e \to \mathbb{R}$ be the dissimilarities defined on nodes' and edges' attributes ($\mathcal{L}_v$ and $\mathcal{L}_e$, respectively). For the sake of generalization, $d_v^{\pi_v}$ and $d_e^{\pi_e}$ might be parametric with respect to π_v and π_e, namely sets of real-valued parameters required to evaluate $d_v^{\pi_v}$ and $d_e^{\pi_e}$, respectively. In nBMF, the substitution cost on nodes and edges is evaluated with the corresponding dissimilarity function ($d_v^{\pi_v}$ and $d_e^{\pi_e}$). Accordingly, it is possible to define the overall substitution costs on nodes and edges (c_{nodes}^{sub} and c_{edge}^{sub}) by summing up the dissimilarity values of all the nodes or edges substituted during the procedure. Conversely, insertion and deletion costs c_{node}^{ins}, c_{edge}^{ins}, c_{node}^{del}, c_{edge}^{del} depend on the difference between the two graphs in terms of nodes and edges set cardinality. The interested reader can found detailed pseudo-codes describing the nBMF in (Baldini et al., 2019a). Additionally, each operation is weighted by w_{node}^{sub}, w_{edge}^{sub}, w_{node}^{ins}, w_{edge}^{ins}, w_{node}^{del}, w_{edge}^{del} individually bounded in $[0, 1]$. At the end of the procedure, the dissimilarities between nodes and edges, say $d_{\mathcal{V}}(\mathcal{V}_1, \mathcal{V}_2)$ and $d_{\mathcal{E}}(\mathcal{E}_1, \mathcal{E}_2)$, are evaluated as:

$$\begin{aligned} d_{\mathcal{V}}(\mathcal{V}_1, \mathcal{V}_2) &= w_{node}^{sub} c_{node}^{sub} + w_{node}^{ins} c_{node}^{ins} + w_{node}^{del} c_{node}^{del} \\ d_{\mathcal{E}}(\mathcal{E}_1, \mathcal{E}_2) &= w_{edge}^{sub} c_{edge}^{sub} + w_{edge}^{ins} c_{edge}^{ins} + w_{edge}^{del} c_{edge}^{del} \end{aligned} \quad (2)$$

Each dissimilarity is normalized by taking into account the different orders between the two graphs:

$$\begin{aligned} d'_{\mathcal{V}}(\mathcal{V}_1, \mathcal{V}_2) &= \frac{d_{\mathcal{V}}(\mathcal{V}_1, \mathcal{V}_2)}{\max\{o_1, o_2\}} \\ d'_{\mathcal{E}}(\mathcal{E}_1, \mathcal{E}_2) &= \frac{d_{\mathcal{E}}(\mathcal{E}_1, \mathcal{E}_2)}{\frac{1}{2}\left(\min\{o_1, o_2\} \cdot (\min\{o_1, o_2\} - 1)\right)} \end{aligned} \quad (3)$$

with $o_1 = |\mathcal{V}_1|$ and $o_2 = |\mathcal{V}_2|$. Finally, the dissimilarity between the two graphs reads as:

$$d(G_1, G_2) = \frac{1}{2}\left(d'_{\mathcal{V}}(\mathcal{V}_1, \mathcal{V}_2) + d'_{\mathcal{E}}(\mathcal{E}_1, \mathcal{E}_2)\right) \quad (4)$$

3.2 Extractor

The subgraph extraction block aims at breaking down a graph into its constituent parts following a stochastic subsampling approach operating in a class-aware fashion. These improvements have been thoroughly described and tested in (Baldini et al., 2019a) and (Baldini et al., 2019b), to which the interested reader is referred to for further details, in order to overcome the main limitation of GRALG: indeed, in its original implementation (Bianchi et al., 2014a), GRALG used to exhaustively extract all subgraphs up to a given user-defined order o from any input graph, resulting in a non-negligible running time and memory footprint. In (Baldini et al., 2019a), we show how a stochastic subsampling can lead to comparable performances with respect to the original (exhaustive) implementation with remarkably lower running times and memory usage. Indeed, this procedure builds a subgraph set with a fixed cardinality W by extracting subgraphs uniformly at random (replacements are allowed) from a set of graphs using either one of two well-known traversal strategies, namely Breadth First Search and Depth First Search. Furthermore, in (Baldini et al., 2019b), we show how the stochastic subsampling granulation procedure can be performed in a class-aware fashion: this allows the exploitation of the ground-truth class labels in the training data, resulting in a more effective alphabet synthesis. The latter, being competitive both in terms of per-

[1] Regardless on whether it is a substitution, deletion or insertion on nodes or edges.

formances and computational complexity, is the strategy we used in this work. Let $\mathcal{S}$ be a set of graphs, each of which is associated to its ground-truth class label L with $L \in \{1,\dots,N\}$ and N being the number of classes. The class-aware stochastic extraction randomly draws a graph $G := \{\mathcal{V},\mathcal{E}\}$ from $\mathcal{S}$ and then randomly draws a vertex $v \in \mathcal{V}$. Then, starting from the node v, a suitable traversal strategy explores G collecting a subgraph $g = \{\mathcal{V}_g,\mathcal{E}_g\}$ with $\mathcal{V}_g \subset \mathcal{V}$ vertices and $\mathcal{E}_g \subset \mathcal{E}$ edges. The subgraph g is inserted in a subgraph set $\mathcal{S}_g^L$ according to the graph's label L. In this way, it is also possible to the define the number of subgraphs W to be sampled from the starting set by fixing how many times the procedure must occur, since each extraction outputs a single graph. The maximum order o of the extracted subgraph, namely the maximum number of vertices for all subgraphs g, is still a user-defined parameter. Conversely to the original work (Bianchi et al., 2014a), this procedure allows to arbitrary choose the cardinality W of the subgraphs set $\bigcup_{L=1}^{N}\mathcal{S}_g^L$ and consequently relieves the memory issues and improves the wall-clock runtime of the algorithm.

