(sparse) Very Large Graphs

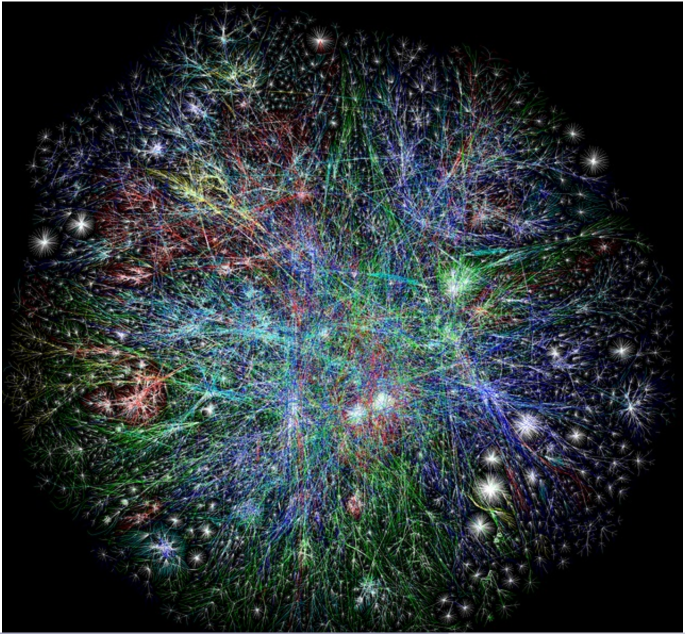
A Very New Kind of (Real) Graphs:

Some Problems & and Some Algorithms

Robert Erra & Alexandre Letois & Mark Angoustures
Projet ING 2 Majeure SCIA 2021

19 mai 2021

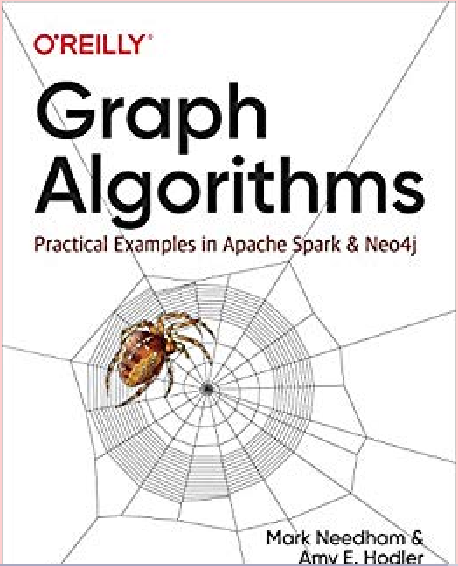
An example of a (s)VLG (Ref [00]) : a partial view of Internet



Version β_0

1. Ces notes sont distribuées telles quelles et sont destinées à évoluer.
2. Chaque nouvelle version sera mise Ãă disposition lorsque le nombre de changements sera significatif.
3. Toute remarque, toute proposition, tout signalement d'erreur, toute idée : n'hésitez pas : verylargegraphs@gmail.com (ou Teams).
4. Par avance merci.

The Rising of Graphs (or Graphs are on the Rise)



Work to do ?

1. 12 hours : 4 "weeks"
2. Week 1 : 3 hours of presentation
3. Weeks 2, 3 and 4 : 9 hours of Laboratory
4. You will have to choose a project (Week 2).

A (Sparse) Very Large Graph [(s)VLG] ? ...

...is a graph, $G = (V, E)$, 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* : $m = O(n)$;
- (we can add this) : it comes from a real-world dataset.

And we can add : these (s)VLGs are generally not static, there are *living* : they evolve dynamically.

So ...

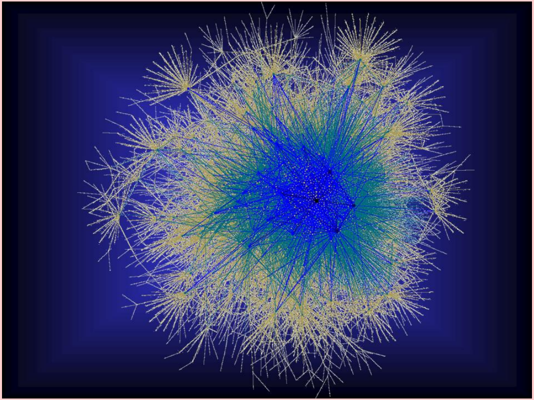
A (Sparse) VLG is *a sparse, large, (possibly) time-evolving graph* that comes from real-world.

Plan

❷ Graphs are on the Rise !

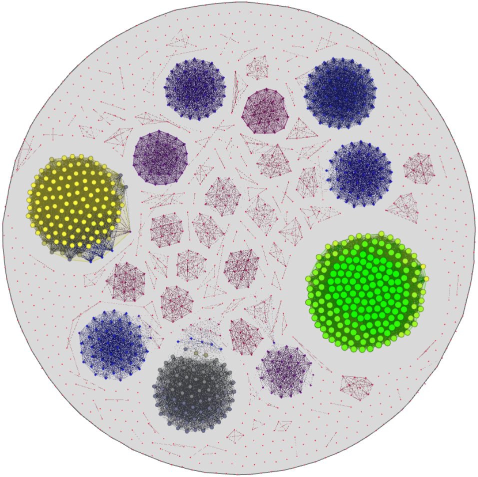
Real-World Data : From [1]

A graph of the BGP (Gateway Protocol) web graph, consisting of major Internet routers (6400 vertices/13000 edges)



http ://blog.zynamics.com/category/vxclass/

. . . where we explored some of the idiosyncrasies of mass-malware vs. the ones in targeted malware . . . a reasonably diverse corpus of recent malware obtained from VirusTotal.

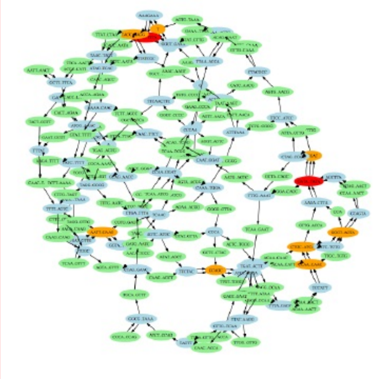


Real-World Data : From [2]

…On a coarse-grained level, the statistics of bio-molecular networks can be studied at the level of the network topology (*i.e.*, the ensemble of its nodes and links).

Real-World Data : From [3]

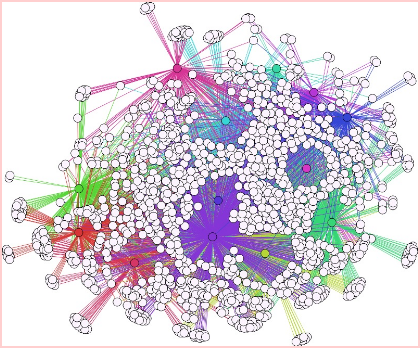
protein-protein and protein-DNA interaction graphs. Topics include predicting protein function from protein interaction networks, comparing interaction networks from multiple organisms, finding common network motifs, inferring missing edges, and visualizing



biological graphs.

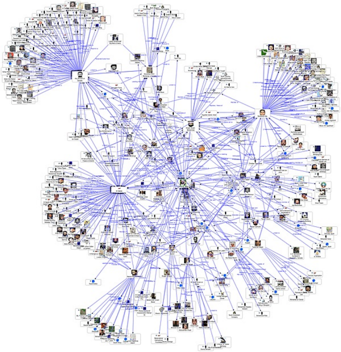
Real-World Data : From [4]

In the *Social Epidemics, Online Social Networks, and Graphs* course, students learn about graph theory, statistics and computational data analysis by studying their own online social network data.



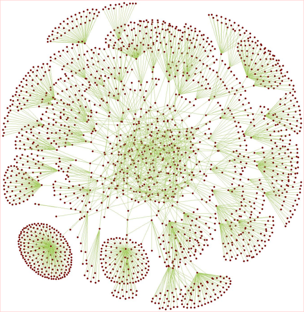
Real-World Data : From [5]

From a visual standpoint, some clusters and centrality are visible. But the density of information makes it difficult to see all the centrality aspects. . . .and the program instantly calculates the Social Network Analysis Metrics for the items on the chart.



Real-World Data : From [6]

...a map of the Internet generated by new algorithms ...at UC San Diego. [...] But it is no ordinary map. It is a (mostly) randomly generated graph that retains the essential characteristics of a specific corner of the Internet but doubles the number of nodes.

