





Graph Embedding in Vector Spaces Using Matching-Graphs

Mathias Fuchs¹  and Kaspar Riesen^{1,2} 

¹ Institute of Computer Science, University of Bern, 3012 Bern, Switzerland
`mathias.fuchs@inf.unibe.ch`, `kaspar.riesen@fhnw.ch`

² Institute for Informations Systems, University of Applied Sciences and Arts
Northwestern Switzerland, 4600 Olten, Switzerland

Abstract. An evergrowing amount of readily available data and the increasing rate at which it can be acquired leads to fast developments in many fields of intelligent information processing. Often the underlying data is complex, making it difficult to represent it by vectorial data structures. This is where graphs offer a versatile alternative for formal representations. Actually, quite an amount of graph-based methods for pattern recognition and related fields have been proposed. A considerable part of these methods rely on graph matching. In our recent work we propose a novel encoding of specific graph matching information. The idea of this encoding is to formalize the stable cores of specific classes by means of graphs (called matching-graphs). In the present paper we propose to use these matching-graphs to create a vectorial representation of a given graph. The basic idea is to produce hundreds of matching-graphs first, and then represent each graph g as a binary vector that shows the occurrence of each matching-graph in g . In an experimental evaluation on three data sets we show that this graph embedding is able to improve the classification accuracy of two reference systems with statistical significance.

Keywords: Graph matching · Matching-graphs · Graph edit distance

1 Introduction and Related Work

Pattern recognition is a major field of research which aims at solving various problems like the recognition of facial expressions [1], the temporal sorting of images [2], or enhancing weakly lighted images [3], to name just a few examples. The field of pattern recognition can be divided in two main approaches. *Statistical approaches*, which rely on *feature vectors* for data representation and *structural approaches*, which use *strings*, *trees*, or *graphs* for the same task. Since graphs are able to encode more information than merely an ordered list of numbers, they offer a compelling alternative to vectorial approaches. Hence, they are widely used and adopted in various pattern recognition tasks that range

Supported by Swiss National Science Foundation (SNSF) Project Nr. 200021.188496.

© Springer Nature Switzerland AG 2021

N. Reyes et al. (Eds.): SISAP 2021, LNCS 13058, pp. 352–363, 2021.

https://doi.org/10.1007/978-3-030-89657-7_26

from predicting the demand of medical services [4], over skeleton based action recognition [5], to the automatic recognition of handwriting [6]. The main drawback of graphs is, however, the computational complexity of basic operations, which in turn makes graph based algorithms often slower than their statistical counterparts.

A large amount of graph based methods for pattern recognition have been proposed from which many rely on *graph matching* [7]. Graph matching is typically used for quantifying graph proximity. *Graph edit distance* [8,9], introduced about 40 years ago, is recognized as one of the most flexible graph distance models available. In contrast with many other distance measures (e.g. *graph kernels* [10] or *graph neural networks* [11]), graph edit distance generally offers more information than merely a dissimilarity score, viz. the information which subparts of the underlying graphs actually match with each other (known as *edit path*).

In a recent paper [12], the authors of the present paper propose to explicitly exploit the matching information of graph edit distance. This is done by encoding the matching information derived from *graph edit distance* into a data structure, called *matching-graph*. The main contribution of the present paper is to propose and research another employment of these matching-graphs. In particular, we use these matching-graphs to embed graphs into a vector space by means of subgraph isomorphism. That is, each graph g is represented by a vector of length of the number of matching-graphs available, where each entry in the vector equals 1 if the matching-graph occurs in g and 0 otherwise.

