Reordering Very Large Graphs for Fun & Profit

Lionel Auroux
LSE - EPITA
lionel.auroux@lse.epita.fr

& Marwann Burelle
LSE - EPITA
marwan.burelle@lse.epita.fr

& Robert ERRA
LSE - EPITA
robert.erra@lse.epita.fr

Abstract

We have made experiments with ***reordering algorithms*** *on sparse very large graphs (VLGs), we have considered only undirected, unweighted, sparse and huge graphs, i.e. $G = (V, E)$ with $n = |V|$ from million to billion of nodes and with $|E| = O(|V|)$. The problem of reordering a matrix to enhance the computation time (and sometimes the memory) is traditional in numerical algorithms but we focus on this short paper on results obtained for the approximate computation of the diameter of a sparse VLG (with some graphs on various different computers). The problem of reordering a graph has already been pointed, explicitly or implicitly by a lot of people, from the numerical community but also from the graph community, like the authors of the Louvain algorithm when they write that choosing an order is thus worth studying since it could accelerate the Louvain algorithm. Our experimental results show clearly that it can be worth (and simple) to preprocess a sparse VLG with a reordering algorithm.*

I. Introduction

Let G be a graph, $G = (V, E)$, with $n = |V|$ nodes and $m = |E|$ edges, we will say it is a very large graph (VLG) if n is very large, or huge, *i.e.* G has at least millions (from some to billions) of vertices and edges, we will say that G is sparse if $|E| = O(n)$ and we consider mainly graphs that come from real life, *i.e.* graphs that comes from a real-world large or huge dataset. We will limit our analysis to graphs that fit in the RAM.

For two isomorphic graphs G and G', will any (polynomial deterministic or not) algorithm on graphs give exactly the same result ? Theoretically this is true. Practically, for very large graphs, sparse or not, but with hundred of millions or billions of edges, this is not true, at least for the cpu time.

The problem of reordering a graph has already been pointed in [4], the authors point that choosing an order is thus worth studying since it could enhance the computation time of the Louvain algorithm because it is a greedy algorithm. We propose here to explore experimentally a such simple reordering algorithm as a preconditionner for very large graphs problems. The full version of this paper will contain the detailed results we have obtained for different graphs, problems and computer architectures. We focus on this short paper on results obtained for the approximate computation of the diameter of a sparse VLG. This problem is time polynomial and in a lot of computer science courses (and textbooks), time polynomial algorithms are presented as fast, by opposition with exponential or subexponential algorithms. This is theoretically true, but for sparse VLGs of billions of edges known exact algorithms can not be used because their time complexity is above the $O(n)$ complexity that acts as a real *wall.* On very large problems we can only use approximate algorithms that have a linear complexity (or in some cases a linearithmic complexity $O(n \log(n))$), and $O(n^2)$ or $O(n^3)$ algorithms are practically useless. Experimental results presented here show that it can be worth (and simple) to preprocess a sparse VLG with a reordering algorithm.

II. Reordering algorithms

I. A little bit of history

The story of reordering algorithms seems to begin with the landmark paper of Cuthill and McKee [5], where the authors explain that an efficient reordering for a sparse matrix is related to the labeling (or reordering) of an undirected graph. This paper has been followed by a lot of algorithms, one of them is the GPS [9] another is the Sloan algorithm, generally a more efficient and faster algorithm [14, 15].

Let $V = \{1, \cdots n\}$ the n vertices of a graph $G = (V, E)$, a *linear ordering*, or reordering of V is a one-to-one function $f : V \to V$ [7]. The Bandwidth Problem for G is to find a reordering f to minimize the bandwidth $\beta(G, f)$ defined as:

$$\beta(G, f) = \max_{(i,j)\in E(G)} |f(i) - f(j)|.$$

For a symmetric matrix A the corresponding problem is to find a permutation matrix P such that $A' = P.A.P^T$ with $\max_{i,j\in\{1,\cdots n\}, A'(i,j)\neq 0} |i-j|$ minimum. We want to minimize the bandwidth by simultaneous row and column permutation. They are other similar problems [8]: the profile problem, the wavefront problem and the fill-in Problem (we want to minimize the fill-in during the numerical Gauss or Cholesky factorization of a sparse and large matrix by reordering it).

