Symmetry Kernel for Graph Classification

Yannick Kuhar
University of Ljubljana; Faculty of Computer and Information Science
Ljubljana, Slovenia *yannick.kuhar@fri.uni-lj.si*

Uroš Čibej
University of Ljubljana; Faculty of Computer and Information Science
Ljubljana, Slovenia *uros.cibej@fri.uni-lj.si*

Abstract

This paper presents a novel way to conduct machine learning analysis on graphs and empirically evaluates it. We can not perform such analysis on non-fixed length feature vectors, so first, we must find a way to represent graphs as such. We propose a graph kernel based on graph automorphisms, also known as graph symmetries. We then empirically evaluate the classification accuracy of three machine learning algorithms, SVM, Random Forest, and AdaBoost, using this novel graph kernel against two existing graph kernels and a naive baseline. The models reach a higher classification accuracy on some datasets using our Symmetry kernels than the graphlet kernel and Weisfeiler-Lehman kernel despite our kernel constructing far smaller feature vectors than the existing approaches.

Keywords: graph symmetries, graph kernels, graph classification, machine learning

1. Introduction

Machine learning on complex data structures has become a popular area of research. In many domains, such as chemistry, bioinformatics, social network analysis, and computer vision, data naturally appears as graphs [1]. To conduct machine learning analysis on data not represented as fixed-length feature vectors, we must first find a way to describe them as such. One way to do this is by using graph kernels. They are functions that map graphs to vectors and use them to measure the similarity between graphs. Researchers have proposed various graph kernels [1]. Some utilize subgraph structure as the basis of the mapping, while others use random walks. Recently, researchers have explored graph kernels based on embeddings created by graph neural networks [2].

The Weisfeiler-Lehman [3] kernel has proven to be successful in practice. It and many other kernels have one drawback, which we highlighted in this study. They construct sizable feature vectors, which can cause computational difficulties, storage problems, and lengthy machine-learning model training times. Our Symmetry kernel is designed to build smaller feature vectors to bypass these problems altogether. In recent years, many contributions have been made to the field of graph kernels. One proposed four hypergraph kernels, exploiting the native multi-scale organization of complex networks [16]. Another defined a kernel based on features constructed by a graph autoencoder framework [17]. Graph kernels have also been successfully used to detect malware [18].

Symmetries naturally appear in structures such as graphs and have been successfully utilized in the domain of graph compression [4]. In this paper, we combine them with a subgraph counting approach [10] used in bioinformatics to describe a graph's local structure to create a novel graph kernel. Our starting point is the Graphlet kernel [9]. It computes the distribution of all smaller graphlets in a larger graph. Those graphs, however, exhibit symmetries. We take the distribution of all graphlets and calculate the distribution of symmetries. The distribution of all symmetries is the feature vector of the graph. Finally, our kernel uses all such vectors to calculate a gram matrix. With it, we train several models using SVM, Random Forest, and AdaBoost. To evaluate our kernel empirically, we calculate the classification accuracy of our models using twelve biochemical datasets.

This paper is structured into five sections. In the second section, we present all preliminaries. The third section describes our graph kernel in detail. In the fourth section, we present our results; in the final section, we give our concluding remarks.

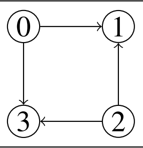
2. Preliminaries

This section defines graph symmetries, formalizes the concept of graph kernels, and describes relevant machine-learning methods.

2.1. Graphs symmetries

A graph is defined as $G = (V, E)$, where $V(G)$ is the set of vertices and $E(G)$ is the set of edges. We assume simple graphs with no isolated vertices. Graph symmetries are a synonym for graph automorphisms. Their standard representation is a permutation of the vertex set, i.e., a bijective function $\pi : V(G) \to V(G)$ that preserves connectivity ($(u, v) \in E(G) \Rightarrow (\pi(u), \pi(v)) \in E(G)$). All such permutations form a permutation group $Aut(G)$ [4], an example is shown in Table 1.

Table 1. On the left is a simple directed graph G with four vertices and four arcs. All permutations of $Aut(G)$ are written in standard cycle notation on the right.