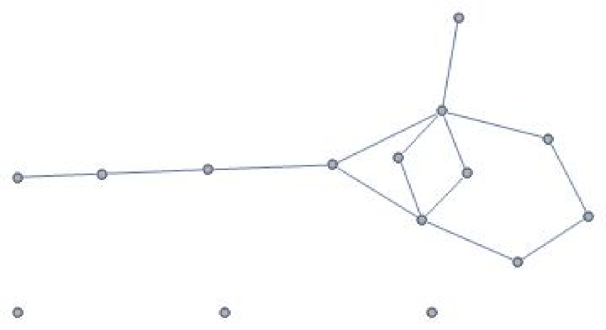


3 Basics definitions

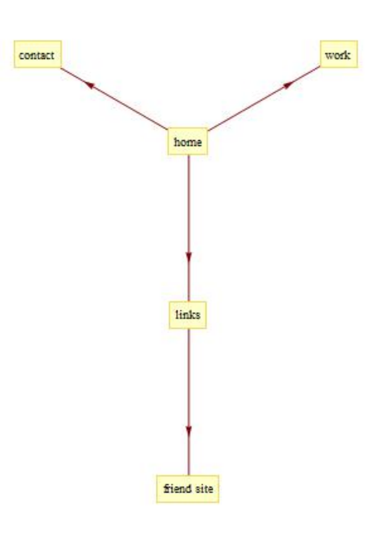
A Graph ?

- $G = (V, E)$
- V : set of nodes/vertices, $|V| = n$
- E : set of edges, $|E| = m$; $E \subset \{1, \cdots n\} \times \{1, \cdots n\}$
- G can be undirected : $(i, j) \in E \Rightarrow (j, i) \in E$
- or G can directed : $(i, j) \in E \not\Rightarrow (j, i) \in E$
- A simple graph : $(i, i) \notin E \Rightarrow$, no loops ; and no multiple edges
- A node or an edge can have attributes (value)
- The degree of a node is the number of its adjacent nodes.

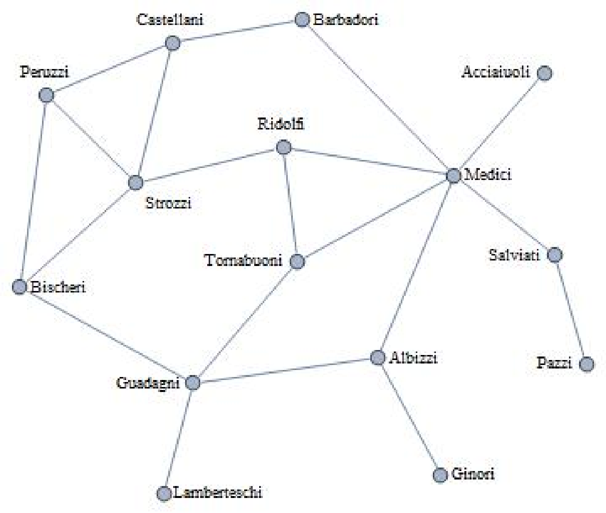
An undirected Graph of 15 nodes



A directed Graph

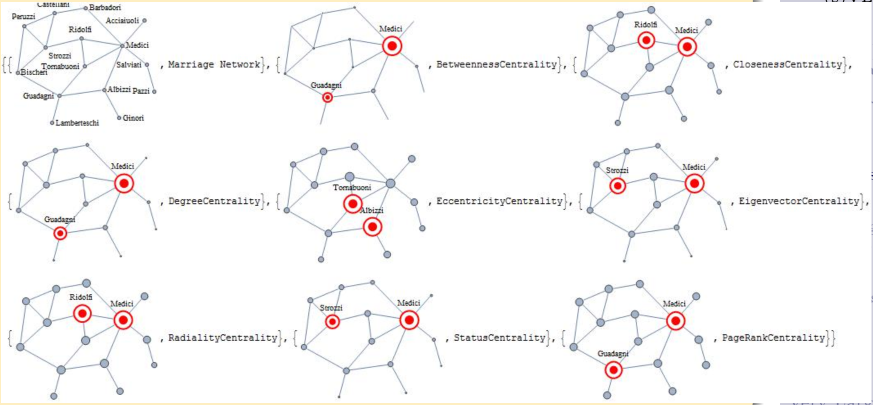


A real (small) example : The Medici graph



(sparse) Very Large Graphs [(s)VLG]

A real (small) example : some results about *node centralities* of the Medici graph



Walks, trails, paths

1. A walk is an alternate sequence of vertices and edges, example : $\{v_2, e_{23}, v_3, e_{37}, v_7\}$
2. The length of a walk is the number of edges
3. A trail is a walk such that no edge occurs more than once
4. A path is a trail such that no (internal) vertex occurs more than once
5. The length of a walk/trail/path is the number of its edges
6. The distance between i and j is the length of the shortest walk between them.

4. Some interesting (real) Problems

Some interesting (real) problems

1. PageRank : the (famous) ranking algorithm from Google : polynomial (quasi-linear) in floating point arithmetic (so considered as a fast algorithm if the large graph is sparse)
2. Diameter and Center of Graphs : polynomial exact algorithms are impracticable [$O(n^3)$ or $O(nm)$] : develop fast algorithms [$O(n)$ or $O(n\log(n))$]
3. Community : NP-complete (so no polynomial algorithm known)
4. A difficult but very interesting problem : compute the approximate diameter and radius/center of the giant component of the graph of the web !

(s)VLG : source of graphs

- Graphs from the web
- Graphs from social networks
- Graphs in biological databases
- CFG of large executable graphs
- etc.
- New discipline : *Graph Database Problems*
- SNAP (Stanford Network Analysis Project) : http ://snap.stanford.edu/ with a lot of graphs at http ://snap.stanford.edu/data/index.html

5 Basics algorithms

Some classical graph algorithms

- BFS : Breadth First search Search (polynomial algorithm)
- DFS : Depth First Search (polynomial algorithm)
- Diameter : the longest shortest path between two nodes (polynomial algorithm)
- Diameter : it is also the maximum of all eccentricities
- Radius : it is the minimum of all eccentricities
- The All Pairs Shortest Paths (APSP) problem : we want all distances between each couple of nodes (polynomial algorithm).

6. A famous algorithm : Pagerank

The "web graph" ...

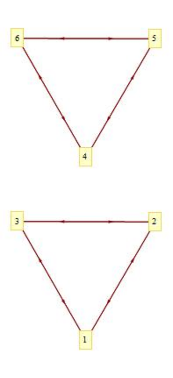
1. is huge (probably today $> 6 \times 10^{10}$ nodes)
2. is moving /changing (dynamical algorithms or incremental algorithms are needed)
3. is not semantic : just hyperlinks !
4. has a huge giant strong connected component with a lot of "islands"
5. is directed (or oriented) but not weighted (unweighted).

PageRank : The Perron-Frobenius Theorem

Let $A \in \mathbb{R}^n x \mathbb{R}^n$ be a positive, column stochastic matrix, *i.e.* each column has a sum=1, then :

- 1 is an eigenvalue of multiplicity one.
- 1 is the largest eigenvalue : all the other eigenvalues are in modulus smaller than 1.
- the eigenvector corresponding v to eigenvalue 1 has all entries positive : $v = (v_1, \cdots v_n) with v_i > 0$.
- In particular, for the eigenvalue 1 there exists a unique eigenvector v^* with the sum of its entries equal to 1 : $A.v^* = v^*$.

Problem with disconnected graphs



PageRank : The Brin & Page's Algorithm

Define the Page Rank matrix (aka the Google matrix) :

$$M = (1-p)A + p.B$$

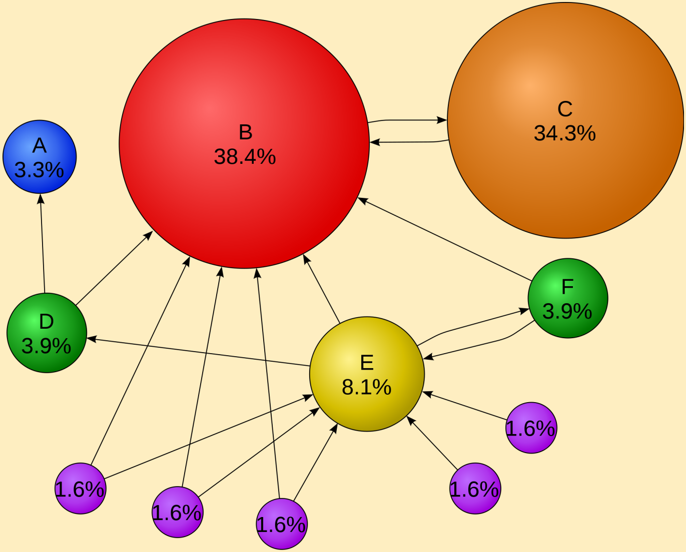
- A : the transition/adjacent matrix of the " full web graph"
- $p \in]0,1[$: a constant, the famous *damping factor* (typical value : 0.15)
- $B = C/n$, with $C(i,j) = 1$ for all i,j
- M remains a column stochastic matrix
- M has only positive entries.