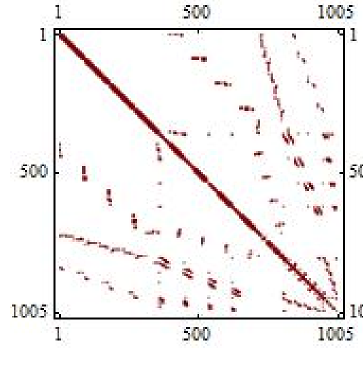

Figure 1: *The adjacency matrix DWT_1005*

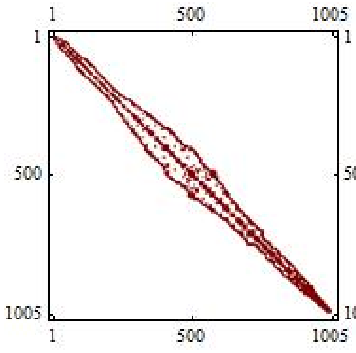

Figure 2: *The matrix DWT_1005, reordered with the Reverse Cuthill-McKee algorithm*

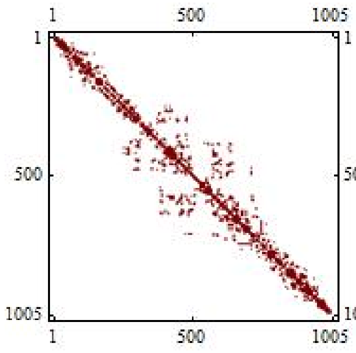

Figure 3: *The matrix DWT_1005, reordered with the Sloan algorithm*

One wants to minimize the *bandwidth*,*the wavefront*, or *the profile* of a matrix, or by extension **of a graph** [5, 14, 15].

The problem of finding a minimum bandwidth reordering is a NP-complete problem [7] so what is a good reordering for a graph or a matrix ? It is a polynomial algorithm that gives a reordering of your graph such that this "new" graph is easier to manipulate for some graph problems. Let us give an example. The figure 1 shows the adjacency matrix DWT_1005 from the Everstine's collection (1011 vertices, 7716 edges. The figures 2 and 3 show the adjacency matrices reordered, we can see that the bandwidth is just better, they have been found, respectively, with the Cuthill-McKee and the Sloan reordering algorithms. The figure 4 shows a toy graph example and the 5 shows the reordered graph.

II. Stategies and tactics to reorder

But how can we reorder a matrix ? Almost all known reordering algorithms for matrices follow the same strategy:

1. Find w as one of the nodes of a pseudodiameter (or hopefully diameter), as proposed in [5, 14, 11, 13].
2. Compute a rooted level structure from w: $L(w) = \{L_0, L_1, \cdots L_r\}$.
3. Renumber the nodes using $L(w)$ with w as node 0.

The rooted Level structure is very similar to a BFS tree obtained by a BFS traversal of the graph. The algorithms Cuthill McKee, Sloan and the others are approximation algorithms and are polynomial:

- Cuthill McKee [5] has a complexity $O(M \log(M)|V|)$,
- Sloan [14, 15] has a complexity of $O(\log(M)|E|)$.

with $M = \max_i(\deg(v_i))$. The results depend of course highly of the tactics to choose the vertex w and the way to compute the $L(w)$.

