Grammatical Path Network: You want cycles, follow this path.

Jason Piquenot[*†] **Louisa Bouzidi**[*†] **Maxime Bérar**[†] **Pierre Héroux**[†] **Jean-Yves Ramel**[‡]

Romain Raveaux[‡] **Sébastien Adam**[†]

Abstract

In this work, we address the challenge of learning from structured data by proposing the Grammatical Path Network (GPN), a novel Graph Neural Network (GNN) designed to efficiently capture cycles in graph structures. Building on recent advancements in GNN expressiveness and substructure counting, GPN combines methodologies from Graph Substructure Networks (GSN) and a framework that translates Context Free Grammars (CFG) into GNNs. The key innovation lies in using CFG to count cycles of length $l+1$ by precomputing paths of length l at the edge level. Our experiments demonstrate that GPN achieves comparable performance to GSN on datasets requiring cycle information, without the need for explicit cycle precomputation. This approach offers a promising direction for developing efficient and expressive GNNs for structured data analysis.

1 Introduction

To address the need for learning from irregular structured data, a wide variety of Graph Neural Networks (GNNs) have been proposed. The pursuit of progressively more expressive models in terms of distinguishability, heavily relying on the well-known Weisfeiler-Lehman test (WL), has led to the development of innovative yet computationally demanding models [1, 2, 3, 4, 5, 6]. On the other hand, other models have been developed by focusing on substructure counting, [7, 8, 9, 10, 11].

Among them, the Graph Substructure Network (GSN) [8] is a model designed to surpass the limitations of the Message Passing Neural Networks (MPNNs) [12]. The core idea of GSN is to encode substructures like paths, cycles and/or cliques. These substructures are precomputed, added as features to the nodes or the edges and fed to multiple Graph Isomorphism Network (GIN) [1] layers. A significant challenge is the selection of appropriate substructures to precompute for a specific task. As noted by [8], further research could focus on developing adaptive methods to dynamically learn the most relevant substructures during training.

To address this challenge, we propose a new model that can count cycles dynamically, without precomputation. This approach reduces the set of candidate substructures that need to be considered, optimizing the precomputation time.

Building on the explicit formulas for counting edge-level cycles from paths proposed by [13], we introduce a CFG G_l whose language encompasses $(l+1)$-cycles based on inputs of l-paths. Leveraging the methodology of [6], which translates Context-Free Grammar (CFG) into GNNs, we propose the Grammatical Path Network (GPN). GPN is a GNN specifically designed to capture cycles of length $l+1$ at the edge level by precomputing paths of length l. Through experiments on a synthetic dataset, we ensure that GPN is able to count cycles at node level. Then, we evaluate GPN against a version of GSN that uses cycle precomputation and demonstrate similar performance without explicitly precomputing these cycles.

[*]Equal contribution.

[†]LITIS Lab, University of Rouen Normandy, France.

[‡]LIFAT Lab, University of Tours, France.

Preprint. Preliminary work.

