Scaling Up Graph Homomorphism Features with Efficient Data Structures

Paul Beaujean, Florian Sikora & Florian Yger
LAMSADE, Université Paris-Dauphine, PSL
{paul.beaujean,florian.sikora,florian.yger}@lamsade.dauphine.fr

Abstract

Typical datasets used in graph classification tasks only contain a few thousand graphs which rarely exceed hundreds of nodes. Graph homomorphism densities are permutation-invariant features that can be directly computed from graph data, and their approximation scales naturally to large graphs. We propose the use of efficient data structures for approximate set membership in the context of a sampling algorithm for graph homomorphism density which enables the use of large-scale datasets containing larger graphs. To validate our findings, we compare this method with existing approaches used for graph homomorphism features in synthetic experiments.

1 Introduction

In current machine learning research focused on graph classification methods, experiments are conducted on a handful of small, well-documented, datasets which most often are graphs representing molecules as studied in biochemistry applications or graphs representing users and their interactions obtained from social network analysis. Unlike modern datasets used in mature application domains of machine learning such as computer vision with the ImageNet database (14.2M images, 20K categories) (Deng et al., 2009) or natural language processing with large Wikipedia corpora (40B characters) (Guo et al., 2020), the datasets commonly used in graph classification, such as those referenced in the TUDataset collection (Morris et al., 2020), are small. For example, biochemistry datasets such as NCI1, ENZYMES, PROTEINS, D&D (Wale et al., 2008; Borgwardt et al., 2005; Dobson & Doig, 2003) or social network datasets such as COLLAB, REDDIT-BINARY, or IMDB-BINARY (Yanardag & Vishwanathan, 2015) contain no more than a few thousands of graphs, each being no larger than a few hundred nodes. Recent work on creating larger datasets has been undertaken with the Open Graph Benchmark (Hu et al., 2020) offering significantly larger datasets (100K molecules) which however contain similarly small graphs.

Small datasets cause many practical problems: (1) they penalize machine learning models that require large training data (2) they make cross-validation difficult since validation and test sets become too small to carry enough statistical power (3) they make it impossible to study model scaling with additional data or larger data. In graph classification tasks in particular, small datasets cast doubt on the ability of models to generalize to larger graphs with similar structure.

Current models for graph classification use either computed features e.g. all approaches commonly found in graph kernels or learned features obtained by a variety of graph neural network architectures (Wu et al., 2020). Graph kernels, because of their inherently quadratic nature, have rarely been studied at scale, save for a few that admit clever reformulations (Kriege et al., 2020). On the other hand, many graph neural networks have been trained on large artificial graph datasets such as direct conversion of MNIST or CIFAR10 images into graphs (Dwivedi et al., 2020). However, these graphs retain geometric properties such as locality that make them qualitatively different from data that is inherently topological such as social networks for example.

This paper attempts to make graph classification possible at scale with large datasets containing graphs of all sizes. For this we focus on: (1) using homomorphism statistics as base topological feature descriptors for graphs, (2) leveraging efficient approximations available for homomorphism densities, and (3) identifying the adequate data structures to obtain a highly scalable implementation.

2 Graph Homomorphism Statistics are Feature Descriptors

A graph morphism is a mapping from the node set of a pattern graph F to the node set of a target graph G. Graph morphisms that preserve adjacency, i.e. all edges $uv \in E(F)$ are mapped to edges $f(u)f(v) \in E(G)$, are called homomorphisms. Graph homomorphisms that also preserve non-adjacency are called isomorphisms. We write that $G_1 \cong G_2$ if and only if there exists an isomorphism between G_1 and G_2. This is a natural way to express that two graphs are identical up to a permutation.

The existence of an homomorphism indicates the presence of a given pattern in a graph in a weaker sense that an isomorphism does. To obtain a global understanding of a graph it can be interesting to count how many patterns are present, which gives rise to the problem of finding the number of homomorphism from F to G which is denoted $\hom(F, G)$ also called homomorphism number. This intuition is formalized by a theorem of Lovász which shows that, by combining homomorphism statistics from multiple pattern graphs, it is possible to uniquely identify graphs up to isomorphism. We write $\hom_{\mathcal{F}}(G) = (\hom(F, G))_{F \in \mathcal{F}}$ to gather multiple homomorphism numbers from pattern graphs into a single vector which can be thought of as a feature descriptor.