The PageRank vector is the **dominant right eigenvector of M**. We can compute it with the *power algorithm.*

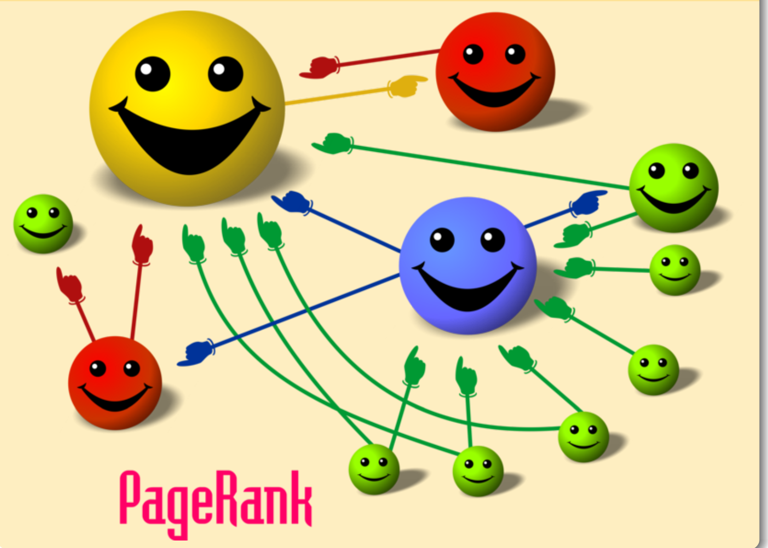
So is it quite easy to do a PageRank algorithm ?

- For a large and dense graph : No. Why ?
- If A is a $(10^{10}, 10^{10})$ dense matrix you need $O(10^{20})$ to store A : IMPOSSIBLE
- So : if the graph is sparse, we store a "compressed" representation of A.
- Use the fact that A is sparse : a list of lists (list of adjacent nodes for each node)
- So : for a sparse Very Large Graph : PageRank is considered as an "easy" algorithm ...

A toy example



A funny cartoon view example (Ref Wikipedia)



Be careful with really *Big Data* : **Small Powers always Win**

For a very large set of data, for example with more than 10^6, we can not use, even if there exists, polynomial algorithms of $O(n^{1+\epsilon})$.
For $n = 10^9$

1. and $\epsilon = 0.1$ we have $O(n^{1+\epsilon}) = n^{1.1} = 7.94328 \times 10^9$
2. and $\epsilon = 0.5$ we have $O(n^{1+\epsilon}) = n^{1.5} = 3.16228 \times 10^{13}$
3. and $\epsilon = 1$ we have $O(n^{1+\epsilon}) = n^2 = 1. \times 10^{18}$
4. and $\epsilon = 1.5$ we have $O(n^{1+\epsilon}) = n^{2.5} = 3.16228 \times 10^{22}$
5. and $\epsilon = 2$ we have $O(n^{1+\epsilon}) = n^3 = 1. \times 10^{27}$! ! ! ! ! ! ! ! !

(3) is expensive, (4) and (5) out of reach ! ! ! A complexity of $O(n)$ or $O(n\log(n)^{1+\epsilon})$ is a real wall.

Be careful with really *Big Data* : there is a practical but real wall

1. Any (classical) polynomial algorithms in $O(n^2)$ or $O(n^3)$ is useless !
2. Any (classical) polynomial algorithms in $O(n^2)$ or $O(n^3)$ have to be approximated by a $O(n)$ algorithm (or in some cases $O(n\log(n)^{1+\epsilon})$)
3. The $O(n)$ (or $O(n\log(n)^{1+\epsilon})$) is a true Wall !

An example

1. Let G has $n = 10^8$ nodes
2. So $n^3 = 10^{24}$
3. With a computer doing 10^{10} FLOPS
4. A $O(n^3)$ algorithm will take 3.17098×10^6 years !

8 How to compute the diameter of a (s)VLG

Diameter & Radius & Center

- Eccentricity of a vertex $v \in V$: maximum graph distance between v and all other vertices of G.
- For a disconnected graph, all vertices are defined to have infinite eccentricity.
- Eccentricity of v : $Ecc(v) = \max_{y \in V} d(v, y)$
- *Maximum eccentricity* : the **graph diameter**. [MAX MAX MIN]
- $\text{Diameter}(G) = \max_{x \in V} Ecc(x) = \max_{x,y \in V} d(x, y)$.
- *Minimum eccentricity* : the **graph radius**. [MIN MAX MIN]
- Radius $R(G) = \min_{x \in V} Ecc(x)$
- Center : the **set** of vertices with eccentricity equal to the graph's radius.

Diameter

- Naive but exact algorithms examine each vertex in turn and performs a BFS starting at the chosen vertex.
- Such a sweep starting at vertex x immediately determines $Ecc(x)$.
- But exact algorithm have time complexity in $O(n^3)$
- For a sparse graph : $O(n\,m)$ if $m << n^2$.
- Too expensive for (s)VLGs [Do not forget $m = O(n)$]
- There exists $O(n^\omega)$ algorithms with $\omega \in]2, 3[$ but not enough "practical" (based on fast matrix multiplication algorithms).

Diameter

- For a NP-complete or NP-hard problems we compute an *approximate* solution
- Such algorithms are called approximation algorithm
- (Approximation algorithms are algorithms used to find approximate solutions to NP-hard optimization problems)
- We will do the same for all problems with a (s)VLG
- We will search for a not too expensive algorithm for (s)VLGs
- The ideal : a linear algorithm that gives a very good approximation.

9 Approximation algorithms for the Diameter Problem

Basic ideas/1 : For $x \in V$ then

- $Ecc(x) \leq D(G) \leq 2Ecc(x)$
- Proof of the lower bound : $Ecc(x) \leq D(G)$: obvious.
- Proof of the upper bound $D(G) \leq 2Ecc(x)$:
  1. Take an internal vertex z between x and y such that $Ecc(x) = (x, y)$
  2. Then $Ecc(x) = d(x, y) \leq d(x, z) + d(z, y) \leq 2Ecc(x)$.

Idea :

1. Use a small set of vertices to refine the bounds, we will have : $e_L \leq D \leq e_U$.
2. If $e_L = e_U$ then we have found the true value $D(G) = e_L = e_U$!

We will call this type of algorithms **self-checking algorithms**.

A toy example : How to use the inequality ...

$$\forall x \in V : Ecc(x) \leq D(G) \leq 2Ecc(x)?$$

- Take $u_1 \in V$.
- Compute $e_1 = Ecc(u_1)$ (with a BFS traversal)
- So we first have : $R \leq e_1 \leq D \leq 2e_1$
- Now Take $u_2 \in V$, different from u_1 of course.
- Compute $e_2 = Ecc(u_2)$ (with a BFS traversal)
- So we have $R \leq \min(e_1, e_2) \leq \max(e_1, e_2) \leq D \leq 2\min(e_1, e_2)$

More generally : we will have for k points/BFS :

$$R \leq \min(e_1, \cdots e_k) \leq \max(e_1, \cdots e_k) \leq D \leq 2\min(e_1, \cdots e_k)$$

Basic ideas/2 : a first strategy

- Take $u \in V$.
- Compute v such that $d(u,v) = Ecc(u)$ with a BFS traversal
- Compute w such that $d(v,w) = Ecc(v)$ with a BFS traversal
- This is called the *Magic Double Sweep algorithm.*
- $+$ Ref [23] : iterate and refine the lower and the upper bounds
- $++$ Ref [23] : use upper bounds given by spanning tree (which is a subgraphs of G)
- Of course, again, if the lower bounds and the the upper bound are equal : we have **found** the true diameter.

Basic ideas/3

The Double Sweep algorithm is a special case of a more general algorithm.