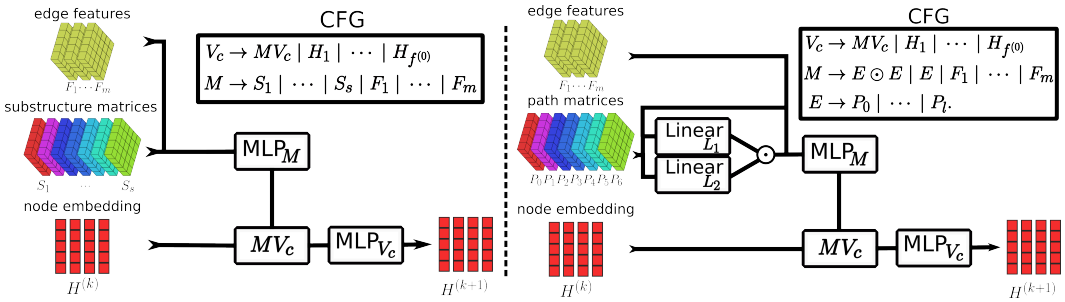

Figure 1: On the left, a schematic representation of the GSN kth layer and its CFG: $\mathrm{MLP}_M$is fed with a set of precomputed substructure matrices and the edge features, producing an arbitrary number of matrices. The node embedding $H^{(k)}$ is then updated by multiplying it with these matrices and passing the result through MLP_{V_c}, yielding the updated node embedding $H^{(k+1)}$. On the right, a schematic representation of a GPN kth layer and its CFG: $\mathrm{MLP}_{\mathrm{M}}$is fed with the results of the batched Hadamard product between two sets of learnable linear combinations of path matrices, the path matrices themselves, and the edge features, producing an arbitrary number of matrices. The node embedding $H^{(k)}$ is then updated by multiplying it with these matrices and passing the result through $\mathrm{MLP}_{\mathrm{V_c}}$, yielding the updated node embedding $H^{(k+1)}$.

This paper is structured as follows. In Section 2, we introduce GSN and derive its corresponding CFG. Section 3 presents our proposed model, GPN. Section 4 details our evaluation of GPN.

2 GSN and its Context Free Grammar

Notations: Let $\mathcal{G}$ be a simple undirected graph with n vertices. A is its adjacency matrix, and I is the identity matrix of the same dimensions as A. For any non-negative integer l, let P_l represent the l-path matrix where $(P_l)_{i,j}$ is the number of l-length paths connecting vertex i to vertex j. Additionally, for $l > 2$, let C_l represent the l-cycle matrix where $(C_l)_{i,j}$ denotes the number of l-cycles connecting vertex i to its adjacent vertex j. Since each vertex can remain stationary in exactly one way, $P_0 = \mathrm{I}$.

As outlined in the introduction, [8] introduces the innovative approach of incorporating precomputed substructure counts within the graph during the aggregation process of GIN layers, leading to the development of the GSN model. At layer k, a set of different substructure counts at the edge level, along with the edge features, is provided as input to a Multi-Layer Perceptron (MLP), denoted as MLP_M, to generate $b^{(k)}$ matrices (see equation (1)). The node embedding matrix $H^{(k)}$, consisting of $f^{(k)}$ column vectors, is then multiplied by the $b^{(k)}$ matrices, resulting in $f^{(k)} \times b^{(k)}$ column vectors. This operation corresponds to the aggregation step in MPNNs. These column vectors are subsequently passed through another MLP, denoted as MLP_{V_c}, which produces the node embedding matrix $H^{(k+1)}$, consisting of $f^{(k+1)}$ column vectors (see equation (2)). The first layer of the model takes the node feature matrix H as input. A schematic representation of a GSN layer is shown in Figure 1.

$$M_1^{(k)}, \cdots, M_{b^{(k)}}^{(k)} = \mathrm{MLP}_M(F_1, \cdots, F_m, S_1, \cdots, S_s) \tag{1}$$

$$H^{(k+1)} = \mathrm{MLP}_{V_c}(M_1^{(k)} H^{(k)}, \cdots, M_{b^{(k)}}^{(k)} H^{(k)}) \tag{2}$$

In the supplementary material of [6] CFGs are derived from existing GNNs to analytic purpose. In this paper, we propose to derive and enhance a CFG from the GSN model. A CFG consists of generative rules that express the substitution of variables into combinations of variables and terminal symbols. A sentence in a CFG is a sequence of terminal symbols produced according to these rules.

In the context of GSN, a set of matrices is generated inside each layer to produce a set of column vectors. Thus, the variables of its corresponding CFG are M for matrices and V_c for column vectors. The MLP in the GSN architecture approximates a pointwise function at both the edge and node levels. However, as noted by [14], it does not change the expressive power of the model. Therefore, the key operation that impacts expressiveness is the aggregation step, where matrix multiplication occurs

between square matrices M and column vectors V_c, producing new column vectors V_c. As a result, the rule $MV_c \to V_c$ is added to the set of rules for generating column vectors.

Lastly, the network's input, comprising the $f^{(0)}$ column vectors of the node embedding matrix H, the s precomputed substructure matrices S_i, and the m edge feature matrices F_i, is incorporated into their respective variables. This leads to the CFG in Figure 1, which corresponds to the GSN architecture.