Theorem 1. *(Lovász (1967)) Given two undirected graphs G_1 and G_2 with at most n nodes. Denoting by $\mathcal{G}_n$ the set of all simple graphs with at most n nodes, we have:*

$$G_1 \cong G_2 \iff \hom_{\mathcal{G}_n}(G_1) = \hom_{\mathcal{G}_n}(G_2).$$

This theorem has been extended to show that graphs whose $\hom$-vectors are close in the sense of a vector norm are also close in terms of cut-distance (Lovász, 2012). In practice, if we want to distinguish graphs from a dataset, it is not necessary to look at all their homomorphism statistics like in Theorem 1. Instead, a few patterns carry most of the information which is similar to graphlet kernels which use subgraph isomorphism statistics (Shervashidze et al., 2009). This is a fortunate situation considering the computational complexity of problems related to homomorphisms.

Despite homomorphisms being less demanding than isomorphisms, the task of finding an homomorphism from a pattern to a graph is already NP-hard from being related to clique-finding. In terms of computational complexity, computing $\hom(F, G)$ is #P-hard which is the equivalent of NP-hardness for counting problems. However, there are cases where it is possible to count homomorphisms efficiently. Indeed, Díaz et al. (2002) give a slice-wise polynomial algorithm parameterized by $\mathrm{tw}(F)$ the treewidth of the pattern graph F with a runtime of $O(\mathrm{tw}(F) \cdot kn^{\mathrm{tw}(F)+1})$. Note that $\mathrm{tw}(F)$ ranges from a minimum of 1 for trees to values such as k for k-by-k grids to a maximum of $n-1$ for a complete graph on n nodes. This turns out to be the best possible algorithm which matches computational complexity lower bounds established by Grohe (2007). Furthermore, even if we relax our assumptions and allow for a multiplicative approximation of $\hom(F, G)$, the problem remains as difficult as counting homomorphisms exactly (Bulatov & Živný, 2020).

Following this line of work, NT & Maehara (2020) have introduced the GHC framework based on homomorphism numbers where the family of patterns $\mathcal{F}$ is an hyper-parameter. They compute exact homomorphism numbers relative to patterns that have low treewidth and use the resulting permutation-invariant vector embeddings as input to traditional classifiers. Their approach has been found to be competitive with both direct embeddings used in combination with graph kernel methods as well as learned graph embeddings obtained via message-passing graph neural networks and has provably high representational power.

The situation is identical for the homomorphism density $t(F, G)$ which is a normalized number of homomorphisms from F to G defined as follows: $t(F, G) = \hom(F, G)/n^k$ where n and k are the number of nodes of G and F respectively. Homomorphism densities are as difficult to compute exactly and it is similarly hard to obtain multiplicative approximations of them.

3 sGHD: Sample Graph Homomorphism Density

The only remaining way to work around the limitations established by previous results in computational complexity is to relax even further the problem and allow additive approximation. By interpreting homomorphism density as being the probability for a random graph morphism to be an homomorphism, it is possible to use a sampling argument to compute an ε-additive approximation of the homomorphism density $t(F, G)$ in time $O\left((k \log n + l) \cdot \varepsilon^{-2} \log \delta^{-1}\right)$. This is achieved by Algorithm 1 which we call sGHD for sample graph homomorphism density.

The sGHD algorithm operates by computing an empirical mean: the first step consists in generating graph morphisms uniformly at random. The second step consists in computing the ratio of homomorphisms in the sample. Each morphism can be seen as a coin flip landing on heads when the morphism is an homomorphism and tails otherwise. In essence the sGHD algorithm is estimating the bias of that coin. Applying Chernoff bounds gives use a conservative guarantee that $O(\varepsilon^{-2} \log \delta^{-1})$ random morphisms will suffice to compute an ε-additive approximation.