3.3 Granulator

The *Granulator* defines a procedure able to synthesize information granules by building the *alphabet* $\mathcal{A} = \{s_1,\dots,s_n\}$, namely a set that collects the relevant substructures s_i starting from a given subgraphs set. The core method in charge of returning suitable information granules is the Basic Sequential Algorithmic Scheme (BSAS) clustering algorithm (Baldini et al., 2019a; Baldini et al., 2019b) that works directly in the graph domain thanks to the dissimilarity measure described in Section 3.1. BSAS needs two parameters: the dissimilarity threshold θ below which a pattern is included into the closest cluster and the maximum number of allowed clusters Q, in order to reasonably bound the number of resulting clusters which can grow indefinitely (especially for small values of θ). According to the Granular Computing paradigm, the granulator shall extract symbols by exploring different levels of abstraction for the problem at hand: indeed, the granulation performs a *clustering ensemble* procedure by generating different partitions of the input data using different θ values. Then, for every cluster $\mathbf{C}$ in the resulting partitions, a cluster quality index $F(\mathbf{C})$ is defined as:

$$F(\mathbf{C}) = \eta \cdot \Phi(\mathbf{C}) + (1-\eta)\cdot\Theta(\mathbf{C}) \tag{5}$$

where the two terms $\Phi(\mathbf{C})$ and $\Theta(\mathbf{C})$ are respectively the compactness and the cardinality of cluster $\mathbf{C}$, defined in turn as:

$$\Phi(\mathbf{C}) = \frac{1}{|\mathbf{C}|-1}\sum_i d(g^*,g_i) \tag{6}$$

$$\Theta(\mathbf{C}) = 1 - \frac{|\mathbf{C}|}{|\mathcal{S}_g|} \tag{7}$$

where g^* is the representative of cluster $\mathbf{C}$ and g_i the i^{th} pattern in the cluster. In this way, we evaluate each cluster by considering its compactness Φ and cardinality Θ, whose importance is weighted by a trade-off parameter $\eta \in [0,1]$. Since the BSAS operates in the graph domain, as discussed in Section 2, the representative g^* is defined as the medoid of the cluster, namely the element that minimizes the pairwise sum of distances among all patterns in the cluster (Martino et al., 2019b). Finally, by thresholding $F(\mathbf{C})$ with a value τ_F, only well-formed clusters (i.e., compact and populated) are promoted to be symbols s belonging to the alphabet $\mathcal{A}$. Since the symbol synthesis occurs on a labelled training set and since the *Extractor* operates in a class-aware fashion (Section 3.2), the clustering ensemble is performed individually on the $\mathcal{S}_g^L$ sets, hence collecting class-related symbols in class-aware alphabets $\mathcal{A}_L$, which are finally merged together in $\mathcal{A} = \bigcup_{L=1}^{N}\mathcal{A}_L$.

3.4 Embedder

The *Embedder* block plays the key role necessary to map patterns from the graph domain towards a geometric (Euclidean) space. Formally, let $\mathcal{G}$ be the graph domain, the *Embedder* is in charge of building the function $\phi : \mathcal{G} \to \mathcal{D}$ that maps the graph domain into an n-dimensional space $\mathcal{D} \subseteq \mathbb{R}^n$. The embedding function relies on the *symbolic histograms* paradigm, exploiting the relevant substructures synthesized by the *Granulator* (i.e., the alphabet). In order to describe this process, let G_{exp} be the set made up of the subgraphs in $G \in \mathcal{G}$, i.e. the expanded representation of G in subgraphs. Given the alphabet set $\mathcal{A} = \{s_1,\dots,s_n\}$, the vectorial representation $\mathbf{h}$ (i.e., the symbolic histogram) of G is a vector whose components represents the number of occurrences of each symbol $s \in \mathcal{A}$ in G_{exp}:

$$\mathbf{h} = \phi^{\mathcal{A}}(G_{exp}) = [occ(s_1,G_{exp}),\dots,occ(s_n,G_{exp})] \tag{8}$$

The function $occ : \mathcal{A} \to \mathbb{N}$ compares a symbol s_j with $j = 1\dots n$ and all the subgraphs $g \in G_{exp}$. If the resulting dissimilarity measure is below a symbol-dependent threshold value $\tau_j = \Phi(\mathbf{C}_j)\cdot\varepsilon$, this counts as a match and the counter is increased by 1. The ε value is a user-defined tolerance parameter and $\mathbf{C}_j$ is the cluster whose MinSOD is s_j. It is worth remarking here that the computational complexity needed to

expand exhaustively G in the set of its subgraph G_{exp} may lead to memory issues, since the number of subgraphs grows exponentially with respect to the order of graph. Furthermore, the number of matches needed to build the embedding space can be very large since all symbols have to be compared with the subgraphs in G_{exp}. This could lead to unfeasible running times even for medium size datasets. For this reason, the set G_{exp} is evaluated with a deterministic algorithm that tries to partially expand $G = \{\mathcal{V}, \mathcal{E}\}$: starting from a seed node $v \in \mathcal{V}$, the same traverse strategy employed in the *Extractor* drives the exploration and the extraction of subgraphs from G, starting from order 1 up until a user-defined value o and collects these substructures in G_{exp}. Since the goal of the procedure is to limit the cardinality of G_{exp}, if a node $v \in \mathcal{V}$ already appears in one of the previously-extracted subgraphs, it will not be eligible to be a seed node for further traversals. The procedure goes on as long as a vertex $v \in \mathcal{V}$ can be legally used as root node for the traversal strategy.