The selection of substructures S_1 to S_s is not only task-specific but also increases computational cost when more substructures are added. As highlighted by [8], future research could explore adaptive techniques for dynamically selecting or learning the most relevant substructures during training. In response to this challenge, the following section proposes an approach that reduces the set of substructures to be precomputed by leveraging CFG analysis.

3 Grammatical Path Network (GPN)

As shown by [13], path and cycle matrices are connected through $C_{l+1} = A \odot P_l, \text{ for } l > 1$.

Thus, adding an intermediate variable to the CFG in Figure (1) with a rule containing the Hadamard product $\odot$ results into a CFG that can count cycles from path matrices. Consequently, we derive the CFG in Figure 1, denoted as G_l.

Proposition 3.1
G_l can count cycles up to length $l+1$ at edge-level.

Proof. Since $P_1 = A$, we have that $P_1 \odot P_k = C_{k+1}$ for $2 \leqslant k \leqslant l$. □

The application of the framework proposed in [6] to G_l results in the Grammatical Path Network. At layer k, the batched Hadamard product of two sets of learnable linear combinations of path matrices (see equation (3)), along with the path matrices and edge features, is fed into a MLP, denoted as $\mathrm{MLP_M}$ (see equation (4)). This process produces $b^{(k)}$ matrices, representing the rule $E \odot E$ and the terminal symbols P_0 through P_l.

As in GSN, the node embedding matrix $H^{(k)}$ is then multiplied by these $b^{(k)}$ matrices, resulting in $b^{(k)} \times f^{(k)}$ column vectors. These vectors are subsequently passed through another MLP, $\mathrm{MLP_{V_c}}$, to produce $f(k+1)$ column vectors (see equation (2)), which corresponds to the rule MV_c and the terminal symbol H. Figure 1 illustrates a GPN layer. The input of the first layer of GPN is the node feature matrix H. Since GPN is derived from G_l, it retains the ability to count cycles as an inherent property.

$$S_1^{(k)}, \cdots, S_{s^{(k)}}^{(k)} = \mathrm{Linear}_1(P_1, \cdots, P_6) \odot \mathrm{Linear}_2(P_1, \cdots, P_6) \tag{3}$$

$$M_1^{(k)}, \cdots, M_{b^{(k)}}^{(k)} = \mathrm{MLP}_M(F_1, \cdots, F_m, P_1, \cdots, P_6, S_1^{(k)}, \cdots, S_{s^{(k)}}^{(k)}) \tag{4}$$

$$H^{(k+1)} = \mathrm{MLP}_{V_c}(M_1^{(k)} H^{(k)}, \cdots, M_{b^{(k)}}^{(k)} H^{(k)}) \tag{5}$$

Precomputation and time complexity: As mentioned in our introduction, [13] developed the more efficient explicit formulas for computing path matrices up to length 6, as mentioned in [15]. These formulas have the same time complexity as matrix multiplication ($O\left(n^3\right)$), while the worst-case complexity of the algorithm used in [8] is $O\left(n^k\right)$, where n is the number of vertices in the graph and k is the path length. The formulas and an evaluation of the precomputation time of GSN an GPN are provided in appendix A and B of the supplementary material.

Since the Hadamard product and the aggregation step MV_c share the same computational complexity, the overall time complexity of GPN scales quadratically with the number of nodes($O\left(n^2\right)$). An experiment on the time computation of GPN compared with GIN can be found in Appendix B of the supplementary material.

4 Experiments

This section aims to respond to two questions.

Q1: Can GPN count cycles at node level? To address **Q1**, we adopt the experimental setup from [9] on the synthetic dataset introduced by [16]. The task involves node-level regression to predict the number of cycles of length 3 to 6 that a node participates in. We evaluate the performance of the GPN model in comparison to various types of Graph Neural Networks (GNNs), categorized by their ability to count cycles at the node level.

