Very Large Graph Project - Clustering and reordering

Sofiene Bourghes[1]

[1]SCIA. Science Cognitive et Informatique Avancée, EPITA, Kremlin-Bicêtre, FR

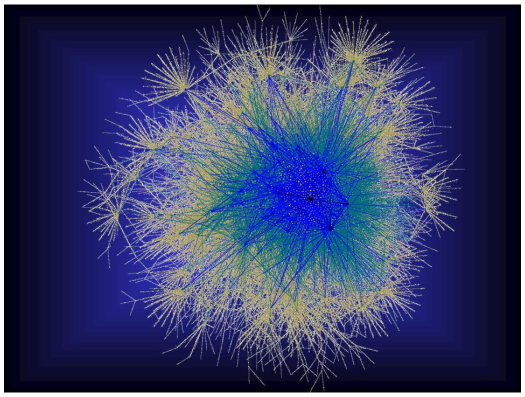

Fig. 1. Internet graphs.

During our Very Large Graphs (VLG) course at EPITA (1), I've made some experiments with clustering and reordering algorithms on sparse very large graphs. The goal of this experiment was to benchmark clustering algorithm with and without reordering the graphs. Is the clustering faster with reordered graph ? Do we found the same amount of community ? What if we choose a random point for reordering, or a more accurate one ?

Very Large Graphs | Experiment | Benchmark | Clustering | Reordering | Louvain | Community

Correspondence: ***sofiene.bourghes@epita.fr***

Introduction. Nowadays, more and more gigantic graphs are starting to emerge in the industry. So big companies like Facebook, Google etc ... are faced with algorithmic problems. Indeed, all methods of graphs that work theoretically are no longer applicable in practice, either due to technical limitations (RAM) or algorithms that simply do not converge. Thus all the polynomial algorithms in $O(n^2)$ or $O(n^3)$ are unusable and must be approximated by algorithms in $O(n)$ or $O(nlog(n)^{1+\epsilon})$. But how defined have a spare very large graphs? G is a graph define by $G = (V, E)$ but such that G has a large number $n = |V|$ of nodes/vertices and $m = |E|$ edges; with m (number of edges) : from some millions to some billions; G is sparse: $m = O(n)$; and as seen in VLG course : it comes from real world dataset.

Clustering without reordering. In order to identify communities inside our sparse very large graphs we will use the Louvain algorithm (2) (https://sites.google.com/site/findcommunities/). This algorithm was developed by Vincent Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre.

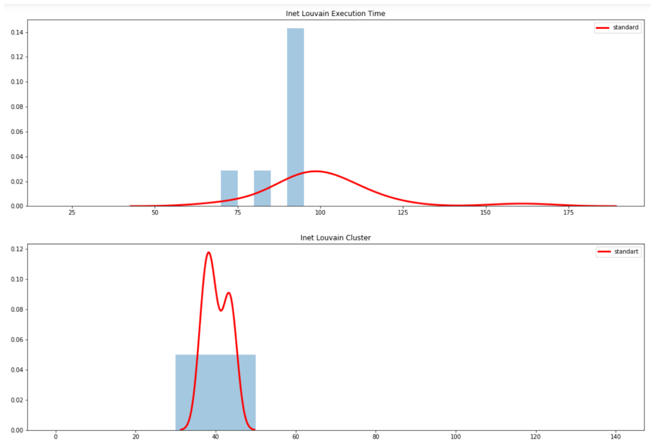

Fig. 2. Inet standard Louvain execution.

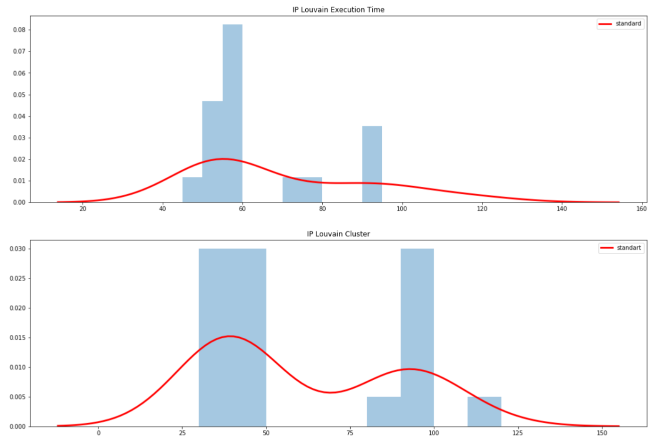

Fig. 3. IP standard Louvain execution.

