# Stochastic Information Granules Extraction for Graph Embedding and Classification

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

Keywords: Pattern Recognition, Supervised Learning, Granular Computing, Graph Embedding, Inexact Graph Matching.

Abstract: Graphs are data structures able to efficiently describe real-world systems and, as such, have been extensively used in recent years by many branches of science, including machine learning engineering. However, the design of efficient graph-based pattern recognition systems is bottlenecked by the intrinsic problem of how to properly match two graphs. In this paper, we investigate a granular computing approach for the design of a general purpose graph-based classification system. The overall framework relies on the extraction of meaningful pivotal substructures on the top of which an embedding space can be build and in which the classification can be performed without limitations. Due to its importance, we address whether information can be preserved by performing stochastic extraction on the training data instead of performing an exhaustive extraction procedure which is likely to be unfeasible for large datasets. Tests on benchmark datasets show that stochastic extraction can lead to a meaningful set of pivotal substructures with a much lower memory footprint and overall computational burden, making the proposed strategies suitable also for dealing with big datasets.

## 1 INTRODUCTION

Graphs are powerful data structures able to capture relationships between elements. This representative power in describing patterns under a structural and topological viewpoint makes graphs a flexible and accurate abstraction especially when nodes and/or edges can be equipped with labels (in this case, we refer to as *labelled graphs*). Indeed, they have been widely used to model a plethora of real-world phenomena, including biological systems (Giuliani et al., 2014; Krishnan et al., 2008; Di Paola and Giuliani, 2017), functional magnetic resonance imaging (Richiardi et al., 2013), computer vision (Bai, 2012) and online handwriting (Del Vescovo and Rizzi, 2007b). On the other hand, it is rather common in pattern recognition to represent the input pattern as a feature vector lying in an $n$-dimensional vector space. This is mainly due to the relatively simple underlying math whether some properties are satisfied. In fact, the resulting space can easily be equipped by an adequate metric satisfying the properties of non-negativity, identity, symmetry and triangle inequality (Pękalska and Duin, 2005; Martino et al., 2018a; Weinshall et al., 1999). This can not be easily achieved in structured domains and, for this reason, the main drawback when representing entities with graphs is the unpractical, non-geometric space to whom they belong to. A rather natural approach to tackle this problem when designing a classification system, is to use an ad-hoc dissimilarity measure working directly in the input space: this allows to reuse some of the well-known pattern recognition techniques for supervised learning, e.g. the $K$-Nearest Neighbour ($K$-NN) algorithm (Cover and Hart, 1967). Related to this approach, we considered *Graph Edit Distances* (GEDs) (Neuhaus and Bunke, 2007) that operate directly in the structured domain (i.e., graphs), measuring the dissimilarity between two graphs, say $G_1$ and $G_2$, as the minimum cost sequence of atomic operations (namely, substitution, deletion and insertion of nodes and/or edges) needed to transform $G_1$ into $G_2$. A very interesting strategy that has gained much attention relies on *Graph Kernels* (Vishwanathan et al., 2010; Ghosh et al., 2018): these methods exploit the so-called *kernel trick*, that is the inner product between graphs in a vector space induced by a (semi)definite positive kernel function. The classification task can heavily rely on well-known kernelized algorithms, the seminal ex-

[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.
Stochastic Information Granules Extraction for Graph Embedding and Classification.
DOI: 10.5220/0008149403910402
In *Proceedings of the 11th International Joint Conference on Computational Intelligence (IJCCI 2019)*, pages 391-402
ISBN: 978-989-758-384-1

ample being Support Vector Machines (Cortes and Vapnik, 1995). The last method, that is closely related to this work, is *Graph Embedding*. In this approach, the input pattern from the structured graphs domain $\mathcal{G}$ is mapped into an embedding space $\mathcal{D}$. Clearly, the designing of the mapping function $\phi : \mathcal{G} \rightarrow \mathcal{D}$ with $\mathcal{D} \subseteq \mathbb{R}^m$ is crucial in this procedure and some effort must be ensured to fill the informative and semantic gap between the two domains. For this purpose, a Granular Computing (Bargiela and Pedrycz, 2008) approach based on the extraction of *information granules* together with *symbolic histograms* (Del Vescovo and Rizzi, 2007a) can be pursued in order to obtain an efficient mapping function able to reflect the information carried by the structured data into the vector space. This method allows the use of common classification and data-driven systems and can achieve not only performance similar to the state-of-art classifiers (Bianchi et al., 2014a), but can also provide useful information through the extracted granules, as they are human-interpretable. Unfortunately, an heavy computational effort is necessary and often, as the dataset size increases, the problem may become unfeasible, especially under the memory footprint viewpoint.

In this paper, starting from the classification system developed by (Bianchi et al., 2014a), we explore an alternative approach for substructures extraction that will be used to synthesize the *alphabet*, i.e. the set of information granules on the top of which the embedding space is built. In particular, a lighter stochastic procedure has been developed and compared to exhaustive method from (Bianchi et al., 2014a); this procedure takes advantage of Breadth First Search (BFS) and Depth First Search (DFS) algorithms for graph traversing.

This paper is organized as follows: in Section 2 we give an overview of Granular Computing both as an information processing paradigm and as a framework in order to build data-driven classification systems for structured data; in Section 3 we introduce GRALG, the graph-based classification system core of this work, highlighting the improvements with respect to its original implementation. Section 4 regards computational results both in terms of performances and computational burden with respect to the original implementation and, finally, Section 5 draws some conclusions and future directions.

## 2 EMBEDDING VIA DATA GRANULATION

Granular Computing is often described as a human-centered information processing paradigm (Howard and Lieberman, 2014; Yao, 2016) based on formal mathematical entities known as *information granules* (Han and Lin, 2010; Bargiela and Pedrycz, 2006). The human-centered computational concept in soft computing and computational intelligence was initially developed by Lofti 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 describe the world around (Pedrycz, 2016). For this reason, Granular Computing can be defined as a framework for analyzing data in complex systems aiming to provide human interpretable results (Livi and Sadeghian, 2016).

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). With this approach, each granule is able to show homogenous semantic information from the problem at hand (Pedrycz, 2010). Furthermore, data at hand can be represented using different levels of 'granularity' and thus different peculiarities of the considered system can emerge (Yao, 2008; Pedrycz and Homenda, 2013; Yao and Zhao, 2012; Wang et al., 2017; Yang et al., 2018). When analyzing a system with high level of detail, one shall expect a huge number of very compact information granules since, straightforwardly, finer details are of interest. On the other hand, the level of abstraction increases when decreasing the granularity level: as a result, one shall expect a lower number of very populated, yet less compact, information granules. Depending on this resolution, a problem 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.