For models unable to count cycles as described in [7], we select GIN [1], an MPNN model. Additionally, we include ID-GNN [17] and NGNN [18] to provide more detailed comparisons. These subgraph GNNs can count cycles of length up to 4 at the node level but are unable to count cycles longer than 4, as noted by [9]. Finally, we compare GPN against PPGN [4], I^2-GNN [9], and G^2N^2 [6], which can count cycles up to length 6.

In all experiments, we employ one layer of GPN with a node dimension of 2 and an edge dimension of 2. The Adam optimizer [19] from PyTorch is used without weight decay or dropout, with an initial learning rate of 0.01, which decreases by a factor of 0.95 with a patience of 25 epochs without amelioration on the validation set. The results are presented in Table 1.

Model	GIN	ID-GNN	NGNN	I^2-GNN	PPGN	G^2N^2	GPN (ours)
triangle	0.3515	0.0006	0.0003	0.0003	0.0003	0.0002	0.0001
4-cycle	0.2742	0.0022	0.0013	0.0016	0.0009	0.0002	0.0001
5-cycle	0.2088	0.0490	0.0402	0.0028	0.0036	0.0018	0.0001
6-cycle	0.1555	0.0495	0.0439	0.0082	0.0071	0.0052	0.0003

Table 1: Normalized MAE results for counting cycles at the node level on a synthetic dataset. Results are taken from [9].

Across all tasks, GPN consistently outperforms GIN. Furthermore, GPN exhibits superior performance compared to more complex models, such as subgraph GNNs, PPGN, and G^2N^2. These results provide a positive answer to **Q1**.

Q2: Can GPN achieve equivalent results to a GSN that uses cycles as inputs? To address **Q2**, we evaluate GPN on the ZINC 12K dataset [20]. We compare GPN with the version of GSN with cycles precomputation, which achieved the best results following the procedure outlined in [21]. The results for GSN are sourced from [8]. To ensure a fair comparison, we configure GPN's hyperparameters to remain under 500K parameters. Node features are one-hot encoded. We employ two layers of GPN with a node dimension of 64 and an edge dimension of 33, resulting in a model with less than 300K parameters. The ADAM optimizer is used, with layer normalization, weight decay at 0.02 but without dropout. The results are presented in Table 2.

Model	parameters	MAE
GSN [8]	$\sim 500K$	0.101 ± 0.010
GPN	$\sim 300K$	0.0972 ± 0.0038

Table 2: Mean MAE over 10 seeds on ZINC 12K. The lower, the better.

GPN achieved results comparable to GSN without the need for precomputing cycle counts. Furthermore, GPN accomplished this using only two layers. This provides a positive answer to **Q2**. We also evaluate GPN on classification tasks, results and discussion can be found in Appendix B.

5 Conclusion

In conclusion, this paper introduces GPN, a novel method for enhancing the expressiveness of GNNs to efficiently capture cycles within graph structures. By precomputing paths instead of explicitly counting cycles, GPN achieves performance comparable to GSN that rely on cycle precomputation. This study highlights CFG as an effective tool for developing more expressive GNNs. Future research could extend this approach to other graph substructures and further refine the balance between expressiveness and computational efficiency.

Acknoledgements

The authors acknowledge the support of the French Agence Nationale de la Recherche (ANR) under grant ANR-21-CE23-0025 (CoDeGNN project).

References

[1] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. 1, 4, 7, 8

[2] Haggai Maron, Heli Ben-Hamu, Nadav Shamir, and Yaron Lipman. Invariant and equivariant graph networks. In *International Conference on Learning Representations*, 2019. 1, 8

[3] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4602–4609, 2019. 1

[4] Haggai Maron, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman. Provably powerful graph networks. *Advances in neural information processing systems*, 32, 2019. 1, 4, 8

[5] Muhammet Balcilar, Guillaume Renton, Pierre Héroux, Benoit Gaüzère, Sébastien Adam, and Paul Honeine. Analyzing the expressive power of graph neural networks in a spectral perspective. In *International Conference on Learning Representations*, 2020. 1