Strategy :

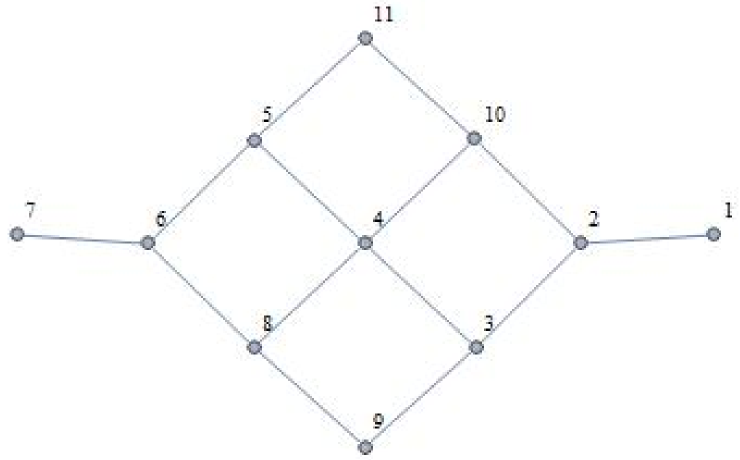
- Take $u \in V$.
- Compute the set F of vertices v such that $v \in F : d(u,v) = Ecc(u)$
- Iterate and refine the lower and the upper bounds

Tactic :

- Choose another $v \in F$.
- Compute again the set F of vertices w such that $w \in F : d(v,w) = Ecc(w)$
- Iterate and refine the lower and the upper bounds

and so on ...

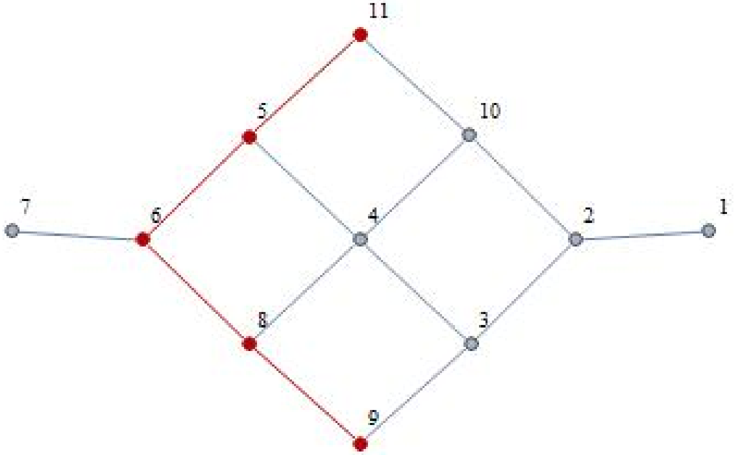
The Shuriken Graph :



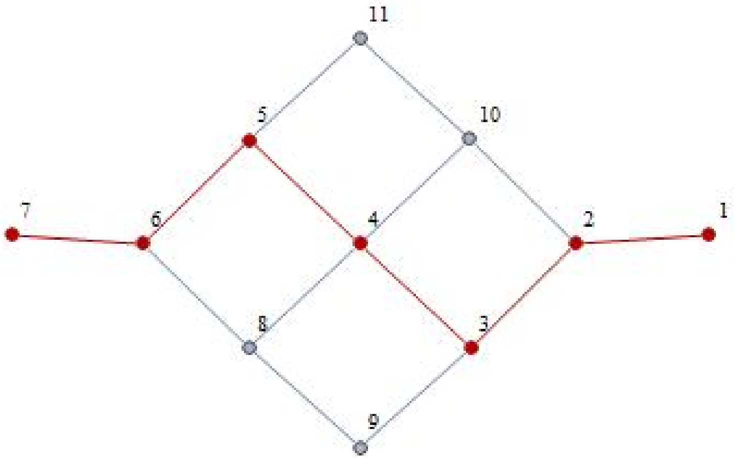
An example : the Shuriken Graph (see previous slide)

- Let $u = 11$ and do a BFS : then $v = 9$ with $d(u, v) = 4$.
- Do a second BFS : then $w = 11 = u$ and so again : 4.
- But $D(G) = 6$ with $D(G) = d(1, 7) = 6$.
- A conjecture (you don't have to test it) :
  1. If $BFS((BFS(u)) = u$ with $v = Last(BFS(u))$ but $D(G) \neq d(u, v)$ and the diameter is not given by a BFS on one of the nodes of the last level then the diameter is given
     - by a BFS on one of the nodes of the last level (if v is not unique).
     - or, by a BFS on one of the nodes of one of the last but one level, or not so far.

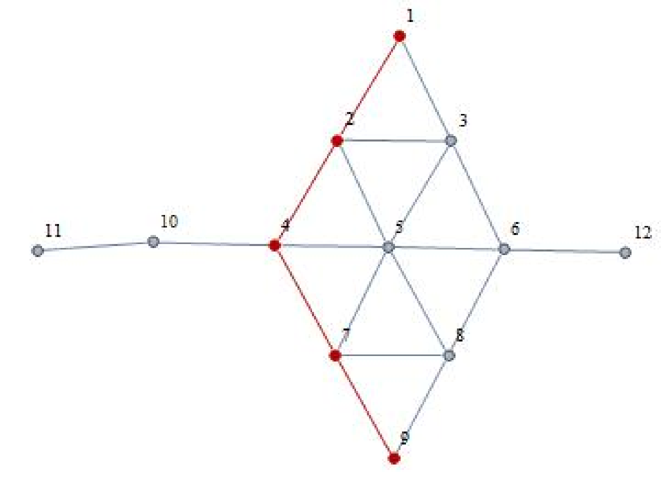
The Shuriken Graph : BFS from 11 to 9 and BFS from 9 to 11 : unfortunately length 4



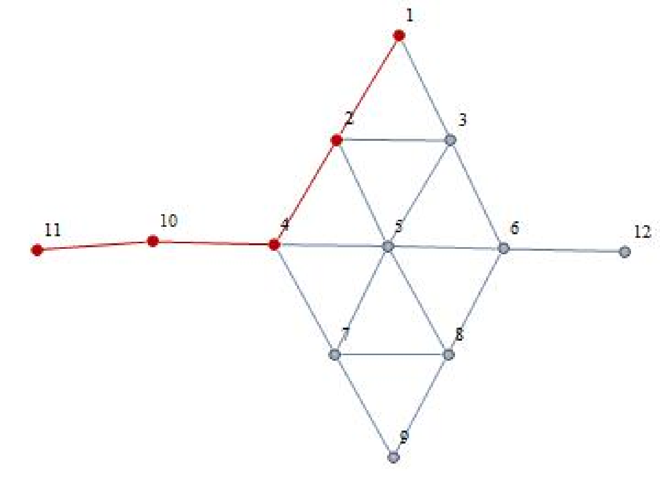
The Shuriken Graph : BFS from 7 to 1 : length 6



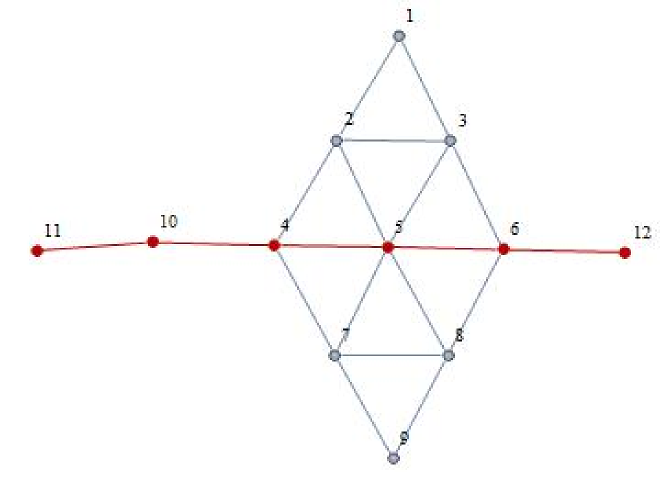
A variant of the Shuriken Graph : BFS From 1 to 9 and BFS from 9 to 1 : length 4



A variant of the Shuriken Graph : BFS from 1 to 9 (depend of the BFS), length 5



A variant of the Shuriken Graph : BFS from 11 to 12 : true diameter 5



Next week

- Go to https ://www-complexnetworks.lip6.fr/ magnien/Diameter/
- Read the paper (if possible)
- Download the files (diam.c and prelim.c and the graphs [(s)VLG] (see *data* on the web page))
- Compile, Test, Understand.
- (not this year) : Play with the Fringe and the Last But One Fringe.

Two Problems 1/

We consider two different but similar problems :

1. *The Graph Problem* : we want an approximation of the diameter and the center of a graph, supposed sparse and very large. Exact values are of course interesting.
2. The *Point Set Problem* : idem for a very large set of high dimensional points.

- Graph $G = (V, E)$: $|V| = n$ nodes and $|E| = m$ edges
- Point Set $\mathcal{S}$: $|\mathcal{S}| = n$ points , $\mathcal{S} \subset \mathbb{R}^d$
- Hypothesis : n very large, $d >> 3$
- We suppose we can compute a distance $d(x, y)$ between any two nodes of G or points of $\mathcal{S}$.