G	Aut(G)
	()
	(13)
	(02)
	(02)(13)

Graph symmetries have been successfully used in graph compression [4], but in this study, we will explore their use to construct machine learning features.

2.2. Machine learning on graphs

Researchers split the field of Machine learning on graphs into three sub-fields, depending on what kind of labels we are predicting:

- Vertex classification: feature vectors represent graph vertices in this space, and machine learning models predict a vertex-level property. For example, we wish to predict the document type in a citation network.
- Edge classification: feature vectors usually combine two vertex feature vectors. Machine learning models here predict whether an edge will be formed between two vertices. For example, we want to know which social network users will be friends.
- Graph classification: feature vectors represent entire graphs. Typically, they are designed by either incorporating statistical properties of graphs or combining all vertex embeddings. Machine learning models predict some graph properties. For example, graphs represent molecules, and we wish to know if a molecule helps combat a specific disease.

This paper will focus on graph classification.

2.3. Graph Kernels

A graph kernel $k(G, G')$ is a function that measures the similarity between graphs G and G'. The function must be:

- symmetric i.e. $k(G, G') = k(G', G)$ and
- positive semi-definite [5].

Typically, they are defined as:

$$k_g(G, G') = \phi(G)^\top \phi(G'),$$

where $\phi(G)$ maps graphs to fixed-length feature vectors. When designing a kernel, the main question is how we define such a mapping. Over the years, researchers have proposed many graph kernels based on different properties. Some kernels use shortest paths [6] or random walks [7], others use limited-sized subgraphs and subtrees [8]. In recent years, graph neural networks [2] have become a popular tool for graph analysis but are out of scope for this paper.

From these various kernels, we have selected two to use in our experiments.

Graphlet Kernel

The graphlet kernel (GK) is a tool used to compare graphs by counting smaller subgraphs (graphlets), usually limited to k vertices for $k \in \{3, 4, 5\}$ [9]. Let

$$\mathbb{G} = \{graphlet(1), graphlet(2), \ldots, graphlet(N_k)\}$$

be the set of graphlets of size k. First, we must define a vector f_G of length N_k. The i-th component of vector f_G is the count of graphlet $graphlet(i) \in \mathbb{G}$ that occurs in the input graph G. The authors then normalize the count vector f_G to account for different input graph sizes.

To compute this kernel exactly is too expensive on large graphs. To solve this issue, the authors resort to sampling. The graphlet distribution is then calculated over the sampled subgraphs in hopes that this distribution approximates the real one. The result is an efficient graph kernel that can be used for comprehensive graph analysis.

Weisfeiler-Lehman Kernel

This graph kernel is based on the 1-dimensional Weisfeiler-Lehman (WL) isomorphism test [3], also known as the "naive node refinement." Each node starts with an initial label or color. The idea of this algorithm is to assign new labels to each node in the following way:

- it collects the labels of all neighboring vertices,
- stores them into a sorted set (this includes the label of the target vertex),
- compresses the label set into a new label and
- assigns it to the target vertex.

The algorithm terminates if the newly created sets of two input graphs G and G' are not the same i.e. graphs G and G' are not isomorphic or the number of iterations h is reached. The feature vector constructed by this kernel is a vector that counts how many times a color has appeared during the color refinement process. Given graph classification tasks, the WL kernel proved competitive regarding runtime and classification accuracy [3].

2.4. Machine learning algorithms

In our experiments, we used three well-known machine-learning algorithms:

- Support vector machine (SVM) is a classifier that builds a decision boundary (linear or not linear) between classes [11]. It is known as a well-performing, out-of-the-box model.
- Random forest is an ensemble classifier that trains a more significant number of decision trees. The main purpose of an ensemble approach is to combat the high variance of decision trees [13].
- AdaBoost is also an ensemble classifier of typically $500 - 1000$ weaker classifiers [12] such as decision trees. When the algorithm trains the first classifier, it assigns weights to the data points based on their results. Misclassified data has increased (boosted) weights. The data point will likely be included in the next classifier's test set based on these weights. This approach yields a successful classifier.

3. Symmetry Kernel

This section defines a novel graph kernel based on the concepts of graph symmetries and the graphlet kernel.

