Project Report on Homomorphisms Are a Good Basis for Counting Small Subgraphs

Dai Cao
400370
Supervisor: Prof.Dr.Pascal Schweitzer

December 12, 2019

Contents

1 Introduction

The purpose of this project[1] is to implement the subgraph counting algorithm stated in [1] in Python as well as apply some experiments to compare it with other state-of-the-art algorithms such as VF2 and VF3 from the perspective of running time and memory consuming so as to verify the effectiveness of the framework *Graph Motif Parameters(GMF)*, using different publicly available databases and random generated networks.

For the sake of clarity, the graphs in the project are all unlabeled, undirected, simple graphs, in particular, these graphs contain neither loops nor parallel edges while they can be either connected or disconnected. In fact, VF2 and VF3 cope up with both directed and undirected graphs, the limitation lies in the homomorphism counting algorithm embedded in GMP. The term subgraph in the following always refers to a partial subgraph if it is not specially specified.

In subsequent sections, we first discuss subgraph counting algorithms chosen to be compared and state their usages. In section 3, we then give an introduction to the implementation of GMP. Section 4 details the datasets we make use of. The subsequent part provides our main empirical results. Finally, we present a summary of the project and possible future works.

2 Subgraph Isomorphisms and Subgraphs Counting

None of subgraph counting algorithms in this project directly determines the existing number of a pattern graph in a target graph. The generation of the problem is *subgraph isomorphism* which is NP-Complete in the general case. In addition, VF2 and VF3, in fact, search for all node-induced subgraph isomorphisms between two graphs.

As to packages for manipulation and statistical analysis of networks, there are a lot of in different programming languages out there, we mainly focus on three Python modules: *networkx*[2], *graph-tool*[3], and *igraph-python*[4] that all of them have VF2 but no VF3 implemented. Graph operations in programs

[1]`https://alg-git.informatik.uni-kl.de/Dai/GraphMotifParameters`

[2]`https://networkx.github.io/`

[3]`https://graph-tool.skewed.de/`

[4]`https://igraph.org/python/`

and experiments are mostly based on the most well-known *networkx*, since it has nice documentation, flexible graph structure, and easy-to-use functions. For visualization, we recommend *graph-tool*, and its core data structures and algorithms of being implemented in C++ confers it a level of performance that is comparable (both in memory usage and computation time) to that of a pure C/C++ library. Besides VF2, *igraph-python* has another competitive subgraph isomorphism solver LAD realized.

2.1 VF2

As mentioned above, apparently a two-step transform process is required to obtain the desired subgraph number from a program who returns the number of node-induced subgraph isomorphisms: node-induced subgraph to subgraph and subgraph isomorphism to subgraph. For *networkx*[5] we call subgraph monomorphism instead to skip the first step while for *graph-tool*[6] turning the option *induced* to be False is enough, *igraph-python* counts subgraph isomorphisms exactly.

Step two is easily achieved by

$$\mathrm{Sub}(H, G) = \mathrm{SubIso}(H, G)/\mathrm{Aut}(H). \tag{1}$$

2.2 VF3

VF3[7], as a further evolution of VF2, is merely accessible in C++ for now, the most straightforward way is to use Python subprocess module to execute programs written in different languages. Furthermore, the two-step transformation is still demanded in VF3, [1] gives the transformation from induced subgraph counting to subgraph counting

$$\mathrm{Sub}(H, G) = \sum_{F \in \mathcal{G}^*} \mathrm{Ext}(H, F) \cdot \mathrm{IndSub}(F, G), \tag{2}$$

the $\mathrm{Ext}(H, F)$ is the number of extensions of H that are isomorphic to F, i.e., $\mathrm{Sub}(H, F)$. Since we deal with small pattern H, the main computation

[5]`https://networkx.github.io/documentation/stable/reference/algorithms/isomorphism.vf2.html`

[6]`https://graph-tool.skewed.de/static/doc/topology.html#graph_tool.topology.subgraph_isomorphism`

[7]`https://github.com/vincarlet/vf3lib`

goes into the second part, it is acceptable to leverage other exsiting subgraph counting programs here like VF2.

3 Relating Counting Subgraphs to Counting Homomorphisms

Now come to the most essential contribution part. The explicit relation between subgraph numbers and homomorphism numbers has been provided in [1] with

$$\mathrm{Sub}(H, G) = \sum_{\rho} \frac{(-1)^{|V(H)|-|V(H/\rho)|} \cdot \prod_{B\in\rho}(|B|-1)!}{\mathrm{Aut}(H)} \cdot \mathrm{Hom}(H/\rho, G). \quad (3)$$

We divide the formula into two parts, the graph partition part and homomorphism counting part, then $\mathrm{Sub}(H, G)$ can be regarded as a multiplication of two vectors.

3.1 Graph Partition

The graph partition generation is completed along with a hierarchy structure, at each layer, new child graphs are generated by merging one pair of non-adjacent vertices in a graph from its upper layer until no possible combination. The node mapping from a new graph to the root graph is also stored meanwhile as a novel partition. Then the first layer of a connected graph is itself, and the last layer will be an edge.

The sum in (3) ranges over all partitions ρ of $V(H)$, it is natural to see that in a middle layer two different graph partitions ρ_1, ρ_2 are possible to form isomorphic quotient graphs H/ρ_1 and H/ρ_2, especially the penultimate layer always ends with a collection of path-2 and triangles. It seems worthwhile to accomplish isomorphism checking between all quotient graphs belonging to the same layer and eliminate redundant before the execution of homomorphism counting in most of the cases, taking the enormous gap between the order of H and the order of target graph into consideration. Moreover, the intuition has already been confirmed in our experiments.