3.5 Classifier

The three blocks in Sections 3.2–3.4 allow us to build an embedding space $\mathcal{D} \subseteq \mathbb{R}^n$, where each graph is mapped with an appropriate vector $\mathbf{h} \in \mathcal{D}$, i.e. its symbolic histogram. Further, such approach allows us to use any classifier designed in $\mathbb{R}^n$ without limitations. In this work, we considered different classification systems:

Support Vector Machines (SVMs) aim at estimating the decision boundary between two classes by a maximal-margin hyperplane. We employed the well-known ν-SVM (Schölkopf et al., 2000) using two different kernels: the linear kernel endowing the dot product between patterns $\mathcal{K}(x,x') = \langle x,x' \rangle$ and a non-linear radial basis function (RBF) kernel, i.e. $\mathcal{K}(x,x') = \exp\{-\gamma\|x-x'\|^2\}$

Neuro-Fuzzy Min-Max classifier relying on the construction of *hyperboxes*, which serve as atomic geometric structures needed to define the decision boundary in the training phase. Specifically, we tested two algorithms for training the model (i.e., building hyperboxes): *Adaptive Resolution Classifier* (ARC) and *Pruning Adaptive Resolution Classifier* (PARC) (Rizzi et al., 2002).

K-Nearest Neighbours (K-NN) in which, when an unseen pattern is considered for classification, its K nearest patterns are selected and the decision is taken according to the most voted class amongst the K patterns (Cover and Hart, 1967).

3.6 Training Phase

After individual and detailed descriptions of the building blocks that made up the GRALG classification system, here we describe how those blocks cooperate together in order to synthesize the final model. To this end, let $\mathcal{S}$ be a dataset of labelled graphs on nodes and/or edges and let $\mathcal{S}_{tr}$, $\mathcal{S}_{vs}$ and $\mathcal{S}_{ts}$ be the training, validation and test set drawn from $\mathcal{S}$:

***Extractor*:** takes as input $\mathcal{S}_{tr}$ and the parameter W and, according to the description given in Section 3.2, extracts N different class-specific subgraphs sets $\mathcal{S}^L_{g,tr}$

***Granulator*:** each class-aware subgraphs set $\mathcal{S}^L_{g,tr}$ is granulated and consequently $\mathcal{A}_L$ class-specific alphabets are constructed

***Embedder*:** the overall alphabet $\mathcal{A} = \bigcup_{L=1}^{N} \mathcal{A}_L$, with $n = |\mathcal{A}|$ symbols, enters the embedding block. First, all graphs in $\mathcal{S}_{tr}$ and $\mathcal{S}_{vs}$ are expanded as described in Section 3.4. Then, for each $G^{tr}_i \in \mathcal{S}^{tr}$ and $G^{vs}_i \in \mathcal{S}^{vs}$, we can construct the associated vector representation $\mathbf{h}^{tr}_i \in \mathcal{D}$ and $\mathbf{h}^{vs}_i \in \mathcal{D}$ such that $\mathcal{D} \subseteq \mathbb{R}^n$

***Classifier*:** given the embedded version of training and validation set ($\mathbf{h}^{tr}$ and $\mathbf{h}^{vs}$), the quality of the mapping function is evaluated by considering a given classifier (amongst the ones presented in Section 3.5) which is trained on $\mathbf{h}^{tr}$ and later evaluated on $\mathbf{h}^{vs}$.

The procedures involved to complete a graph classification are very sensitive to specific parameters that can strongly affect the overall performances. In order to face this issue, two different optimizations take place separately.

3.6.1 Alphabet Optimization Phase

A first stage of optimization aims to synthesize an optimal set of symbols $\mathcal{A}^*$ and properly tune the hyperparameters of the chosen classifier, by deploying an evolutive strategy, i.e. a genetic algorithm. In order to purse this goal, the following parameters (and corresponding procedures) are considered:

***GED*:** the dissimilarity measure in the graph domain requires a fine tuning of the 6 weights for the edit operations: $\mathcal{W} = \{w^{sub}_{node}, w^{sub}_{edge}, w^{ins}_{node}, w^{ins}_{edge}, w^{del}_{node}, w^{del}_{edge}\}$. Depending on the dataset, both vertices and edges can require a parametric dissimilarity measure $d^{\pi_v}_v$ and $d^{\pi_e}_e$ with parameters $\Pi = \{\pi_v, \pi_e\}$ whose values are optimized as well.

***Granulator*:** in order to synthesize an optimal alphabet, we optimize Q (maximum number of allowed

clusters for BSAS), τ_F (MinSOD quality threshold for being promoted to a symbol) and η (trade-off parameter that weights compactness and cardinality in Eq. (5)).

***Classifier*:** the set of hyperparameters $\mathcal{C}$ that depends on the considered classifier:

- ν-SVM: only the regularization term $\nu \in (0,1]$.
- RBF ν-SVM: along with $\nu \in (0,1]$, we also tune the kernel shape $\gamma \in (0,100]$.
- ARC/PARC: we optimize the $\lambda \in (0,1)$ parameter used as trade-off between the error on training set and the network complexity. Furthermore, we optimize the type of membership function (to be chosen between Trapezoidal and Simpson's) and the decay parameter $\mu \in (0,1)$ associated with it.
- K-NN: the number $K \in [1, 2\sqrt{|\mathcal{S}_{tr}|}]$ of nearest patterns involved in the voting.