A mainstream approach in order to synthesize a possibly meaningful set of information granules can be found in data clustering. Since its direct connection with the concept of 'granules-as-groups', cluster analysis has been widely explored in the context of granular computing (Pedrycz, 2005; Pedrycz, 2013). When designing a clustering method for information granules synthesis, the parameters of the algorithm must be tuned in an appropriate way in order to select the relevant features at a suitable resolution (granularity) for the problem at hand. According to (Ding et al., 2015), typically three main factors can impact the resulting data partitioning: (dis)similarity measure, threshold parameter and cluster representatives. The *threshold* defines whether a given pattern belongs or not to a specific cluster. In our point of view, this threshold changes the granularity and therefore the level of detail considered. A typical clustering algorithm that endows a threshold in order to determine pattern-to-cluster assignments is the Basic Sequential Algorithmic Scheme (BSAS) (Theodoridis and Koutroumbas, 2008) that performs a so-called free clustering procedure, i.e. the number of clusters shall not be defined a-priori as in other data clustering paradigms, notably *k*-clustering (Martino et al., 2017; Martino et al., 2018b; Martino et al., 2019). Varying the threshold parameter impacts on how patterns will be aggregated into clusters. A suitable *(dis)similarity function* is in charge to measure the (dis)similarity in order to aggregate data entities in a proper manner. Since the clustering procedure is usually performed in the input (structured) domain, 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 (Del Vescovo et al., 2014) mainly due to the following reason: 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 (Martino et al., 2017). The clusters representatives from the outcoming partition can be considered as symbols belonging to an *alphabet* $\mathcal{A} = \{s_1, ..., s_m\}$: 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 *m*-length integer-valued vector which counts in position *i* the number of occurrences of the $i^{\text{th}}$ symbol drawn from the alphabet. 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 limitations.

## 3 THE GRALG CLASSIFICATION SYSTEM

GRALG (GRanular computing Approach for Labelled Graphs) is a general purpose classification system suitable for dealing with graphs and based on Granular Computing. GRALG has been originally proposed in (Bianchi et al., 2014a) and lately successfully applied in the context of image classification (Bianchi et al., 2014b; Bianchi et al., 2016). In this Section, the main blocks of the system are described separately (Sections 3.1–3.4), along with the way they cooperate in order to perform the training (Section 3.5) and testing phases (Section 3.6).

### 3.1 Extractor

The goal of this block regards the extraction of substructures from the input set $\mathcal{S} \subset \mathcal{G}$. In the original GRALG implementation, this procedure used to compute exhaustively the set of possible subgraphs from any given graph $G \in \mathcal{S}$. The maximum order $o$, namely the maximum number of vertices for all subgraphs, is an input parameter which must be defined by the end-user. Obviously, the complexity of the procedure strongly depends on this parameter: in fact, the asymptotically combinatorial behaviour of an exhaustive extraction makes this method unfeasible for large graphs and/or for high value of $o$, both in terms of running time and memory usage. The procedure used to expand each node of a given graph to a possible subgraph of order 2, caching in memory the resulting substructures, and then expanding and storing them iteratively until the desired maximum order $o$ is reached. At the end of the extraction procedure, the resulting set of substructures $\mathcal{S}^g$ is returned.

#### 3.1.1 Random Subgraphs Extractor based on BFS and DFS

The new procedure randomly draws a graph $G \in \mathcal{S}$ and then selects a seed node $v \in G$ for a traversal strategy based on either BFS or DFS in order to extract a subgraph. Both the extractions (graph $G$ from $\mathcal{S}$ and node $v$ from $G$) are performed with uniformly random distribution. Alongside $o$ (maximum subgraph order), a new parameter $W$ determines the cardinality of $\mathcal{S}^g$.

Algorithm 1: Random Extractor.

```
procedure EXTRACTRND(Graph Set $\mathcal{S} = \{G_1, \dots, G_n\}$ with $G = \{\mathcal{V}, \mathcal{E}\}$, $W$ max size of subgraphs set, empty set of subgraphs $\mathcal{S}^g$, $o$ max order of extracted subgraph)
    while $|\mathcal{S}^g| \leq W$ do
        for $order = 1$ to $o$ do
            Random extract a graph $G$ from $\mathcal{S}$
            Random extract a vertex $v$ from $\mathcal{V}$
            $g = \text{EXTRACT}(G, v, order)$
            $\mathcal{S}^g = \mathcal{S}^g \cup g$
    return Subgraph Set $\mathcal{S}^g$ with $|\mathcal{S}| = W$
```

Algorithm 1, which summarizes this procedure, relies on a procedure called EXTRACT (separately described in Algorithm 2) that performs a graph traverse using one of the two well-known algorithms:

**Breadth First Search:** Starting from a node $v$, BFS performs a traverse throughout the graph exploring first the adjacent nodes of $v$, namely those with unitary distance, and then moving farther only after the neighbourhood is totally discovered. A First-In-First-Out policy is in charge to organize the list of neighbours for the considered vertex, in order to give priority to adjacent nodes. The algorithm can be summerized as follow:

1. Select the starting vertex $v$.
2. Push $v$ in a queue list $Q$.
3. Pop $u$ the first element of the queue from $Q$.
4. For each neighbour $s$ of $u$, push $s$ to $Q$ if $s$ is not mark as *visited*.
5. Mark $u$ as a *visited* vertex.
6. Repeat 3-5 until $Q$ is empty.

**Depth First Search:** In this strategy, a given graph is traversed starting from a seed vertex $v$, but unlike the BFS search, the visit follows a path with increasingly distance from $v$ and backtracks only after all the vertices from the selected path are discovered. A Last-In-First-Out policy is in charge to organize the list of neighbours for the considered vertex, in order to visit in-depth vertices first. The steps of the algorithm are:

1. Select the starting vertex $v$.
2. Push $v$ in a stack list $S$.
3. Pop $u$ the last element from stack $S$.
4. For each neighbour $s$ of $u$, push $s$ in $S$ if $s$ is not marked as *visited*.
5. Mark $u$ as *visited*.
6. Repeat 3-5 until $S$ is empty.

Algorithm 2: BFS/DFS graph extraction.

```
procedure EXTRACT(Graph $G$, Vertex $v$, order actual order of extracted subgraph)
    graph $g$ of vertex $\mathcal{V}_g$ and $\mathcal{E}_g$ initially empties
    repeat
        $\{\mathcal{V}_g, \mathcal{E}_g\} \leftarrow$ BFS/DFSsearch with seed node $v$
    until $|\mathcal{V}_g| = order$
    $g = \{\mathcal{V}_g, \mathcal{E}_g\}$
    return $g$
```

In Algorithm 2, these methods are employed to populate the set of vertices $\mathcal{V}_g$ and edges $\mathcal{E}_g$ for the subgraph $g$: a vertex is added to $\mathcal{V}_g$ as soon as it is marked as *visited*, whereas an edge is added to $\mathcal{E}_g$ by considering the current and the last visited vertices.