We will use three sparse very large graphs to execute the Louvain algorithms:

Graph List			
Name	Size	Nodes	Links
Inet	66Mo	1719038	23935356
Ip	83Mo	2250499	39692736
P2P	793Mo	5792298	295442546

Using the Louvain algorithm we found (mean on 20 execution for inet and ip)

Graph List		
Name	Compute Time	Community
Inet	101.55 sec	40.3
Ip	70.1 sec	61.1
P2P	984 sec	106

Looking at inet and ip plot graph, we can't easily see a correlation between Louvain compute time and the resulting clusters (see figure 2 and figure 3).

Reordering. Reordering a sparse very large graph is a task that theoretically speeds up some calculations on this type of graph. Indeed it allows, for example, to have a contiguous graph in memory and to avoid jumps too frequent and important from one memory address to another (quote of his article). We will first use the reordering technique presented in *"Reordering Very Large Graphs for Fun Profit"* (3).

- First we choose a node at random to start our BFS.
- Then we reorder the graph according to the result of the BFS (with our node = 0).

In order to do this reordering, I used and modified the code of Clémence Magnien (https://www-complexnetworks.lip6.fr/ magnien/Diameter) (4) (5):

- Use of its implementation of graphs and queues.
- Modification of its BFS function to return the list of traversed nodes.
- Use and modification of its renumbering implementation.

Then I implemented the reordering algorithm detailed above. Here are the results obtained on the three sparse very large graph studied previously after running Louvain on the reordered graph:

Graph List		
Name	Compute Time	Community
Inet	57.8 sec	89.1
Ip	50.25 sec	96.7
P2P	- sec	-

Is Louvain better with random reordering ?
Looking at inet and ip plot graph, we can clearly see that random reordering improve Louvain execution speed (see figure 4 and figure 5). Random reordering almost divided by two the compute time, but doesn't preserve the same number of clusters. We find almost twice more clusters than standard graph.
For now, we can't say if that's a benefits or an error. In fact it depends on the importance of the number of clusters. But random reordering definitively speed up Louvain execution.

Min and Max degree Reordering. We can now try a more "*determinist*" reordering. We will use the same algorithm explained in the reordering section, except the starting point. Instead of choosing a random node to start our BFS, we search for the node with the maximum/minimum degree.

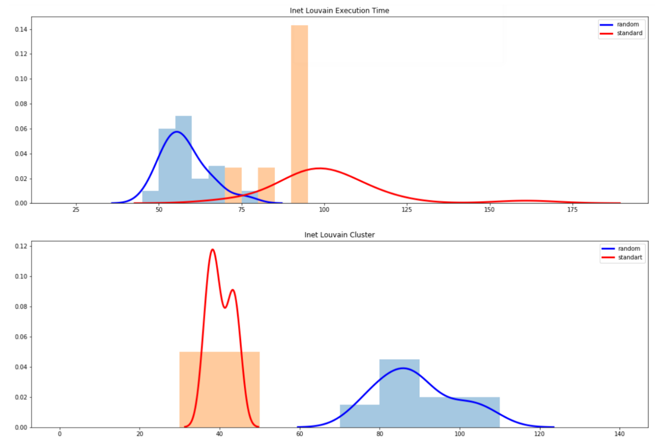

Fig. 4. Inet random reordering Louvain execution.

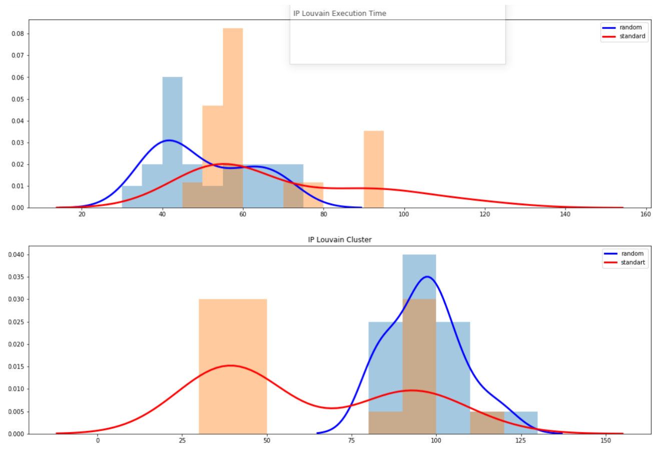

Fig. 5. IP random reordering Louvain execution.