The proposed process of creating vector space embedded representations based on found substructures is similar in spirit to approaches like frequent substructure based approaches [14], subgraph matching kernels [15] or graphlet approaches [16]. The common idea is to first generate a set of subgraphs and treat them as features. In [14] a graph g is represented by a vector that counts the number of times a certain subgraph occurs in g . The subgraphs that are used for embedding are derived via *FSG algorithm* [14]. Related to this in [15] a *Subgraph Matching Kernel (SMKernel)* is proposed. This kernel is derived from the common subgraph isomorphism kernel and counts the number of matchings between subgraphs of fixed sizes in two graphs. Another related approach uses *graphlets* for embedding [16]. Graphlets are small induced subgraphs of fixed size that contain a given set of nodes including all edges.

The major principle of our approach is similar to that of [14–16]. However, the main difference of the above mentioned approaches to our proposal lies in the creation of the subgraphs. We employ graph edit distance to create matching-graphs as basic substructures. These matching-graphs offer a natural way of defining significant and large sets of subgraphs that can be readily used for embedding.

The remainder of this paper is organized as follows. Section 2 makes the paper self-contained by providing basic definitions and terms used throughout this paper. Next, in Sect. 3 the general procedure for creating a matching-graph is explained together with a detailed description of the vector space embedding

for graphs. Eventually, in Sect. 4, we empirically confirm that our approach is able to improve the classification accuracy of two reference systems. Finally, in Sect. 5, we conclude the paper and discuss some ideas for future work.

2 Graphs and Graph Edit Distance – Basic Definitions

2.1 Graph and Subgraph

Let L_V and L_E be finite or infinite label sets for nodes and edges, respectively. A *graph* g is a four-tuple $g = (V, E, \mu, \nu)$, where

- V is the finite set of nodes,
- $E \subseteq V \times V$ is the set of edges,
- $\mu : V \rightarrow L_V$ is the node labeling function, and
- $\nu : E \rightarrow L_E$ is the edge labeling function.

A part of a graph, called a *subgraph*, is defined as follows. Let $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ be graphs. Graph g_1 is a subgraph of g_2 , denoted by $g_1 \subseteq g_2$, if

- $V_1 \subseteq V_2$,
- $E_1 \subseteq E_2$,
- $\mu_1(u) = \mu_2(u)$ for all $u \in V_1$, and
- $\nu_1(e) = \nu_2(e)$ for all $e \in E_1$.

Obviously, a subgraph g_1 is obtained from a graph g_2 by removing some nodes and their incident edges, as well as possibly some additional edges from g_2 .

Two graphs g_1 and g_2 are considered *isomorphic* if there is a matching part for each node and edge of g_1 in g_2 (and vice versa). In this regard it is also required that the labels on the nodes and edges exactly correspond (if applicable).

In close relation to graph isomorphism is *subgraph isomorphism*. Intuitively speaking a subgraph isomorphism states whether a graph is contained in another graph. More formally a graph g_1 is subgraph isomorphic to a graph g_2 if there exists a subgraph $g \subseteq g_2$ that is isomorphic to g_1 . The concept of subgraph isomorphism is one of the building blocks used in our embedding framework (see Sect. 3).

2.2 Graph Matching

When graphs are used to formally represent objects or patterns, a measure of distance or similarity is usually required. Over the years several dissimilarity measures for graphs have been proposed. Some of the most prominent ones would be *graph kernels* [10], *spectral methods* [17], or *graph neural networks* [18].

A kernel is a function that implicitly maps data to a feature space, by representing it in the form of pairwise comparisons [19]. Intuitively, graph kernels measure the similarity between pairs of graphs and thus provide an embedding

in a – typically unknown – feature space. Graphs can also be represented in the form of their *laplacian matrix*. The eigenvalues and eigenvectors of these matrices are known to contain information about the branching and clustering of the nodes and can be used for the definition of various similarity measures [17]. Another emerging graph matching method makes use of *deep neural networks*. Some approaches use neural networks to map the graphs into an Euclidean space [11], while other approaches directly take pairs of graphs as input and output a similarity score [18].

A further prominent graph matching method, which is actually employed in the present paper, is *graph edit distance* [8,9]. One of the main advantages of graph edit distance is its high degree of flexibility, which makes it applicable to virtually any kind of graphs.

