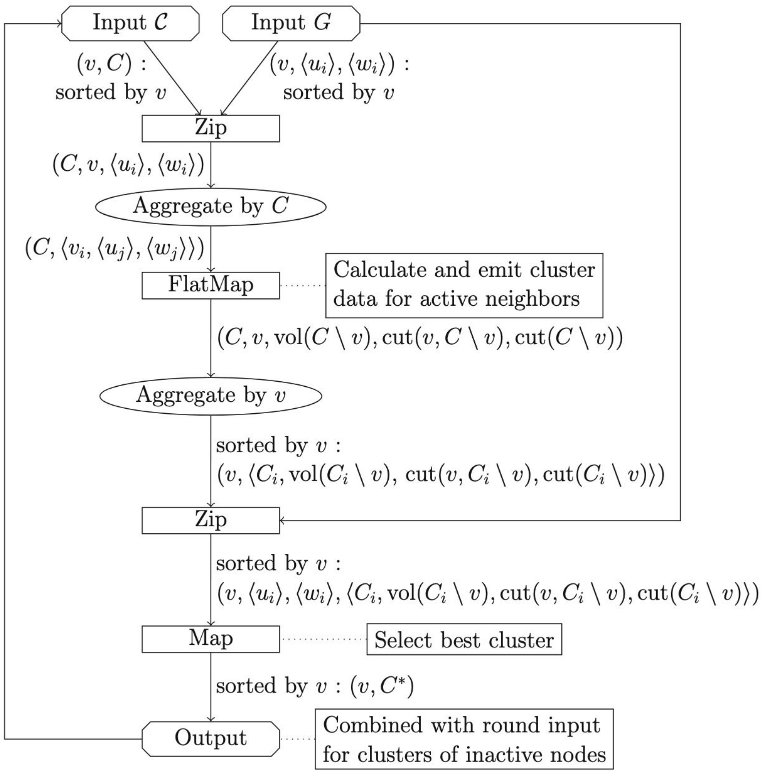
Distributing the Louvain Algorithm Using a Distributed Framework

Finn Archinuk
Ezra MacDonald
Alison Ziesel

April 06 2023
Code: github.com/aziesel/CSC502_Project

Learning Objectives

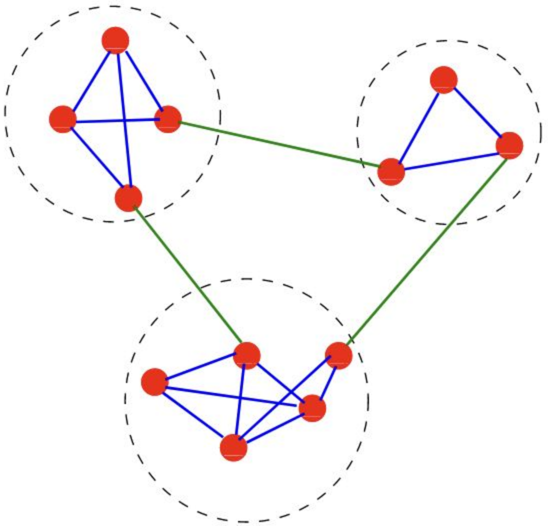
- Understand Louvain Algorithm
- Translate C++ implementation to PySpark
- Optimize implementation



Hamann *et al.* 2018

Communities in Networks

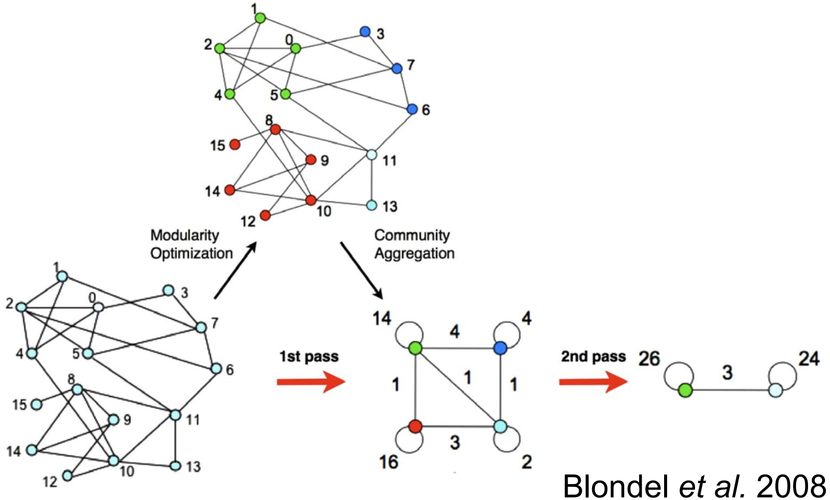
- What are communities?
  - Communities are a subset of nodes that are densely connected to each other and loosely connected to other nodes in the graph
- Applications
  - Network analysis
  - Recommender systems
  - similarity based algorithms
  - Any problem where you have a graph
- How are they defined?
  - Weights and graph edges
  - Edges with a greater weight in general indicate a closer relationship between two vertices, holding everything else constant



