Investigating the performance of different algorithms for the densest subgraph problem/arboricity

Nadya Voronova
Boston University
Boston, USA
voronova@bu.edu

Abstract

The main goal of the project is to investigate the performance of various algorithms for DSP/arboricity on different datasets.

1 Introduction

In this project I give my attention to the Densest Subgraph Problem. The input to this problem is a graph G and the expected output is

$$dens(G) = \max_{S \subset V(G)} \frac{|E(S)|}{|S|}$$

where $E(S)$ is the set of edges induced by S.

This problem is widely encountered in the real-life. Most of the time real-world graphs a relatively sparse and an existence of a dense subgraph usually is a sign of some abnormality. Because of this DSP is a problem of current interest both in the theoretical community and in graph/data mining community.

DSP is closely related to another problem which is arboricity computation problem. Arboricity of a graph G is the number of disjoint forests that can cover the whole graph. And it's closely related to the DSP because

$$dens(G) \leq arb(G) \leq dens(G) + 1$$

In this project I investigated what algorithms for estimation of these characteristics of the graph exist and compared their performance of different datasets.

2 Project description

The project consisted of three stages:

- Reading the papers and identifying the algorithms to compare
- Implementing the algorithms that have not been implemented yet and setting up the already existing implementations
- Running the algorithms on different datasets and comparing their performance both in terms of time and the approximation

The papers I read were:

Submitted as a project report for CS599 T1 class at BU for Fall2021.

- Paper [1] which presents an approximation algorithm for DSP problem. The main idea behind the algorithm is the following. If it only ran with number of iterations set to 1 it will just peel a node with the lowest degree one out of time and record the highest density of the subgraphs it gets during this procedure. This algorithm is guaranteed to output a 2-approximation of the optimal solution for DSP. If the number of iterations is greater than one then for each iteration the algorithm continues the same process but now it bases the decision of "what node to delete?" not only on the degrees of the vertices but also on the weights that depend of previous iterations of the algorithm. By the authors' conjecture this algorithm gives $1 + O(\frac{1}{\sqrt{T}})$ approximation where T is the number of iterations.
- Paper [2] which presents an approximation algorithm for computing of arboricity. In this paper the algorithm doesn't need to process the whole graph. Instead of this it quires specific nodes in the graph and estimates the arboricity based on this small sample. The main idea for the estimation is that graphs with small arboricity can be decomposed to a layered decomposition where nodes in each layer don't have too many edges to a different layer. This gives an algorithm that can answer to a decision gap problem "Is the input graph a graph with small arboricity or with big arborocity?". And the search version of the problem is done by doing binary search of the threshold for the decision problem. This algorithm outputs a value α such that $\frac{arb(G)}{200\log^2(n)} \leq \alpha \leq alpha$.
- Paper [3] which presents an exact algorithm for the DSP problem based on the min-cut/max-flow approach (for the parameter $k = 2$).
- Paper [4] which presents an approximation streaming algorithm for DPS for a dynamic graph. The main idea of this paper was to just sample a subset of edges and run the flow-based algorithm on the sampled graph.

3 Implementation and experiments details

I decided to compare the performance of the Greedy++ algorithm (from paper [1]), arboricity approximation algorithm (from paper [2]), the algorithm that solves the problem exactly (from paper [3]) and the approximation algorithm that uses the idea from paper [4] of sampling some of the edges and then running the exact algorithm on the sampled subgraph.

I used the the implementation for the Greedy++ available here `https://www.dropbox.com/s/jzouo9fjoytyqg3/code-greedy%2B%2B.zip?dl=0` . I also discovered that the implementation of the flow-based algorithm from the paper [3] is the part of this codebase so I used it instead of writing my own version. I implemented the arboricity approximation algorithm (the code can be found here `https://github.com/nvoronova/algorithms_for_DSP`) in C++. I chose C++ because the Greedy++ algorithm as well as the flow-based algorithms are both implemented in C++. For the last approach with the sampling I used a script in Python to go from the input file containing the whole input to the input file containing only the sample with the nodes re-enumerated.

I decided to use the following datasets for my experiments:

- Complete graph on 1k nodes
- A graph on 1k nodes with only one edge
- Social circles from Facebook (anonymized), 4k nodes. (`http://snap.stanford.edu/data/ego-Facebook.html`)
- Social network of LastFM users from Asia, 7k nodes. (`http://snap.stanford.edu/data/feather-lastfm-social.html`)
- Collaboration network of Arxiv High Energy Physics, 12k nodes. (`http://snap.stanford.edu/data/ca-HepPh.html`). Denoted as HepPh in the results.
- Collaboration network of Arxiv Condensed Matter, 23k nodes. (`http://snap.stanford.edu/data/ca-CondMat.html`). Denoted as CondMat in the results.