[6] Jason Piquenot, Aldo Moscatelli, Maxime Berar, Pierre Héroux, Romain Raveaux, Jean-Yves Ramel, and Sébastien Adam. G^2n^2 : Weisfeiler and lehman go grammatical. In *The Twelfth International Conference on Learning Representations*, 2024. 1, 2, 3, 4

[7] Zhengdao Chen, Lei Chen, Soledad Villar, and Joan Bruna. Can graph neural networks count substructures? *Advances in neural information processing systems*, 33:10383–10395, 2020. 1, 4

[8] Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1):657–668, 2022. 1, 2, 3, 4, 7, 8

[9] Yinan Huang, Xingang Peng, Jianzhu Ma, and Muhan Zhang. Boosting the cycle counting power of graph neural networks with I^2-GNNs. In *The Eleventh International Conference on Learning Representations*, 2023. 1, 4

[10] Caterina Graziani, Tamara Drucks, Fabian Jogl, Monica Bianchini, Thomas Gärtner, et al. The expressive power of path-based graph neural networks. In *Forty-first International Conference on Machine Learning*. 1

[11] Raffaele Paolino, Sohir Maskey, Pascal Welke, and Gitta Kutyniok. Weisfeiler and leman go loopy: A new hierarchy for graph representational learning. *arXiv preprint arXiv:2403.13749*, 2024. 1

[12] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017. 1

[13] AN Voropaev and SN Perepechko. The number of fixed length cycles in undirected graph explicit formula in case of small lengths. *Discrete and Continuous Models and Applied Computational Science*, (2):6–12, 2012. 1, 3, 7, 8

[14] FlorisF Geerts. On the expressive power of linear algebra on graphs. *Theory of Computing Systems*, Oct 2020. 2

[15] Pierre-Louis Giscard, Nils Kriege, and Richard C Wilson. A general purpose algorithm for counting simple cycles and simple paths of any length. *Algorithmica*, 81:2716–2737, 2019. 3

[16] Lingxiao Zhao, Wei Jin, Leman Akoglu, and Neil Shah. From stars to subgraphs: Uplifting any gnn with local structure awareness. *arXiv preprint arXiv:2110.03753*, 2021. 4

[17] Jiaxuan You, Jonathan M Gomes-Selman, Rex Ying, and Jure Leskovec. Identity-aware graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10737–10745, 2021. 4

[18] Muhan Zhang and Pan Li. Nested graph neural networks. *Advances in Neural Information Processing Systems*, 34:15734–15747, 2021. 4

[19] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[20] John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012. 4

[21] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023. 4

[22] Christopher Morris, Nils M. Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. In *ICML 2020 Workshop on Graph Representation Learning and Beyond (GRL+ 2020)*, 2020. 7

[23] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011. 8

[24] Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *Advances in neural information processing systems*, 32, 2019. 8

[25] Muhan Zhang, Zhicheng Cui, Marion Neumann, and Yixin Chen. An end-to-end deep learning architecture for graph classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018. 8

[26] Pim de Haan, Taco S Cohen, and Max Welling. Natural graph networks. *Advances in Neural Information Processing Systems*, 33:3636–3646, 2020. 8

[27] Soheil Kolouri, Navid Naderializadeh, Gustavo K Rohde, and Heiko Hoffmann. Wasserstein embedding for graph learning. *arXiv preprint arXiv:2006.09430*, 2020. 8

[28] Tianle Cai, Shengjie Luo, Keyulu Xu, Di He, Tie-yan Liu, and Liwei Wang. Graphnorm: A principled approach to accelerating graph neural network training. In *International Conference on Machine Learning*, pages 1204–1215. PMLR, 2021. 8

This document provides additional content to the main paper.