## 3.2 Granulator

This module is in charge to compute the alphabet symbols $\mathcal{A}$ starting from the subgraphs belonging to the set $\mathcal{S}^g$, as returned by the *Extractor* defined in Algorithm 1. The information granules are synthesized by performing the BSAS clustering algorithm on $\mathcal{S}^g$. The BSAS algorithm relies on two parameters $Q$ and $\theta$, respectively the maximum number of allowed clusters and a threshold dissimilarity below which a pattern can be included in its nearest cluster[1]. Regarding $\theta$, it is worth noting that different values lead to different partitions and a binary search is deployed to generate an ensemble of partitions, each of which is obtained with a different value for $\theta$. For every cluster $\mathbf{C}$ in the resulting partitions, a cluster quality index $F(\mathbf{C})$ is defined as:

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

where the two terms $\Phi(\mathbf{C})$ and $\Theta(\mathbf{C})$ are defined respectively as:

$$\Phi(\mathbf{C}) = \frac{1}{|\mathbf{C}|-1} \sum_i d(g^*, g_i) \tag{2}$$

$$\Theta(\mathbf{C}) = 1 - \frac{|\mathbf{C}|}{|\mathcal{S}^g_{tr}|} \tag{3}$$

where, in turn, $g^*$ is the representative of cluster $\mathbf{C}$ and $g_i$ the $i^{\text{th}}$ pattern in the cluster. In other words, the quality index (1) sees a convex linear combination between the *compactness* $\Phi(\mathbf{C})$ and the *cardinality* $\Theta(\mathbf{C})$, weighted by a parameter $\eta \in [0,1]$. From Section 2, $g^*$ is the MinSOD of cluster $\mathbf{C}$, defined as the element that minimizes the sum of pairwise distances with respect to all other patterns in the cluster. The dissimilarity measure driving both Eq. (2) and the overall clustering procedure is defined as a weighted GED, described in details in Section 3.2.1. Eq. (1) needs to be evaluated for all clusters in the partitions (regardless of the corresponding $\theta$), yet only representatives belonging to clusters whose quality index is above a threshold $\tau_F$ are eligible to be included in $\mathcal{A}$: in this way, only well-formed clusters (i.e., compact and populated) are considered.

### 3.2.1 Dissimilarity Measure and Inexact Graph Matching

The core dissimilarity measure in GRALG is a weighted GED, which is based on the same ratio-

[1] If a pattern cannot be included in one of the available clusters, it can be used to initialize a new cluster, provided that the number of already-available clusters is below $Q$.

nale behind other well-known edit distances, such as the Levenshtein distance between strings (Cinti et al., 2019). Accordingly, it is possible to define some edit operations on graphs: deletion, insertion, substitution of both nodes and edges. Each of these operations can be possibly associated to a weight in order to tune the penalty induced by a particular transformation. In GRALG, six weights for edit operations are taken into account in order to establish the importance of substitutions, deletions and insertions for vertices and edges.

Formally speaking, the GED between $G_1$ and $G_2$ can be defined as a function $d : \mathcal{G} \times \mathcal{G} \rightarrow \mathbb{R}$, such that:

$$d(G_1, G_2) = \min_{(e_1,\ldots,e_k)\in\mathcal{X}(G_1,G_2)} \sum_{i=1}^{k} c(e_i) \qquad (4)$$

where $\mathcal{X}(G_1, G_2)$ is the (possibly infinite) set of prospective edit operations needed to transform the two graphs into one another. Obviously, defining the costs $c(\cdot)$ for edit operations is the crucial facet in any GED. The optimal match described in Eq. (4) is unpractical due to exponential complexity (Bunke and Allermann, 1983; Bunke, 1997; Bunke, 2000; Bunke, 2003), thus a suitable algorithm for a suboptimal search is mandatory (Tsai and Fu, 1979). In light of these observations, let us now describe the dissimilarity measure used in GRALG.

Let $G_1 = (\mathcal{V}_1, \mathcal{E}_1, \mathcal{L}_v, \mathcal{L}_e)$, $G_2 = (\mathcal{V}_2, \mathcal{E}_2, \mathcal{L}_v, \mathcal{L}_e)$ be two fully labelled graphs with nodes and edges labels set $\mathcal{L}_v$ and $\mathcal{L}_e$ and let $o_1 = |\mathcal{V}_1|$, $o_2 = |\mathcal{V}_2|$, $n_1 = |\mathcal{E}_1|$, $n_2 = |\mathcal{E}_2|$ be the number of nodes and edges in the two graphs, respectively. For the sake of generalization, the two graphs are likely to have different sizes, hence we suppose $o_1 \neq o_2$ and $n_1 \neq n_2$. Further, let us define suitable dissimilarity measures between vertices and edges, respectively $d_v^{\pi_v} : \mathcal{L}_v \times \mathcal{L}_v \rightarrow \mathbb{R}$ and $d_e^{\pi_e} : \mathcal{L}_e \times \mathcal{L}_e \rightarrow \mathbb{R}$, possibly depending on some parameters $\pi_v$ and $\pi_e$ (Wang and Sun, 2015; Di Noia et al., 2019). The strategy adopted in GRALG is called *node Best Match First* (nBMF) (Bianchi et al., 2016): by following a greedy strategy, nBMF matches most similar nodes first and then matches edges induced by those pairs. The procedure can be divided in two consecutive routines called VERTEX NBMF and EDGE NBMF, respectively.

Let us start from the former, technically described in Algorithm 3. The first node from $\mathcal{V}_1$ is selected and matched with the most similar node from $\mathcal{V}_2$ according to $d_v^{\pi_v}$. This pair is included in the set of node matches $\mathcal{M}$. Nodes involved in the pair are then removed from their respective sets and the procedure iterates until either $\mathcal{V}_1$ or $\mathcal{V}_2$ is empty. In terms of edit operations, each match counts as a (node) substitution and the overall cost associated to nodes substitutions is given by the sum of their respective dissimilarities. The overall cost for nodes insertions and deletions is strictly related to the difference between the two orders. Specifically, if $o_1 > o_2$, then we consider $(o_1 - o_2)$ node insertions. Conversely, if $o_1 < o_2$, then we consider $(o_2 - o_1)$ node deletions.

Algorithm 3: Node Best Match First Routine 1.

```
 1: procedure VERTEX NBMF($G_1, G_2$)
 2:     $minDissimilary \leftarrow \infty$
 3:     $\mathcal{M} \leftarrow \emptyset$
 4:     $c_{node}^{sub} = 0$
 5:     repeat
 6:         Select a node $v_a \in \mathcal{V}_1$
 7:         for all nodes $v_b \in \mathcal{V}_2$ do
 8:             if $d_v^{\pi_v}(v_a, v_b) \leq minDissimilarity$ then
 9:                 $minDissimilarity = d_v^{\pi_v}(v_a, v_b)$
10:                 $\mathcal{V}_1 = \mathcal{V}_1 \setminus v_a$ and $\mathcal{V}_2 = \mathcal{V}_2 \setminus v_b$
11:                 append $(v_a, v_b) \rightarrow \mathcal{M}$
12:                 $c_{node}^{sub}$ += $minDissimilarity$
13:     until $\mathcal{V}_1 = \emptyset \vee \mathcal{V}_2 = \emptyset$
14:     if $o_1 > o_2$ then
15:         $c_{node}^{ins} = (o_1 - o_2)$
16:     else if $o_1 < o_2$ then
17:         $c_{node}^{del} = (o_2 - o_1)$
18:     return $\mathcal{M}$, $c_{node}^{sub}$, $c_{node}^{ins}$, $c_{node}^{del}$
```

Now the procedure moves towards edges (Algorithm 4). For each pair of nodes in $\mathcal{M}$, the procedure checks whether an edge between the two nodes exists in both $\mathcal{E}_1$ and $\mathcal{E}_2$: if so, this counts as an edge substitution and its cost is given by the dissimilarity between edges according to $d_e^{\pi_e}$. Conversely, if the two nodes are connected on $G_1$ only, this counts as an edge insertion; if the two nodes are connected on $G_2$ only, this counts as an edge deletion.