Fortunato & Castellano 2007

Louvain Algorithm

- What does it do?
  - divides network into most modular, or closely related communities by the given weighted edge metric
- Why is it useful?
  - It’s fast
  - Has properties that can exploited to increase parallelizability
  - It’s relatively simple to implement
  - It’s unsupervised and outperforms many similar modularity optimization methods [Aynaud et. al 2011]
- How does it work?
  - Iteratively maximize the modularity score until:
    1. A fixed number of rounds has been reached
    2. No moves to another cluster can be made in a given round



Blondel *et al.* 2008

Network Modularity

- Σ_{in} = Sum of all the edge weights between nodes within the community C
- Σ_{tot} = Sum of all the edge weights for nodes within the community including edges connected to other communities

$$Q_c = \frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2$$

Blondel et al. 2008

- In general higher modularity scores = better cluster
- vol(C) = sum of weights connecting each node within a cluster
- cut(C) = Sum of weights of all vertices v ∈ C for pairs (u, v) such that v ∈ C and u∈∉C

$$\mathcal{Q}(\mathcal{C}) := \sum_{C \in \mathcal{C}} \frac{\mathrm{vol}(C) - \mathrm{cut}(C)}{\mathrm{vol}(V)} - \sum_{C \in \mathcal{C}} \frac{\mathrm{vol}(C)^2}{\mathrm{vol}(V)^2}.$$

Hamann *et al.* 2018

Louvain Algorithm

Algorithm LouvainAlgorithm(Graph G)

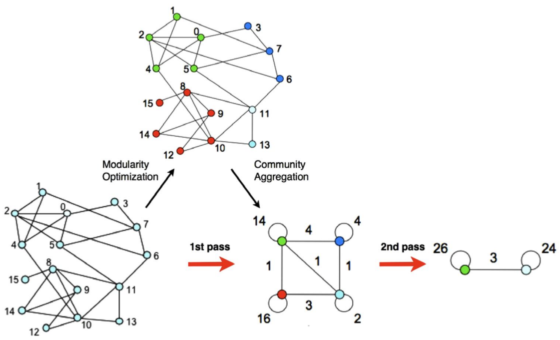
Require: $(x_1, y_1), ..., (x_m, y_m)$ where $x_i \in X$ and $y_i \in \{-1, 1\}$

$C \leftarrow index(G)$ Map the index of each node to its own cluster
$G' = G$
$q \leftarrow -\infty$
while $q < Q(G', C)$ **do**
  $q \leftarrow Q(G', C)$
  $C \leftarrow MoveNodes(G')$ // Phase 1
  $G' \leftarrow Aggregate(G', C)$ // Phase 2
  $C \leftarrow$ each node of G' in it's own community
end while
return G'

Algorithm MoveNodes(Graph G)

Require: C the index of communities for each nodes of G
Require: $(x_1, y_1), ..., (x_m, y_m)$ where $x_i \in X$ and $y_i \in \{-1, 1\}$

$C \leftarrow index(G)$ Map the index of each node to its own cluster
$G' = G$
$q \leftarrow -\infty$
while at least one node moves or reached max iterations **do**
  for random $v \in V(G)$ **do**
    $maxQ \leftarrow -\infty$
    $maxC \leftarrow$ community of v
    for each neighbor u of v **do**
      $deltaQ = \Delta Q$ between v and n
      if $maxQ < deltaQ$ **then**
        $maxQ \leftarrow deltaQ$
        $maxC \leftarrow$ community of u
      end if
    end for
    $C \leftarrow Update(C, v)$ // Update cluster for node v
  end for
end while
return G'