A Explicit formulas for path counting at edge level

The most effective explicit formulas discovered to date for calculating the number of 2 through 6 paths connecting two nodes were proposed by [13]. Let J denote a matrix of the same dimensions as A, where all off-diagonal elements are set to one and all diagonal elements are set to zero. The 2-path formula is

$$P_2 = \mathrm{J} \odot A^2. \tag{6}$$

The 3-path formula is

$$P_3 = \mathrm{J} \odot A^3 - (\mathrm{I} \odot A^2)A - A(\mathrm{I} \odot A^2) + A. \tag{7}$$

The 4-path formula is

$$\begin{aligned} P_4 = {} & \mathrm{J} \odot A^4 - \mathrm{J} \odot (A(\mathrm{I} \odot A^2)A) + 2(\mathrm{J} \odot A^2) \\ & - (\mathrm{I} \odot A^2)(\mathrm{J} \odot A^2) - (\mathrm{J} \odot A^2)(\mathrm{I} \odot A^2) \\ & - A(\mathrm{I} \odot A^3) - (\mathrm{I} \odot A^3)A + 3A \odot A^2. \end{aligned} \tag{8}$$

The 5-path formula is

$$\begin{aligned} P_5 = {} & \mathrm{J} \odot A^5 - (\mathrm{I} \odot A^4)A - A(\mathrm{I} \odot A^4) - (\mathrm{I} \odot A^3)(\mathrm{J} \odot A^2) - (\mathrm{J} \odot A^2)(\mathrm{I} \odot A^3) \\ & - (\mathrm{I} \odot A^2)(\mathrm{J} \odot A^3) - (\mathrm{J} \odot A^3)(\mathrm{I} \odot A^2) - \mathrm{J} \odot (A(\mathrm{I} \odot A^3)A) + 3A \odot A^3 \\ & + 2A(\mathrm{I} \odot A^2)(\mathrm{I} \odot A^2) + 2(\mathrm{I} \odot A^2)(\mathrm{I} \odot A^2)A + 3A \odot A^2 \odot A^2 + (\mathrm{I} \odot A^2)A(\mathrm{I} \odot A^2) \\ & - \mathrm{J} \odot (A(\mathrm{I} \odot A^2)A^2) - \mathrm{J} \odot (A^2(\mathrm{I} \odot A^2)A) + 3\mathrm{J} \odot ((A \odot A^2)A) + 3\mathrm{J} \odot (A(A \odot A^2)) \\ & + (\mathrm{I} \odot (A(\mathrm{I} \odot A^2)A))A + A(\mathrm{I} \odot (A(\mathrm{I} \odot A^2)A)) - 6(\mathrm{I} \odot A^2)A - 6A(\mathrm{I} \odot A^2) \\ & - 4A \odot A^2 + 3\mathrm{J} \odot A^3 + 4A. \end{aligned} \tag{9}$$