Algorithm 4: Node Best Match First Routine 2.

```
1: procedure EDGE NBMF($G_1, G_2$)
2:     for all $(v_a, v_b) \in \mathcal{M}$ from VERTEX NBMF do
3:         if $\exists e_a \in \mathcal{E}_1, e_b \in \mathcal{E}_2 \mid e_a = (v_a, v_b) \wedge e_b = (v_a, v_b)$ then
4:             $c_{edge}^{sub}$ += $d_e^{\pi_e}(e_a, e_b)$
5:         else if $\exists e_a \in \mathcal{E}_1 \mid e_a = (v_a, v_b)$ then
6:             $c_{edge}^{ins}$ += 1
7:         else if $\exists e_b \in \mathcal{E}_2 \mid e_b = (v_a, v_b)$ then
8:             $c_{edge}^{del}$ += 1
9:     return $c_{edge}^{sub}$, $c_{edge}^{ins}$, $c_{edge}^{del}$
```

By defining $c_{edge}^{sub}$, $c_{edge}^{ins}$, $c_{edge}^{del}$, $c_{node}^{sub}$, $c_{node}^{ins}$, $c_{node}^{del}$ as the overall edit costs on nodes and edges and by defining

$w^{sub}_{node}$, $w^{sub}_{edge}$, $w^{ins}_{node}$, $w^{ins}_{edge}$, $w^{del}_{node}$, $w^{del}_{edge}$ as the aforementioned non-negative six weights which reflect the importance of the three atomic operations (insertion, deletion, substitutions) on nodes and edges, the total dissimilarity measures on vertices and edges of $G_1$ and $G_2$, say $d_{\mathcal{V}}(\mathcal{V}_1, \mathcal{V}_2)$ and $d_{\mathcal{E}}(\mathcal{E}_1, \mathcal{E}_2)$, are respectively computed as:

$$\begin{aligned} d_{\mathcal{V}}(\mathcal{V}_1, \mathcal{V}_2) &= w^{sub}_{node} \cdot c^{sub}_{node} + w^{ins}_{node} \cdot c^{ins}_{node} + w^{del}_{node} \cdot c^{del}_{node} \\ d_{\mathcal{E}}(\mathcal{E}_1, \mathcal{E}_2) &= w^{sub}_{edge} \cdot c^{sub}_{edge} + w^{ins}_{edge} \cdot c^{ins}_{edge} + w^{del}_{edge} \cdot c^{del}_{edge} \end{aligned} \tag{5}$$

In order to avoid skewness due to the different sizes between $G_1$ and $G_2$, the latter can be normalized as follows:

$$\begin{aligned} d'_{\mathcal{V}}(\mathcal{V}_1, \mathcal{V}_2) &= \frac{d_{\mathcal{V}}(\mathcal{V}_1, \mathcal{V}_2)}{\max(o_1, o_2)} \\ d'_{\mathcal{E}}(\mathcal{E}_1, \mathcal{E}_2) &= \frac{d_{\mathcal{E}}(\mathcal{E}_1, \mathcal{E}_2)}{\frac{1}{2}\left(\min(o_1, o_2) \cdot (\min(o_1, o_2) - 1)\right)} \end{aligned} \tag{6}$$

And finally:

$$d(G_1, G_2) = \frac{1}{2}\left(d'_{\mathcal{V}}(\mathcal{V}_1, \mathcal{V}_2) + d'_{\mathcal{E}}(\mathcal{E}_1, \mathcal{E}_2)\right) \tag{7}$$

## 3.3 Embedder

This block aims at the definition of an embedding function $\phi : \mathcal{G} \to \mathcal{D}$ that maps the graphs space $\mathcal{G}$ into an $m$-dimensional space $\mathcal{D} \subseteq \mathbb{R}^m$.

The embedding relies on the symbolic histograms paradigm (Del Vescovo and Rizzi, 2007a; Del Vescovo and Rizzi, 2007b). After the alphabet $\mathcal{A} = \{s_1, \ldots, s_m\}$ has been computed by the *Granulator* module, the embedding function $\phi^{\mathcal{A}} : \mathcal{G} \to \mathbb{R}^m$ consists in assigning an integer-valued vector $\mathbf{h}^{(i)}$ (the symbolic histogram) to each graph $G_i$ such that:

$$\mathbf{h}^{(i)} = \phi^{\mathcal{A}}(G_i) = [occ(s_1), \ldots, occ(s_m)] \tag{8}$$

where $occ : \mathcal{A} \to \mathbb{N}$ counts the occurrences of the subgraphs $s_j \in \mathcal{A}$ in the input graph $G_i$. The counting process of the symbols $s_j$ in $G_i$ is performed thanks to the same GED described in Section 3.2.1 between $s_j$ and the subgraphs of $G_i$. $\mathbf{h}^{(i)}_j$ is increased only when the dissimilarity between a subgraph of $G_i$ and the symbol $s_j$ reach a symbol-dependent threshold value $\tau_j = \Phi(\mathbf{C}_j) \cdot \varepsilon$, where $\varepsilon$ is a user-defined tolerance parameter and $\mathbf{C}_j$ is the cluster whose MinSOD is $s_j$. The resulting embedding space is defined as the space spanned by the symbolic histograms of the form (8).

A not negligible issue of this procedure is the computational burden related to the subgraphs extraction and comparison: the former exhaustive procedure used to extract all subgraphs up to a desired order from a given graph $G_i$; then, for each subgraph, it used to compute the GED with respect to all symbols in $\mathcal{A}$. In order to pursue the goal of avoiding an exhaustive extraction, a lighter procedure has been deployed and described in Algorithm 5. In this case, the algorithm explores a graph by performing a traverse starting from each node, which acts as seed node for the BFS or DFS strategy[2] in order to extract subgraphs. Furthermore, for limiting the number of subgraphs, if a node $v \in G$ already appears in one of the previously extracted subgraphs, it will not be later considered as a prospective seed node.

Algorithm 5: Extraction procedure for Embedder.

```
procedure EXTRACTEMBED(Graph $G = \{\mathcal{V}, \mathcal{E}\}$, empty set $\mathcal{S}^{ge}$, $o$ max order of extracted subgraph)
    for all Vertices $v$ in $\mathcal{V}$ do
        $g :=$ empty graph
        for $order = 1$ to $o$ do
            $g = \text{EXTRACT}(G, v, order)$
            $\mathcal{S}^{ge} = \mathcal{S}^{ge} \cup g$
        $\mathcal{V} = \mathcal{V} \setminus \mathcal{V}_g$
    return Subgraph Set $\mathcal{S}^{ge}$
```

## 3.4 Classifier

The classification module in GRALG relies on the $K$-NN decision rule. In order to assign the class label to a previously-unseen pattern, $K$-NN looks the $K$ nearest pattern and the classes they belong to and the test pattern is classified according to the most frequent class amongst the $K$ nearest patterns.
The performance of the whole system is defined as the accuracy on a given validation/test set, in turn defined as the ratio of correctly classified patterns. It is worth stressing that the classification procedure is performed in a metric space due to the embedding procedure, therefore the $K$-NN is equipped with a plain Euclidean distance between vectors (i.e., symbolic histograms).

