Graph Product Representations

Maximilian Seeliger[1] (✉), Fabian Jogl[1], and Thomas Gärtner[1]

Technische Universität Wien, Vienna 1040, Austria
{maximilian.seeliger,fabian.jogl,thomas.gaertner}@tuwien.ac.at

Abstract. We propose a novel algorithm for generating expressive graph and node representations utilizing graph products. These representations, based on simple substructure counts in product graphs, aim to encode structural information that is often missed by standard message passing graph neural networks (MPNNs).

MPNNs allow to learn vector representations of graphs, that are used in critical domains like drug discovery, social network analysis, protein folding and transportation networks. Their expressiveness is limited by the Weisfeiler-Leman (WL) graph isomorphism test, meaning they are unable to distinguish certain non-isomorphic graphs and fail to recognize important substructures, which restricts their overall capability. Our approach, called *Product Substructure Count (PSC)*, addresses this by utilizing graph products to transform graphs. The transformed graphs encode extensive structural information within simple substructures, such as cycles. By counting these substructures at both the graph and node levels, we can generate embeddings that represent the graph as a whole as well as individual nodes.

We show that PSC representations outperform WL in isomorphism testing and improve the representational capacity of MPNNs across multiple benchmark datasets.

Keywords: Graph Product · Representation Learning · Subgraph Counting.

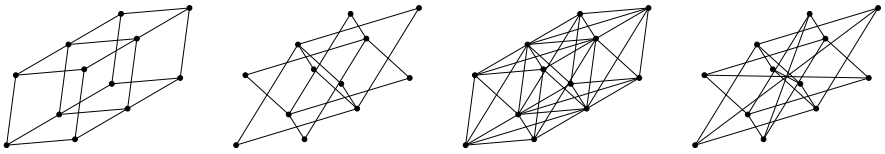

Fig. 1. Graph products between a cycle with four vertices (C_4) and a path of length 2 (P_3). From left to right: Cartesian, direct, strong and modular

1 Introduction

We propose a new method for creating graph and node representations based on graph products. We experimentally analyze the ability of graph products to improve the expressiveness of message passing graph neural networks (MPNNs). MPNNs are a widely used [13,19,22,24] type of graph neural network (GNN) that learn task-specific estimators for graph structured data. They do so, by passing and aggregating messages to and from node neighborhoods iteratively, starting from some initial features given by the input graph. Morris et al. [17] and Xu et al. [24] have shown that MPNNs are limited in their expressive power by the Weisfeiler-Leman (WL) [23] isomorphism test. This means that there exist pairs of non-isomorphic graphs which any MPNN will map to the same embedding regardless of the width or depth of the MPNN. To address these limitations, prior work explores k-WL tests as a theoretical foundation for GNN architectures that pass information over k-tuples of nodes instead of individual nodes [17,12,4]. While these approaches are strictly more expressive, they scale poorly: already 2-WL is computationally expensive and higher-k models quickly become impractical [27]. The k-WL hierarchy is coarse and fails to capture subtle structural differences. Our graph product-based method introduces a more fine-grained perspective, that measures expressiveness between the levels of the k-WL hierarchy.

Graph products combine two graphs to one product graph. There are many different variants of graph products that each define unique construction rules for the product graph. Graph products have previously been used to utilize graph structure in machine learning problems. The random walk kernel [10] applies the direct graph product for efficient similarity calculations based on the number of common walks across graphs and the subgraph matching kernel [14] utilizes the relation between subgraph isomorphism and cliques in the modular product graph for computation. Shi et al. [20] and Einzade et al. [8] learn to decompose product graphs into their factors, making use of their spectral properties. Einzade et al. [7] also provide a domain specific GNN architecture for multidomain data, which can be seen as Cartesian product graphs. Leskovec et al. [15] use the direct graph product to generate synthetic graphs with properties similar to real-world networks.

In this work, graph products are used to encode structural properties of graphs within simple subgraphs. We first transform individual graphs by applying the graph product with the same fixed factor graph to the whole dataset. Then we create feature vectors with the counts of subgraphs in the product graph, either for the whole graphs or for individual vertices (only counting those subgraphs that the vertex participates in). This approach, called *Product Substructure Count (PSC)*, allows to choose the type of considered subgraphs, the graph product and factor graph in an application specific way, in order to focus on relevant structural aspects for a given task. We find that PSC outperforms the Weisfeiler-Leman test as a graph isomorphism heuristic in distinguishing small non-isomorphic graphs. Additionally, PSC features improve training per-

formance when used as initial node features in MPNNs, particularly in social graph datasets.