Algorithm 1 sGHD

```
Require: $G$ an undirected graph on $n$ nodes, $F$ a pattern graph on $k$ nodes and $l$ edges, $\varepsilon > 0$ the
    requested additive precision, $1 - \delta \in (0, 1)$ the desired confidence.
Ensure: $\bar{t}$ such that $\mathbb{P}(|t(F, G) - \bar{t}| > \varepsilon) \leq \delta$
 1: $N \leftarrow O(\varepsilon^{-2} \log \delta^{-1})$
 2: for $i = 1$ to $N$ do
 3:     $f_i \sim (\mathcal{U}(0, n-1))_{[k]}$
 4: end for
 5: $\bar{t} \leftarrow \frac{1}{N} \sum_{i=1}^{N} \prod_{uv \in E(F)} \mathbb{1}_{E(G)}(f_i(u) f_i(v))$
 6: return $\bar{t}$
```

It is possible to extend the approach of NT & Maehara to approximate homomorphism densities with no significant loss in classification accuracy. The main advantage comes from the inherent scalability of Algorithm 1 which does not depend on the size of the target graph G outside of the initial pseudo-random number generation required to sample morphisms uniformly at random. Furthermore, the algorithm does not depend on the treewidth of F. Due to the efficiency of PRNG implementations, the practical running time of Algorithm 1 amounts to $O\left(|E(F)|\, \varepsilon^{-2} \log \delta^{-1}\right)$ which is close to constant for small pattern graphs and fixed precision. While Algorithm 1 is already efficient, we propose to speed up the computation of the homomorphism test implemented via the expression $\sum_{i=1}^{N} \prod_{uv \in E(F)} \mathbb{1}_{E(G)}(f_i(u) f_i(v))$ by focusing on membership queries for the target graph G.

4 Approximate Membership

The problem of set membership is one of the fundamental building blocks in the study of data structures. A wealth of data structures and algorithms provide various trade-offs and guarantees going from sequential search in an unsorted array, to tries, and hash tables. Of particular note is perfect hashing which allows for constant-time set membership queries at the cost of linear space (the space required to represent a perfect hashing function is proportional to the size of the set). In some way, this space complexity is optimal and cannot be improved upon.

Surprisingly, a simple probabilistic data structure, the Bloom filter (Bloom, 1970), allows to implement set membership close to this optimal space complexity. Each element of the set is represented by no more than a dozen of bits and the membership query is realized in constant time by applying a few hash functions to the element to be queried. Bloom filters are implemented via an array of bits which are filled at indices corresponding to the outputs of the different hash functions. The randomness of this data structure is solely contained in the choice of hash functions and is perfectly deterministic once the hash functions have been chosen.

Bloom filters alleviate the need to store the elements themselves unlike e.g. hash tables and other traditional data structures. This convenience in the specific context of the set membership problem comes at the cost of a false positive rate. Indeed, when a Bloom filter returns that an element is not in the set, that answer is always truthful. However a positive answer does not guarantee that the element is actually in the set.

With regards to computing the sample homomorphism density, each graph G can thus be compressed into a Bloom filter containing information about its edges. In practice this means that each edge, usually represented by a pair of unsigned integers each coded over 32, 64, or 128 bits is compressed

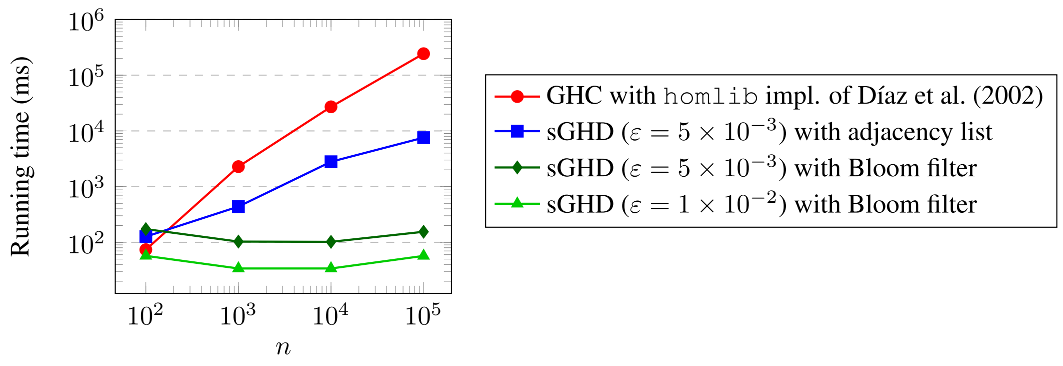

Figure 1: Running time of homomorphism density algorithms w.r.t. $K_3 \to G(n, \log^2 n/n)$