Using the minimum degree we found (mean on 20 execution for inet and ip) :

Graph List		
Name	Compute Time	Community
Inet	15,6 sec	149.3
Ip	65.0 sec	85.3
P2P	- sec	-

And using the maximum degree (mean on 20 execution for inet and ip) :

Graph List		
Name	Compute Time	Community
Inet	60.35 sec	90.3
Ip	52.4 sec	88.65
P2P	- sec	-

Is Louvain better with minimum and maximum reordering ?
First, we can compare min and max reordering with standard Louvain execution. As random reordering, compute the clusters is faster but give different results.
Maximum reordering give more scattered result, with a *"flatter"* histogram. Instead minimum reordering return more *"steady"* result (see figure 6 and figure 7).
Maximum reordering is a bit faster than minimum reordering,

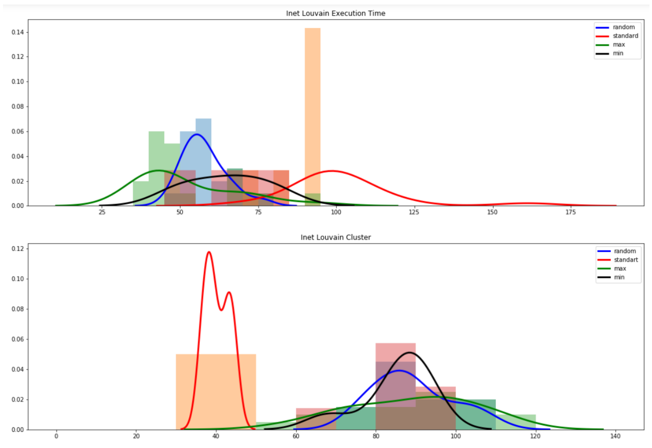

Fig. 6. Inet min/max reordering Louvain execution.

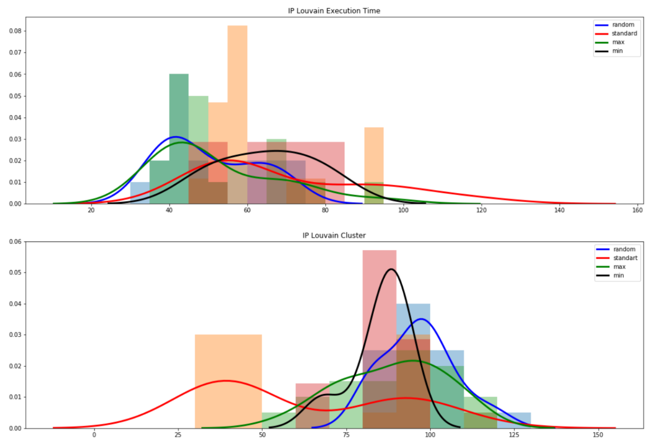

Fig. 7. IP min/max reordering Louvain execution.

but minimum reordering return more clusters.
What about min/max reordering against random ? Looking at inet and ip plot graph, we can not see a clear difference between the different method. (see figure 6 and figure 7). We can't say if using the maximum degree node or the minimum degree node make any real difference, or benefits.

Conclusion. We first tested Louvin without reordering, the results thus obtained were used as a reference. After defining and implementing our reordering algorithm, we performed tests to compare with the reference. This study allowed us to establish that reordering makes it possible to accelerate the calculation of communities, but the number of communities found is greater. This work has allowed us to rule out the hypothesis that reordering only affects the speed of calculation. A question then emerges: do we use the best reordering method? To try to answer it, we implemented a reordering based on the highest degree node and the smaller degree node. The tests carried out allowed us to establish that the starting node does not allow a significant improvement of the results. However, it remains very difficult to evaluate the results obtained. Indeed, a shorter calculation time is often appreciated, however if it affects the number of clusters obtained by the algorithm it is not necessarily an improvement. In the future, we could look at other re-scaling implementations such as choosing the largest related component, or modify the BFS to choose the highest or lowest node.

Annex. The tests are performed from the script provided in the project sources. Each result is obtained after 20 iterations of Louvain, in order to obtain a distribution and an exploitable average. Following the execution time and the amount of RAM taken by the P2P graph, we were able to execute it only once. Each plot represents the data obtained in the form of two histograms: distribution of the execution times and distribution of the obtained cluster numbers.
All tests in this article were performed with a PC having the following features Windows 10 - WSL Ubuntu, I7-7700HQ, 16 GB RAM. The WSL Ubuntu still performs poorly on IO operations, so computation times can be sometimes significant.

Bibliography

1. Robert Erra. Very large graph. In *EPITA - SCIA*, 2019.
2. Renaud Lambiotte Etienne Lefebvre Vincent D. Blondel, Jean-Loup Guillaume. Fast unfolding of communities in large networks. 2008.
3. Robert Erra Lionel Auroux, Marwann Burelle. Reordering very large graphs for fun profit. 2015.
4. Matthieu Latapy Clemence Magnien. Diameter of massive graphs source code. In *http://www-rp.lip6.fr/ magnien/Diameter*, 2007.
5. Fast computation of empirically tight bounds for the diameter of massive graphs.