For both problems, we consider data from real-world datasets, so :

- Very large means : hundred of millions or billions of nodes/points.
- Graphs are sparse : $m = O(n)$
- For the Graph Problem classical exact algorithms can not be used because of their cubic complexity $O(n^3)$
- Points are high dimensional and some algorithms are quite expensive if the dimension is high.

There is growing interest in these two problems

1. The *Graph Problem* is concerned with *web graphs*
2. The *Point Set Problem* appears when we want to measure the quality of clusters obtained with Machine-Learning algorithms like K-means or KNN.

G : unweighted (or weighted) undirected graph

- *Eccentricity* of a node u as : $ecc(u) = \max_{v \in V} d(u,v)$
- *Set Eccentricity* of a node u as : $\mathcal{E}\rfloor\rfloor(u) = \{v | d(u,v) = ecc(u)\}$
- *Diameter* : $D = \max_x ecc(x) = \max_{x,y} d(x,y)$
- *Radius* : $R = \min_x ecc(x) = \min_x \max_y d(x,y)$.
- *Center* : set of all vertices of minimum eccentricity (radius !)
- Some inequalities for the Graph Problem :

$$\forall u : ecc(u) \leq D(G) \leq 2ecc(u)$$

$$R(G) \leq D(G) \leq 2R(G)$$

- We will use these definitions for both problems !

Graph Problem

1. For large n : any exact $O(n^3)$ or $O(n^2)$ algorithm is computationally impracticable
2. We can only use $O(n)$ or $O(n \log(n))$ approximate algorithms

Point Set Problem

1. For $d = 2$ or 3 there is a fast algorithm
2. Compute the Convex Hull
3. Diameter points are on the Convex Hull
4. For $d > 3$ the Convex Hull can be computed only with a $O(n^{d/2})$ algorithm
5. So, again : we can only use $O(n)$ or $O(n \log(n))$ approximate algorithms

10 Two (different but similar) Problems
- Known approximate algorithms
- A new algorithm for the Graph Problem
- Some Open Problems as a Conclusion

Almost all fast approximate algorithms follows the following

Multiple-Sweep Strategy

1. Choose $u \in V$, make a *Breadth First Search* (BFS) : we will have a node $v \in \mathcal{E}cc(u)$ so : $v = \arg ecc(u)$
2. Use a (small) set of vertices to refine the bounds : $e_L \leq D(G) \leq e_U$
3. If $e_L = e_U$ then we have found the true value $D(G) = e_L = e_U$: it's a **self-checking algorithm** !

Tactics ? How to choose u ?

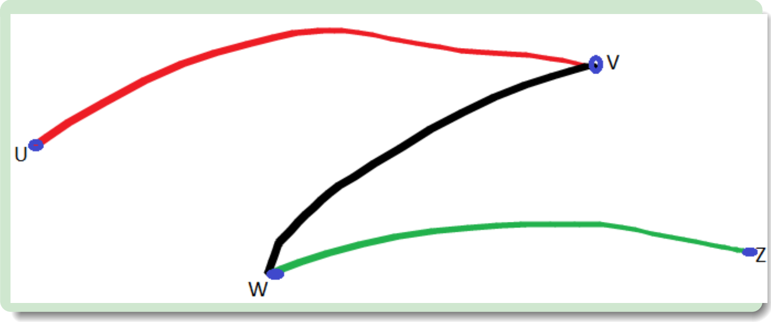
1. Choose u of low degree/high degree/random degree ?
2. Our tactic : compute an approximation c of the Center and choose some nodes $u \in \mathcal{E}cc(c)$

The fastest known approximate algorithm :

Malandain & Boissonat [2001]

1. Choose $u \in \mathcal{S}$
2. First Sweep : Find a point $v \in \mathcal{E}cc(u)$ so : $v = \arg ecc(u)$
3. Do (possibly) Multiple Sweeps
4. Consider the "best" quasi diameter $[u, v]$ obtained
5. Compute c : the middle of $[u, v]$
6. Define $r = d(u, v)/2$
7. Find all points in the ball $B(c, r)$
8. Eliminate all these points from $\mathcal{S}$
9. Iterates with the remainder points
10. Stop when *stopping criterium* is true.

Multiple Sweep for the Graph Problem

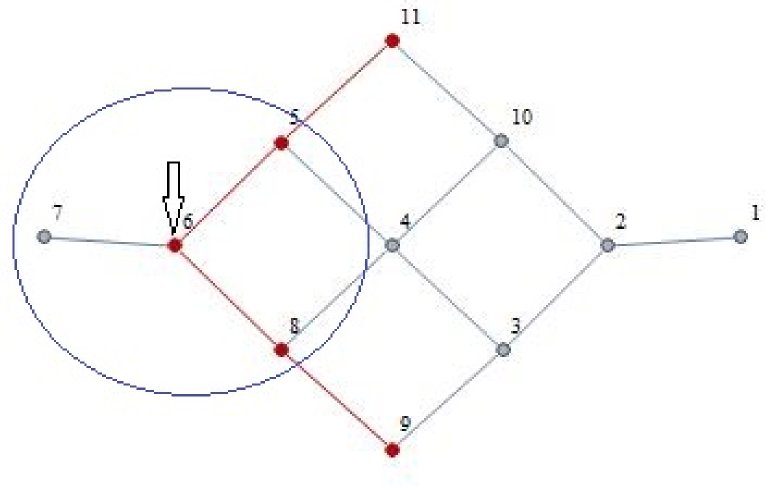


A new algorithm for the Graph Problem 2/

Description

1. Choose $u \in V$
2. First Sweep : Find a point $v \in \mathcal{E}cc(u)$ so : $v = \arg ecc(u)$
3. Do (possibly) Multiple Sweeps
4. Consider the "best" quasi diameter $[u, v]$ obtained (and e_L, e_U)
5. Compute c : the middle of $[u, v]$
6. Define $r = \lfloor d(u, v)/2 - 1 \rfloor$ (or $r = \lfloor d(u, v)/2 - 2 \rfloor$)
7. Find all points in the ball "$B(c, r)$"
8. Eliminate all these points from V
9. Iterates with the remainder points to refine e_L, e_U
10. If we want the exact diameter ; iterate till $e_L = D(G) = e_U$.

The algorithm on a toy example : the *Shuriken* Graph : $u = 11$, BFS gives $ecc(u) = 4 = d(u, v = 9)$



Plan

10 Two (different but similar) Problems
- Known approximate algorithms
- A new algorithm for the Graph Problem
- Some Open Problems as a Conclusion

About the "new" algorithm :

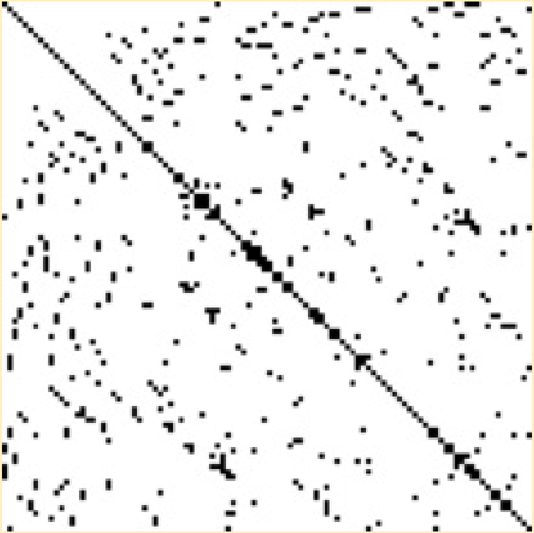
- *Approximate* diameter : it can be very fast (tests with real graphs of 4 10^8 nodes)
- *Exact* diameter :
  1. Can we obtain in $O(n)$ the true value for some real world sparse large graphs ?
  2. At least for graphs with low hyperbolicity ?
  3. Or for a specific class of graphs ?
- Can we find a "self-checking" algorithm for the Point Set Problem ?

11 Representing/Storing a graph

Sparse ?

- (Wikipedia) : a sparse matrix is a matrix in which most of the elements are zero.
- By contrast, if most of the elements are nonzero, then the matrix is considered dense.
- The fraction of zero elements (non-zero elements) in a matrix is called the *sparsity* (density).

A Sparse Matrix/Graph (Black= non zero element)



The Yale Format for storing a sparse matrix

- Store only the non zero elements, and the index (i,j) in two separate arrays
- Example : $A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 5 & 4 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{bmatrix}$
- will be stored with
  1. $A = [1, 5, 4, 3, 2]$
  2. with $IA = [1, 2, 2, 3, 4]$
  3. and $JA = [3, 1, 2, 2, 3]$

The LIL Format :LIst of Lists

- Store one list per row, *i.e.* per node
- With each entry containing the column index and the value.
- Sorted by column index for faster lookup.
- Very good for incremental matrix construction.