down to a constant dozen of bits, regardless of the number of nodes of G. Furthermore, each homomorphism test corresponds to computing grouped queries which must all return true. Since the queries are independent, this implies a lower false positive rate for the grouped query than its constituent queries. Finally, the Bloom filter associated with G is constructed once as a read-only data structure which can be queried repeatedly in parallel when computing the sample homomorphism density of G with regards to queries representing the edge set of some pattern graph F.

There exist many variants of the Bloom filter which may support less operations or instead display additional properties. We note in particular cuckoo filters (Fan et al., 2014) which leverage the properties of cuckoo hashing to lower the number of memory accesses required for each membership query. More recently, XOR filters (Graf & Lemire, 2020), have been proposed to focus entirely on the set membership problem from the point of view of a static set which does not change over time. This specialization allows for a lower amount of bits per key which enables faster querying than most alternatives, albeit at the cost of a slower construction.

5 Experimental Evaluation

We describe experiments which attempt to compare two currently available approaches to compute graph homomorphism features. The first method is the C++ implementation of the algorithm of Díaz et al. (2002) given by the `homlib` library of NT & Maehara (2020). This algorithm includes a tree decomposition routine together with a dynamic programming algorithm to compute homomorphism numbers. Obtaining homomorphism densities is done by dividing by n^k which is the cardinality of all morphisms from a pattern graph F with k nodes to a target graph G with n nodes. The second method is our implementation of Algorithm 1 with two variants: exact edge membership queries to an adjacency list and approximate queries to a Bloom filter representing the edge set of G. Our implementation is written in the Rust programming language (Matsakis & Klock, 2014) and will be made available as a library at a later date. Our experiments use Erdős-Rényi random graphs $G(n,p)$ to control the presence of subgraphs via the edge density p. This allows us to manipulate the target homomomorphism density for many small pattern graphs. Furthermore, this method allows us to generate families of graphs of varying sizes which retain common properties.

The `homlib` library implements graphs using adjacency lists and as such demonstrates great performance on smaller graphs. However, its running time is highly dependent on the density of the target graph. Consider for example a random graph $G(1000, 9 \times 10^{-3})$ (disconnected w.h.p.) which contains very few triangles, for which `homlib` computes an exact triangle density of 7.9×10^{-7} in only 8ms. The running time drastically increases to 102ms for $G(1000, 4 \times 10^{-2})$ (connected w.h.p.) which has a notably larger density of 6.4×10^{-5}. More importantly, the running time of `homlib` scales linearly with the size of G as shown in Figure 1. Surprisingly, increasing the treewidth parameter of the pattern graph F, from triangle to K_5 the complete graph over 5 nodes, incurs at most a linear increase in the running time which does not match the worst-case time complexity.

For the sample homomorphism density implementation of Algorithm 1 we set a requested additive error of $\pm 1 \times 10^{-2}$ (this is not 1% error) and $\pm 5 \times 10^{-3}$ with 95% confidence, i.e. no more than 5% of the sample densities exceed the error bound. In practice it is clear this theoretical error bound is extremely conservative. For example if we consider a graph $G(400, 0.05)$ with approximately 4000 edges, the exact triangle homomorphism density is 1.26×10^{-4} which should be well below the "detection threshold" of $\pm 1 \times 10^{-2}$ obtained by the Chernoff bounds. However, the sample homomorphism density correctly identifies the order of magnitude with a sample density of 1.94×10^{-4}.