The 6-path formula is

$$\begin{aligned} P_6 = {} & \mathrm{J} \odot A^6 - (\mathrm{I} \odot A^5)A - A(\mathrm{I} \odot A^5) - (\mathrm{I} \odot A^2)(\mathrm{J} \odot A^4) - (\mathrm{J} \odot A^4)(\mathrm{I} \odot A^2) \\ & - (\mathrm{I} \odot A^4)(\mathrm{J} \odot A^2) - (\mathrm{J} \odot A^2)(\mathrm{I} \odot A^4) - \mathrm{J} \odot (A(\mathrm{I} \odot A^4)A) + 3A \odot A^4 \\ & - (\mathrm{J} \odot A^3)(\mathrm{I} \odot A^3) - (\mathrm{I} \odot A^3)(\mathrm{J} \odot A^3) - \mathrm{J} \odot (A(\mathrm{I} \odot A^2)A^3) - \mathrm{J} \odot (A^3(\mathrm{I} \odot A^2)A) \\ & - \mathrm{J} \odot (A(\mathrm{I} \odot A^3)A^2) - \mathrm{J} \odot (A^2(\mathrm{I} \odot A^3)A) + 4A(\mathrm{I} \odot A^2)(\mathrm{I} \odot A^3) + 4(\mathrm{I} \odot A^3)(\mathrm{I} \odot A^2)A \\ & + 6A \odot A^2 \odot A^3 + (\mathrm{I} \odot A^2)A(\mathrm{I} \odot A^3) + (\mathrm{I} \odot A^3)A(\mathrm{I} \odot A^2) + 3\mathrm{J} \odot ((A \odot A^3)A) \\ & + 3\mathrm{J} \odot (A(A \odot A^3)) + (\mathrm{I} \odot (A(\mathrm{I} \odot A^3)A))A + A(\mathrm{I} \odot (A(\mathrm{I} \odot A^3)A)) - \mathrm{J} \odot (A^2(\mathrm{I} \odot A^2)A^2) \\ & + 2A(\mathrm{I} \odot A^2)(\mathrm{I} \odot A^2) + 2(\mathrm{I} \odot A^2)(\mathrm{I} \odot A^2)A + \mathrm{J} \odot A^2 \odot A^2 \odot A^2 + (\mathrm{I} \odot A^2)(\mathrm{J} \odot A^2)(\mathrm{I} \odot A^2) \\ & + 3\mathrm{J} \odot ((A \odot A^2)A^2) + 3\mathrm{J} \odot (A^2(A \odot A^2)) + (\mathrm{I} \odot (A(\mathrm{I} \odot A^2)A))(\mathrm{J} \odot A^2) + (\mathrm{J} \odot A^2)(\mathrm{I} \odot (A(\mathrm{I} \odot A^2)A)) \\ & + \mathrm{J} \odot ((\mathrm{I} \odot A^2)A(\mathrm{I} \odot A^2)A) + \mathrm{J} \odot (A(\mathrm{I} \odot A^2)A(\mathrm{I} \odot A^2)) + 2\mathrm{J} \odot (A(\mathrm{I} \odot A^2)(\mathrm{I} \odot A^2)A) \\ & + (\mathrm{I} \odot (A(\mathrm{I} \odot A^2)A^2))A + A(\mathrm{I} \odot (A(\mathrm{I} \odot A^2)A^2)) + (\mathrm{I} \odot (A^2(\mathrm{I} \odot A^2)A))A + A(\mathrm{I} \odot (A^2(\mathrm{I} \odot A^2)A)) \\ & + 3\mathrm{J} \odot ((A \odot A^2 \odot A^2)A) + 3\mathrm{J} \odot (A(A \odot A^2 \odot A^2)) - 12(\mathrm{I} \odot A^2)(A \odot A^2) - 12(A \odot A^2)(\mathrm{I} \odot A^2) \\ & - 4\mathrm{J} \odot A^2 \odot A^2 - 8A \odot (A(A \odot A^2)) - 8A \odot ((A \odot A^2)A) - 3A \odot (A(\mathrm{I} \odot A^2)A) \\ & + 3\mathrm{J} \odot (A(A \odot A^2)A) + \mathrm{J} \odot (A(\mathrm{I} \odot (A(\mathrm{I} \odot A^2)A))A) - 4\mathrm{J} \odot (A(A \odot A^2)) - 4\mathrm{J} \odot ((A \odot A^2)A) \\ & + 4\mathrm{J} \odot A^4 - 5A(\mathrm{I} \odot A^3) - 5(\mathrm{I} \odot A^3)A - 4(\mathrm{I} \odot (A(A \odot A^2)))A - 4A(\mathrm{I} \odot (A(A \odot A^2))) \\ & - 4(\mathrm{I} \odot ((A \odot A^2)A))A - 4A(\mathrm{I} \odot ((A \odot A^2)A)) - 7(\mathrm{I} \odot A^2)(\mathrm{J} \odot A^2) - 7(\mathrm{J} \odot A^2)(\mathrm{I} \odot A^2) \\ & - 10\mathrm{J} \odot (A(\mathrm{I} \odot A^2)A) + 44A \odot A^2 + 12\mathrm{J} \odot A^2. \end{aligned} \tag{10}$$

B Experiments on TUD [22]

We evaluate GPN on the classical TUD benchmark ([22]), using the evaluation protocol of [1]. Results of GNNs and Graph Kernel are taken from [8]. For each of the 5 experiments related to this

Table 3: Results on TUD dataset. The metric is accuracy, the higher, the better.