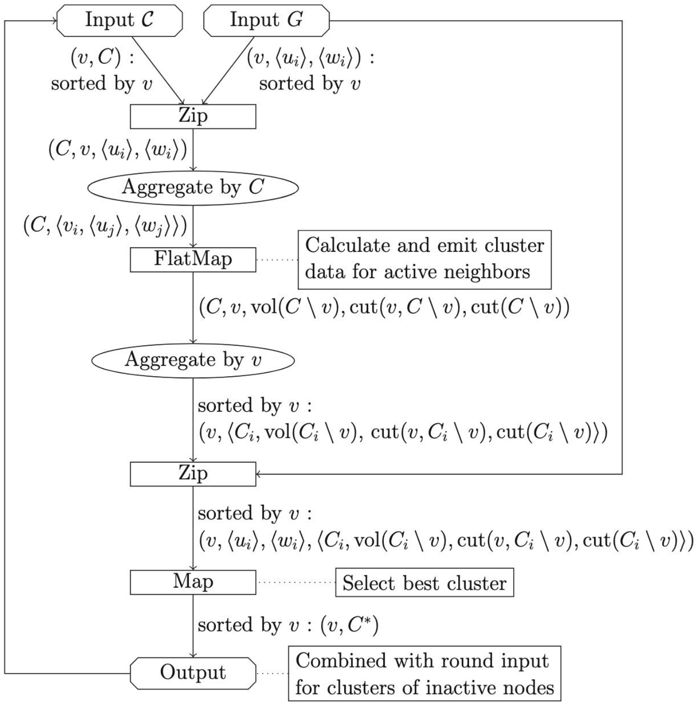
Algorithm Aggregate(Graph G, Partition C)

$G' \leftarrow$ aggregate nodes in the same community based on C

return G'

Distributed Clustering Using Modularity and Map Eq.

- Group graph inputs with the individual vertices and their corresponding cluster assignments
- Generate cluster groupings based on vertices that are randomly assigned to move in this round
- Compute the volumes and cuts for each generated clustering
- Sort respective clusterings and update



Hamann *et al.* 2018

Opportunities for Parallelization

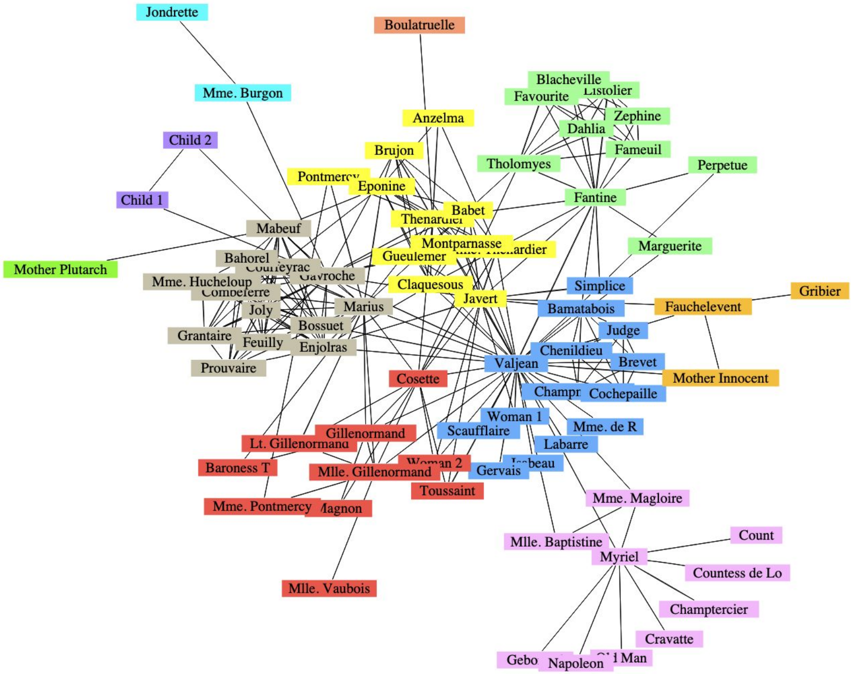
- Evaluation of the modularity gain for all vertices (Computing ΔQ)
    - Each vertex computes the modularity gain for joining its neighbours’ communities
    -
    - This results in a considerable amount of
- Clustering that’s required at each round requires multiple sort operations
-