List of degrees and edges :[21]

* INPUT FORMAT *
****

- The first line must be the number n of nodes ;
- Then comes a series of lines of the form 'i j' meaning that node 'i' has degree 'j',
- And then a series of lines of the form 'u v' meaning that nodes 'u' and 'v' are linked together.
- The nodes must be numbered from 0 to n-1. There must be no loop (u,u).

List of edges : Example : [21]

3

0 2

1 2

2 2

0 1

0 2

2 1

(3 nodes, thus numbered from 0 to 2, node 0 has degree 2, node 1 has degree 2, and node 2 has degree 2 too, and the links are 0 1, 0 2 and 2 1)

Store your (sparse) matrix : and save Space & Time

A toy example in scikit/python :

$$\begin{bmatrix} 14 & 0 & 19 & 0 \\ 0 & 17 & 0 & 0 \\ 0 & 0 & 18 & 0 \\ 0 & 0 & 0 & 15 \end{bmatrix}$$

```
>>> # csr_matrix(arg1[, shape, dtype, copy])
>>> # Compressed Sparse Row matrix
>>> from scipy import sparse
>>> from numpy import array
>>> I = array([0,0,1,2,3])
>>> J = array([0,2,1,2,3])
>>> V = array([14,19,17,18,15])
>>> A = sparse.csr_matrix(I,J,V),shape=(4,4))
```

12 Complements

Complements

The rest of the slides are just complements, there do not concerne the projects of this year. There are related to previous years.

13 Fringe and Last But One Fringe Algorithm

The Fringe and the Last But One Fringe

Algorithme 1 : The Fringe and the Last But One Fringe

(See [101] and [103,104])

Donnees : a graph G (connex) of n nodes ;

Procedures externes : **BFS** ;

Sortie : Approximate Diameter(G)

Debut :

(1) Choose a vertex x_0 (example : maximum degree) ;

(2) Use a BFS to compute $d(x_0, v)$;

for all $v \in \{0, \cdots n-1\}/\{v_0\}$ $K = Ecc(x_0)$;

(3) $Inf = K$; $Sup = 2K$;

(4) Take the Fringe $F_k = \{f_1, \cdots f_w\}$

(...) the last level of $BFS(x_0)$;

Continue

The Fringe and the Last But One Fringe

Algorithme : The Fringe and the Last But One Fringe

(See [101] and [103,104])

For $i = 1$ **To** w ;

Compute $BFS(f_i)$; this gives $Ecc(f_i)$

If $Ecc(f_i) > Inf$ **Then**

$Inf = Ecc(f_i)$;

'[If] $Inf = Sup$ **Then** $\{D = Inf$; Return D ;

EndIf

EndFor ;

If $Ecc(f_i) > Inf$ **Then** $Inf = Ecc(f_i)$;

(5) **If** $Inf \geq Sup - 2$ **Then** $\{D = Inf$; Return D ; $\}$

Else

$\{Sup = Sup - 2$; $k = k - 1$; $\}$

Continue with the Previous Fringe **Goto (4)** ;

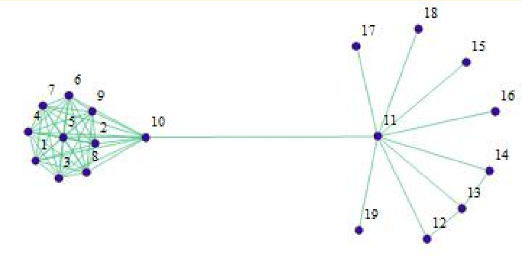
Fin.

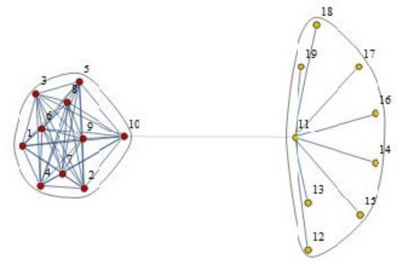
14 Communities in a (s)VLG

Community in a graph (see [102] and [106] and 107)

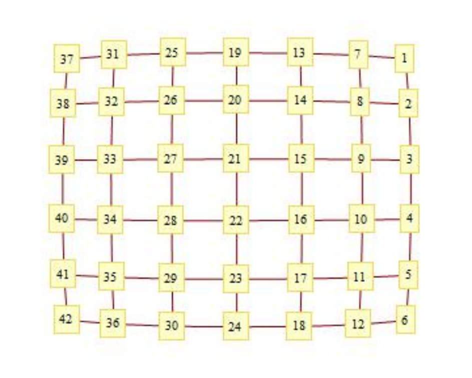
- groups of vertices within which connections are dense,
- but between which connections are sparser.
- (See next figure).

A toy example and its communities

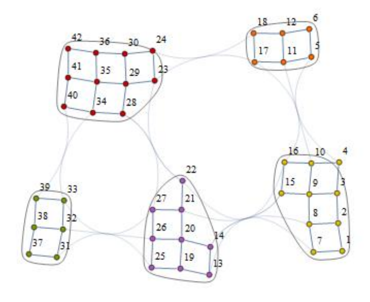




Another toy example



Its communities



http ://digitalinterface.blogspot.fr/2013/05/community-detection-in-graphs.html : 3 communities

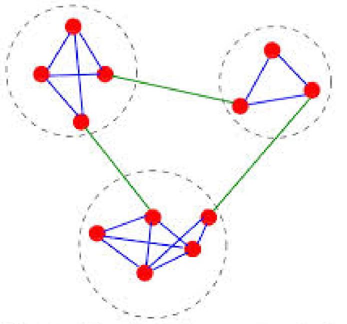


FIG. 1 A simple graph with three communities, enclosed by the dashed circles. Reprinted figure with permission from Ref. (Fortunato and Castellano, 2009). ©2009 by Springer.

Community in a graph : a mathematical definition of the *Modularity* function $Q(\mathcal{C})$:

- We suppose $\mathcal{C} = C_1 \bigcup C_2 \cdots C_{n_c}$, $\mathcal{C}$ is a community division (no overlapping communities : $C_i \bigcap C_j = \oslash$).
- $Q(\mathcal{C}) = \frac{1}{2m} \sum_{i,j} (A_{i,j} - \frac{k_i k_j}{2m}) \delta(C_i, C_j)$
- k_i : degree of node i,
- k_j : degree of node j
- and $\delta(C_i, C_j) = 1$ if nodes i and j are in the same community, 0 if otherwise.

Community in a graph : another formulation via communities

- The only contributions to the sum come from vertex pairs belonging to the same cluster, so, we can group these contributions together and rewrite the sum over the vertex pairs as a sum over the clusters :

$$Q(\mathcal{C}) = \sum_{i=1}^{n_c} [\frac{l_{C_i}}{m} - (\frac{d_{C_i}}{2m})^2]$$

- n_c : the number of different communities
- l_{C_i} the total number of edges joining vertices of community C_i
- and d_{C_i} the sum of the degrees of the vertices of community C_i.

Modularity function : 4 identical expressions for the most popular function

1. $$Q(\mathcal{C}) = \frac{1}{2m}\sum_{i,j}(A_{i,j} - \frac{k_i k_j}{2m})\delta(C_i, C_j)$$
2. $$Q(\mathcal{C}) = \sum_{i=1}^{n_c}[\frac{l_{C_i}}{m} - (\frac{d_{C_i}}{2m})^2]$$
3. $$Q(\mathcal{C}) = \sum_{C_i \in \mathcal{C}}[\frac{E(C_i)}{m} - (\frac{Vol(C_i)}{2m})^2]$$
4. $$Q(\mathcal{C}) = \sum_{C_i \in \mathcal{C}}[\frac{Vol(C_i)}{2m} - (\frac{Vol(C_i)}{2m})^2]$$

- $E(C_i)$: number of edges inside C_i
- $Vol(C_i)$: volume of C_i equal to $\sum_{i \in C_i} k_i$

Community in a graph : another formulation via communities

- Q is a numerical index of how good a community division $\mathcal{C}$ is.
- $Q(\mathcal{C}) \in [-1, 1]$.
- Finding $\arg_{\mathcal{C}} \max Q(\mathcal{C})$ is a NP-complete problem.