Contributions.

- We propose a novel algorithm, called *Product Substructure Count (PSC)*, for extracting structural features from graphs based on graph products, that enhance the expressivity of MPNNs (Section 3).
- We experimentally evaluate PSC by comparing its ability to distinguish graphs to the Weisfeiler-Leman isomorphism test and utilizing the extracted features together with MPNNs for representation learning (Section 4).

2 Background

A graph G is a tuple $G = (V_G, E_G)$ comprising of a set of vertices V_G and a set of undirected edges $E_G \subseteq \{\{u, v\} \mid \forall u, v \in V_G, u \neq v\}$. We call a graph $H = (V_H, E_H)$ a subgraph of G if $V_H \subseteq V_G$ and $E_H \subseteq E_G$. We say H is a path if V_H can be arranged as a sequence $(v_1, v_2, \ldots, v_n)$, with $n = |V_H|$, such that the edges are exactly $E_H = \{\{v_i, v_{i+1}\} \mid 1 \leq i < n\}$. A cycle is a path where the first and last vertices in the sequence are the same, i.e. $v_1 = v_n$. Further, we say two graphs G and H are isomorphic, denoted $G \cong H$, if there exists a bijection $f : V_G \to V_H$ such that $\forall u, v \in V_G : \{u, v\} \in E_G \Leftrightarrow \{f(u), f(v)\} \in E_H$.

Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be two undirected graphs. We define four standard graph products—*Cartesian*, *direct*, *strong* and *modular*—each constructing a new graph over the vertex set $V_G \times V_H$ with distinct edge sets [11]. Figure 1 provides an example for each definition.

- The **Cartesian Product** $(G \square H)$ has an edge between vertices $(u, u'), (v, v') \in V_{G \square H}$ if and only if $(u = v$ and $\{u', v'\} \in E_H)$ or $(u' = v'$ and $\{u, v\} \in E_G)$.
- The **Direct Product** $(G \times H)$, also called tensor product, has an edge between $(u, u'), (v, v') \in V_{G \times H}$ if and only if $\{u, v\} \in E_G$ and $\{u', v'\} \in E_H$.
- The **Strong Product** $(G \boxtimes H)$ combines the Cartesian and direct products. Vertices $(u, u'), (v, v') \in V_{G \boxtimes H}$ are adjacent if and only if $(u = v$ and $\{u', v'\} \in E_H)$ or $(u' = v'$ and $\{u, v\} \in E_G)$ or $(\{u, v\} \in E_G$ and $\{u', v'\} \in E_H)$.
- The (weak) **Modular Product** $(G \nabla H)$ has an edge between $(u, u'), (v, v') \in V_{G \nabla H}$ if and only if $\big(\{u, v\} \in E_G \Leftrightarrow \{u', v'\} \in E_H\big)$ and $u \neq v$, $u' \neq v'$. Note that this definition of the edge set is equal to the union $E_{G \times H} \cup E_{\overline{G} \times \overline{H}}$ where $\overline{G}$ and $\overline{H}$ are the complement graphs of G and H.

In this work, we consider the setting where one operand of the graph product is fixed. We are given a tuple $(\circ, F)$, where $\circ \in \{\square, \times, \boxtimes, \nabla\}$ is a graph product and F is a graph. We use graph products as a parameterized graph transformation where $\circ_F(G) := G \circ F$. In this case, we call F the *factor graph*. Different

choices of $(\circ, F)$ can transform structures from the original graphs in non-trivial ways. For example, Lemma 1 and Lemma 2 in Appendix B show cases where paths are encoded in the cycle space of the product graph.

Algorithm 1 Graph-Level Product Substructure Count (G-PSC)

function GRAPHPSC(G)
  $P \leftarrow \circ_F(G)$
  $\mathcal{S} \leftarrow$ `FindSubgraphs`(P) ▷ *Return a list of subgraphs*
  `countVector` $\leftarrow$ zero vector of length $\max_{S \in \mathcal{S}} |S|$
  for all $S \in \mathcal{S}$ **do**
    `countVector`$[|S|] \leftarrow$ `countVector`$[|S|] + 1$
  return `countVector`

3 Algorithm

We propose an algorithm called *Product Substructure Count (PSC)*, that extract structural information from product graphs for use as features in learning tasks. It does so by counting specific subgraphs (e.g. cycles, cliques, paths) within the product graph. PSC has three primary degrees of freedom: (1) the choice of which type of subgraph to count, (2) the type of graph product $\circ$ to apply and (3) the factor graph F to use. The algorithms can extract features at both the graph level (count subgraphs globally) and the node level (count subgraphs containing the node).