Why PySpark

- The paper by Hamann et. al which we chose to focus on for this project drew many parallels with the MapReduce model
- Non-parallel implementations in Python exist
- Python integration into data science
  - Data science has lots of interest in network/community analysis
- No currently publicly available PySpark implementation of the Louvain algorithm
- Other implementations: Spark, Thrill (Hamann 2018)
  - Other implementations are thesis work…

Experimentation: *Les Misérables* dataset

- Toy example: Les Misérables character interaction network
  - Edges indicated the count of co-appearances of characters
- Used for development
- No real ground truth



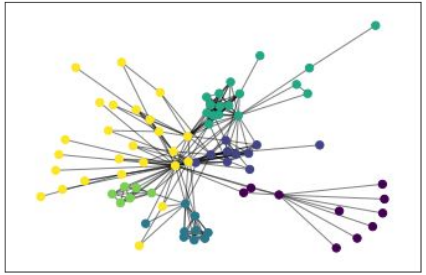
Knuth 1993

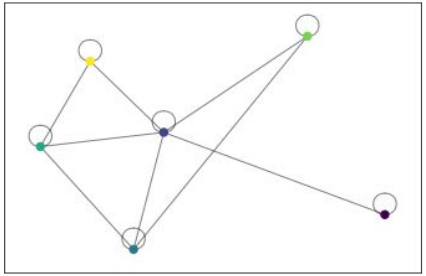
Results: Les Miserables

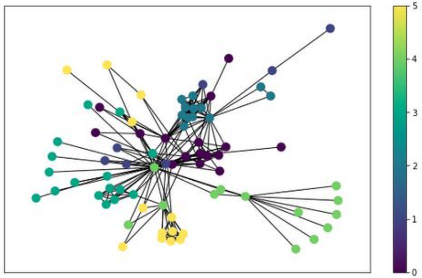
- Took a collaborative approach to development
- Prioritized having at least one working implementation
- Some implementations work better than others
- Gave everyone more development time
- More opportunity to explore unique implementation approaches

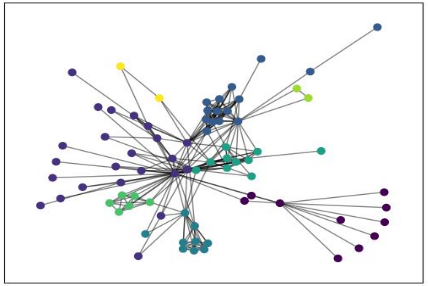
<u>Two implementations:</u>

1) 52/77 (70.1%) concurrence
2) 60/77 (77.9%) concurrence









Experimentation: ArXiv dataset

- A collection of 2.2 million+ metadata records of articles published on ArXiv retrieved from Kaggle.
- Explored the connection between coauthors
  - Edges were created based on co-authoring a publication
  - Edge weights were computed based on the number of shared publications
- No ground truth has been established for the dataset… unsupervised