3.2 Homomorphism Counting

There are two public Python modules on Github for homomorhpism counting: *homsearch*[8] and *homlib*[9]. Their core branching algorithms are implemented in C++, with Cython and pybind11 interface to Python respectively.

The package *homlib* implements [2] upon which the most crucial Proposition 1.6 in [1] is based. For more details, the homomorphism counting algorithm consists of two phases: first to find a nice tree decomposition of the pattern graph using greedy heuristics in [3, 4], then second to use [2]'s dynamic programming algorithm on the nice tree decomposition. We have *homlib* successfully installed and run on a small test but fail to have it applied because of a sudden interruption for unknown issues during experiments.

homsearch seems to be much more stable and empirical. Additionally, we are able to speed up the execution of $\mathrm{Hom}(H, G)$ by calculate the number of homomorphisms from each connected component of H to each connected component of G and do the corresponding additions and multiplications to achieve final value for disconnected graphs H and G. Nevertheless, a notable shortcoming of the module is it is not implemented for graphs larger than 4096.

4 Experiments

All experiments are performed on macOS Catalina with 16G memory and 2.7 GHz processor, so the quantity, the order and the size of evaluated graphs are selected moderately to avoid running out of memory and time. The multiprocessing module of Python is also employed to boost performance.

4.1 Datasets

The small patterns H are chosen to be graphs with order less than 7, such as motifs[10], small graphs[11], or special patterns such as cliques, paths, trees and so on. From the perspective of practicality, specific classes of patterns who have their own much more effective counting algorithms are avoided, e.g.

[8] `https://github.com/gavento/homsearch`

[9] `https://github.com/spaghetti-source/homlib`

[10] `https://github.com/nkahmed/PGD`

[11] `http://www.graphclasses.org/smallgraphs.html`

edges, independent nodes, stars. We examine the performance of aforementioned subgraph counting algorithms on a set of real-world networks from KONECT[12] as well as a fair corpus of Erdős-Rényi random graphs.

4.2 Experiment setting and results

In many cases, the graph motif parameters algorithm outperforms VF2 and VF3, see Figure 1. Maybe VF3 is only recommended when dealing with very dense large connected graphs. The table 1 gives explicit running times on small cliques and paths with the same real-world graphs as targets to deliver a more concrete sense. The memory usage is not discussed because of no significant difference observed.

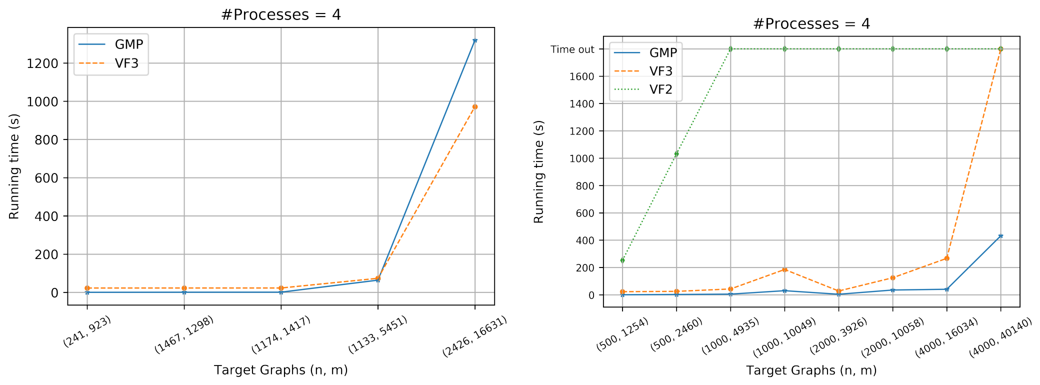

Figure 1: Running time of three subgraph counting algorithms: GMP, VF2 and VF3, counting numbers of subgraphs for a set of motifs no larger than 5 in a target graph, and exceeding half an hour is regarded as time out. On the left, target graphs are real-world networks, the right are ER random graphs.

5 Summary

It is truly surprising to see the outstanding performance of the new subgraph counting algorithm GMP given there is no theoretical evidence and considerable improvements from other twos in many cases. It may not fair to compare the VF2 who is written in Python with GMP whose cores are in

[12]`http://konect.uni-koblenz.de/`

Time(s)	Chicago		Euroroad		U. Rovira		Hamsterster	
	GMP	VF3	GMP	VF3	GMP	VF3	GMP	VF3
K_3	0.023	2.045	0.028	2.040	0.488	2.068	3.721	2.634
K_4	0.034	2.044	0.036	2.046	1.250	2.165	36.707	5.999
K_5	0.050	2.042	0.055	2.040	3.634	2.266	393.729	50.002
P_2	0.102	4.115	0.201	4.097	2.311	4.167	15.950	4.696
P_3	0.336	10.290	0.683	10.273	43.482	11.813	654.739	59.666

Table 1: GMP vs. VF3

C++, we also test VF2 from *graph-tool* who is claimed to have comparable efficiency as C++, it still the worst even though in some cases VF2 beats VF3. So a possible work to do is figuring out the applications for each one. One of the limitations of GMP is it only suita unlabeled undirected graphs, then the other direction could be extending it to more general graphs.

References

[1] Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2017, pages 210–223, New York, NY, USA, 2017. ACM.

[2] Josep Díaz, Maria J. Serna, and Dimitrios M. Thilikos. Counting h-colorings of partial k-trees. In *Proceedings of the 7th Annual International Conference on Computing and Combinatorics*, COCOON '01, pages 298–307, London, UK, UK, 2001. Springer-Verlag.

[3] Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations i. upper bounds. *Inf. Comput.*, 208(3):259–275, March 2010.

[4] Aaron B. Adcock, Blair D. Sullivan, and Michael W. Mahoney. Tree decompositions and social graphs. *CoRR*, abs/1411.1546, 2014.