Dataset	MUTAG	PTC	Proteins	NCI1	IMDB-B
WL kernel [23]	90.4±5.7	59.9±4.3	75.0±3.1	**86.0±1.8**	73.8±3.9
GNTK [24]	90.0±8.5	67.9±6.9	75.6±4.2	84.2±1.5	76.9±3.6
DGCNN [25]	85.8±1.8	58.6±2.5	75.5±0.9	74.4±0.5	70.0±0.9
IGN [2]	83.9±13.0	58.5±6.9	76.6±5.5	74.3±2.7	72.0±5.5
GIN [1]	89.4±5.6	64.6±7.0	76.2±2.8	82.7±1.7	75.1±5.1
PPGNs [4]	90.6±8.7	66.2±6.6	77.2±4.7	83.2±1.1	73.0±5.8
Natural GN [26]	89.4±1.60	66.8±1.79	71.7±1.04	82.7±1.35	74.8±2.01
WEGL [27]	N/A	67.5±7.7	76.5±4.2	N/A	75.4±5.0
GIN+GraphNorm [28]	91.6±6.5	64.9±7.5	77.4±4.9	82.7±1.7	76.0±3.7
GSNs [8]	92.2±7.5	68.2±7.2	76.6±5.0	83.5±2.0	**77.8±3.3**
GPN	**97.3±3.5**	**73.8±3.9**	**80.1±2.8**	84.45±1.1	74.8±4.6

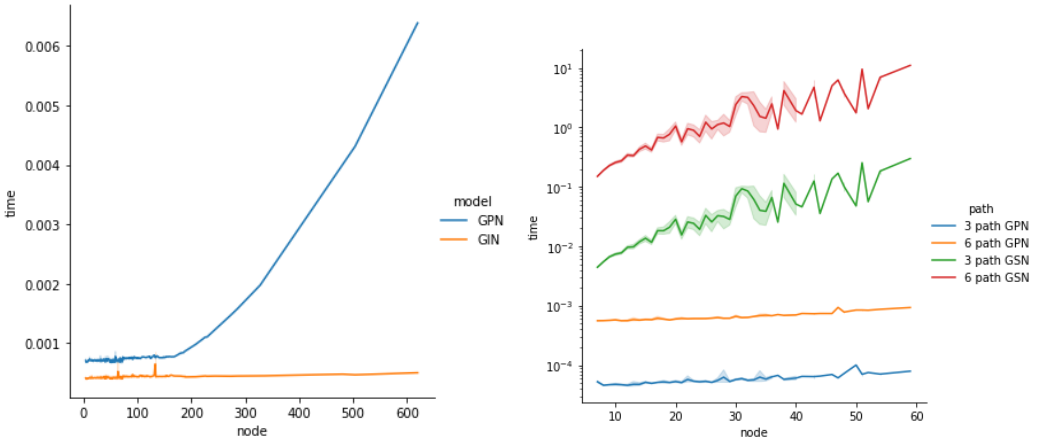

Figure 2: On the left, time consumption of GPN and GIN on PROTEINS dataset. On the right, time preconsumption of GPN and GSN (log scale) on IMDB-MULTI dataset for path of length 3 and 6.

dataset, we use 3 layers of GPN, with node dimension of 64, edge dimension of 8. We do not use dropout, except on NCI1 dataset where it is set to 0.2. Complete results are given in Table 3.

For the MUTAG, PTC, and NCI1 datasets, GSN achieved its best performance by precomputing cycles. We initially expected GPN to yield comparable results; however, GPN exceeded expectations, delivering results significantly better than GSN. On the Proteins dataset, GPN also outperformed GSN, despite GSN utilizing 4-clique precomputation to achieve its best results. We hypothesize that GPN may be capturing a broader range of substructures than initially anticipated, which warrants further investigation. For the IMDB-B dataset, where GSN precomputes 5-cliques, GPN's performance was below that of GSN.

We evaluated the computational time of GPN on the Proteins dataset, as well as the precomputation times of both GPN and GSN on the IMDB-MULTI dataset, with the results shown in Figure 2. This analysis confirms the quadratic inference complexity of GPN. However, for graphs with fewer than 150 nodes, the results indicate that the computation time ratio between GIN and GPN is below 2. Regarding precomputation, the evaluation underscores the efficiency gains provided by the [13] formulae.