PhD project
Sorting nodes to scale to massive real-world networks

2020

1 Societal and scientific context

Various real-life situations generate data that fits on a network: international trade, traffic jam, scientific collaboration... Our collective ability to gather and organise this data makes increasingly massive graphs available: online social networks can track millions of users and their billions of interactions, web crawlers have indexed trillions of web pages, and the 10^{14} synaptic connections of our brain could be mapped entirely over the next years.

While algorithms are sometimes considered as efficient when they run in polynomial time (as opposed to exponential), the size of these datasets makes it impossible to run quadratic procedures even with the fastest current supercomputers[1]. In order to keep pace with growing real-life networks and their computer-generated counterparts, it is crucial to design quasi-linear algorithms.

However, some typical graph problems are proved to be NP-hard: it means that worst cases cannot be solved in polynomial time. One possibility is to use heuristics with the aim of obtaining good approximations in reasonable time. An other way is to restrict the problem to real-world graphs, that have been shown to present specific and favourable characteristics. For instance, research has shown that networks from very different fields had power-law degree distributions and smaller diameters than expected in a random graph. It has also been observed that many algorithms with high worst-case complexity behaved quasi-linearly on real-life networks.

2 Summary of the thesis

Real datasets are here the motivation to build new theoretical tools that specifically adapt to them. That is the approach taken by the LIP6 team in the LiMass project (Linear algorithms for Massive real-world graphs)[2]. The PhD project is supervised by this team. Additionally, regular visits to the CRI Paris (Center for Research and Interdisciplinarity) are planned, where the Network Ties team[3] collects and analyses data from various sources.

Sorting the nodes of an input graph in a relevant order has proved to be a key subroutine for solving many problems in massive graphs. Such order-dependent algorithms are particularly interesting to develop efficient algorithms, as the relevant order of the nodes can generally be computed in quasi-linear time.

For instance, the degree ordering can be obtained by first computing the degree of each node (time $\mathcal{O}(m+n)$) and then sorting them. We can do that in time $\mathcal{O}(n\log(n))$ with quick sort, or even $\mathcal{O}(n)$ using bucket sort, as degrees are bounded by n. Similarly, the degeneracy ordering has become one of the most popular node ordering; it can be obtained in linear time by repeatedly removing a node of minimal degree (graph pealing algorithm [2]).

Examples of problems benefiting from sorting nodes in a relevant order include finding a dense subgraph [6], listing triangles [9], listing k-cliques [5, 3], finding or listing maximal cliques [11, 8], counting k-motifs [10] and compressing graphs [1, 7].

The PhD candidate will work on (i) finding better orders to use in existing order-dependent graph algorithms, (ii) suggesting new efficient order-dependent graph algorithms and (iii) applying these tools to several large real-world networks (both well-known and newly-collected datasets).

[1] en.wikipedia.org/wiki/TOP500
[2] sites.google.com/view/limass
[3] research.cri-paris.org/.../network-ties

3 Preliminary results to be continued by the PhD candidate

3.1 Listing triangles and other motifs

We here focus on listing triangles and show how a relevant order η on the nodes can be used to obtain an efficient algorithm leveraging some specific graphs properties:

Algorithm 1 Listing triangles

```
Let η be a total order on the nodes of the input graph G
function TRIANGLES(G)
    for each node u of G⃗ do                ▷ G⃗: directed version of G according to η
        for each node v in Δ⁺_u do          ▷ Δ⁺_u: out-neighbours of u
            for each node w ∈ Δ⁺_u ∩ Δ⁺_v do
                output triangle {u, v, w}
```

We denote Δ_u^+ the set of out-neighbours of node u in the directed acyclic graph $\vec{G}$, that is the set of neighbours v of u such that $\eta(u) < \eta(v)$.

Algorithm 1 lists all triangles in the input graph. We denote d_u^+ the out-degree of node u in $\vec{G}$, d_u^- its in-degree and d_u its total degree in G. Depending on how the intersections are done and which data-structure is used, the asymptotic running time can be in $\mathcal{O}\big(m + \sum_{u\in V} d_u^+ \cdot d_u^-\big)$, $\mathcal{O}\big(m + \sum_{u\in V}(d_u^+)^2\big)$, $\mathcal{O}\big(m + \sum_{uv\in E} \min(d_u^+, d_v^+)\big)$ or $\mathcal{O}\big(m + \sum_{uv\in E} \min(d_u^+, d_v^-)\big)$. Therefore, the complexity highly depends on the input order η.