In bioinformatics, the graphlet distribution has been a successful tool for network analysis [10]. As mentioned, the graphlet kernel uses all possible graphlets with $k \in \{3, 4, 5\}$ vertices. All such graphlets exhibit symmetries. We construct the graph features by first computing the graphlet distribution of an input graph G. We compute the $Aut(g)$ for each graphlet g in the graphlet distribution and then construct the feature vectors based on the distribution of symmetries. For example, consider graphs H and H' shown in Figure 1, their graphlet distributions are:

$$\phi_{graphlet}(H) = (8, 2, 6, 0, 1, 4, 0, 0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0) \text{ and}$$

$$\phi_{graphlet}(H') = (16, 2, 16, 4, 1, 6, 0, 0, 10, 2, 0, 4, 8, 2, 2, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0).$$

The i-th component of each of the above vectors corresponds to the i-th graphlet. In all graphlets with $k \in \{3, 4, 5\}$ vertices there exist seven lengths of symmetries:
$(2), (2)(2), (2)(3), (3), (3)(2), (4)$ and (5). The distributions on our test graphs are then:

$$\phi_{symmetries}(H) = (24, 11, 0, 0, 0, 2, 0) \text{ and}$$
$$\phi_{symmetries}(H') = (60, 45, 0, 8, 0, 2, 8).$$

To account for different sizes of graphs, we will run experiments on a normalized variant of this kernel.

3.1. Efficient graphlet counting

To calculate the distributions explained in Section 3 in a reasonable time, we used the efficient Orca [10] graphlet counting algorithm. It aims to calculate the number of times some vertex x appears in each orbit O_i across all graphlets with $k \in \{3, 4, 5\}$ vertices in graph G by building and solving a system of equations.

The right sides of the equations contain graph properties. The left sides of the equations are relations between orbit counts o_i. The key observation that makes this approach possible is that if some vertex x is in a k-vertex graphlet, it is also in a $(k-1)$-vertex graphlet. The authors use this observation in reverse. For example, every 4-vertex graphlet can be constructed by adding

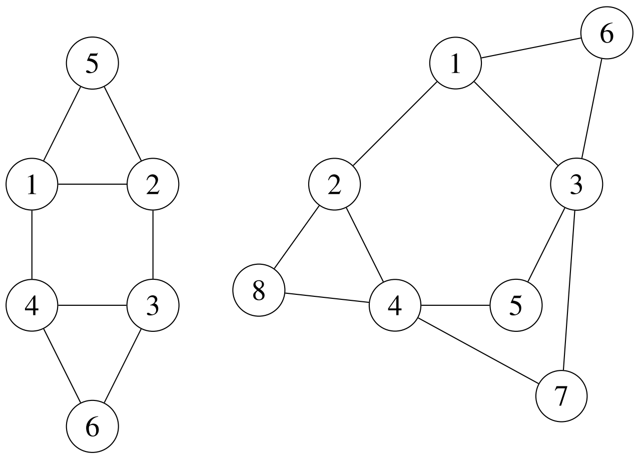


Fig. 1. Example graphs H and H'.

a vertex to a 3-vertex graphlet. They then find the relations between orbits by enumerating all 3-vertex graphlets that touch some vertex x and count all extensions while considering that all summations must consider symmetries.

This strategy is sound when counting orbits of graphlets with $k \in \{4\}$ vertices. The authors constructed a list of equations by adding vertices to 3-vertex and observing the results. This approach is not practical for 5-vertex graphlets as it would yield a large number of equations that are not linearly independent; therefore, a different approach is necessary. For each orbit, the authors chose some vertex y. They then observed how the corresponding graphlet's orbits and the vertex of interest x behaved if they added edges between y and other vertices in the graphlet.

In all experiments, Orca has proven to be the fastest algorithm compared to the existing approaches and will be used to calculate the graphlet distribution by our kernel.

4. Results

This section briefly describes the datasets we used and our machine-learning evaluation pipeline. It will also present our results.

4.1. Data

The datasets AIDS, Mutagenicity, NCI1, NCI109, BZR, COX2, DHFR, MUTAG, and PTC are graphs created from small molecules. The vertices of these graphs represent the atoms, and the edges represent chemical bonds [1]. The binary classification task on these datasets is to predict specific properties of these molecules' toxicity and activity against aids or cancer. PROTEINS_full, on the other hand, is a set of graphs derived from bioinformatics [1].