To summarize, the genetic code reads as:

$$[Q \quad \tau_F \quad \eta \quad \mathcal{W} \quad \Pi \quad \mathcal{C}] \tag{9}$$

whereas the fitness function J_{alph} reads as the classifier accuracy $\omega \in [0,1]$ on the validation set:

$$J_{alph} = \omega(\mathcal{S}_{vs}) = \frac{\sum_{i=1}^{|\mathcal{S}_{vs}|} \delta(y_i, \hat{y}_i)}{|\mathcal{S}_{vs}|} \tag{10}$$

with

$$\delta(y_i, \hat{y}_i) = \begin{cases} 1 & \text{if } y_i = \hat{y}_i \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

and where, in turn, $\hat{y}_i$ and y_i are (respectively) the predicted label and the ground-truth label for i-th pattern in $\mathcal{S}_{vs}$. Standard genetic operators (mutation, selection, crossover and elitism) take care of moving from one generation to the next. The best individual is retained at the end of the evolution, specifically the portions of the genetic code $\mathcal{W}^\star$ and $\Pi^\star$, along with the alphabet $\mathcal{A}^\star$ synthesized using its genetic code.

3.6.2 Feature Selection Phase

The cardinality of $\mathcal{A}^\star$ and consequently the embedding space $\mathcal{D} \subseteq \mathbb{R}^{|\mathcal{A}^\star|}$ may be too large, impacting the classification performance and the interpretability of the final model. To this end, we implement a feature selection phase still based on genetic optimazation. Formally, let $\mathbf{m} \in \{0,1\}^{|\mathcal{A}^\star|}$ be a projection mask, then:

1. we perform the component-wise product between the mask and the embedded vectors of the training set:
$$\overline{\mathbf{h}}_{tr} = \mathbf{m} \odot \mathbf{h}_{tr} \tag{12}$$
by further neglecting components corresponding to 0's in $\overline{\mathbf{h}}_{tr}$. Hence, it is possible to figure $\overline{\mathbf{h}}_{tr}$ as lying in a (possibly) reduced embedding space $\overline{\mathcal{D}} \subseteq \mathbb{R}^m$ with $m \leq n$
2. according to the mask $\mathbf{m}$ employed in step 1, the validation set $\mathbf{h}_{vs} \in \mathcal{D}$ is reduced in the same way, leading to $\overline{\mathbf{h}}_{vs} \in \overline{\mathcal{D}}$
3. the classification system is trained on $\overline{\mathbf{h}}_{tr}$ using the hyperparameters set $\mathcal{C}$. The accuracy on the (reduced) validation set $\overline{\mathbf{h}}_{vs}$ is finally computed.

Following this scheme, a genetic algorithm drives the optimization of the mask $\mathbf{m}$ and, at the same time, the classifier hyperparameters $\mathcal{C}$ by maximizing a fitness function J_{fs} that reads as the linear convex combination between the classifier accuracy on the validation set and the cost of the mask weighted by a trade-off value $\alpha \in [0,1]$:

$$J_{fs} = \alpha \cdot \omega(\mathcal{S}_{vs}) + (1-\alpha) \cdot \mu \tag{13}$$

where the cost of the mask $\mu \in [0,1]$ is defined as:

$$\mu = 1 - \frac{|\{i : \mathbf{m}_i = 1\}|}{|\mathbf{m}|} \tag{14}$$

When the evolution is completed, the best individual is retained, as it encodes the optimal projection mask $\mathbf{m}^\star$, able to return the reduced alphabet $\underline{\mathcal{A}}^\star$, and the optimal set of parameters $\mathcal{C}^\star$ used to train the classifier in the embedding space $\overline{\mathcal{D}}$.