Given two graphs g_1 and g_2 , the general idea of graph edit distance is to transform g_1 into g_2 using some *edit operations*. A standard set of edit operations is given by *insertions*, *deletions*, and *substitutions* of both nodes and edges. Sometimes, in other applications additional operations like merging and splitting are used. We denote the substitution of two nodes $u \in V_1$ and $v \in V_2$ by $(u \rightarrow v)$, the deletion of node $u \in V_1$ by $(u \rightarrow \varepsilon)$, and the insertion of node $v \in V_2$ by $(\varepsilon \rightarrow v)$, where ε refers to the empty node. For edge edit operations we use a similar notation.

A set $\{e_1, \dots, e_t\}$ of t edit operations e_i that transform a source graph g_1 completely into a target graph g_2 is called an *edit path* $\lambda(g_1, g_2)$ between g_1 and g_2 . Let $\mathcal{T}(g_1, g_2)$ denote the set of all edit paths transforming g_1 into g_2 while c denotes the cost function measuring the strength $c(e_i)$ of edit operation e_i . The graph edit distance between $g_1 = (V_1, E_1, \mu_1, \nu_1)$ and $g_2 = (V_2, E_2, \mu_2, \nu_2)$ can now be defined as follows.

$$d_{\lambda_{\min}}(g_1, g_2) = \min_{\lambda \in \mathcal{T}(g_1, g_2)} \sum_{e_i \in \lambda} c(e_i) \quad , \quad (1)$$

Optimal algorithms for computing the edit distance are computationally demanding, as they rely on combinatorial search procedures. In order to counteract this problem we use the often used approximation algorithm BP [20]. The approximated graph edit distance between g_1 and g_2 computed by algorithm BP is termed $d_{\text{BP}}(g_1, g_2)$ from now on.

3 Graph Embedding by Means of Matching-Graphs

The general idea of the proposed approach is to embed a given graph into a vector space by means of *matching-graphs*. These matching-graphs are built by extracting information on the matching of pairs of graphs and by formalizing and encoding this information in a data structure. Matching-graphs can be interpreted as denoised core structures of their respective class. The idea of matching-graphs emerged in [12] where this data structure is employed for the first time for improving the overall quality of graph edit distance. In the next subsection we first formalize the graph embedding, and in Subsection 3.2 we then describe in detail the creation of the matching-graphs.

3.1 Graph Embedding Using Matching-Graphs

Let g be an arbitrary graph stemming from a given set of graphs. Using a set $\mathcal{M} = \{m_1, \dots, m_N\}$ of matching-graphs, we embed g as follows

$$\varphi(g) = (sub(m_1, g), \dots, sub(m_N, g)),$$

where

$$sub(m_i, g) = \begin{cases} 1 & \text{if } m_i \subseteq g \\ 0 & \text{else} \end{cases}$$

That is, for our embedding we employ subgraph isomorphism that provides us with a binary similarity measure which is 1 or 0 for subgraph-isomorphic and non-subgraph-isomorphic graphs, respectively. There are various algorithms available that can be applied to the subgraph isomorphism problem. Namely various tree search based algorithms [21,22], as well as decision tree based techniques [23]. In the present paper we employ the VF2 algorithm which makes use of efficient heuristics to speed up the search [22].

Obviously, our graph embedding produces binary vectors with a dimension that is equal to the number of matching-graphs actually available. This specific graph embedding is similar in spirit to the frequent substructure approaches [14], the subgraph matching kernel [15], or graphlet kernel [16] reviewed in the introduction of the present paper. However, the special aspect and novelty of our approach is the employment of matching-graphs for embedding.