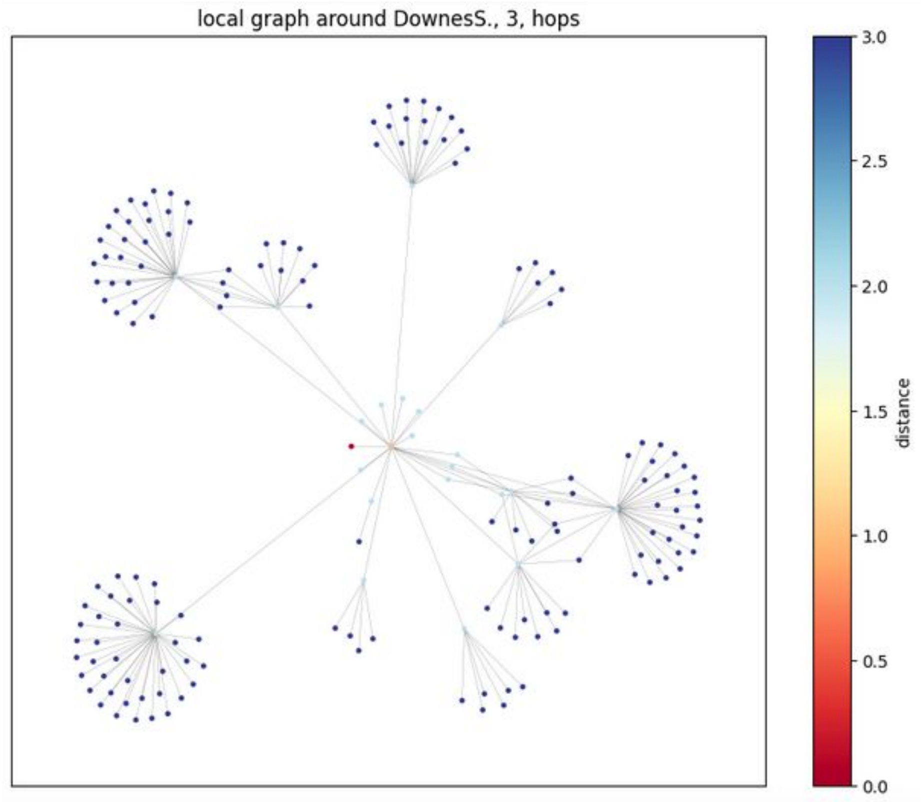
"root" : { 14 items
"id" : string "0704.0001"
"submitter" : string "Pavel Nadolsky"
"authors" : string "C. Bal\'azs, E. L. Berger, P. M. Nadolsky, C.-P. Yuan"
"title" : string "Calculation of prompt diphoton production cross sections at Tevatron and LHC energies"
"comments" : string "37 pages, 15 figures; published version"
"journal-ref" : string "Phys.Rev.D76:013009,2007"
"doi" : string "10.1103/PhysRevD.76.013009"
"report-no" : string "ANL-HEP-PR-07-12"
"categories" : string "hep-ph"
"license" : NULL
"abstract" : string " A fully differential calculation in perturbative quantum chromodynamics is presented for the production of massive photon pairs at hadron colliders. All next-to-leading order perturbative contributions from quark-antiquark, gluon-(anti)quark, and gluon-gluon subprocesses are included, as well as all-orders resummation of initial-state gluon radiation valid at next-to-next-to-leading logarithmic accuracy. The region of phase space is specified in which the calculation is most reliable. Good agreement is demonstrated with data from the Fermilab Tevatron, and predictions are made for more detailed tests with CDF and DO data. Predictions are shown for distributions of diphoton pairs produced at the energy of the Large Hadron Collider (LHC). Distributions of the diphoton pairs from the decay of a Higgs boson are contrasted with those produced from QCD processes at the LHC, showing that enhanced sensitivity to the signal can be obtained with judicious selection of events. "
"versions" : [2 items
0 : {...} 2 items
1 : {...} 2 items