Community in a graph : The Louvain Algorithm

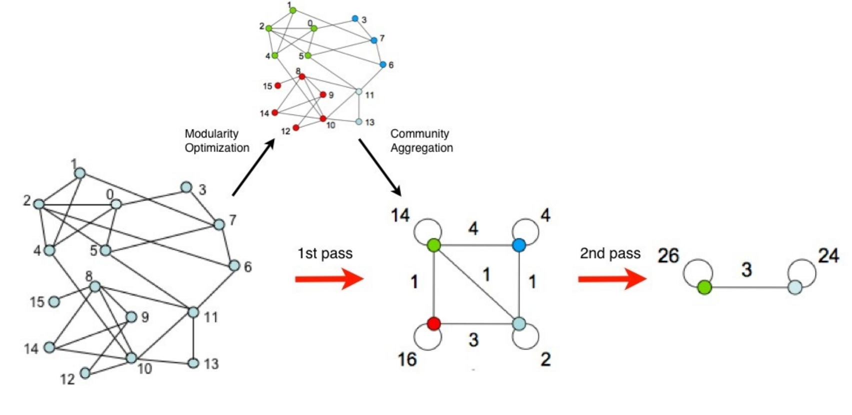
- THE paper to read, Title : *Fast unfolding of communities in large networks*
- Author : Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre,
- Journal of Statistical Mechanics : Theory and Experiment 2008 (10)
- Very Fast : quite linear
- Code (and paper) : https ://sites.google.com/site/findcommunities/

Community in a graph : The Louvain Algorithm

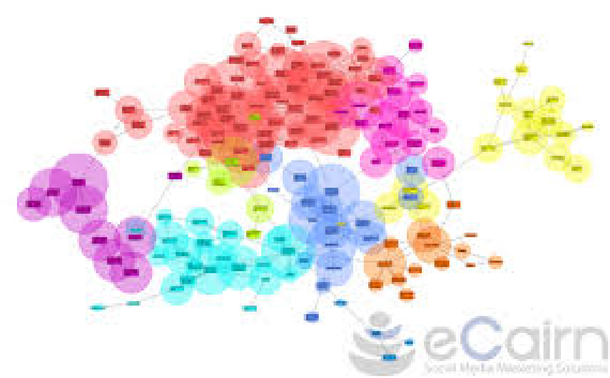
The method consists of two phases. First, it looks for "small" communities by optimizing modularity in a local way. Second, it aggregates nodes of the same community and builds a new network whose nodes are the communities. These steps are repeated iteratively until a maximum of modularity is attained.

https ://sourceforge.net/projects/louvain/

Ancienne :
https ://sites.google.com/site/findcommunities/



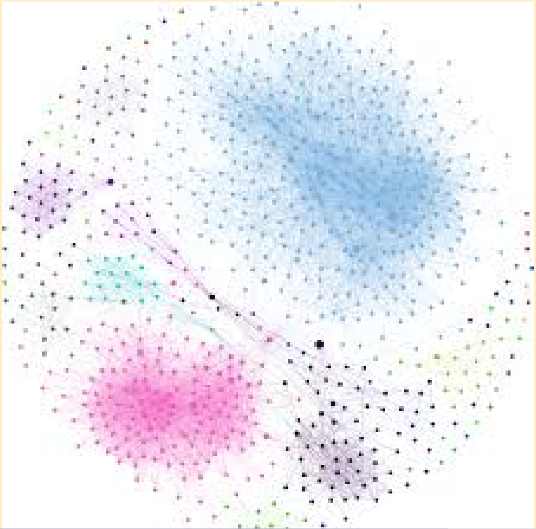
http ://blog.ecairn.com/2012/03/22/digital-influence-renaissance-solis/



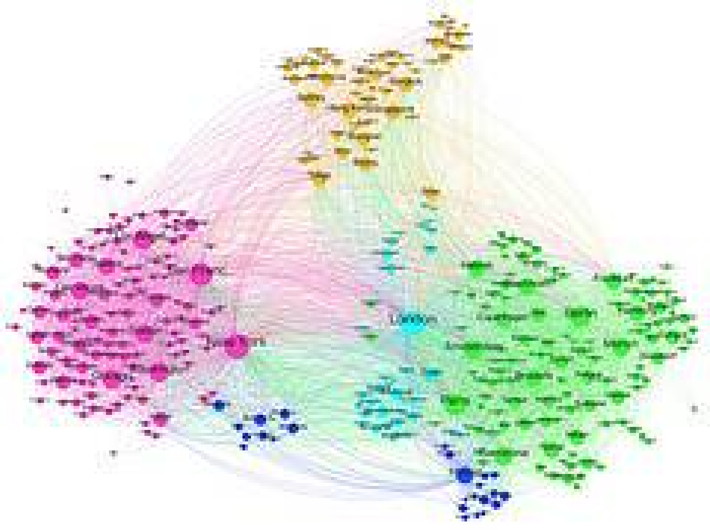
http ://www.hackdiary.com/

VLG : example of communities

https ://wisonets.wordpress.com/tag/force-based-algorithms-graph-drawing/



https ://www.flickr.com/photos/mbiddulph/sets/72157627176352483/deta



15 Reordering a Graph for Fun & Profit : *MOD versus MUD*

Reordering a graph ? G and G' two isomorphic graphs

- Theorically solving a problem $\mathcal{P}$ on G is stricly equivalent to solve the problem on G'.
- But what about practically with (s)VLG ? Surprisingly it is sometimes better (times) to solve the problem $\mathcal{P}$ on a renumbered version G' of G.
- How to renumber ? When ? and Why ?
- We will experimentally explore this on : a BFS traversal, diameter approximation, and the community computation.

MOD versus MUD

- MUD paradigm :Massive Unordered Data (MAP/REDUCE paradigm)
- MOD :Massive Ordered Data (example : graphs, numerical matrices . . .)

MUD :Massive Unordered Data (from Ref [31])

`The Massive, Unordered, Distributed-data (MUD) model was recently introduced by Feldman and al. [32] as an abstraction of part of the infrastructure used at Google. It is related to the MapReduce framework presented in [33]. In the multi-round, multi-key MUD model, n data records are distributed arbitrarily between M machines. Each machine maps each record to (key, value) pairs. All pairs corresponding to the same key are then 'reduced' to a single record. This reduction is performed by an O(polylogn)-space streaming computation. The process repeats for a total of l rounds. The model is very powerful and it was proven that any EREW-PRAM algorithm can be simulated in the multi-round, multi-key MUD model if the number of keys and rounds is sufficiently large [...]. In practice we are primarily interested in computing with a small number of keys and rounds. What can be computed given k keys and l rounds?`

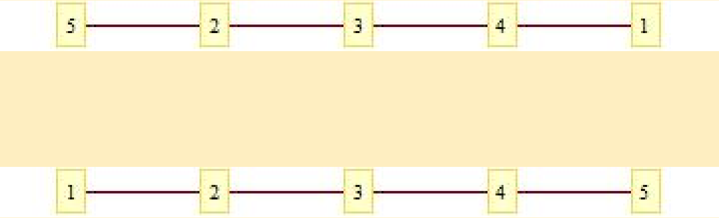
Reordering/Renumbering a matrix

- Problem comes from Numerical Analysis : algorithms to solve large linear problems $Ax = b$: A is large and sparse (Difference methods, Finite Elements Methods etc.)
- For each problem (direct solving of $Ax = b$, A is symmetric, or A is unsymmetric, iterative solving ...) soon it appeared that renumbering the x_i could be interesting. [Cuthill-Mackee algorithm : 1969]
- But renumbering the Matrix means renumbering the vertices of $G(A)$ (the graph G such that its adjacency matrix is A) !
- Be careful : Cuthill-Mackee, Sloan, Gibbs-Poole etc. can be very expensive, but Sloan can be used.

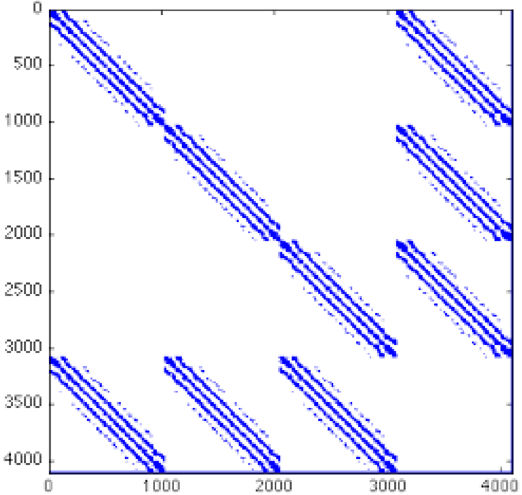
A simple Reordering/Renumbering of a graph/matrix

- Choose a node v_0
- Make a BFS traversal from v_0, let $L = BFS(v_0) = \{v_0, v_1, v_2, \cdots v_{n-1}\}$
- Renumber the graph using L.
- Save the graph.
- Ideas to choose v_0 : make a double sweep. Use the last vertex as v_0.