3.2 Creating Matching-Graphs

In order to produce the N matching-graphs for embedding, we pursue the following procedure. We consider a pair of graphs g_i, g_j for which the graph edit distance is computed. Resulting from this a (suboptimal) edit path $\lambda(g_i, g_j)$ can be obtained. For each edit path $\lambda(g_i, g_j)$, two matching-graphs $m_{g_i \times g_j}$ and $m_{g_j \times g_i}$ are now built (one for the source graph g_i and one for the target graph g_j). These matching-graphs contain all nodes of g_i and g_j that are substituted according to edit path $\lambda(g_i, g_j)$. All nodes that are deleted in g_i or inserted in g_j are not considered in either of the two matching-graphs.

We observe isolated nodes in some experiments. Graph edit distance can handle isolated nodes. However we still decide to remove isolated nodes from our matching-graphs because we aim at building as small as possible cores of the graphs that are actually connected. Note that we also remove incident edges of nodes that are not included in the resulting matching-graphs.

An edge $u_1, u_2 \in E_i$ that connects two substituted nodes $u_1 \rightarrow v_1$ and $u_2 \rightarrow v_2$, is added to the matching-graph $m_{g_i \times g_j}$, if, and only if, there is an edge $(v_1, v_2) \in E_j$ available.

Using the described procedure for creating a matching-graph out of two input graphs, we now employ a simplified version of an iterative algorithm [13] that builds a set of matching-graphs. The algorithm takes as input k sets of graphs

$G_{\omega_1}, \dots, G_{\omega_k}$ with graphs from k different classes $\omega_1, \dots, \omega_k$ as well as the number of matching-graphs kept from one iteration to another (see Algorithm 1).

Algorithm 1: Algorithm for iterative matching-graph creation.

input : sets of graphs from k different classes $\mathcal{G} = \{G_{\omega_1}, \dots, G_{\omega_k}\}$, the maximum number n of matching-graphs to keep in each iteration
output: sets of matching-graphs for each of the k different classes $\mathcal{M} = \{M_{\omega_1}, \dots, M_{\omega_k}\}$

```

1 Initialize  $\mathcal{M}$  as the empty set:  $\mathcal{M} = \{\}$ 
2 foreach set of graphs  $G \in \mathcal{G}$  do
3   Initialize  $M$  as the empty set:  $M = \{\}$ 
4   foreach pair of graphs  $g_i, g_j \in G \times G$  with  $j > i$  do
5      $M = M \cup \{m_{g_j \times g_i}, m_{g_i \times g_j}\}$ 
6   end
7   do
8      $M' =$  a subset of  $n$  random elements of  $M$ 
9     foreach pair of graphs  $m_i, m_j \in M' \times M'$  with  $j > i$  do
10        $M = M \cup \{m_{m_j \times m_i}, m_{m_i \times m_j}\}$ 
11     end
12   while  $M$  has changed in the last iteration
13    $\mathcal{M} = \mathcal{M} \cup M$ 
14 end

```

The algorithm iterates over all k sets (classes) of graphs from \mathcal{G} (main loop of Algorithm 1 from line 2 to line 14). For each set of graphs G and for all possible pairs of graphs g_i, g_j stemming from the current set G , the initial set of matching-graphs M is produced first (line 3 to 6). Eventually, we aim at iteratively building matching-graphs out of pairs of existing matching-graphs. The motivation for this procedure is to further reduce the size of the matching-graphs and to find small core-structures that are often available in the corresponding graphs. Due to computational limitations, we have to randomly select a subset of size n from the current matching-graphs in M (line 8)¹. Based on this selection, the next generation of matching-graphs is built. This process is continued until no more changes occur in set M . Finally, set \mathcal{M} – actually used for graph embedding – is compiled as the union of all matching-graphs individually produced for all available classes.

The dimension of the created vectors directly depends on the number of matching-graphs. Hence, our method might result in feature vectors that are initially very large. In order to reduce potential redundancies and select informative matching-graphs, we apply a recursive feature elimination based on feature weights of random forests on our graph embeddings [24].

¹ Qualitative results of our research show, that the finally created matching-graphs do not differ substantially regardless the initial random set of graphs. Hence, the process of creating matching-graphs is not executed in multiple iterations.