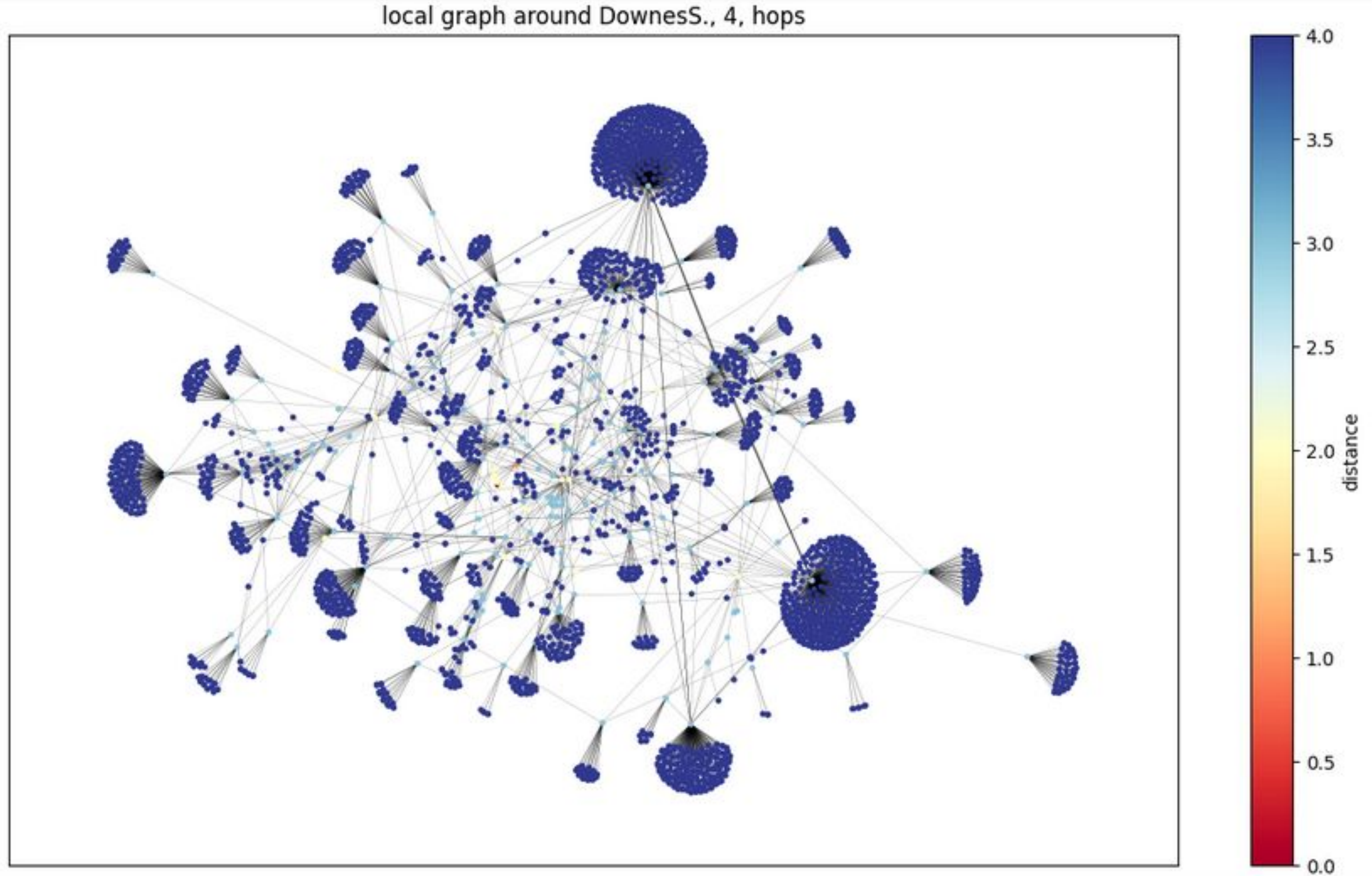
Results: Arxiv Figures

Visualization: “DownesS.” only has 1 direct connection. This figure is of the local neighbourhood.

Preprocessing: Removed “Collaborations”

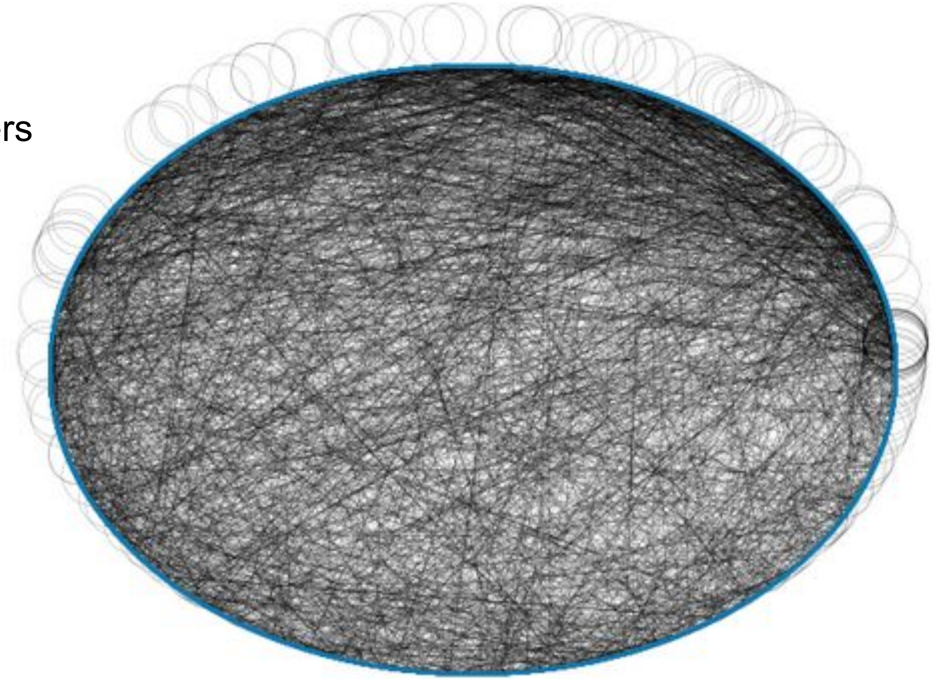
The next slide is a continuation to 4 hops. “Fireworks” are highly connected individuals.