We used these datasets because they have become the standard for evaluating graph kernels [1, 3, 9].

4.2. Experimental setting

We tested our symmetric kernel against the graphlet kernel, counting all graphlets with up to 5 vertices and the WL subtree kernel with four iterations, using vertex histogram as its basis.

We use SVM, Random forest, and Adaboost machine learning algorithms. First, we split each dataset according to the standard 70:30 split. Then, we fine-tune each model's parameters using grid search Cross-validation. We fine-tune the regularization parameter, the internal kernel type, and the kernel coefficient for SVM. For Random forest, we fine-tune the number of decision trees and the function that measures the split quality. For Adaboost, we fine-tuned

Table 2. The datasets used in our experiments alongside some of their properties. N is the number of graphs in a dataset. Classes shows the number of properties we predict, and the Largest class is the percentage of the most common class. Lastly, $|V(G)|$ and $|E(G)|$ are the average number of vertices and edges, respectively.

Name	N	Classes	Largest class	|V(G)|	|E(G)|
AIDS	2000	2	0.8000	15.69	16.20
Mutagenicity	4337	2	0.5536	30.32	30.77
NCI1	4110	2	0.5005	29.87	32.30
NCI109	4127	2	0.5038	29.68	32.13
PROTEINS_full	1113	2	0.5957	39.06	72.82
BZR	405	2	0.7816	35.75	38.36
COX2	467	2	0.7816	41.22	43.45
DHFR	756	2	0.6098	42.43	44.54
MUTAG	188	2	0.6648	17.93	19.79
PTC_FM	349	2	0.5926	14.11	14.48
PTC_FR	351	2	0.6553	14.56	15.00
PTC_MM	336	2	0.6161	13.97	14.32

the number of estimators and the learning rate. Lastly, we train 30 models with each algorithm, each trained on a bootstrapped training set sample. All reported results are the means of the evaluated models.

We will evaluate the quality of all models with classification accuracy (CA), which is the percentage of all true positive and true negative classifications in the test set.