Graph-Level Product Substructure Count (G-PSC). The G-PSC algorithm (see Algorithm 1) creates a vector representation of a graph by counting the number of substructures of different sizes found in its product graph. The runtime of G-PSC is dominated by three main steps:

1. **Product Graph Construction.** Building the product graph $P = \circ_F(G)$ has complexity $\mathcal{O}(|V_P|^2)$, since all node pairs need to be checked for adjacency. The product has size $|V_P| = |V_G| \cdot |V_F|$, but in practice, $|V_F|$ is small and fixed, so $|V_P| = \mathcal{O}(|V_G|)$.
2. **Subgraph Detection.** Finding all subgraphs of interest is the most expensive step. We denote the time complexity as a function $g(P)$ that depends on the type of subgraph.
3. **Counting Step.** Counting substructures by size and populating the representation vector takes $\mathcal{O}(|\mathcal{S}|)$ time. However, since we find the set $\mathcal{S}$ in step 2, it cannot be bigger than $\mathcal{O}(g(P))$.

The total time complexity of G-PSC is $\mathcal{O}\left(|V_P|^2 + g(P)\right)$.

Node-Level Product Substructure Count (N-PSC). The N-PSC algorithm (see Algorithm 2) generates a feature vector for each node in the original graph by counting how many substructures (of each size) its corresponding nodes in the product graph participate in. The runtime analysis of N-PSC uses the same first two steps as G-PSC and continues as follows:

3. **Vector Initialization.** Initializing a vector of length k (the maximum subgraph size) for each node in G takes $\mathcal{O}(k \cdot |V_G|)$
4. **Counting Participation.** Each subgraph contributes to the count vectors of all nodes it corresponds to in G. Counting these participations takes $\mathcal{O}(k \cdot |\mathcal{S}|)$

The total time complexity of N-PSC is $\mathcal{O}\left(|V_P|^2 + g(P) + k \cdot |V_G| + k \cdot |\mathcal{S}|\right)$.

Algorithm 2 Node-Level Product Substructure Count (N-PSC)

```
function NodePSC(G)
    P ← ∘_F(G)
    S ← FindSubgraphs(P)                          ▷ Return a list of subgraphs
    nodeVectors ← empty dictionary
    for all u ∈ V(G) do
        nodeVectors[u] ← zero vector of length max_{S∈S} |S|
    for all S ∈ S do
        for all (u, f) ∈ S do          ▷ note, V_P = V_G × V_F and S is subgraph of P
            nodeVectors[u][|S|] ← nodeVectors[u][|S|] + 1
    return nodeVectors
```

4 Experiments

To evaluate the effectiveness of the Product Substructure Count (PSC) framework, we conduct two sets of experiments: one focusing on the ability of PSC to distinguish non-isomorphic graphs through invariant structural embeddings and another assessing its utility as a feature extractor for downstream graph learning tasks. The first set highlights PSC's potential in the context of graph comparison and isomorphism testing, while the second investigates its capacity to enhance message passing neural networks (MPNNs) by providing structurally rich node-level features. Together, these experiments aim to validate the practical applicability of the PSC approach.

4.1 Distinguishing Graphs

The graph isomorphism problem—determining whether two graphs are structurally identical—remains a computationally challenging task. Although a quasi-polynomial time algorithm exists [1], the exact complexity status of the problem is unresolved. It defines the complexity class GI, which is believed to lie

strictly between P and NP-complete. Algorithms in this complexity class commonly require too much time for practical applications. In practice, the graph isomorphism problem is solved using heuristics based on graph invariants, such as degree sequences or cycle counts, to achieve fast and approximately correct results.

To be useful in this context, the PSC algorithm must produce graph embeddings invariant under automorphisms (otherwise the same graph could result in two different embeddings). We investigate G-PSC by calculating the features for each of the 995 non-isomorphic, unlabeled, connected graphs with up to seven nodes and compare the resulting vectors for collisions—cases where non-isomorphic graphs are mapped to identical representations.

Baselines. We consider two baselines: (B1) counting the chosen subgraphs on the original graphs without transformation and (B2) the WL algorithm, a strong method for graph comparison.

PSC Setup. We choose chordless cycles (i.e. only consecutive vertices in the cycle are adjacent) as the subgraphs to count, resulting in a complexity of $g(P) = \mathcal{O}((|V_P| + |E_P|) \cdot c)$, where c is the number of chordless cycles [6,21]. We run the algorithm for all four considered graph products—Cartesian, direct, strong and modular—and apply factor graphs of different graph families and sizes: complete graphs (K_n; n vertices and they are pairwise adjacent), path graphs (P_n; a path of length $n-1$) and star graphs (S_n; a center node with n arms).