In the case of our implementation of Algorithm 1 using Bloom filters we select a 1% false positive rate. Compared to the standard implementation, the Bloom filter variant seems entirely agnostic to the size of G. Surprisingly, we observe a behavior that is comparable to `homlib` in the sense that lower density is associated with shorter running time. We notice that low density in the target graph results in a higher frequency of negative membership queries which short-circuit the entire group query. As described by its theoretical time complexity, the most important factor influencing the running time of Algorithm 1 is the requested additive error ε. Figure 1 reveals that a fixed additive error of $\varepsilon = \pm 1 \times 10^{-2}$ results in running times ranging from 30ms to 60ms while raising the precision to $\varepsilon = \pm 5 \times 10^{-3}$ increases the running time to values ranging from 100ms to 200ms. We underline that this additional cost is needed when the target density becomes small. For example, the triangle homomorphism density of a random graph $G(n, \log^2 n)$ with $n = 10^5$ is 2.1×10^{-8} which cannot be detected (the sample density is 0) with a precision of $\varepsilon = \pm 1 \times 10^{-2}$. However, a slightly better precision with $\varepsilon = \pm 5 \times 10^{-3}$ gives a coarse approximation of this value which is "only" off by 2 orders of magnitude. This detection threshold can also be used as an implicit filter discarding low-frequency patterns. Finally, we observe like with `homlib` that the size of F has very little impact on the running time.

We have conducted additional experiments with cuckoo filters and XOR filters which reproduce the known tradeoffs between fast filter construction and fast filter querying. However the cuckoo filter implementation we have tested was slower approximately by 10% to 20% on both construction and querying than Bloom filters while XOR filters were up to 30% faster than Bloom filters at the cost of more than double the construction time.

6 Conclusion

We propose the use of approximate set membership data structures to scale up the computation of sample graph homomorphism densities and demonstrate in experiments that this approach is scalable and appropriate for graphs of varying connectivity. In particular, this paves the way towards on-demand feature generation in the context of feature representation learning where the family of patterns is the representation to be learned. More generally, this approach enables the use of much larger datasets containing larger and more diverse graphs in graph classification tasks. We hope that this efficient method for feature generation encourages the creation of larger datasets which would contribute to the study of machine learning models for graph classification at scale.

Acknowledgments

This work was supported by the ANR project STAP (ANR-17-CE23-0021). FY acknowledges the support of the ANR as part of the "Investissements d'avenir" program, reference ANR-19-P3IA-0001 (PRAIRIE 3IA Institute).

References

Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

Karsten M Borgwardt, Cheng Soon Ong, Stefan Schönauer, SVN Vishwanathan, Alex J Smola, and Hans-Peter Kriegel. Protein function prediction via graph kernels. *Bioinformatics*, 2005.

Andrei Bulatov and Stanislav Živný. Approximate counting CSP seen from the other side. *ACM Transactions on Computation Theory*, 12:1–19, 05 2020. doi: 10.1145/3389390.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Josep Díaz, Maria Serna, and Dimitrios M Thilikos. Counting H-colorings of partial k-trees. *Theor. Comput. Sci.*, 2002.

Paul D Dobson and Andrew J Doig. Distinguishing enzyme structures from non-enzymes without alignments. *J. Mol. Biol*, 2003.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. Cuckoo filter: Practically better than bloom. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pp. 75–88, 2014.

Thomas Mueller Graf and Daniel Lemire. Xor filters: Faster and smaller than bloom and cuckoo filters. *Journal of Experimental Algorithmics (JEA)*, 25:1–16, 2020.

Martin Grohe. The complexity of homomorphism and constraint satisfaction problems seen from the other side. *J. ACM*, 54(1), March 2007. ISSN 0004-5411. doi: 10.1145/1206035.1206036. URL `https://doi.org/10.1145/1206035.1206036`.

Mandy Guo, Zihang Dai, Denny Vrandečić, and Rami Al-Rfou. Wiki-40B: Multilingual language model dataset. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pp. 2440–2452, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL `https://www.aclweb.org/anthology/2020.lrec-1.297`.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 2020.

László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 1967.

László Lovász. *Large networks and graph limits*. American Mathematical Soc., 2012.

Nicholas D Matsakis and Felix S Klock. The Rust language. *ACM SIGAda Ada Letters*, 34(3): 103–104, 2014.

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 Workshop on Graph Representation Learning and Beyond (GRL+)*, 2020. URL `www.graphlearning.io`.

Hoang NT and Takanori Maehara. Graph homomorphism convolution. In *ICML*, 2020.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *AIStat*, 2009.

Nikil Wale, Ian A Watson, and George Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *Knowl. Inf. Syst.*, 2008.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Trans. Neural Networks Learn. Syst.*, 2020.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *KDD*, 2015.