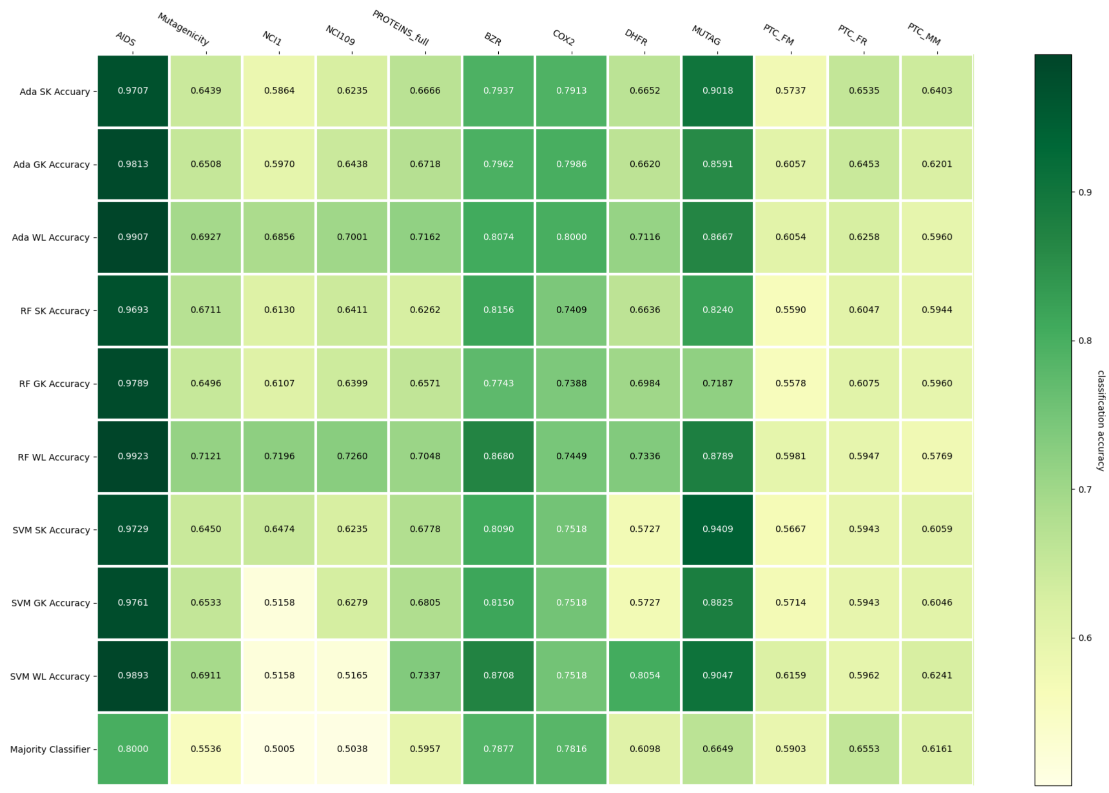
4.3. Analysis

Table 3. The dimensionality of the feature vectors constructed by each kernel used in our experiments.

Name	Symmetry Kernel	Graphlet Kernel	WL Kernel
AIDS	7	29	28961
Mutagenicity	7	29	60896
NCI1	7	29	71738
NCI109	7	29	72766
PROTEINS_full	7	29	94873
BZR	7	29	6774
COX2	7	29	5316
DHFR	7	29	8669
MUTAG	7	29	1982
PTC_FM	7	29	5786
PTC_FR	7	29	5979
PTC_MM	7	29	5495

In this subsection, we will compare the CAs of all resulting models. All results are shown as a heatmap in Figure 2. The CA of each method on each dataset is shown as a colored cell according to the intensity scale.

All models, using all three kernels, performed well on the AIDS dataset. Their accuracy was close to 100%. On the other hand, all models performed poorly on the three PTC datasets and COX2. Their CA was close to the majority classifier. Only AdaBoost outperformed the



	AIDS	Mutagenicity	NCI1	NCI109	PROTEINS_full	BZR	COX2	DHFR	MUTAG	PTC_FM	PTC_FR	PTC_MM
Ada SK Accuary	0.9707	0.6439	0.5864	0.6235	0.6666	0.7937	0.7913	0.6652	0.9018	0.5737	0.6535	0.6403
Ada GK Accuracy	0.9813	0.6508	0.5970	0.6438	0.6718	0.7962	0.7986	0.6620	0.8591	0.6057	0.6453	0.6201
Ada WL Accuracy	0.9907	0.6927	0.6856	0.7001	0.7162	0.8074	0.8000	0.7116	0.8667	0.6054	0.6258	0.5960
RF SK Accuracy	0.9693	0.6711	0.6130	0.6411	0.6262	0.8156	0.7409	0.6636	0.8240	0.5590	0.6047	0.5944
RF GK Accuracy	0.9789	0.6496	0.6107	0.6399	0.6571	0.7743	0.7388	0.6984	0.7187	0.5578	0.6075	0.5960
RF WL Accuracy	0.9923	0.7121	0.7196	0.7260	0.7048	0.8680	0.7449	0.7336	0.8789	0.5981	0.5947	0.5769
SVM SK Accuracy	0.9729	0.6450	0.6474	0.6235	0.6778	0.8090	0.7518	0.5727	0.9409	0.5667	0.5943	0.6059
SVM GK Accuracy	0.9761	0.6533	0.5158	0.6279	0.6805	0.8150	0.7518	0.5727	0.8825	0.5714	0.5943	0.6046
SVM WL Accuracy	0.9893	0.6911	0.5158	0.5165	0.7337	0.8708	0.7518	0.8054	0.9047	0.6159	0.5962	0.6241
Majority Classifier	0.8000	0.5536	0.5005	0.5038	0.5957	0.7877	0.7816	0.6098	0.6649	0.5903	0.6553	0.6161

Fig. 2. Classification accuracies achieved by our three classifiers on each graph kernel we tested.

MC on COX2, creating comparable models using all kernels. On Mutagenicity, NCI1, NCI109, PROTEINS_full, and DHFR SK reached similar results to GK but performed worse than the WL kernel. Similarly, for BZR, except here, SK performed better, sometimes reaching comparable performances to the WL kernel. SK performed best on the MUTAG dataset, outperforming GK and the WL kernel.