Results. As shown in Table 1, PSC consistently outperforms B1, with some configurations even surpassing B2. The modular product achieves perfect distinction, while Cartesian and strong products with complete graphs underperform due to their excessive number of chords. With a complete factor graph, the Cartesian and strong products edge construction rule connects all occurrences of the same vertex from the original graph (i.e. the vertices $(v, f_1), (v, f_2) \in V_P$ are always connected). This introduces chords into cycles that would otherwise be part of the representation of the graph. We hypothesize, that a high performing combination of graph product and factor graph finds a trade-off regarding the sparsity of the product graph.

Table 1. Number of pairs of non-isomorphic graphs with the same embedding per method. For comparison, baseline B1 results in 6989 collisions and B2 in 20 collisions.

	Graph Products				
F	$\square$	$\times$	$\boxtimes$	∇	
K_3	1245	0	3952	0	Compl.
K_5	1244	0	3952	0	
K_7	1244	0	3952	0	
P_3	1	332	11	0	Path
P_5	0	12	0	0	
P_7	0	5	0	0	
S_3	0	350	5	0	Star
S_5	0	362	5	0	
S_7	0	327	5	0	

To qualitatively evaluate the effect of graph products on the embeddings, we reduce their dimensionality and visualize them in 2D space. We use principal

component analysis (PCA), a linear transformation onto a new orthonormal basis, which captures the maximum variance. We observe that the graph product transformation alters the hierarchical structure of the embeddings compared to the original graphs, potentially increasing the degrees of freedom in the representation. Notably, the transformation preserves correlations between structural properties, such as the edge count, and the position of graphs in the embedding space. (Figure 2 and more detailed in Appendix A).

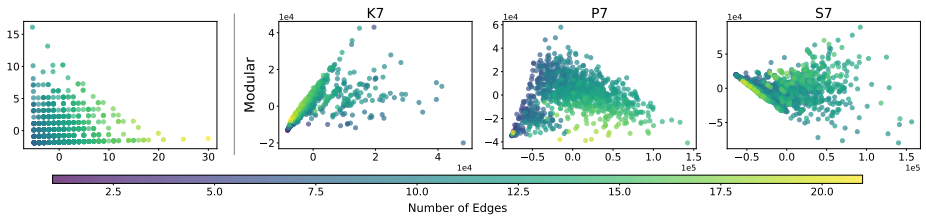

Fig. 2. PCA dimensionality reduction of the G-PSC embeddings for chordless cycles on the set of all connected graphs with up to seven vertices: from left to right: without transformation (B1), modular product with K_7, with P_7 and with S_7.

4.2 Learning on PSC Features

All embeddings produced by MPNNs build on the initial node representation given by node features. Thus, graph learning tasks are significantly more difficult when none or only few features are available. We utilize the N-PSC framework to compute initial features and apply them in conjunction with MPNNs in a classification setup. We focus on fundamental cycle bases as the subgraphs of interest with PSC. These bases span the entire cycle space, allowing any cycle present in the graph to be reconstructed by computing the symmetric difference of a subset of the basis cycles. Further, they can be efficiently computed even on large graphs, leading to a complexity of $g(P) = \mathcal{O}(|E_P| \cdot \log |V_P|)$ (cf. Appendix B).

Baseline. In addition to the node features provided by the dataset, we use one-hot encodings of the node degree and concatenate them for the baseline `deg`. We use a 3-layer GIN [24] with ReLU activations, batch norm and jumping knowledge [25]. Models are trained for 200 epochs with an initial learning rate of $\eta = 0.001$ which is halved every 50 epochs. Hidden channels ($\in \{64, 128\}$) and batch size ($\in \{32, 256\}$) are tuned and results are averaged over 10-fold cross-validation.

PSC Features. To assess the capability of PSC we evaluate two configurations, that concatenate additional N-PSC features to the initial features from the base-

line. The first configuration ($\square K_3$) uses the Cartesian product and K_3 as a fixed factor and the second (∇P_3) uses the modular product and P_3 as a fixed factor.

Setup. We evaluate our approaches on social (IMDB [26], Reddit [26]), synthetic (SYNTHETIC [9], SYNTHIE [16]) and biological (MUTAG [5], ENZYMES [3]) datasets (see Appendix C for more details). Following Xu et al. [24], we report training accuracies as a proxy measure for the expressiveness of the models. A more expressive model should be able to distinguish between graphs from the training dataset better and therefore be able to fit closer to the training distribution. The implementation is publicly available on GitHub[1].