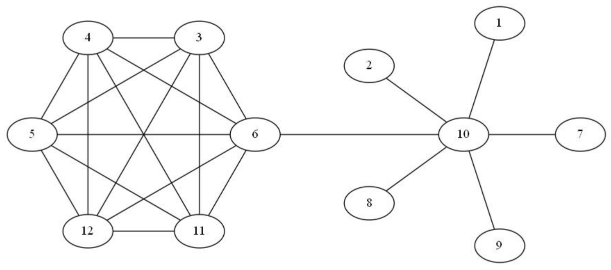

Figure 4: *Original toy graph example*

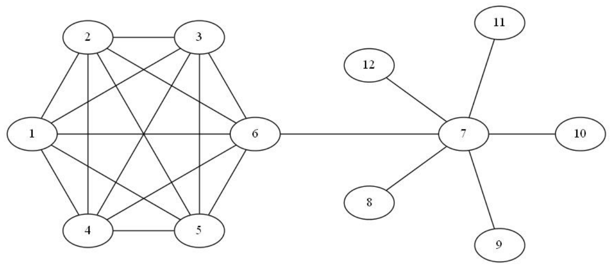

Figure 5: *Reordered toy graph*

III. Our reordering algorithm

Some experiments with Boost Graph Library (BGL) implementation of Cuthill-MCkee and Sloan algorithm have shown that these algorithms were slow on the real sparse very large graphs we have tested. We wanted a fast and simple algorithm, with a linear complexity, to respect the $O(n)$ wall, so we have used a very simple, but fast algorithm: just pick a random point w and compute the BFS from w and use the BFS tree to renumber the graph vertices. This gives the following algorithm:

1. Choose a node w of the graph G and compute the BFS of the graph with w as the root.
2. Renumber the nodes following the BFS traversal with w node 0.

The complexity of this algorithm is just the linear complexity of a BFS traversal, so it is $O(|E| + |V|)$

for a sparse VLG and it is generally really faster than Cuthill-McKee and Sloan algorithms on VLGs. Typically, for the approximate diameter computation via BFS traversals, a reordering costs approximately more than a BFS computation but less than two BFS computations so it can be used without problems on sparse VLGs.

III. Why does reordering algorithms works so well ?

The memory hierarchy of a computer system can be very complex. Between the classical RAM memory and the registers of the processor there can be different cache levels of memory. It seems that, at least for the approximate diameter computation the main reason that explains why the reordering algorithm works so well is that it minimizes cache misses. A lot of papers have presented some ideas similar to our main idea, implicitly or explicitly : to reorder a huge date structure can be a good idea to save time or to accelerate an algorithm. For example, Mueller and al. [12] compares different reordering algorithms for large graph visualization, and Apostolico and Drovandi [1] have shown the interest to reorder a VLG (by a BFS) to have a better compression ratio.

All these examples are concerned by the *locality principle*, an expression coined by D.J. Denning in the years 60, he has written [6] *the locality principle will be useful wherever there is an advantage in reducing the apparent distance from a process to the objects it accesses.* And reordering a graph (or any huge structure) seems to augments the data locality.

I. Methodology and some results

We have renumbered some VLGs and we have compared time with two inputs: the initial graphs and the renumbered graphs obtained by our linear reordering algorithm, on different computers.

For the approximate diameter computation we have observed an acceleration **from 10% to 40%** (using the software diam.c [10], whithout any change, from authors of [11]. The acceleration seems to highly depend on the hardware and notably to depend thoroughly on the *memory hierarchy*, we have observed a lower number of cache misses (measured with perf) which seems to be the main reason.

To try to understand what really happens we have also written our own program to compute the approximate diameter of a sparse VLG, and the results are similar, as we will see now.

II. Some Experiments with the perf linux command

We have made some measures with the `perf` linux command, which is a powerful profiler tool for Linux 2.6+ based systems. We have used a computer with a dual core AMD opteron 2220, at 2.8 GHz, with 16 Gb of memory, 48 Kb of L1 cache, 1 Mb of L2 cache and 16 Mb of L3 cache memory. We have used our own program for the approximate computation of the diameter, but the results are similar to those obtained with diam.c [10].

II.1 The (small) graph 3elt

It seems that the graph has not to be too small to see an improvement, for example for the graph 3elt with 4720 vertices, 13722 edges (indeed a very small graph) the following table shows no amelioration (ABFS means *average time for a BFS* traversal) except for the cache misses:

Graph	Initial	Reordered
Loading time	0. 0190s	0. 0266s
Cycles	241,948,601	200,120, 628
Cache misses	467,126	204,880
ABFS	$4,8\ 10^{-3}$s	$5,2\ 10^{-3}$s

II.2 The p2p graph

For this graph with 5,792,297 vertices and 142,038,401 edges; the results show a smaller loading time, a smaller ABFS and a dramatic decrease on the number cache misses:

Graph	Initial	Reordered
Loading time	112.24s	102.94s
Cycles	$782\ 10^{9}$	$667\ 10^{9}$
Cache misses	919,506,845	200,823,282
ABFS	23.45s	19.08s

II.3 The web graph

And for the huge graph `web` with 39,459,925 vertices and 783,027,125 edges; the results show again a smaller loading time, a smaller ABFS and, a large decrease on the number cache misses:

Graph	Initial	Reordered
Loading time	420.47s	387.21s
Cycles	$13,201\ 10^{9}$	$10,604\ 10^{9}$
Cache misses	9,358,538,202	5,028,589, 731
ABFS	65.03s	50.69s

IV. Conclusion and future work

We need more experiments to better understand and explain the results we have observed. And some interesting research questions have raised:

- Is it possible to find a specialized reordering algorithm, with a linear complexity, for a specific task on a targeted computer architecture ?
- How can we reorder efficiently a very large *directed* graph (and how) ?

- Is it interesting to find a reordering algorithm for weighted graphs (and how to do it) ?
- A lot of sparse VLGs are real-world data and are generally not static : could we define an *incremental* reordering algorithms for dynamic graph ?
- If a VLG, sparse or not, does not fit in the RAM, can we design an efficient external memory reordering algorithm ?
- Is it worth to reorder a VLG for other problems, possibly with a specific reordering algorithm, for example: the graph isomorphism and the subgraph isomorphism problems for a VLG, or finding, listing and counting all triangles in a VLG (sparse or not) ?
- Our experiments for the computation of communities via the Louvain algorithm show an acceleration **from 20% to 50%**, using the software [2, 3], without any change, from authors of the Louvain algorithm [4]. It seems that a lower number of cache misses is not the main reason, the Louvain algorithm is a greedy algorithm and it seems that with the reordered graphs the algorithm just works better (as proposed by the authors themselves). Can we find a specialized reordering algorithm to accelerate the computation of communities?

But our results are clear: they show it can be really worth to use a reordering algorithm as a preconditionner for sparse VLGs so, more research need to be done on the problems we have presented.

References

[1] A. Apostolico and G. Drovandi. Graph compression by BFS. *Algorithms*, 2(3):1031–1044, 2009.

[2] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. https://perso.uclouvain.be/vincent.blondel/research/louvain.html.

[3] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. https://sites.google.com/site/findcommunities/.

[4] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[5] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proceedings of the 1969 24th National Conference*, ACM '69, pages 157–172. ACM, 1969.

[6] P.J. Denning. The Locality Principle. *Communications of the ACM*, 48(7):19–24, dcembre 2005.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979.

[8] A. George and J. W. Liu. *Computer Solution of Large Sparse Positive Definite.* Prentice Hall Professional Technical Reference, 1981.

[9] N. E. Gibbs, W. G. Poole, Jr., and Paul K. Stockmeyer. A comparison of several bandwidth and profile reduction algorithms. *ACM Trans. Math. Softw.*, 2(4):322–330, December 1976.

[10] C. Magnien. http://www-rp.lip6.fr/ magnien/Diameter/.

[11] C. Magnien, M. Latapy, and M. Habib. Fast computation of empirically tight bounds for the diameter of massive graphs. *ACM Journal of Experimental Algorithmics*, 13, 2008.

[12] C. Mueller, B. Martin, and A. Lumsdaine. A comparison of vertex ordering algorithms for large graph visualization. In *APVIS 2007, 6th International Asia-Pacific Symposium on Visualization 2007, Sydney, Australia, 5-7 February 2007*, pages 141–148, 2007.

[13] J. K. Reid and J. A. Scott. Ordering symmetric sparse matrices for small profile and wavefront. *Int. J. for Num. Methods in Eng.*, 45(12):1737–1755, 1999.

[14] S. W. Sloan. An algorithm for profile and wavefront reduction of sparse matrices. *International Journal for Numerical Methods in Engineering*, 23(2):239–251, 1986.

[15] S. W. Sloan. A fortran program for profile and wavefront reduction. *International Journal for Numerical Methods in Engineering*, 28(11):2651–2679, 1989.