Graph Homomorphism Features: Why Not Sample?

Paul Beaujean [1] Florian Sikora [1] Florian Yger [1]

[1] LAMSADE, Université Paris-Dauphine

ECML PKDD 2021
VIRTUAL
13-17 September

What is a graph morphism?

Graph morphisms are functions that map the set of nodes from a **pattern** graph F to the set of nodes of a **target** graph G. Graph morphisms that preserve adjacency are called **homomorphisms**.

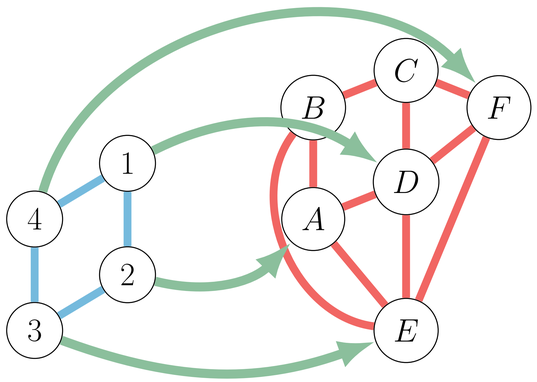

Figure: Example of an homomorphism $f : C_4 \to G$

Graph homomorphisms that also preserve non-adjacency are called isomorphisms. We say 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.

Computational Complexity

- Finding a graph homomorphism is **NP-hard** because of its relation to finding cliques
- Counting graph homomorphisms is **#P-hard** however
- Díaz et al. (2002) give a **slice-wise polynomial algorithm** depending on the **treewidth of** F
- Allowing some **multiplicative approximation** does **not** make the problem **easier**

Homomorphism statistics: a canonical vector representation of graphs

Combining homomorphism statistics from **several pattern graphs** reveals deep information which characterizes graphs and can be used as a **permutation-invariant vector representation**.

- Homomorphism **numbers**: absolute counts (between 0 and n^k)
- Homomorphism **densities**: ratio of homomorphisms to morphisms (between 0 and 1)

Lovász on homomorphism densities

Theorem

Given two undirected graphs G_1 and G_2 with the same number of nodes n, and $\mathcal{G}_n$ the set of all simple graphs with at most n nodes, we have:

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

- $t(F, G)$ = number of homomorphisms from F to G / number of morphisms from F to G
- $t_{\mathcal{G}_n}(G) = (t(F, G))_{\mathcal{G}_n}$ where $\mathcal{G}_n$ is the set of graphs on at most n nodes
- permutation-invariant vector representation of graphs
- can be extended to distance between t-vectors = cut-distance between graphs

sGHD: sample graph homomorphism density

It is possible to obtain a **fast** ε**-additive approximation** of the graph homomorphism density from F to G with **no condition on the treewidth of** F. By sampling homomorphisms uniformly at random this can be done in $O\left((k \log n + l) \cdot \varepsilon^{-2} \log \delta^{-1}\right)$ time.

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

Optimizing set membership $uv \in E(G)$

When implementing sGHD the main bottleneck comes from handling the large number of random set membership queries. **Bloom filters** are efficient data structures which offer **fast approximate set membership** queries without the need to store the corresponding data structure. Bloom filters have a **false positive rate** which can be compensated by a larger memory footprint.

- False positive rate of Bloom filters partially compensates undersampling in practice
- Allows a highly scalable parallel implementation of sGHD
- No noticeable penalty on machine learning tasks compared to exact set membership

Approximate homomorphism densities for large-scale graph learning

To handle **large datasets** containing **large graphs** we propose to use approximate homomorphism densities as permutation-invariant feature descriptors parameterized by a given **family of pattern graphs**.

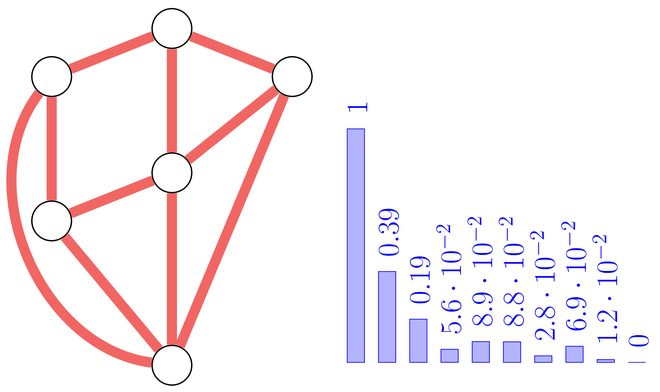

Figure: G and its approximate homomorphism densities from 10 small pattern graphs

For performance and simplicity we focus only on families of small pattern graphs. This approach is justified by results by Lovász on **graph algebras** involving products of homomorphism densities.

Experimental results

We compare two implementations of the sGHD algorithm with the standard implementation of the graph homomorphism numbers-based GHC algorithm of NT & Maehara (2020). Experiments are conducted on **Erdős-Rényi random graphs** with an edge density that guarantees the presence of several subgraph patterns. We obtain a **quasi-constant runtime** for sGHD with Bloom filter.

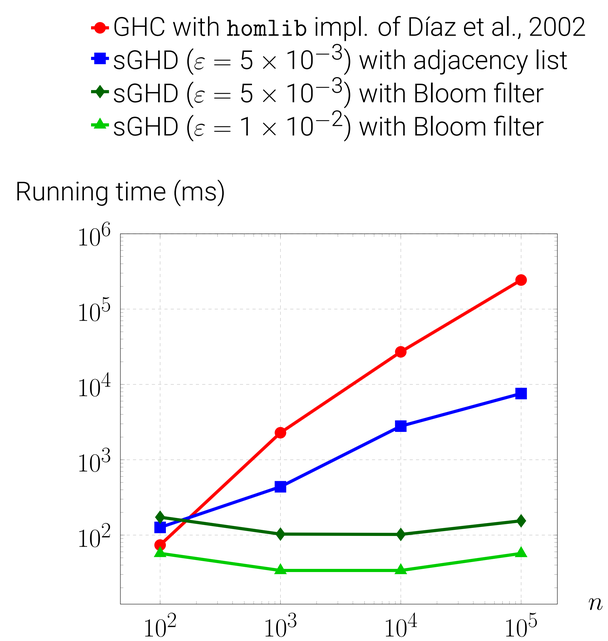

Figure: Running time of homomorphism density algorithms w.r.t. $K_3 \to G(n, \log^2 n/n)$

Additional insights

- Low runtime increase from computing homomorphism statistics from larger cliques K_5, K_6
- Actual precision obtained is 2 orders of magnitude above requested precision ε
- Graph sparsity short-circuits edge membership queries leading to good performance

Future work

1. Additional experiments on currently available large datasets such as OGB (small graphs), MalNet (large graphs), and new social network datasets currently being extracted
2. Representation learning via selection of the family of pattern graphs
3. Investigation into polynomial models and their relationship with graph algebras

References

[1] Burton H Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 1970.

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

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

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