Results. Table 2 report the models' training accuracies on the respective datasets. We see that our approach consistently outperforms or equalizes the baseline on every dataset in its ability to fit the data. In particular, social graph datasets benefit the most from the additional features. We hypothesize, that due to the clustered nature of social graphs, they benefit from additional information about global structural properties, that are inherently hard to capture with message passing.

dataset	deg	$\square K_3$	∇P_3
IMDB-BINARY	0.891 ± 0.005	0.901 ± 0.003	$\mathbf{0.926} \pm 0.005$
IMDB-MULTI	0.614 ± 0.009	0.628 ± 0.005	$\mathbf{0.640} \pm 0.005$
REDDIT-BINARY	0.986 ± 0.002	$\mathbf{0.996} \pm 0.001$	0.995 ± 0.002
SYNTHETIC	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
SYNTHETIC (no attr)	0.580 ± 0.013	0.584 ± 0.014	$\mathbf{0.593} \pm 0.022$
SYNTHIE	1.000 ± 0.000	1.000 ± 0.000	1.000 ± 0.000
MUTAG	0.990 ± 0.006	$\mathbf{1.000} \pm 0.000$	0.995 ± 0.003
ENZYMES	0.984 ± 0.004	0.984 ± 0.005	$\mathbf{0.996} \pm 0.001$

Table 2. Following Xu et al. [24], we report mean and standard deviation of the training accuracy as a proxy measure for expressivity for all datasets over the baseline and the two N-PSC configurations. Green rows indicate datasets, on which our proposed approach outperforms the baseline and yellow rows mark datasets with equal performance across all approaches.

5 Discussion and Conclusion

Summary. We demonstrated the utility of our approach through both theoretical analysis and extensive experiments. PSC, particularly in its form using basis cycles, effectively increased the expressiveness of graph representations by capturing substructures that are inherently difficult for traditional GNNs based

[1] Link to GitHub repository: https://github.com/max-seeli/graph-gumbo

on message passing to learn. Our proposed graph products not only expand the feature space but also provided meaningful separations between graph instances, as evidenced by our experiments on distinguishing graphs with up to seven nodes. Notably, counting chordless cycles in the modular product allows our method to distinguish all non-isomorphic graphs in the dataset, surpassing even the commonly used WL test. The experimental evaluation of PSC-based features, combined with the Graph Isomorphism Network, consistently improved the training accuracy of classification tasks across multiple benchmark datasets. Particularly in social graph datasets, the addition of node-level product substructure counts led to higher accuracies, highlighting the effectiveness of our method to model data from this domain.

Future Work. PSC introduces several design choices such as the graph product, factor graph, and substructure type. As these parameters strongly depend on the task at hand we leave their choice and the effect on the trade-off between representational power and computational cost open for future work. It is known that certain substructures can be computationally intensive to count in large product graphs. In future work, we plan to investigate approximation and sampling techniques to reduce this computational bottleneck.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Babai, L.: Graph isomorphism in quasipolynomial time. In: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing. pp. 684–697 (2016)
2. Bender, M.A., Farach-Colton, M.: The level ancestor problem simplified. Theoretical Computer Science **321**(1), 5–12 (2004)
3. Borgwardt, K.M., Ong, C.S., Schönauer, S., Vishwanathan, S., Smola, A.J., Kriegel, H.P.: Protein function prediction via graph kernels. Bioinformatics **21**(suppl_1), i47–i56 (2005)
4. Chen, Z., Chen, L., Villar, S., Bruna, J.: Can graph neural networks count substructures? Advances in neural information processing systems **33**, 10383–10395 (2020)
5. Debnath, A.K., Lopez de Compadre, R.L., Debnath, G., Shusterman, A.J., Hansch, C.: Structure-activity relationship of mutagenic aromatic and heteroaromatic nitro compounds. correlation with molecular orbital energies and hydrophobicity. Journal of medicinal chemistry **34**(2), 786–797 (1991)
6. Dias, E.S., Castonguay, D., Longo, H., Jradi, W.A.R.: Efficient enumeration of chordless cycles. arXiv preprint arXiv:1309.1051 (2013)
7. Einizade, A., Malliaros, F., Giraldo, J.H.: Continuous product graph neural networks. Advances in Neural Information Processing Systems **37**, 90226–90252 (2024)
8. Einizade, A., Sardouie, S.H.: Learning product graphs from spectral templates. IEEE Transactions on Signal and Information Processing over Networks **9**, 357–372 (2023)