local graph around DownesS., 4, hops
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
distance

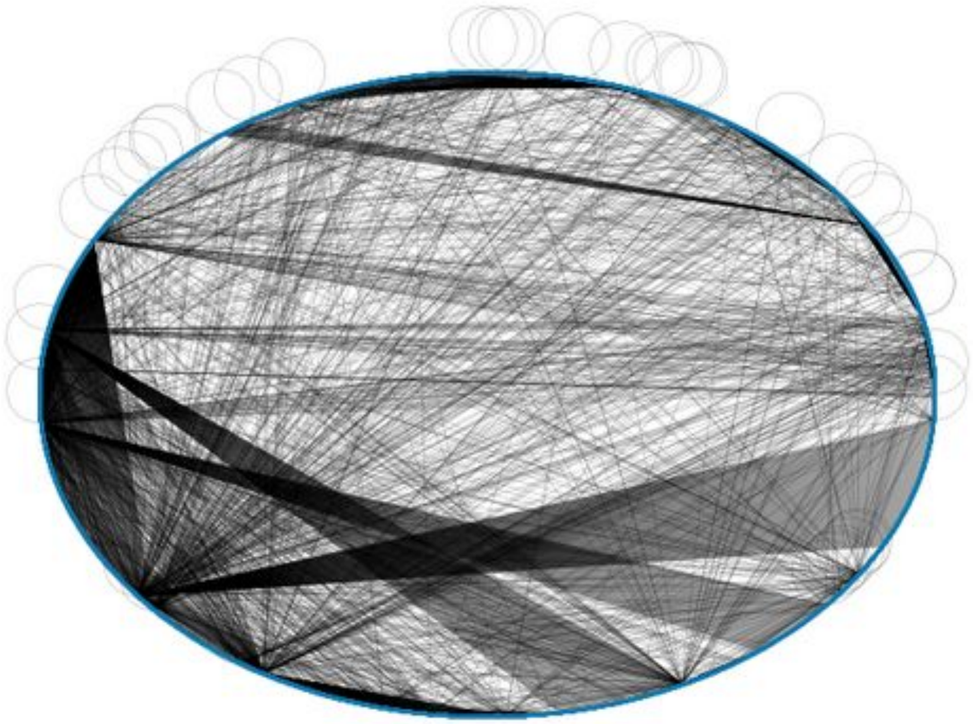
Authors were compacted.
These are a random set of
5000 *edges* of those compacted clusters

Highly connected authors make
this very difficult.



First 5000 compacted edges

Some obvious structure



Author were clustered starting from 100k singletons to ~15k clusters. These clusters were then compacted.

The compacted clusters went through a single iteration (10 subrounds).
This is the resulting cluster and neighbours of a *single* cluster.

Because of highly connected authors, compaction isn't very useful. Another challenge is highly connected authors make small communities hard to distinguish.