4 Experimental Evaluation

4.1 Experimental Setup

From an experimental point of view we aim at answering the following question. Can the created feature vectors (based on our novel matching-graphs) be used to improve the classification accuracy of existing procedures where the graph matching distances are directly used as a basis for classification? In order to answer this question, we compare our embedding with two reference systems in a classification experiment.

The first reference system is a k -nearest-neighbor classifier (k -NN) that directly operates on d_{BP} (denoted as k -NN(d_{BP})). The second reference system is a Support Vector Machine (denoted as SVM($-d_{BP}$)) that exclusively operates on a similarity kernel $\kappa(g_i, g_j) = -d_{BP}(g_i, g_j)$ [25]. For classifying the embedded graphs, we also employ an SVM that operates on the embedding vectors (using standard kernel functions for feature vectors). We denote our novel approach as SVM_{vec}.

We chose the above mentioned classifiers as a baseline, because our goal is to leverage the power of graph edit distance to build a novel graph representation. That is, we decide to compare our novel method with these classifiers that are often used in conjunction with graph edit distance.

The proposed approach is evaluated on three different data sets representing molecules. The first two sets stem from the IAM graph repository [26]² (AIDS and Mutagenicity) and the third originates from the National Cancer Institute [27]³ (NCI1).

Each data set consists of two classes. The AIDS data set consists of two classes that represent molecules with activity against HIV or not. Mutagenicity consists of molecules with or without the *mutagen* property, whereas the NCI1 data set consists of chemical compounds that contain activity against non-small cell lung cancer or not. For all data sets the nodes contain a discrete label (symbol of the atom) whereas the edges have no labels.

For the experimental evaluation each data set is split in to three predefined random disjoint sets for training, validation, and testing. Details about the size of the individual splits can be found in Table 1.

4.2 Validation of Metaparameters

For the BP algorithm that approximates the graph edit distance the following parameters are commonly optimized. The costs for node and edge deletions, as well as a weighting parameter $\alpha \in [0, 1]$ that is used to trade-off the relative importance of node and edge costs. However, for the sake of simplicity we employ unit cost of 1.0 for both deletions and insertions of both nodes and edges and optimize the weighting parameter α only.

² www.iam.unibe.ch/fki/databases/iam-graph-database.

³ <https://ls11-www.cs.tu-dortmund.de/staff/morris/graphkerneldatasets>.

Table 1. We show the total number of graphs for each data set as well as the corresponding number of graphs in the training, validation, and test sets.

Data set	Total	Training	Validation	Test
AIDS	2,000	250	250	1,500
Mutagenicity	4,337	1,500	500	2,337
NCI1	4,110	2,465	822	823

For the creation of the matching-graphs – actually also dependant on graph edit distance – the same cost parameters are employed. For the iterative matching-graph creation process (Algorithm 1) we set the number of matching-graphs considered for the next iteration to $n = 200$ for all data sets. The stop criterion of the iterative process checks whether or not the last iteration resulted in a change of the underlying set M . Hence, the final number of matching-graphs to be employed for graph embedding is self-controlled by the algorithm.

As discussed in Sect. 3.1 the dimension of the created vectors initially refers to the number of matching-graphs. As our method might generate thousands of matching-graphs, the dimension of the resulting vectors can be very large. Hence, we apply the feature selection process as discussed in Sect. 3.2.

In Fig. 1 we can see the cross validation accuracy as a function of the number of features after each step of the recursive feature elimination process. It is clearly visible that if the dimension of the vectors becomes too small, the validation accuracy drops by a large margin. However, before this significant drop the accuracy remains relatively stable. In Table 2 we compare the number of selected features and the total amount of available features for all data sets. On AIDS and Mutagenicity about 4% of the originally available features are selected, while on NCI1 about 13% of the features are finally used.

Table 2. The amount of features created for each data set and the final amount used after feature selection.