9. Feragen, A., Kasenburg, N., Petersen, J., de Bruijne, M., Borgwardt, K.: Scalable kernels for graphs with continuous attributes. Advances in neural information processing systems **26** (2013)
10. Gärtner, T., Flach, P., Wrobel, S.: On graph kernels: Hardness results and efficient alternatives. In: Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003. Proceedings. pp. 129–143. Springer (2003)
11. Hammack, R.H., Imrich, W., Klavžar, S.: Handbook of product graphs, vol. 2. CRC press Boca Raton (2011)
12. He, J., Cheng, M.: Orthogonal bases for equivariant graph learning with provable k-wl expressive power. Journal of Machine Learning Research **26**(29), 1–35 (2025)
13. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings (2017)
14. Kriege, N., Mutzel, P.: Subgraph matching kernels for attributed graphs. In: Proceedings of the 29th International Coference on International Conference on Machine Learning. p. 291–298. ICML'12, Omnipress, Madison, WI, USA (2012)
15. Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., Ghahramani, Z.: Kronecker graphs: an approach to modeling networks. Journal of Machine Learning Research **11**(2) (2010)
16. Morris, C., Kriege, N.M., Kersting, K., Mutzel, P.: Faster kernels for graphs with continuous attributes via hashing. In: 2016 IEEE 16th International Conference on Data Mining (ICDM). pp. 1095–1100. IEEE (2016)
17. Morris, C., Ritzert, M., Fey, M., Hamilton, W.L., Lenssen, J.E., Rattan, G., Grohe, M.: Weisfeiler and leman go neural: Higher-order graph neural networks. Proceedings of the AAAI Conference on Artificial Intelligence **33**(01), 4602–4609 (Jul 2019). https://doi.org/10.1609/aaai.v33i01.33014602, https://ojs.aaai.org/index.php/AAAI/article/view/4384
18. Paton, K.: An algorithm for finding a fundamental set of cycles of a graph. Communications of the ACM **12**(9), 514–518 (1969)
19. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE transactions on neural networks **20**(1), 61–80 (2008)
20. Shi, C., Mishne, G.: Learning cartesian product graphs with laplacian constraints. In: International Conference on Artificial Intelligence and Statistics. pp. 2521–2529. PMLR (2024)
21. Uno, T., Satoh, H.: An efficient algorithm for enumerating chordless cycles and chordless paths. In: International Conference on Discovery Science. pp. 313–324. Springer (2014)
22. Veličković, P., Cucurull, G., Casanova, A., Romero, A., Lio, P., Bengio, Y.: Graph attention networks. arXiv preprint arXiv:1710.10903 (2017)
23. Weisfeiler, B., Leman, A.: The reduction of a graph to canonical form and the algebra which appears therein. nti, Series **2**(9), 12–16 (1968)
24. Xu, K., Hu, W., Leskovec, J., Jegelka, S.: How powerful are graph neural networks? In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=ryGs6iA5Km
25. Xu, K., Li, C., Tian, Y., Sonobe, T., Kawarabayashi, K.i., Jegelka, S.: Representation learning on graphs with jumping knowledge networks. In: International conference on machine learning. pp. 5453–5462. PMLR (2018)
26. Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. pp. 1365–1374 (2015)

27. Zhao, L., Shah, N., Akoglu, L.: A practical, progressively-expressive gnn. Advances in Neural Information Processing Systems **35**, 34106–34120 (2022)

A Additional Figures

Figure 3 shows the PCA dimensionality reduction of embeddings created from counting chordless cycles on the set of all connected graphs with up to seven vertices without transforming the graphs. In Figure 4 we can see the a dimensionality reduction of the embeddings when using G-PSC with various different configurations.

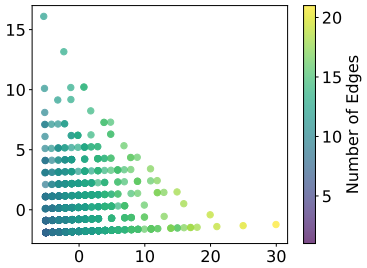

Fig. 3. PCA dimensionality reduction of the G-PSC embeddings for chordless cycles on the set of all connected graphs with up to seven vertices, without transformation.