Two views of the same graph : A and B are isomorphic

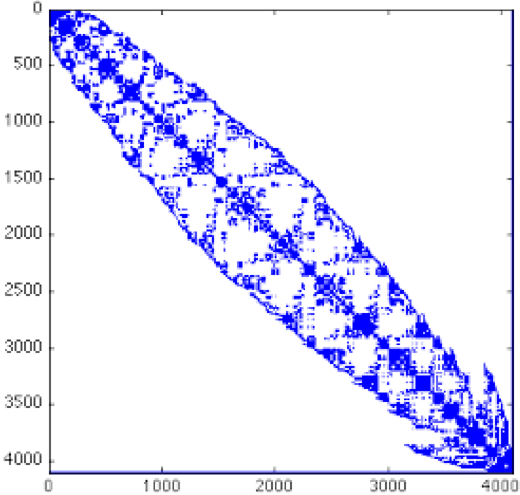


http ://rbfzone.blogspot.fr/2011/12/cuthill-mckee-ordering-of-boostublas.html



Reordering a matrix : Reordered Matrix : locality

http ://rbfzone.blogspot.fr/2011/12/cuthill-mckee-ordering-of-boostublas.html



Indirection : test case 1/2

```
int A[n];
int B1[n],B2[n],B3[n];
.....
/* Loop 1 B1[i]=i */
for(i=0;i<n,i++)
   A[B1[i]]=i;
/* Loop 2 B1[i]=C[i]*/
for(i=0;i<n,i++)
   A[B2[i]]=i;
/* Loop 3 B3[i]=C[D[i]] */
for(i=0;i<n,i++)
   A[B3[i]]=i;
```

Indirection : 2/2

- Take a large n : $n > 10^6$, $n > 10^7$ or $n > 10^8$
- Let $B1 = \{0, \cdots n-1\}$, compute the cpu time needed for the loop 1
- Let $B2 = \{n-1, \cdots 0\}$, compute the cpu time needed for the loop 2
- Let $B3 = \{n-1, \cdots 0\}$, again compute the cpu time needed for the loop 3
- Let $B3$ be a random permutation of $\{0, \cdots n-1\}$, again compute the cpu time needed for the loop

Plan

16 WorkingPackage to do for 2018 ?

What ? See the file "Working Packages 2018 ..."

WorkPackages (WP) : what ?

- Choose a project between :
  1. Project 1 (diameter : classical version with a modification)
  2. Project 2 (diameter : a new "suppression" algorithm)
  3. Project "special" : A personnel project about (Sparse) Very Large Graphs (but you need to tell us before and we have to sent you our answer).
- Do the job (see after)
- How ? sent a **unique file** (zip, tar)to : **verylargegraphs@gmail.com**
- When ? Dead line : before June, 1th 2018 23h42.
- Any question ? Sent it to **verylargegraphs@gmail.com**

Work to do : : WorkPackages (WP) 2015-2016 (old)

- Choose a WorkPackages from 1 to 4.
- Do the job.
- Write a file *RESULTS_ <YOUR NAME>.xxx* (max 4 pages in any format) within : Your name, the WP you have chosen, and all the results (specially timing results or any problem)
- Send :
  1. Every programs (with YOUR NAME)
  2. The file RESULTS_<YOUR NAME>.xxx (including the timing results)
  3. In a single message please
  4. At **verylargegraphs@gmail.com**

Work to do : Workpackage 1 2015-2016 (old)

- Take the graphs inet and ip : compute and save the giant connected component (see diam.c) : inetgiant and ipgiant
- Implement the double sweep and/or the Fringe algorithms (with bounds)
- Write a reordering algorithms as presented (use a simple BFS from a random node of the giant connected component or a simple BFS from the max degree node)
- Test on inet/inetgiant + ip/ipgiant (for example with time.h)
  1. with diam.c on unordered graphs
  2. with diam.c on reordered graphs
  3. with your version of the double sweep
  4. and he Fringe algorithms (with bounds)
  5. with community.cpp on unordered graphs and

Work to do Workpackage 2 : go to (Ref [21]) 2015-2016 (old)

- Take the graphs inet and ip : compute and save the giant connected component (see diam.c) : inetgiant and ipgiant
- Implement the double sweep and/or he Fringe algorithms (with bounds)
- Use « my » reordering algorithms : send an email to verylargegraphs@gmail.com : but you will losse some points
- Test on inet/inetgiant + ip/ipgiant (for example with time.h)
  1. with diam.c on unordered graphs
  2. with diam.c on reordered graphs
  3. with your version of the double sweep
  4. and the Fringe algorithms (with bounds)
  5. with community.cpp on unordered graphs and

Work to do : Workpackage 3 : play with community.cpp 2015-2016 (old)

- Take the graphs inet and ip : compute and save the giant component (see diam.c) : inetgiant and ipgiant
- Use the resultst of community.cpp to write a reordering algorithms
- Implement (with time.h) the double sweep and/or the Fringe algorithms (with bounds)
- Test (with time.h) on inet/inetgiant + ip/ipgiant
  1. with diam.c on unordered graphs
  2. with diam.c on reordered graphs
  3. with your version of the double sweep
  4. and he Fringe algorithms (with bounds)
  5. and (again) with community.cpp on unordered graphs and reordered graphs
- Send everything

Work to do : Workpackage 4 : play with diam.c and community.cpp 2015-2016 (old)

- If you choose this WP : please write me to discuss : this is a very explorating WP
- Take the graphs inet and ip : compute and save the giant component (see diam.c) : inetgiant and ipgiant
- Use the results of community.cpp to write a reordering algorithms
- Implement a « Fringe like » algorithm using results of community.cpp : Basically this means you create a new graph where each node is a community, we will call this graph the *metagraph*. Compute the diameter of this very smaller metagraph, save the communities involved (and some of the vertices involved,explain how you choose these vertices)) and use them on the true graphs with diam.c using (or your modified newdiam.c) nodes of the community involded. This is a very promising algorithm.

Work to do : Workpackage 4 : play with diam.c and community.cpp :2015-2016 (old)

- Test (with time.h) on inet/inetgiant + ip/ipgiant
  1. with diam.c on unordered graphs
  2. with diam.c on reordered graphs
  3. with newdiam.c on unordered graphs
  4. with newdiam.c on reordered graphs
  5. and (again) with community.cpp on unordered graphs and reordered graphs
  6. and with you program
- Send everything ...

Plan

17 Some Lectures

References

- (Ref [00]) http ://matthewgress.com/galleries/Web/Internet-Graph-1069646562-LGL-2D-4096x4096
- (Ref [1]) : http ://www.math.cornell.edu/ mec/Winter2009/RalucaRemus/Lecture2/lecture2.html
- (Ref [2]) : http ://www.thp.uni-koeln.de/ lassig/projects.html
- (Ref [3]) : http ://www.cs.umd.edu/class/fall2007/cmsc858l/
- (Ref [4]) : http ://ecsite.cs.colorado.edu/subject-areas/civics/
- (Ref [5]) : http ://www.fmsasg.com/SocialNetworkAnalysis/
- (Ref [6]) : http ://www.jacobsschool.ucsd.edu/news/news_releases/release.sfe ?id=685
- (Ref [21]) : https ://www-complexnetworks.lip6.fr/ magnien/Diameter/
- (Ref [22]) : http ://mathworld.wolfram.com/GraphEccentricity.html
- (Ref [31] : http ://sublinear.info/index.php ?title=Open_Problems :17)
- (Ref [32]) : Jon Feldman, S. Muthukrishnan, Anastasios Sidiropoulos, Cliff Stein, and Zoya Svitkina. On the complexity of processing massive, unordered, distributed data. 2006.
- (Ref [33]) : Jeffrey Dean and Sanjay Ghemawat. MapReduce : Simplified data processing on large clusters. In OSDI, pages 137-150, 2004.

References

- (Ref [101]) : https ://eudml.org/doc/104856
- (Ref [102]) : https ://sites.google.com/site/findcommunities/
- (Ref [102]) : https ://sourceforge.net/projects/louvain/
- (Ref [103]) : http ://piluc.dsi.unifi.it/lasagne/ ?page_id=137
- (Ref [104]) : http ://piluc.dsi.unifi.it/lasagne/wp-content/uploads/2014/10/leo_PhD_thesis.pdf
- (Ref [105]) : http ://www.cs.princeton.edu/ chazelle/courses/BIB/pagerank.htm
- (Ref [106]) : http ://arxiv.org/abs/0906.0612
- (Ref [107]) : http ://arxiv.org/pdf/1308.0971v1.pdf
- (Ref [108]) : M. E. Newman and M. Girvan. *Finding and evaluating community structure in networks*. Physical review E, 69(2), 2004.
- (REf [109]) :https ://hal.archives-ouvertes.fr/hal-01171295/document