	AIDS	Mutagenicity	NCI1
Total features	4,955	86,752	4,544
Selected features	199	4,139	618

For the optimization of the SVM that operates on our embedded graphs we evaluate three different standard kernels for feature vectors, viz. the Radial Basis Function (RBF), Linear kernel, and a Sigmoid kernel [28]. For all functions we optimize parameter C , which is the trade off between margin maximization and error minimization. In case of RBF and Sigmoid kernel also parameter γ is optimized (all optimizations are conducted by means of an exhaustive grid search).

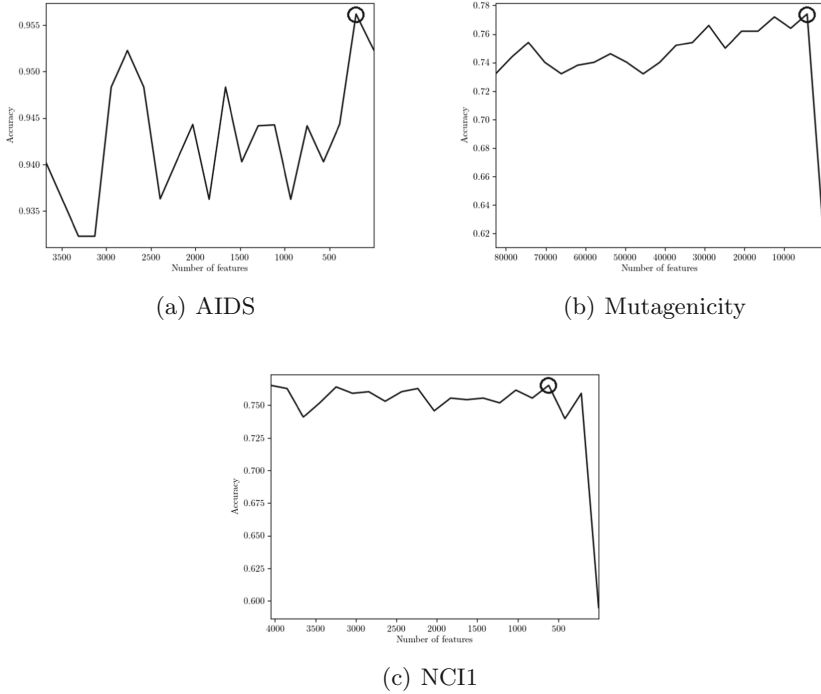


Fig. 1. Cross validation accuracy as a function of the number of features during the recursive feature elimination process. The global optimum is indicated with a small circle.

4.3 Test Results and Discussion

In Table 3 we show the classification accuracies of both reference systems, viz. k -NN(d_{BP}) and SVM($-d_{BP}$), as well as the results of our novel approach SVM_{vec} on all data sets.

We observe that our approach achieves better classification results compared to both baseline classifiers on all data sets. On the Mutagenicity data set our approach outperforms both reference systems with statistical significance. On AIDS and NCI1 we achieve a statistically significant improvement compared with the first and second reference system, respectively. The statistical significance is based on a Z-test using a significance level of $\alpha = 0.05$.

A more detailed analysis of the validation and test results on the Mutagenicity data set brings to light the following interesting result (see Table 4). While for both reference systems the validation and test accuracies are more or less stable, we observe a massive overfitting of our novel approach. That is, the classification accuracy drops from 88.2% on the validation set to 76.3% on the test set. Note that this effect is visible on this specific data set only and needs further investigations in future work.

Table 3. Classification accuracies of two reference systems (a k -NN classifier that operates on the original edit distances (k -NN(d_{BP})) and an SVM that uses the same edit distances as kernel values (SVM($-d_{BP}$))) and our proposed system (an SVM using the embedded graphs (SVM_{vec})). Symbol \circ/\circ indicates a statistically significant improvement over the first and second system, respectively. We are using a Z-test at significance level $\alpha = 0.05$.