## 3.5 Training Phase

The four blocks described in Sections 3.1–3.4 carry out the atomic functions in GRALG and herein we describe how they jointly work in order to synthesize a classification model. Let $\mathcal{S} \subset \mathcal{G}$ be a dataset of labelled graphs on nodes and/or edges and let $\mathcal{S}_{tr}$, $\mathcal{S}_{vs}$ and $\mathcal{S}_{ts}$ be three non-overlapping sets (training, validation and test set, respectively) drawn from $\mathcal{S}$.

[2]The *Embedder* must follow the same traverse strategy as the *Extractor*: both of them shall use either DFS or BFS.

Table 1: Characteristic of IAM datasets used for testing: size of Training (tr), Validation (vl) and Test (ts) set, number of classes (*# classes*), types of nodes and edges labels, average number of nodes and edges, whether the dataset is uniformly distributed amongst classes or not (*Balanced)*.

| **Database** | **size (tr, vl, ts)** | **# classes** | **node labels** | **edge labels** | **Avg # nodes** | **Avg # edges** | **Balanced** |
|---|---|---|---|---|---|---|---|
| Letter-L | 750, 750, 750 | 15 | $\mathbb{R}^2$ | none | 4.7 | 3.1 | Y |
| Letter-M | 750, 750, 750 | 15 | $\mathbb{R}^2$ | none | 4.7 | 3.2 | Y |
| Letter-H | 750, 750, 750 | 15 | $\mathbb{R}^2$ | none | 4.7 | 4.5 | Y |
| GREC | 286, 286, 528 | 22 | string + $\mathbb{R}^2$ | tuple | 11.5 | 12.2 | Y |
| AIDS | 250, 250, 1500 | 2 | string + integer + $\mathbb{R}^2$ | integer | 15.7 | 16.2 | N |

The training procedure starts with the *Extractor* (Section 3.1) that expands graphs in $\mathcal{S}_{tr}$ using either BFS or DFS in order to return the set of subgraphs $\mathcal{S}_{tr}^{g}$ which are used as the main input for the *Granulator* module.

### 3.5.1 Optimized Alphabet Synthesis via Genetic Algorithm

The *Granulator* block (Section 3.2) depends on several parameters whose suitable values are strictly problem and data-dependent and are hardly known a-priori. For this reason, a genetic algorithm is in charge of automatically tune these parameters in order to sythesize the alphabet $\mathcal{A}$.

The genetic code is given by:

$$[Q \quad \tau_F \quad \eta \quad \Omega \quad \Pi] \tag{9}$$

where:

- $Q$ is maximum number of allowed clusters for the BSAS procedure
- $\tau_F$ is the threshold that discards low quality clusters in order to form the alphabet
- $\eta$ is the trade-off parameter for weighting compactness and cardinality in the cluster quality index (1)
- $\Omega = \{w_{node}^{sub}, w_{edge}^{sub}, w_{node}^{ins}, w_{edge}^{ins}, w_{node}^{del}, w_{edge}^{del}\}$ is the set composed by the six weights for the GED (see Section 3.2.1)
- $\Pi = \{\pi_v, \pi_e\}$ is the set of parameters for the dissimilarity measures between nodes $d_v^{\pi_v}$ and edges $d_e^{\pi_e}$, if applicable (see Section 3.2.1).

Each individual from the evolving population considers the set of subgraphs $S_{tr}^{g}$ extracted from $S_{tr}$ and runs several BSAS procedures with different threshold values $\theta$ where at most $Q$ clusters can be discovered in each run and where the dissimilarity between graphs is evaluated using the nBMF procedure as in Section 3.2.1 by considering the six weights $\Omega$ and (possibly) the parameters $\Pi$, if the vertices and/or nodes dissimilarities are parametric themselves. At the end of the clustering procedures, each cluster is evaluated thanks to the quality index (1) using the parameter $\eta$ for weighting the convex linear combination and clusters whose value is above $\tau_F$ are discarded and their representatives will not form the alphabet. Once the alphabet $\mathcal{A}$ is synthesized, the *Embedder* (Section 3.3) extracts $\mathcal{S}_{tr}^{ge}$ and $\mathcal{S}_{vs}^{ge}$ from $\mathcal{S}_{tr}$ and $\mathcal{S}_{vs}$ and exploits $\mathcal{A}$ in order to map both the training set and the validation set towards a metric space (say $\mathcal{D}_{tr}$ and $\mathcal{D}_{vs}$) using the same GED previously used for BSAS, along with the corresponding parameters $\Omega$ and $\Pi$. The classifier is trained on $\mathcal{D}_{tr}$ and its accuracy is evaluated on $\mathcal{D}_{vs}$. The latter serves as the fitness function for the individual itself. Standard genetic operators (mutation, selection, crossover and elitism) take care of moving from one generation to the next. At the end of the evolution, the best individual is retained, especially the portions of the genetic code $\Omega^\star$ and $\Pi^\star$, along with the alphabet $\mathcal{A}^\star$ synthesized using its genetic code.

### 3.5.2 Feature Selection Phase

The *Granulator* may produce a large set of symbols in $\mathcal{A}^\star$, hence the dimension of the embedding space may result large as well. In order to shrink the dimensionality of the embedding space (i.e., the set of meaningful symbols), a feature selection procedure still based on genetic optimization is in charge to discard unpromising features, hence reducing the number of symbols in $\mathcal{A}^\star$, with a projection mask $\mathbf{m} \in \{0,1\}^{|\mathcal{A}^\star|}$: features corresponding to 1's are retained, whereas features corresponding to 0's are discarded. The projection mask is the genetic code for this second genetic optimization stage.

In this optimization step, each individual from the evolving population projects $\mathcal{D}_{tr}$ and $\mathcal{D}_{vs}$ on the subspace marked by non-zero elements in $\mathbf{m}$, say $\overline{\mathcal{D}}_{tr}$ and $\overline{\mathcal{D}}_{vs}$. The classifier is trained on $\overline{\mathcal{D}}_{tr}$ and its accuracy is evaluated on $\overline{\mathcal{D}}_{vs}$. The fitness function is defined as a convex linear combination between the classifier accuracy on $\overline{\mathcal{D}}_{vs}$ and the cost $\mu$ of the mask $\mathbf{m}$ defined as:

$$\mu = |\mathbf{m} == 1| \ / \ |\mathbf{m}| \tag{10}$$

weighted by a parameter $\alpha \in [0,1]$ which weights performances and sparsity. At the end of the evolution the best projection mask $\mathbf{m}^\star$ is retained and used in order to consider the reduced alphabet $\overline{\mathcal{A}}^\star$.

### 3.6 Synthesized Classification Model

From the two genetic optimization procedures, $\Pi^\star$, $\Omega^\star$ and $\overline{\mathcal{A}}^\star$ are the main actors which completely characterize the classification model, hence the key components in order to classify previously unseen test data. Specifically, given a set of test data $\mathcal{S}_{ts}$, the *Embedder* evaluates $\mathcal{S}_{ts}^{ge}$ and performs the symbolic histograms embedding by matching symbols in $\overline{\mathcal{A}}^\star$ using the GED equipped with parameters $\Omega^\star$ and $\Pi^\star$ (if applicable).

The $K$-NN classifier is trained on $\overline{\mathcal{D}}_{tr}^\star$, namely the training set projected using the best projection mask $\mathbf{m}^\star$ and the final performance is evaluated on the embedded test data.