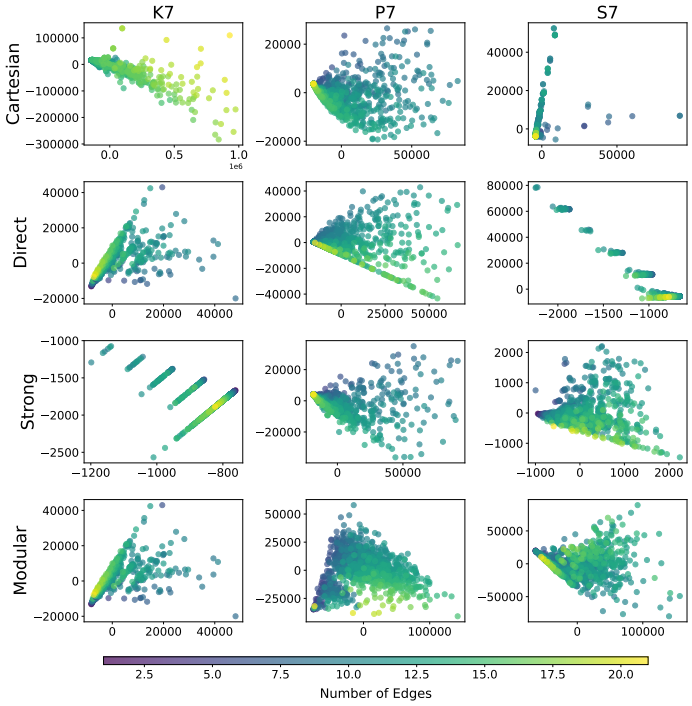

Fig. 4. PCA dimensionality reduction of the G-PSC embeddings for chordless cycles with the respective graph product and factor graph combination on the set of all connected graphs with up to seven vertices.

B Fundamental Cycle Basis

The motivation for studying cycle basis stems from their ability to provide a polynomial characterization of the potentially exponential number of all cycles in a graph. This offers insights into the graph's cyclic structure while maintaining a manageable upper bound on the number of subgraphs to consider, making their practical use feasible. Moreover, in product graphs, the cyclic structure captures additional information (e.g. about paths) depending on the specific type of graph product being considered.

Lemma 1. *Given a graph G and a factor graph F with $|E_F| \geq 1$. For every path P in graph G there is a cycle of length $2|P| + 2$ in the Cartesian product graph $G \square F$*

Proof. Given $|E_F| \geq 1$, we take any edge $\{f_1, f_2\} \in E_F$ and induce the subgraph S of $G \square F$ with the nodes $\{(v, f_1) \mid v \in V_G\} \cup \{(v, f_2) \mid v \in V_G\}$. By construction of the Cartesian product, S consists of two isomorphic copies of G, where each corresponding vertex is connected. Therefore, any path $P = (v_1, v_2, \ldots, v_n)$ in G has a corresponding cycle

$$C = ((v_1, f_1), (v_2, f_1), \ldots, (v_n, f_1), (v_n, f_2), (v_{n-1}, f_2), \ldots, (v_1, f_2), (v_1, f_1)).$$

This cycle walks along the path P in both isomorphic copies for a total length of $2|P|$ and takes two additional edges to move across them at the endpoints. □

Lemma 2. *Given a graph G and a factor graph F. Every edge $\{v_1, v_2\} \in E_G$ entails a copy of all even sized cycles of F in the direct product graph $G \times F$.*

Proof. By construction of the direct product, $\forall f_1, f_2 \in V_F : \{f_1, f_2\} \in E_F \rightarrow \{(v_1, f_1), (v_2, f_2)\} \in E_{G \times F}$. Therefore, any given cycle $C = (c_1, c_2, \ldots, c_n, c_1)$ in F with an even cardinality n transforms to the cycle

$$C' = ((v_1, c_1), (v_2, c_2), (v_1, c_3), \ldots, (v_2, c_n), (v_1, c_1))$$

in the product graph, where v_1 and v_2 are alternating. □

There are many similar lemmas underscoring the structure-capturing effect, the cycle space has in product graphs. Both Lemma 1 and 2 also apply to the strong product and the latter even applies to the modular product, since the edge sets of those products are supersets.

Multiple efficient algorithms have been found for the various types of cycle basis. In the following, we present one for fundamental cycle basis developed by [18].