Conclusions

- The Louvain algorithm is an easily parallelizable algorithm that returns quality results and can be fast
- PySpark can be used to improve the computational efficiency of the Louvain algorithm
- We got acceptably close results of the non-distributed implementation
- The ArXiv dataset was a worthy adversary

References

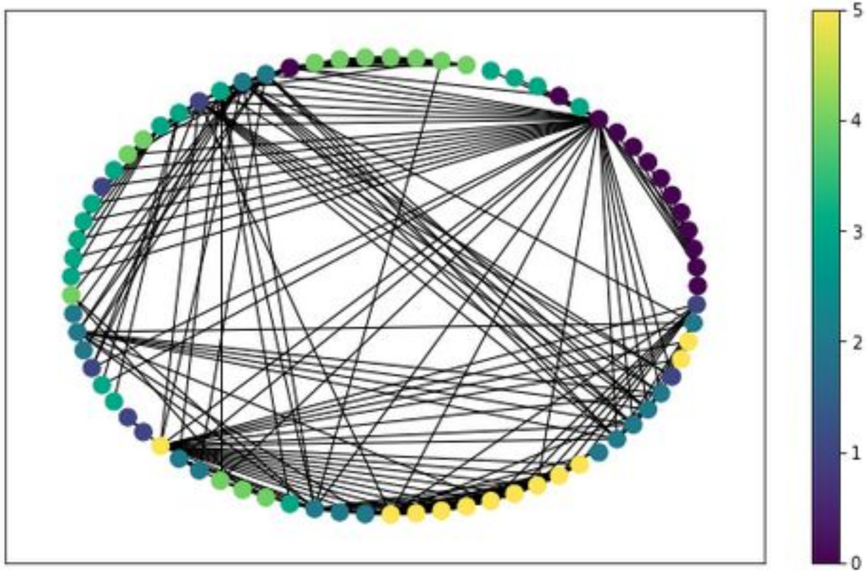
Aynaud, T.; Blondel, V. D.; Guillaume, J.; Lambiotte, R. "Multilevel local optimization of modularity". *Graph Partitioning*. John Wiley & Sons: 315–345. (2011)

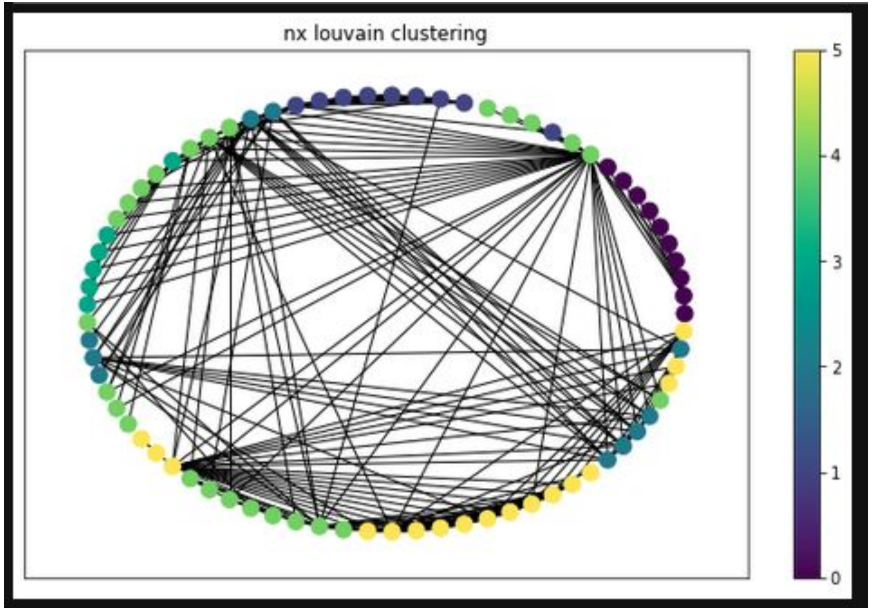
Blondel, V.D. *et al.* "Fast unfolding of communities in large networks." *Journal of Statistical Mechanics: Theory and Experiment* (2008)

Hamann, M. *et al.* "Distributed graph clustering using modularity and map equation." *Euro-Par 2018: Parallel Processing: 24th International Conference on Parallel and Distributed Computing* (2018)

https://github.com/taynaud/python-louvain

Hagberg, A.A. *et al.* "Exploring network structure, dynamics and function using NetworkX" *Proceedings of the 7th Python in Science Conference (SciPy2008)* (2008)


5
4
3
2
1
0


nx louvain clustering
5
4
3
2
1
0