Data Set	Reference Systems		Ours
	k -NN(d_{BP})	SVM($-d_{BP}$)	SVM _{vec}
AIDS	98.6	99.4	99.6 $\circ/-$
Mutagenicity	72.4	69.1	76.3 \circ/\circ
NCI1	74.4	68.6	76.7 $-/\circ$

Table 4. The difference of validation and test accuracies using the different classifiers on the Mutagenicity data set. The effect of overfitting of our novel system is clearly observable.

Data Set	Reference Systems				Ours	
	k -NN(d_{BP})		SVM($-d_{BP}$)		SVM _{vec}	
	va	te	va	te	va	te
Mutagenicity	74.8	72.4	69.8	69.1	88.2	76.3

5 Conclusions and Future Work

In the present paper we propose to use matching-graphs – small, pre-computed core structures extracted from a training set – to build vector representations of graphs. The matching-graphs are based on the edit path between pairs of graphs. In particular, the resulting matching-graphs contain only nodes that are actually substituted via graph edit distance. First, we build a relatively large set of N small matching-graphs by means of an iterative procedure. Eventually, we embed our graphs in an N -dimensional vector space such that the i -th dimension corresponds to the i -th matching-graph. More formally, each entry of the resulting vector represents whether or not the corresponding matching-graph occurs as a subgraph in the graph to be embedded. Finally, we reduce the dimension of the created vectors by means of a standard feature selection. Hence we follow the paradigm of overproducing and selecting features.

By means of an experimental evaluation on three graph data sets, we empirically confirm that our novel approach is able to statistically significantly improve the classification accuracy when compared to classifiers that directly operate on the graphs.

For future work we see several rewarding paths to be pursued. First, we aim at evaluating the procedure on additional data sets and in this regard apply it also on data sets with continuous labels. Furthermore it could be interesting to

employ the vectorized representation in conjunction with other classifiers and compare our approach with other subgraph or graphlet based kernels.

References

1. Jain, N., Kumar, S., Kumar, A., Shamsolmoali, P., Zareapoor, M.: Hybrid deep neural networks for face emotion recognition. *Pattern Recogn. Lett.* **115**, 101–106 (2018)
2. Padilha, R., Andaló, F.A., Lavi, B., Pereira, L.A., Rocha, A.: Temporally sorting images from real-world events. *Pattern Recogn. Lett.* **147**, 212–219 (2021)
3. Li, C., Guo, J., Porikli, F., Pang, Y.: LightenNet: a convolutional neural network for weakly illuminated image enhancement. *Pattern Recogn. Lett.* **104**, 15–22 (2018)
4. Jin, R., Xia, T., Liu, X., Murata, T., Kim, K.S.: Predicting emergency medical service demand with bipartite graph convolutional networks. *IEEE Access* **9**, 9903–9915 (2021)
5. Feng, D., Wu, Z., Zhang, J., Ren, T.: Multi-scale spatial temporal graph neural network for skeleton-based action recognition. *IEEE Access* **9**, 58256–58265 (2021)
6. Fischer, A., Suen, C.Y., Frinken, V., Riesen, K., Bunke, H.: A fast matching algorithm for graph-based handwriting recognition. In: Kropatsch, W.G., Artner, N.M., Haxhimusa, Y., Jiang, X. (eds.) *Graph-Based Representations in Pattern Recognition. GBRPR 2013. Lecture Notes in Computer Science*, **7877**, 194–203. Springer, Berlin, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38221-5_21
7. Conte, D., Foggia, P., Sansone, C., Vento, M.: Thirty years of graph matching in pattern recognition. *Int. J. Pattern Recogn. Artif. Intell.* **18**(03), 265–298 (2004)
8. Bunke, H., Allermann, G.: Inexact graph matching for structural pattern recognition. *Pattern Recogn. Lett.* **1**(4), 245–253 (1983)
9. Sanfeliu, A., Fu, K.S.: A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Syst. Man Cybern.* **3**, 353–362 (1983)
10. Vishwanathan, S.V.N., Schraudolph, N.N., Kondor, R., Borgwardt, K.M.: Graph kernels. *J. Mach. Learn. Res.* **11**, 1201–1242 (2010)
11. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. *IEEE Trans. Neural Netw.* **20**(1), 61–80 (2008)
12. Fuchs, M., Riesen, K.: Matching of matching-graphs - a novel approach for graph classification. In: *Proceedings of the 25th International Conference on Pattern Recognition, ICPR 2020, Milano, Italy, 10–15 January 2021*
13. Fuchs, M., Riesen, K.: Iterative creation of matching-graphs - finding relevant substructures in graph sets. In: *Proceedings of the 25th Iberoamerican Congress on Pattern Recognition, CIARP25 2021, Porto, Portugal, 10–13 May 2021*
14. Deshpande, M., Kuramochi, M., Wale, N., Karypis, G.: Frequent substructure-based approaches for classifying chemical compounds. *IEEE TKDE* **17**(8), 1036–1050 (2005)
15. Kriege, N., Mutzel, P.: Subgraph matching kernels for attributed graphs (2012). arXiv preprint: [arXiv:1206.6483](https://arxiv.org/abs/1206.6483)
16. Shervashidze, N., Vishwanathan, S.V.N., Petri, T., Mehlhorn, K., Borgwardt, K.: Efficient graphlet kernels for large graph comparison. In: *Artificial Intelligence and Statistics*, pp. 488–495. PMLR (2009)
17. Kannan, N., Vishveshwara, S.: Identification of side-chain clusters in protein structures by a graph spectral method. *J. Mol. Biol.* **292**(2), 441–464 (1999)