- Social network of Github developers, 37k nodes. (`http://snap.stanford.edu/data/github-social.html`). Denoted as git in the results.
- Gemsec Facebook dataset, artist, 50k nodes. (`http://snap.stanford.edu/data/gemsec-Facebook.html`)

The experiments were performed on a single machine, with an Intel® Core™ i7-8565U CPU @ 1.80GHz × 4 cores, and 10GB of main memory

4 Experiments results and challenges on the way

4.1 Experiments results

Here are the results of the experiments for the artificial data: two graphs on 1k nodes, one is complete, and another has only one edge.

	1k nodes, complete		1k nodes, 1 edge	
	result	time (in seconds)	result	time (in seconds)
exact	499.5	4.029	0.5	0.004
greedy++, 1 iter	499.5	0.019	0.5	0.003
greedy++, 2 iter	499.5	0.032	0.5	0.002
greedy++, 10 iter	499.5	0.118	0.5	0.002
arboricity	1	0.561	1	0.073
sampling	318.02	2.344	-	-

You can see that the algorithm for arboricity didn't do much and also took longer that almost all other. This is something that you'll be able to see in all other experiments.

	facebook, 4k nodes		lastfm, 7k nodes		HepPh, 12k nodes	
	result	time	result	time	result	time
exact	77.346	0.53	14.793	0.174	238.004	2.409
greedy++, 1 iter	77.3465	0.006	14.7937	0.003	238.134	0.031
greedy++, 2 iter	77.3465	0.007	14.7937	0.005	238.134	0.092
greedy++, 10 iter	77.3465	0.027	14.7937	0.017	238.114	0.278
arboricity	1	2.117	1	9.332	1	16min
sampling	-	-	-	-	-	-

And the results for bigger datasets

	CondMat, 23k		git, 37k		facebook, 50k	
	result	time	result	time	result	time
exact	26.833	1.642	30.248	3.468	58.137	17.211
greedy++, 1 iter	58	0.022	30.248	0.028	216	0.098
greedy++, 2 iter	58	0.056	30.248	0.061	216	0.197
greedy++, 10 iter	58	0.166	30.248	0.241	216	0.844
arboricity	1	15min	1	2min	1	5min
sampling	-	-	-	-	-	-

4.2 Challenges

While I was running the experiments I encountered several challenges, such as:

- The arboricity approximation algorithm has a big approximation gap. So for $n < 100k$ the output 1 would be a valid output no matter what is the input since

$$\frac{arb(G)}{200\log^2(n)} \leq \frac{n}{400\log^2(n)} < 1$$

.

On the other hand, because of the size of the data we need to remember for this algorithm, it can't really handle big graphs. So my only hope for this algorithm was that it performs significantly better on the real-world data that the theoretical guaranties. Unfortunately, it was outputting 1 for all inputs I ran it on.

- The sampling for the forth approach works the following way. Each edge is sampled with probability $c\varepsilon^{-2}\log(n)\frac{n}{m}$, where ε is the approximation parameter and c is a constant. And this expression is bigger than 1 for sparse graphs. Most of the graphs I used as the inputs were sparse so for most of the graphs this algorithm was not usable.
- While I was performing the experiments using the implementation for [1] I noticed a couple of things that need further work. The most important is that for several datasets the output of the exact version was far away from the output provided by the approximation version. For example, on the dataset facebook_big the exact algorithm output was 58, but the approximation was 216 which is inconsistent with the conjecture and my understanding of the algorithm. This could be due to an error in the implementation or due to an error in the creation of the input, I didn't have time to investigate this further. Another thing that was challenging was that the code is leaking memory. It requests a lot of memory through **new** but never frees this memory using **delete**. Because of this and of the fact that I don't have a lot of RAM (used virtual machine to run the experiments) it took me a while to run some of the bigger experiments. And the last thing is that it expects the nodes to start from 1 which is never mentioned in the documentation.

5 Summary

For the relatively small sparse graphs the Greedy++ works the best both in terms of the performance and in terms of the approximation.

The sampling technique could be useful for the dense graphs but most of the real-world graphs are not dense enough for this technique to be valid.

The arboricity approximation algorithm only makes sense for really big graphs. But in order to use it for these graphs the implementation should use significantly less memory and I personally not quite sure if there is an easy way to reduce the memory usage without changing the algorithm.

References

[1] Digvijay Boob, Yu Gao, Richard Peng, and Saurabh Sawlani, and Charalampos Tsourakakis, and Di Wang, and Junxing Wang, (2020) Flowless: Extracting Densest Subgraphs Without Flow Computations. *Proceedings of The Web Conference 2020.*

[2] Eden T., Mossel S., Ron D. Approximating the Arboricity in Sublinear Time. *//arXiv preprint arXiv:2110.15260. – 2021.*

[3] Mitzenmacher, M., Pachocki, J., Peng, R., Tsourakakis, C., & Xu, S. C. (2015, August). Scalable large near-clique detection in large-scale networks via sampling. *In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 815-824).*

[4] McGregor, Andrew, et al. "Densest subgraph in dynamic graph streams." *International Symposium on Mathematical Foundations of Computer Science. Springer, Berlin, Heidelberg,* 2015.