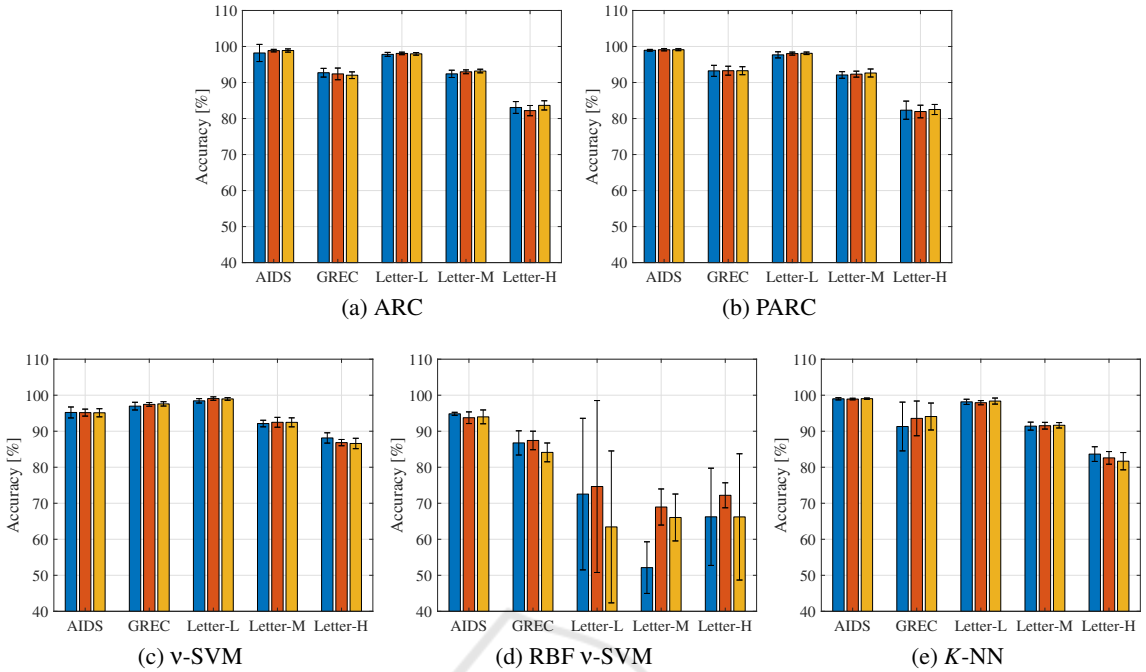
3.7 Synthesized Classification Model

After the two optimizations stages are over, the final classification performances can be evaluated on the test set. First, the test set $\mathcal{S}_{ts}$ is embedded in the vector space $\overline{\mathcal{D}}$. To this aim, the embedder block specifically equipped with parameters $\mathcal{W}^\star$ and $\Pi^\star$ for the GED dissimilarity measure, outputs the vector set $\mathbf{h}_{ts}$ by taking advantage of the alphabet $\overline{\mathcal{A}}^\star$. The classifier returned by the optimization phase (i.e., trained on the projected vectors $\overline{\mathbf{h}}_{tr}$ with the hyperparameters $\mathcal{C}^*$) is tested on the embedded test set $\mathbf{h}_{ts} \in \overline{\mathcal{D}}$, returning the overall accuracy of the GRALG system.

4 EXPERIMENTS

Five different datasets from the IAM repository (Riesen and Bunke, 2008) have been considered for testing:

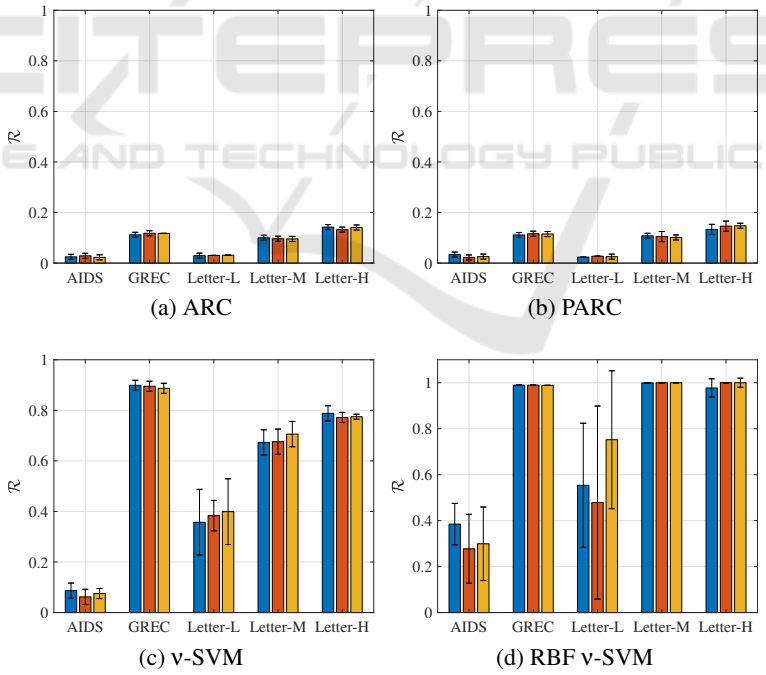
Letter: a triad of datasets where each graph represents a letter drawing with different level of distortions: low (L), medium (M) and high (H). Match-



(a) ARC (b) PARC

(c) ν-SVM (d) RBF ν-SVM (e) K-NN

Figure 1: Accuracy comparison for the 5 classifiers. Blue, red and yellow bars correspond to subsampling rates $W = 10\%, 30\%, 50\%$, respectively. Whiskers indicate the standard deviation.



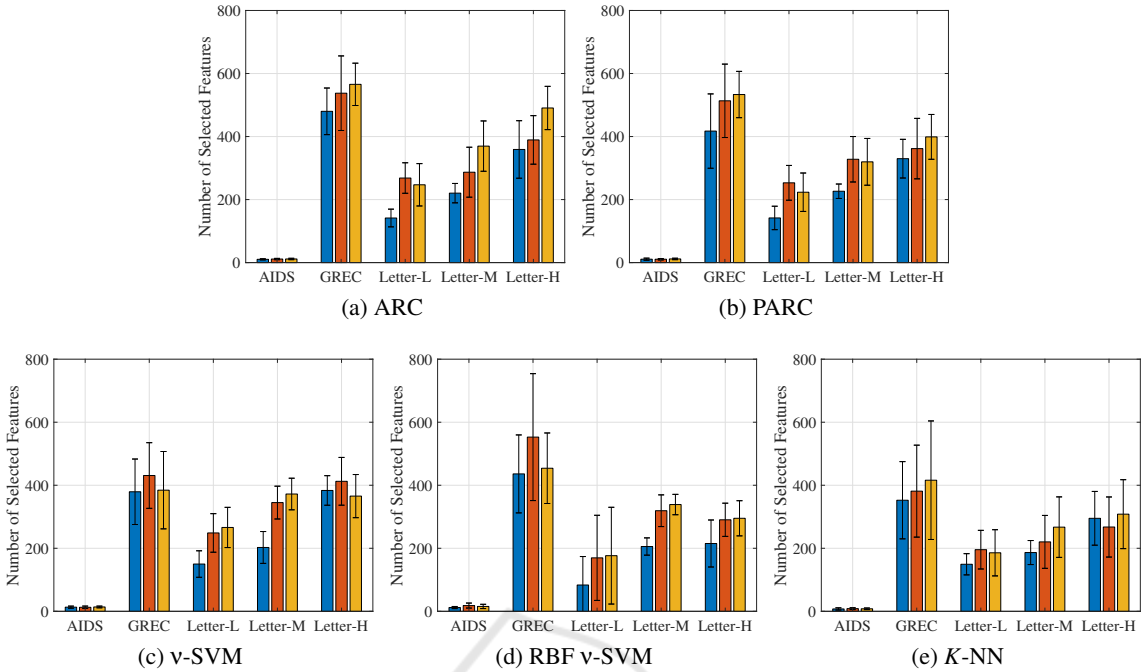
(a) ARC (b) PARC

(c) ν-SVM (d) RBF ν-SVM

Figure 2: Complexity ratio for SVM and Min-Max classifiers. For bar legend, see caption of Fig. 1.