18. Li, Y., Gu, C., Dullien, T., Vinyals, O., Kohli, P.: Graph matching networks for learning the similarity of graph structured objects. In: International Conference on Machine Learning, pp. 3835–3845. PMLR (2019)
19. Cristianini, N., Shawe-Taylor, J.: An Introduction To Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press, Cambridge (2000)
20. Riesen, K., Bunke, H.: Approximate graph edit distance computation by means of bipartite graph matching. *Image Vision Comput.* **27**(7), 950–959 (2009)
21. Ullmann, J.R.: An algorithm for subgraph isomorphism. *J. ACM (JACM)* **23**(1), 31–42 (1976)
22. Cordella, L.P., Foggia, P., Sansone, C., Vento, M.: A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* **26**(10), 1367–1372 (2004)
23. Messmer, B.T., Bunke, H.: A decision tree approach to graph and subgraph isomorphism detection. *Pattern Recogn.* **32**(12), 1979–1998 (1999)
24. Darst, B.F., Malecki, K.C., Engelman, C.D.: Using recursive feature elimination in random forest to account for correlated variables in high dimensional data. *BMC Genet.* **19**(1), 1–6 (2018)
25. Neuhaus, M., Bunke, H.: Bridging the Gap Between Graph Edit Distance And Kernel Machines, **68**. World Scientific, Singapore (2007)
26. Riesen, K., Bunke, H.: IAM graph database repository for graph based pattern recognition and machine learning. In: da Vitoria Lobo, N., et al. (eds.) Structural, Syntactic, and Statistical Pattern Recognition. SSPR/SPR 2008. Lecture Notes in Computer Science, **5342**, 287–297. Springer, Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-89689-0_33
27. Wale, N., Karypis, G.: Comparison of descriptor spaces for chemical compound retrieval and classification. In: Sixth International Conference on Data Mining (ICDM 2006), pp. 678–689 (2006). <https://doi.org/10.1109/ICDM.2006.39>
28. Lin, H.T., Lin, C.J.: A study on sigmoid kernels for SVM and the training of non-PSD kernels by SMO-type methods. *Neural Comput.* **3**(1–32), 16 (2003)