## 4 TEST AND RESULTS

For addressing the proposed improvements over the original GRALG implementation, different graph datasets from the IAM repository (Riesen and Bunke, 2008) are considered (see Table 1 for list and description). Since labelled graphs on both nodes and edges have been considered, suitable dissimilarity measures have to be defined as well (cf. Section 3.2.1):

- *Letter:* Node labels are real-valued 2-dimensional vectors $\mathbf{v}$ of $x, y$ coordinates and therefore the dissimilarity measure $d_v$ between two given nodes, say $v^{(a)}$ and $v^{(b)}$, is defined as the plain Euclidean distance:

  $$d_v(v^{(a)}, v^{(b)}) = \|\mathbf{v}^{(a)} - \mathbf{v}^{(b)}\|_2$$

  Conversely, edges are not labelled.

- *AIDS:* Node labels are composed by a string value $S_{chem}$ (chemical symbol), an integer $N_{ch}$ (charge) and a real-valued 2-dimensional vector $\mathbf{v}$ of $x, y$ coordinates. For any two given nodes, their dissimilarity is evaluated as:

  $$d_v(v^{(a)}, v^{(b)}) = \|\mathbf{v}^{(a)} - \mathbf{v}^{(b)}\|_2 + |N_{ch}^{(a)} - N_{ch}^{(b)}| + \\ + d_s(S_{chem}^{(a)}, S_{chem}^{(b)})$$

  where $d_s(S_{chem}^{(a)}, S_{chem}^{(b)}) = 1$ if $S_{chem}^{(a)} \neq S_{chem}^{(b)}$, and 0 otherwise. Conversely, the edge dissimilarity is discarded since not useful for the classification task.

- *GREC:* Node labels are composed by a string (*type*) and a real-valued 2-dimensional vector $\mathbf{v}$. The dissimilarity measure $d_v$ between two different nodes is then defined as:

  $$d_v(v^{(a)}, v^{(b)}) = \begin{cases} 1 & \text{if } type^{(a)} = type^{(b)} \\ \|\mathbf{v}^{(a)} - \mathbf{v}^{(b)}\|_2 & \text{otherwise} \end{cases}$$

  Edge labels are defined by an integer value $freq$ (frequency) that defines the number of *(type,angle)*-pairs where, in turn, *type* is a string which may assume two values (namely, *arc* or *line*) and *angle* is a real number. Given two edges, say $e^{(a)}$ and $e^{(b)}$ their dissimilarity is defined as follows:

  1. If $freq^{(a)} = freq^{(b)} = 1$

  $$d_e(e^{(a)}, e^{(b)}) = \begin{cases} \alpha \cdot d^{line}(angle^{(a)}, angle^{(b)}) \\ \quad \text{if } type^{(a)} = type^{(b)} = line \\ \beta \cdot d^{arc}(angle^{(a)}, angle^{(b)}) \\ \quad \text{if } type^{(a)} = type^{(b)} = arc \\ \gamma \quad \text{otherwise} \end{cases}$$

  2. If $freq^{(a)} = freq^{(b)} = 2$

  $$d_e(e^{(a)}, e^{(b)}) = \begin{cases} \frac{\alpha}{2} \cdot d^{line}(angle_1^{(a)}, angle_1^{(b)}) + \\ + \frac{\beta}{2} \cdot d^{arc}(angle_2^{(a)}, angle_2^{(b)}) \\ \quad \text{if } type^{(a)} = type^{(b)} = line \\ \frac{\alpha}{2} \cdot d^{line}(angle_2^{(a)}, angle_2^{(b)}) + \\ + \frac{\beta}{2} \cdot d^{arc}(angle_1^{(a)}, angle_1^{(b)}) \\ \quad \text{if } type^{(a)} = type^{(b)} = arc \\ \gamma \quad \text{otherwise} \end{cases}$$

  3. If $freq^{(a)} \neq freq^{(b)}$

  $$d_e(e^{(a)}, e^{(b)}) = \delta$$

  where $d^{line}$ and $d^{arc}$ are the module distance normalized respectively in $[-\pi, \pi]$ and $[0, arc_{max}]$. $\alpha, \beta, \gamma, \delta \in [0,1]$ is the set of parameters $\Pi$ defined in Section 3.5.1 which shall be optimized by the genetic algorithm.

Table 2: Number of subgraphs extracted ($o = 5$) by the exhaustive procedure.

| Dataset | $\lvert\mathcal{S}_{tr}^g\rvert$ | $\lvert\mathcal{S}_{vl}^g\rvert$ | $\lvert\mathcal{S}_{ts}^g\rvert$ |
|---|---|---|---|
| Letter-L | 21165 | 20543 | 21435 |
| Letter-M | 8582 | 8489 | 8560 |
| Letter-H | 8193 | 7976 | 8111 |
| GREC | 27119 | 28581 | 50579 |
| AIDS | 35208 | 35692 | 220108 |

The implementation has been developed in C++, using the SPARE[3] (Livi et al., 2014) and Boost libraries[4]. Tests have been performed on a workstation with Linux Ubuntu 18.04, 4-cores Intel i7-3770K@3.50GHz equipped with 32GB of RAM.

For the sake of benchmarking, the number of subgraphs extracted from the training set, validation set and test set by the former exhaustive procedure has been reported in Table 2. In our tests, we followed the random extraction procedure defined in Algorithm 1, setting up the maximum number of allowed subgraphs $W$ equal to a given percentage of $|S_{tr}^{g}|$ (cf. Table 2). The subgraphs for the embedding strategy are extracted by following the procedure described in Algorithm 5. Both of the traverse strategies (BFS and DFS) have been considered for comparison and the number of resulting subgraphs needed for the embedding procedure are reported in Table 3.
The system parameters are defined as follow:

- $W = 10\%,\ 30\%,\ 50\%$ of $|S_{tr}^{g}|$
- $o = 5$ the maximum order of the extracted subgraphs
- 20 individuals for the population of both genetic algorithms
- 20 generations for the first genetic algorithm (alphabet optimization)
- 50 generations for the second genetic algorithm (feature selection)
- $\alpha = 1$ in the fitness function for the second genetic algorithm (no weight to sparsity)
- $K = 5$ for the $K$-NN classifier
- $\epsilon = 1.1$ as tolerance value for the symbolic histograms evaluation.