Note that if G has degeneracy δ, taking the degeneracy order for η makes all four variations in $O(\delta \cdot m)$, because each node u fulfils $d_u^+ \leq \delta$ and $d_u^- \leq d_u$. While finding the ordering that minimises $\sum_{u\in V} d_u^+ \cdot d_u^-$ is NP-hard [5], the degeneracy ordering enabled us to list all triangles in any real-world graphs we have considered as long as it was fitting in the RAM of the machine at hand, exhibiting the scalability of Algorithm 1.

It is still an open questions whether the sums of $(d_u^+)^2$, $\min(d_u^+, d_v^+)$ or $\min(d_u^+, d_v^-)$ are NP-hard to minimise or even to minimise within a constant factor (which is enough as we evaluate big-$\mathcal{O}$ asymptotic complexities). The PhD student will work on such theoretical questions as well as deriving a better running time for Algorithm 1. A generalisation to other motifs than triangles will be considered, and more advanced properties than the degeneracy will be investigated.

3.2 Graph compression

Although the largest graphs can fit in the main memory of the best supercomputers in adjacency array format, such a machine is rarely at hand. A relevant alternative is to work with compressed structures that allow some specific operations without a full decompression: a graph algorithm does not need the graph itself, but rather the ability to do some specific operations on this graph. Thus, after identifying the operations needed by a given algorithm the input graph can in principle be compressed into a lightweight structure discarding anything unnecessary.

For instance, a web-graph (URLs connected by hypertext links) can be compressed using around 1 bit per link while allowing to iterate over the neighbours of a given node [1]. This is done with a lexicographical order on URLs that leverages two properties of web-graphs: locality (most links are intra-domain) and similarity (nodes in the same domain have very similar neighbourhoods).

However, it is still an open question whether other real-world graphs can be compressed as effectively. Adaptations to social networks exist [4], but the compression rate is capped between 3 and 6, requiring about 10 bits per link. It is also open whether other graph operations can be maintained, such as checking the adjacency of two nodes or adding/removing nodes and edges in a dynamic setting.

4 Application to real-world datasets

The developed algorithms will be used in collaboration with the Network Ties team of Marc Santolini on a large Twitter dataset. It consists of 30 million nodes and 1Tb of temporal interaction data. For further collection, an access to the Decahose[4] (10% of Twitter data since 2012) was granted in the context of a Domaine d'Interêt Majeur from the Ile-de-France region to study the evolution and segregation of digital communities (ENS-CRI). The compression method will be used for data reduction to allow for the visualisation of the evolution of the resulting network[5].

5 Expected results

The thesis will lead to state-of-the-art algorithms for solving important problems in massive real-world networks. Each proposed algorithm will be associated with an open source and publicly available implementation and a scientific publication. We target important applied conferences (such as WWW, IJCAI, KDD, WSDM or AAAI) and also more theoretical ones (such as STOC, FOCS, SODA or ICALP).

References

[1] P. Boldi and S. Vigna. The webgraph framework i: compression techniques. In *WWW*, 2004.

[2] M. Charikar. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 2000.

[3] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 1985.

[4] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD*, 2009.

[5] M. Danisch, O. D. Balalau, and M. Sozio. Listing k-cliques in sparse real-world graphs. In *WWW*, 2018.

[6] M. Danisch, T.-H. H. Chan, and M. Sozio. Large scale density-friendly graph decomposition via convex programming. In *WWW*, 2017.

[7] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita. Compressing graphs and indexes with recursive graph bisection. In *KDD*, 2016.

[8] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in large sparse real-world graphs. *Journal of Experimental Algorithmics (JEA)*, 2013.

[9] M. Latapy. Main-memory triangle computations for very large (sparse (power-law)) graphs. *Theoretical Computer Science*, 2008.

[10] A. Pinar, C. Seshadhri, and V. Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *WWW*, 2017.

[11] R. A. Rossi, D. F. Gleich, A. H. Gebremedhin, and M. M. A. Patwary. Fast maximum clique algorithms for large graphs. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, 2014.

[4] developer.twitter.com/. . . /decahose

[5] Examples in *Large Graph Visualization Tools and Approaches*: towardsdatascience.com