Table 1: Number of subgraphs extracted from $\mathcal{S}_{tr}$ ($o = 5$) by the exhaustive procedure.

Letter-L	**Letter-M**	**Letter-H**	**GREC**	**AIDS**
8193	8582	21165	27119	35208



(a) ARC (b) PARC

(c) ν-SVM (d) RBF ν-SVM (e) K-NN

Figure 3: Selected number of symbols comparison for the 5 classifiers. For bar legend, see caption of Fig. 1.

Table 2: Average K value for K-NN classifier, with standard deviation.

Sample %	AIDS	GREC	Letter1	Letter2	Letter3
10	5 ± 4	3 ± 2	4 ± 4	1 ± 0	3 ± 3
30	8 ± 5	2 ± 1	5 ± 4	1 ± 0	4 ± 2
50	7 ± 4	2 ± 1	4 ± 4	1 ± 0	5 ± 5

ing between vertices is evaluated with a plain Euclidean distance, whereas edge matching is a simple delta distance, being unlabelled.

GREC: patterns in this dataset are graph representation of symbols taken from architectural and electronic drawings. Since node labels are composed by different data structures, a custom dissimilarity measure is involved in the vertex matching, as well as the edge dissimilarity function. In both cases, dissimilarity measures are parametric with respect to five real-valued parameters bounded in $[0,1]$ whose values are stored in the set Π (cf. Eq. (9)).

AIDS: graph instances in this dataset are molecular compounds where atoms are represented as nodes and covalent bonds as edges. Nodes dissimilarity is a custom (non parametric) dissimilarity measure.

Further properties of these datasets and formal definition of nodes and vertices dissimilarities can be found in (Baldini et al., 2019b).

The software is developed in C++ using the SPARE library (Livi et al., 2014). Additional dependencies include LibSVM (Chang and Lin, 2011) for ν-SVMs and the Boost Graph Library[2] for handling graph data structures. In all tests, we adopt the Class-Aware granulator (Baldini et al., 2019b) along with the stochastic sampling method (Baldini et al., 2019a) described in Section 3.2, where a Breadth First Search strategy has been employed for graph traversing. In this way, the cardinality of the set $\bigcup_{L=1}^{N} \mathcal{S}_{g,tr}^{L}$ needed to the granulator has been fixed to a given number W as percentage of the number of subraphs returned by the exhaustive procedure (Bianchi et al., 2014a), reported in Table 1. In order to take into account the intrinsic randomness in the model synthesis procedure, we run GRALG 10 times (both training and testing phases), showing the results on the test set in terms of average and standard deviation. For each of the five datasets, training, validation and test splits are kept unchanged with respect to the ones available in the IAM repository.

The system parameters have been chosen as follows:

- $W = 10\%, 30\%, 50\%$ of the maximum number of

[2]http://www.boost.org/

subgraphs that can be drawn from the training set (Table 1)

- $Q \in [1, 500]$
- $o = 5$ maximum subgraphs vertices
- 20 individuals per population of both genetic algorithms
- 20 generations for the first genetic algorithm (alphabet optimization)
- 50 generations for the second genetic algorithm (feature selection)
- $\alpha = 0.99$ in the fitness function for the second genetic algorithm (cf. Eq. (13))
- $\varepsilon = 1.1$ as tolerance value for the symbolic histograms evaluation.

As the model complexity is concerned, we define $\mathcal{R}$ as the ratio between complexity of the synthesized model and its maximal attainable complexity:

- for SVMs, $\mathcal{R}$ reads as the ratio between the number of support vectors and $|\mathcal{S}_{tr}|$ (Martino et al., 2020a);
- for ARC/PARC, one takes the ratio between the number of hyperboxes and $|\mathcal{S}_{tr}|$ (Rizzi et al., 2002);
- K-NN notably has the maximum complexity, which equals $|\mathcal{S}_{tr}|$, since all pairwise distances with respect to the training data must be evaluated for classification. Notwithstanding that, for the sake of completeness, we report the resulting number of neighbours in Table 2.

In Fig. 1, we show the accuracy (both in terms of average and standard deviation) obtained by the five classifiers on the embedded version of the test set $\mathcal{S}_{ts}$, alongside the cardinality of the optimized alphabet $\overline{\mathcal{A}^*}$ in Fig. 3. The latter simply reads as the number of selected features after the second genetic optimization and it can be considered as a measure of model interpretability (the lower, the better). The good performances obtained by ν-SVM are striking: this classifier outperforms the competitors especially for *GREC* and *Letter-H*. On the other hand, its kernelized counterpart shows generally poor performances with respect to the rest of competitors. Notably, the accuracy achieved on the three *Letter* datasets are far from being comparable with the other methods. This could be explained by considering that even if a feature selection mechanism is employed, the resulting embedding space is generally large, making unnecessary or, as in this case, disadvantageous the projection mapping due to kernel functions (Martino et al., 2019a; Martino et al., 2020b). By observing the number of selected features, the K-NN classifier emerges as the one that generally shows the smallest number of symbols in the final alphabet, especially for *Letter* datasets, while keeping comparable performances in terms of accuracy with the linear ν-SVM. When we compare the classifiers in terms of structural complexity via the $\mathcal{R}$ score, as can be seen in Fig. 2, Min-Max networks clearly show remarkable behaviour with respect to SVMs. Indeed, even if ν-SVMs have (slightly) better performances in terms of accuracy, Min-Max networks strikingly outperform them in terms of structural complexity. Further, it is worth noting that RBF ν-SVMs in three cases (*GREC*, *Letter-M* and *Letter-H*) tend to elect all patterns as support vectors: a clear sign that they strive in discriminating patterns in a Hilbert space.