Table 3: Number of subgraphs extracted for the embedding block using Algorithm 5 with BFS and DFS.

| Dataset | Traverse | $\lvert S_{tr}^{ge}\rvert$ | $\lvert S_{vl}^{ge}\rvert$ | $\lvert S_{ts}^{ge}\rvert$ |
|---|---|---|---|---|
| Letter-L | BFS | 5451 | 5371 | 5428 |
| | DFS | 4266 | 4192 | 4253 |
| Letter-M | BFS | 5311 | 5293 | 5243 |
| | DFS | 4336 | 4234 | 4213 |
| Letter-H | BFS | 4513 | 4355 | 4305 |
| | DFS | 4495 | 4391 | 4290 |
| AIDS | BFS | 6701 | 6833 | 41149 |
| | DFS | 11776 | 11893 | 71294 |
| GREC | BFS | 5141 | 5119 | 9508 |
| | DFS | 6076 | 6219 | 11223 |

[3]https://sourceforge.net/projects/libspare/
[4]http://www.boost.org/

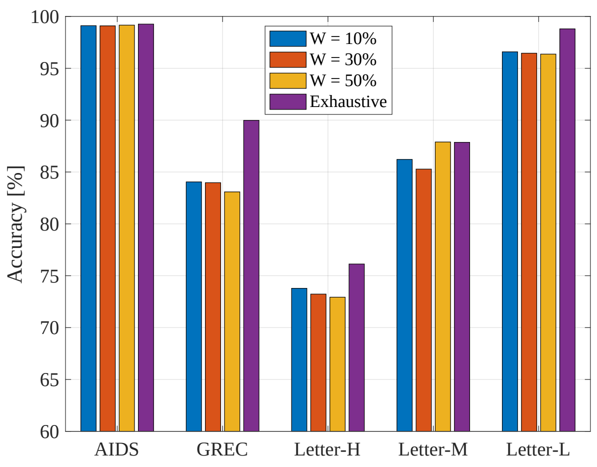
In Figure 1, we compare the performances achieved by the exhaustive procedure in terms of accuracy on the test set (in percentage) and total wall-clock time (in minutes) against the proposed subsampling procedures. Due to the intrinsic randomness in the training procedures, results herein presented have been averaged across five runs. The random extraction procedure has been tested with three different values of $W$, up to a maximum subgraph order $o$. It is noteworthy that aim of our analyses is to investigate on how the subsampling rate impacts on accuracy, memory footprint and running times: as such, all parameters except $W$ itself have been kept constant.

By matching Figures 1a and 1b, it is possible to see that the novel strategies lead to comparable results (in terms of accuracy) with those obtained by the exhaustive procedure for every value of $W$. The only remarkable shift can be observed for GREC (approximately 5%). It is worth remarking that the performances of the classification block are strongly influenced by the efficiency of the mapping function in preserving the graph input space properties into the $\mathbb{R}^m$ space. This can be achieved only if the information granules extracted are indeed meaningful representatives of the considered dataset(s). For all datasets, clearly some properties emerge even by performing a strong subsampling of the prospective subgraphs.

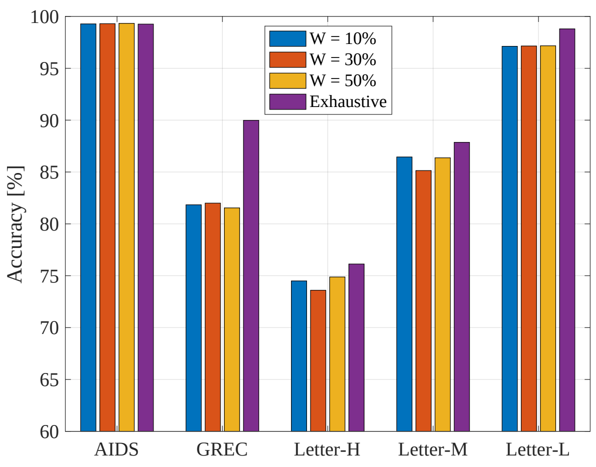
Other than comparable results in terms of accuracy, remarkable improvements in terms of running time can be observed as well (Figures 1c and 1d). This is due to the lower number of subgraphs returned by the *Extractor* driving mainly the *Granulator* and due to the traverse strategy adopted by the *Embedder* before the evaluation of the symbolic histograms. Recalling Section 3.5.1, the genetic algorithm must repeat several times the entire procedure of granulation, embedding and classification in order to optimize the parameters involved. This task involves the GED computation many times, which can be very intensive and time consuming. By matching Table 1 and Figures 1c–1d, clearly the advantages of subsampling are more and more evident as the dataset size increases and/or in presence of complex semantic information on nodes/edges, as their dissimilarity measures impact the overall GED computational burden.

## 5 CONCLUSIONS

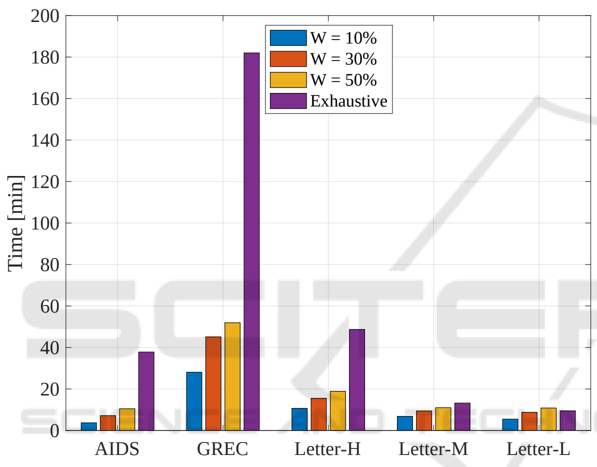
In this paper, we addressed the possibility of designing a Granular Computing-based classification system for labelled graphs by performing stochastic extraction procedures on the training data in order to im-



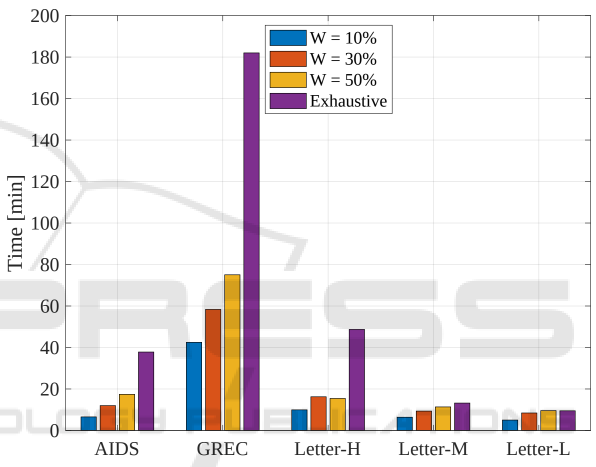
(a) Accuracy on the Test Set (BFS)



(b) Accuracy on the Test Set (DFS)



(c) Overall Running Time (BFS)



(d) Overall Running Time (DFS)

Figure 1: Comparison between the exhaustive procedure and the proposed stochastic sampling.

prove the information granulation procedure both in terms of running time and memory footprint. The hypothesis behind a stochastic granulation procedure is that the information (regularities), whether present in the dataset, can still be observed if subsamples of the dataset itself are considered. In plainer words, meaningful clusters are still visible.

In order to prove this concept, we equipped GRALG with a different granulation procedure that instead of finding information granules on the entire set of possible subgraphs, such subgraphs are extracted by performing stochastic extraction procedures driven by well-known graph traversing algorithms, namely DFS and BFS. These two strategies are also considered when building the embedding space, since the symbolic histograms paradigm relies on counting how many times the symbols from the alphabet appear in the original graphs. Indeed, DFS and BFS have been used to traverse the input graphs and match the resulting subgraphs with the alphabet.

This lightweight procedure for extracting subgraphs both at granulation stage and at embedding stage drastically outperforms the former exhaustive procedure in terms of memory footprint and running times and, at the same time, results in terms of accuracy on the test set are comparable with respect to the former case. The achieved results somehow prove our hypothesis, at least for the considered datasets, showing that clustering techniques may be promising for synthesizing information granules even with random subsampling. This is particularly crucial in Big Data scenarios, where the memory footprint is a delicate issue and where redundancies and noisy patterns can easily be found in massive datasets.