The computational complexity of the Algorithm 3 is $\mathcal{O}(m \log n)$, where m is the number of edges and n is the number of vertices in the graph G. The critical operation influencing this complexity is the identification of fundamental cycles, which requires finding the shortest path between two vertices in the spanning tree T. By utilizing efficient tree algorithms—specifically, the *Lowest Common*

Algorithm 3 Fundamental Cycle Basis [18]

```
function FundamentalCycleBasis(G)
    T ← ({v}, ∅)                          ▷ Take any vertex v as root of spanning tree
    X ← V_G
    while V_T ∩ X ≠ ∅ do
        Select any vertex z from V_T ∩ X
        for all {z, w} ∈ E_G do
            if w ∈ V(T) then
                Fundamental cycle with edge {z, w} and the shortest path from z to
                w in T
            else
                Add edge {z, w} to spanning tree T
            Remove {z, w} from E_G
        X ← X \ {z}
```

Ancestor (LCA) method with binary lifting [2]—we can continuously maintain a datastructure of T during its generation in a total of $\mathcal{O}(n \log n)$ cumulative time. This datastructure allows each shortest path query to be answered in $\mathcal{O}(\log n)$ time. As each edge is considered exactly once before being removed, the total time spent on processing edges and identifying cycles sums up to $\mathcal{O}(m \log n)$, resulting in the overall time complexity.

C Datasets

In the experimental evaluation, we utilize a diverse set of datasets to assess our approach. The datasets span multiple domains, including synthetic, biological, chemical and social graph benchmarks. Below, we provide a detailed description of each category, Table 3 gives an overview over their central properties and in Figure 5 example graphs from each dataset class are shown.

Table 3. Characteristics of the datasets.

Dataset	Graphs	Classes	Avg. Nodes	Avg. Edges
IMDB-BINARY	1000	2	19.77	96.53
IMDB-MULTI	1500	3	13.00	65.94
REDDIT-BINARY	2000	2	429.63	497.75
SYNTHETIC	300	2	100.00	196.00
SYNTHIE	400	4	95.00	172.93
MUTAG	188	2	17.93	19.79
ENZYMES	600	6	32.63	62.14

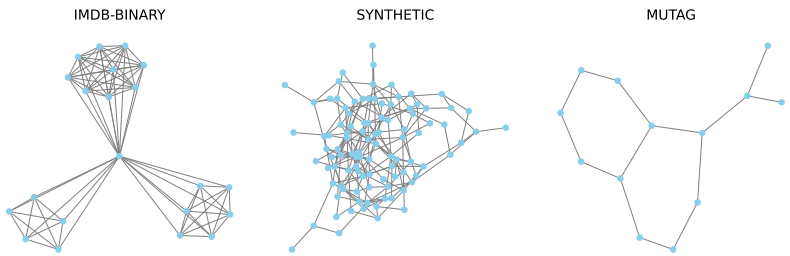

Fig. 5. Example graphs (left to right): *IMDB-BINARY*, *SYNTHETIC* and *MUTAG*

Social Graphs. We consider datasets representing social networks, specifically *IMDB* and *REDDIT* [26]. These datasets are chosen to evaluate the model's proficiency in learning from real-world social community structures.

- *IMDB-BINARY*: Contains 1,000 graphs, each representing a set of actors appearing together in films, used for binary classification of movie genres.
- *IMDB-MULTI*: Consists of 1,500 graphs, similar to *IMDB-BINARY*, but used for multi-class classification involving multiple movie genres.
- *REDDIT-BINARY*: Contains 2,000 graphs representing Reddit discussion threads, used for binary classification of different discussion types.

Synthetic Datasets. We incorporate two synthetic datasets, *SYNTHETIC* and *SYNTHIE*, which are relevant in evaluating the model's ability to identify and represent abstract graph structures under controlled generation rules.

- *SYNTHETIC* [9] is a dataset of 300 synthetic graphs derived from a random base graph with 100 nodes and 196 edges, featuring normally distributed node attributes. Each graph in class A is formed by reconnecting 5 edges and permuting 10 node attributes and class B is generated with the same process but switched numbers.
- *SYNTHETIC (no attr)* is a variant of *SYNTHETIC*, where the node attributes are ignored and only the rewiring of 5 and 10 edges define the respective classes A and B.
- The *SYNTHIE* dataset [16] consists of 400 graphs, categorized into four classes, each featuring 15 real-valued node attributes. These were derived from two initial Erdős-Rényi graphs by altering 25% of the edges to generate seed sets, from which connected graphs were formed by randomly adding edges between sampled seeds. Node attributes from two distinct classes were then assigned based on the origin of the seed, resulting in the formation of the final graph classes.

Biology and Chemistry. We employ datasets from biology and chemistry, specifically *MUTAG* and *ENZYMES*.

- *MUTAG* [5]: Contains 188 graphs representing chemical compounds, represented as graphs, where nodes correspond to atoms and edges denote bonds. The task is to predict the mutagenic effects of aromatic and heteroaromatic compounds.
- *ENZYMES* [3]: Consists of 600 protein tertiary structures obtained from the BRENDA enzyme database. The task is to connect the protein with the appropriate enzyme.