5 CONCLUSIONS

In this paper, we proposed a comparison between different supervised learning algorithms for classifying graphs in geometric spaces thanks to a graph embedding procedure. Specifically, starting from GRALG, whose embedding strategy relies on symbolic histograms, we considered a ν-SVM equipped with linear and Gaussian kernels, a Min-Max neuro-fuzzy network with two different training methods (namely, ARC and PARC) and a simple K-NN decision rule. The classifiers are evaluated by taking into account both the accuracy on the test set and the $\mathcal{R}$ score for the structural complexity. If on one hand these two indices can summarize their classification and generalization abilities, they do not consider the complexity (dimensionality) of the underlying embedding space. Consequently, the number of selected features are considered as a futher performance measure of the overall classification system. At least for the considered datasets, our tests returned linear ν-SVM as generally the most performing method and its kernelized counterpart as the least performing one: this suggests that non-linear kernels that implicitly map patterns into higher dimensions may not work properly with graph embedding strategies based on symbolic histograms since the corresponding embedding vectors are likely to reside in an already high dimensional space. Notwithstanding their good performance, if compared with ARC and PARC classifiers, SVMs tend to have higher structural complexity. Under the model interpretability viewpoint, K-NN seems to be the most promising classifier.

REFERENCES

Bacciu, D., Micheli, A., and Sperduti, A. (2018). Generative kernels for tree-structured data. *IEEE transactions on neural networks and learning systems*, 29(10):4932–4946.

Bai, L., Hancock, E. R., Han, L., and Ren, P. (2012). Graph clustering using graph entropy complexity traces. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 2881–2884.

Baldini, L., Martino, A., and Rizzi, A. (2019a). Stochastic information granules extraction for graph embedding and classification. In *Proceedings of the 11th International Joint Conference on Computational Intelligence - Volume 1: NCTA, (IJCCI 2019)*, pages 391–402. INSTICC, SciTePress.

Baldini, L., Martino, A., and Rizzi, A. (2019b). Towards a class-aware information granulation for graph embedding and classification. In *Computational Intelligence: 11th International Joint Conference, IJCCI 2019 Vienna, Austria, September 17-19, 2019 Revised Selected Papers*. To appear in.

Bargiela, A. and Pedrycz, W. (2003). *Granular computing: an introduction*. Kluwer Academic Publishers, Boston.

Bargiela, A. and Pedrycz, W. (2006). The roots of granular computing. In *2006 IEEE International Conference on Granular Computing*, pages 806–809.

Bianchi, F. M., Grattarola, D., Alippi, C., and Livi, L. (2019). Graph neural networks with convolutional arma filters. *arXiv preprint arXiv:1901.01343*.

Bianchi, F. M., Livi, L., Rizzi, A., and Sadeghian, A. (2014a). A granular computing approach to the design of optimized graph classification systems. *Soft Computing*, 18(2):393–412.

Bianchi, F. M., Scardapane, S., Livi, L., Uncini, A., and Rizzi, A. (2014b). An interpretable graph-based image classifier. In *2014 International Joint Conference on Neural Networks (IJCNN)*, pages 2339–2346.

Bianchi, F. M., Scardapane, S., Rizzi, A., Uncini, A., and Sadeghian, A. (2016). Granular computing techniques for classification and semantic characterization of structured data. *Cognitive Computation*, 8(3):442–461.

Borgwardt, K. M. and Kriegel, H. P. (2005). Shortest-path kernels on graphs. In *Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 8 pp.–.

Bunke, H. (2003). Graph-based tools for data mining and machine learning. In Perner, P. and Rosenfeld, A., editors, *Machine Learning and Data Mining in Pattern Recognition*, pages 7–19, Berlin, Heidelberg. Springer Berlin Heidelberg.

Chang, C.-C. and Lin, C.-J. (2011). Libsvm: A library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, 2(3):27.

Cinti, A., Bianchi, F. M., Martino, A., and Rizzi, A. (2020). A novel algorithm for online inexact string matching and its fpga implementation. *Cognitive Computation*, 12(2):369–387.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

Ding, S., Du, M., and Zhu, H. (2015). Survey on granularity clustering. *Cognitive neurodynamics*, 9(6):561–572.

Frattale Mascioli, F. M., Rizzi, A., Panella, M., and Martinelli, G. (2000). Scale-based approach to hierarchical fuzzy clustering. *Signal Processing*, 80(6):1001 – 1016.

Ghosh, S., Das, N., Gonçalves, T., Quaresma, P., and Kundu, M. (2018). The journey of graph kernels through two decades. *Computer Science Review*, 27:88 – 111.

Hamilton, W., Ying, Z., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Advances in neural information processing systems*, pages 1024–1034.

Han, L., Hancock, E. R., and Wilson, R. C. (2011). Characterizing graphs using approximate von neumann entropy. In Vitrià, J., Sanches, J. M., and Hernández, M., editors, *Pattern Recognition and Image Analysis*, pages 484–491, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kondor, R. I. and Lafferty, J. (2002). Diffusion kernels on graphs and other discrete structures. In *Proceedings of the 19th international conference on machine learning*, volume 2002, pages 315–322.

Leone Sciabolazza, V. and Riccetti, L. (2020). Diffusion delay centrality: Decelerating diffusion processes across networks. *Available at SSRN 3653030*.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics doklady*, 10(8):707–710.

Li, W. (2013). Modularity embedding. In Lee, M., Hirose, A., Hou, Z.-G., and Kil, R. M., editors, *Neural Information Processing*, pages 92–99, Berlin, Heidelberg. Springer Berlin Heidelberg.