Nonetheless, the overall system keeps the peculiar properties typical of information granulation-based systems, namely the human-interpretability of the synthesized model. Indeed, the resulting information granules can give insights to field-experts about the modelled system. This aspect is stressed by the second genetic optimization, which is in charge of shrinking the alphabet size, hence finding the subset of information granules better related to the semantic behind the classification problem at hand.

As already mentioned, subsampling procedures are appealing especially in Big Data scenarios. As such, future research avenues can consider the implementation of the proposed alphabet synthesis techniques in parallel and distributed frameworks (Dean and Ghemawat, 2008; Zaharia et al., 2010), eventually following multi-agent paradigms (Cao et al., 2009; Altilio et al., 2019), or by means of dedicated hardware (Tran et al., 2016; Cinti et al., 2019) in order to properly face massive datasets and/or datasets with non-trivial semantic information on both nodes and edges. Thanks to these paradigms, the dataset can be shred across several computational units and, most importantly, the GED evaluation can be performed in parallel, being it the most computationally expensive step in the synthesis procedure.

## REFERENCES

Altilio, R., Di Lorenzo, P., and Panella, M. (2019). Distributed data clustering over networks. *Pattern Recognition*, 93:603 – 620.

Bai, X. (2012). *Graph-Based Methods in Computer Vision: Developments and Applications: Developments and Applications*. IGI Global.

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

Bargiela, A. and Pedrycz, W. (2008). Toward a theory of granular computing for human-centered information processing. *IEEE Transactions on Fuzzy Systems*, 16(2):320–330.

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.

Bunke, H. (1997). On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18(8):689 – 694.

Bunke, H. (2000). Graph matching: Theoretical foundations, algorithms, and applications. In *Proceedings of Vision Interface*, pages 82–88.

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.

Bunke, H. and Allermann, G. (1983). Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245 – 253.

Cao, L., Gorodetsky, V., and Mitkas, P. A. (2009). Agent mining: The synergy of agents and data mining. *IEEE Intelligent Systems*, 24(3):64–72.

Cinti, A., Bianchi, F. M., Martino, A., and Rizzi, A. (2019). A novel algorithm for online inexact string matching and its fpga implementation. *Cognitive Computation*.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297.

Cover, T. M. and Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Del Vescovo, G., Livi, L., Frattale Mascioli, F. M., and Rizzi, A. (2014). On the problem of modeling structured data with the minsod representative. *International Journal of Computer Theory and Engineering*, 6(1):9.

Del Vescovo, G. and Rizzi, A. (2007a). Automatic classification of graphs by symbolic histograms. In *2007 IEEE International Conference on Granular Computing (GRC 2007)*, pages 410–416. IEEE.

Del Vescovo, G. and Rizzi, A. (2007b). Online handwriting recognition by the symbolic histograms approach. In *2007 IEEE International Conference on Granular Computing (GRC 2007)*, pages 686–686. IEEE.

Di Noia, A., Martino, A., Montanari, P., and Rizzi, A. (2019). Supervised machine learning techniques and genetic optimization for occupational diseases risk prediction. *Soft Computing*.

Di Paola, L. and Giuliani, A. (2017). *Protein–Protein Interactions: The Structural Foundation of Life Complexity*, pages 1–12. American Cancer Society.

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

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.

Giuliani, A., Filippi, S., and Bertolaso, M. (2014). Why network approach can promote a new way of thinking in biology. *Frontiers in Genetics*, 5:83.

Han, J. and Lin, T. Y. (2010). Granular computing: Models and applications. *International Journal of Intelligent Systems*, 25(2):111–117.

Howard, N. and Lieberman, H. (2014). Brainspace: Relating neuroscience to knowledge about everyday life. *Cognitive Computation*, 6(1):35–44.

Krishnan, A., Zbilut, J. P., Tomita, M., and Giuliani, A. (2008). Proteins as networks: usefulness of graph theory in protein science. *Current Protein and Peptide Science*, 9(1):28–38.

Livi, L., Del Vescovo, G., Rizzi, A., and Frattale Mascioli, F. M. (2014). Building pattern recognition applications with the spare library. *arXiv preprint arXiv:1410.5263*.

Livi, L. and Sadeghian, A. (2016). Granular computing, computational intelligence, and the analysis of non-geometric input spaces. *Granular Computing*, 1(1):13–20.

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., Rizzi, A., and Frattale Mascioli, F. M. (2017). Efficient approaches for solving the large-scale k-medoids problem. In *Proceedings of the 9th International Joint Conference on Computational Intelligence - Volume 1: IJCCI,*, pages 338–347. INSTICC, SciTePress.

Martino, A., Rizzi, A., and Frattale Mascioli, F. M. (2018b). Distance matrix pre-caching and distributed computation of internal validation indices in k-medoids clustering. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Martino, A., Rizzi, A., and Frattale Mascioli, F. M. (2019). 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.

Neuhaus, M. and Bunke, H. (2007). *Bridging the gap between graph edit distance and kernel machines*, volume 68. World Scientific.

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

Pedrycz, W. (2010). Human centricity in computing with fuzzy sets: an interpretability quest for higher order granular constructs. *Journal of Ambient Intelligence and Humanized Computing*, 1(1):65–74.

Pedrycz, W. (2013). Proximity-based clustering: a search for structural consistency in data with semantic blocks of features. *IEEE Transactions on Fuzzy Systems*, 21(5):978–982.

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.

Richiardi, J., Achard, S., Bunke, H., and Van De Ville, D. (2013). Machine learning with brain graphs: predictive modeling approaches for functional imaging in systems neuroscience. *IEEE Signal Processing Magazine*, 30(3):58–70.

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.

Theodoridis, S. and Koutroumbas, K. (2008). *Pattern Recognition*. Academic Press, 4 edition.

Tran, H.-N., Cambria, E., and Hussain, A. (2016). Towards gpu-based common-sense reasoning: Using fast subgraph matching. *Cognitive Computation*, 8(6):1074–1086.

Tsai, W.-H. and Fu, K.-S. (1979). Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Transactions on systems, man, and cybernetics*, 9(12):757–768.

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.

Wang, F. and Sun, J. (2015). Survey on distance metric learning and dimensionality reduction in data mining. *Data Mining and Knowledge Discovery*, 29(2):534–564.

Wang, G., Yang, J., and Xu, J. (2017). Granular computing: from granularity optimization to multi-granularity joint problem solving. *Granular Computing*, 2(3):105–120.

Weinshall, D., Jacobs, D. W., and Gdalyahu, Y. (1999). Classification in non-metric spaces. In Kearns, M. J., Solla, S. A., and Cohn, D. A., editors, *Advances in Neural Information Processing Systems 11*, pages 838–846. MIT Press.

Yang, J., Wang, G., and Zhang, Q. (2018). Knowledge distance measure in multigranulation spaces of fuzzy equivalence relations. *Information Sciences*, 448:18–35.

Yao, Y. (2016). A triarchic theory of granular computing. *Granular Computing*, 1(2):145–157.

Yao, Y. and Zhao, L. (2012). A measurement theory view on the granularity of partitions. *Information Sciences*, 213:1–13.

Yao, Y.-Y. (2008). The rise of granular computing. *Journal of Chongqing University of Posts and Telecommunications (Natural Science Edition)*, 20(3):299–308.

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.

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10. USENIX Association.