Regarding feature vector dimensionality, SK constructed feature vectors far smaller than the existing approaches, as seen in Table 3. SK and GK construct constant-size feature vectors as input graph size increases. However, The WL kernel constructs larger feature vectors as the size of input graphs increases. One notable observation regarding dimensionality is that SK performs comparatively to GK and the WL kernel despite constructing far smaller feature vectors.

5. Conclusion

In this paper, we proposed a novel graph kernel based on graph automorphisms (symmetries) and tested it on 12 graph datasets using three well-known machine-learning algorithms. We compared their classification accuracy using our novel graph kernel SK to the accuracy reached using GK, the WL kernel, and the majority classifier.

The main benefit of our kernel is that it reaches a performance comparable to GK on ten of the twelve datasets while constructing feature vectors of smaller dimensionality, as shown in Table 3. In some cases, it is comparable, and in two cases, it is better than the state-of-the-art WL kernel, which constructs vectors with much higher dimensionality. The ratios between kernel performances repeat themselves regardless of the machine learning model used to evaluate them.

The kernel we presented is based on vertex symmetries, but there are two other types of symmetries we have not explored. We could define a graph kernel based on edge symmetries in the domain of undirected graphs and arc symmetries in the domain of directed graphs. Once we have those, we can compare their performance against SK, GK, the WL kernel, and graph neural networks. We could use more shallow machine learning models and deep ones to make the comparison more comprehensive.

Another direction worth exploring would be the extension of GK with the Jesse Algorithm [15]. Currently, GK is limited to graphlets with $k \in \{3, 4, 5\}$ vertices. Jesse would allow us to go beyond this limit.

References

1. Kriege, N.M., Johansson, F.D., Morris C.: A survey on graph kernels. Applied Network Science 5(1), 1-42 (2020)
2. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 (2016)
3. Shervashidze, N., et al. "Weisfeiler-Lehman graph kernels." Journal of Machine Learning Research 12(9) (2011)
4. Čibej, U., Mihelič, J.: Graph automorphisms for compression. Open Computer Science 11(1), 51-59 (2021), https://doi.org/10.1515/comp-2020-0186
5. Vishwanathan, S., Vichy, N., et al.: Graph kernels Journal of Machine Learning Research 11, 1201-1242 (2010)
6. Borgwardt, K.M., Kriegel, H.-P.: Shortest-path kernels on graphs. In: Fifth IEEE international conference on data mining (ICDM'05), IEEE (2005)
7. Ramon, J., Gärtner, T.: Expressivity versus efficiency of graph kernels. In: Proceedings of the first international workshop on mining graphs, trees, and sequences (2003)
8. Horváth, T., Gärtner, T., Wrobel, S.: Cyclic pattern kernels for predictive graph mining. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (2004)
9. Shervashidze, N., et al.: Efficient graphlet kernels for large graph comparison. Artificial intelligence and statistics. PMLR (2009)
10. Hočevar, T., Demšar, J.: A combinatorial approach to graphlet counting. Bioinformatics 30(4), 559-565 (2014)
11. Huang, S., et al.: Applications of support vector machine (SVM) learning in cancer genomics. Cancer genomics & proteomics 15(1), 41-51 (2018)
12. Hastie, T., et al.: Multi-class adaboost. Statistics and its Interface 2(3), 349-360, (2009)
13. Ho, T.K.: Random decision forests. In: Proceedings of 3rd international conference on document analysis and recognition. Vol. 1, IEEE (1995)
14. Morris, C., et al.: Tudataset: A collection of benchmark datasets for learning with graphs. arXiv preprint arXiv:2007.08663 (2020)
15. Melckenbeeck, I., et al.: Efficiently counting all orbits of graphlets of any order in a graph using autogenerated equations. Bioinformatics 34(8), 1372-1380 (2018)
16. Martino, A., Rizzi, A.: (Hyper) graph kernels over simplicial complexes. Entropy 22(10), 1155 (2020)
17. Hou, Z., et al.: Graphmae: Self-supervised masked graph autoencoders. In: Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (2022)
18. Nafiiev, A., Rodionov, A.: Malware Detection System Based on Static and Dynamic Analysis Using Machine Learning. Theoretical and Applied Cybersecurity 5(2) (2023)