Livi, L., Del Vescovo, G., Rizzi, A., and Frattale Mascioli, F. M. (2014). Building pattern recognition applications with the SPARE library. *CoRR*, abs/1410.5263.

Livi, L. and Rizzi, A. (2013). Graph ambiguity. *Fuzzy Sets and Systems*, 221:24–47.

Martino, A., De Santis, E., Giuliani, A., and Rizzi, A. (2020a). Modelling and recognition of protein contact networks by multiple kernel learning and dissimilarity representations. *Entropy*, 22(7).

Martino, A., Giuliani, A., and Rizzi, A. (2018a). Granular computing techniques for bioinformatics pattern recognition problems in non-metric spaces. In Pedrycz, W. and Chen, S.-M., editors, *Computational Intelligence for Pattern Recognition*, pages 53–81. Springer International Publishing, Cham.

Martino, A., Giuliani, A., and Rizzi, A. (2019a). (hyper)graph embedding and classification via simplicial complexes. *Algorithms*, 12(11).

Martino, A., Giuliani, A., Todde, V., Bizzarri, M., and Rizzi, A. (2020b). Metabolic networks classification

and knowledge discovery by information granulation. *Computational Biology and Chemistry*, 84:107187.

Martino, A., Rizzi, A., and Frattale Mascioli, F. M. (2018b). Supervised approaches for protein function prediction by topological data analysis. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Martino, A., Rizzi, A., and Frattale Mascioli, F. M. (2019b). Efficient approaches for solving the large-scale k-medoids problem: Towards structured data. In Sabourin, C., Merelo, J. J., Madani, K., and Warwick, K., editors, *Computational Intelligence: 9th International Joint Conference, IJCCI 2017 Funchal-Madeira, Portugal, November 1-3, 2017 Revised Selected Papers*, pages 199–219. Springer International Publishing, Cham.

Mizui, Y., Kojima, T., Miyagi, S., and Sakai, O. (2017). Graphical classification in multi-centrality-index diagrams for complex chemical networks. *Symmetry*, 9(12).

Neumann, M., Garnett, R., Bauckhage, C., and Kersting, K. (2016). Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2):209–245.

Niepert, M., Ahmed, M., and Kutzkov, K. (2016). Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023.

Pedrycz, W. (2001). Granular computing: an introduction. In *Proceedings Joint 9th IFSA World Congress and 20th NAFIPS International Conference*, volume 3, pages 1349–1354. IEEE.

Pedrycz, W. (2005). *Knowledge-based clustering: from data to information granules*. John Wiley & Sons.

Pedrycz, W. (2016). *Granular computing: analysis and design of intelligent systems*. CRC press.

Pedrycz, W. and Homenda, W. (2013). Building the fundamentals of granular computing: A principle of justifiable granularity. *Applied Soft Computing*, 13(10):4209 – 4218.

Pękalska, E. and Duin, R. P. (2005). *The dissimilarity representation for pattern recognition: foundations and applications*. World Scientific.

Pękalska, E., Duin, R. P., and Paclík, P. (2006). Prototype selection for dissimilarity-based classifiers. *Pattern Recognition*, 39(2):189–208.

Riesen, K. and Bunke, H. (2008). Iam graph database repository for graph based pattern recognition and machine learning. In *Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*, pages 287–297. Springer.

Rizzi, A., Del Vescovo, G., Livi, L., and Frattale Mascioli, F. M. (2012). A new granular computing approach for sequences representation and classification. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Rizzi, A., Panella, M., and Frattale Mascioli, F. M. (2002). Adaptive resolution min-max classifiers. *IEEE Transactions on Neural Networks*, 13(2):402–414.

Schölkopf, B. and Smola, A. J. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.

Schölkopf, B., Smola, A. J., Williamson, R. C., and Bartlett, P. L. (2000). New support vector algorithms. *Neural computation*, 12(5):1207–1245.

Shawe-Taylor, J. and Cristianini, N. (2004). *Kernel methods for pattern analysis*. Cambridge University Press.

Shervashidze, N. and Borgwardt, K. M. (2009). Fast subtree kernels on graphs. In *Advances in neural information processing systems*, pages 1660–1668.

Shervashidze, N., Schweitzer, P., Van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9).

Shervashidze, N., Vishwanathan, S. V. N., Petri, T., Mehlhorn, K., and Borgwardt, K. M. (2009). Efficient graphlet kernels for large graph comparison. In van Dyk, D. and Welling, M., editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, volume 5 of *Proceedings of Machine Learning Research*, pages 488–495. PMLR.

Vishwanathan, S. V. N., Schraudolph, N. N., Kondor, R., and Borgwardt, K. M. (2010). Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Philip, S. Y. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*.

Xiao, B. and Hancock, E. R. (2005). Graph clustering using heat content invariants. In Marques, J. S., Pérez de la Blanca, N., and Pina, P., editors, *Pattern Recognition and Image Analysis*, pages 123–130, Berlin, Heidelberg. Springer Berlin Heidelberg.

Xiao, B., Hancock, E. R., and Wilson, R. C. (2009). Graph characteristics from the heat kernel trace. *Pattern Recognition*, 42(11):2589–2606.

Ye, C., Wilson, R. C., and Hancock, E. R. (2014). Graph characterization from entropy component analysis. In *2014 22nd International Conference on Pattern Recognition*, pages 3845–3850.

Zadeh, L. A. (1979). Fuzzy sets and information granularity. *Advances in fuzzy set theory and applications*, 11:3–18.

Zadeh, L. A. (1997). Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. *Fuzzy sets and systems*, 90(2):111–127.

Zhang, S., Tong, H., Xu, J., and Maciejewski, R. (2019). Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):11.