Clustering in a Few Rounds

Ravi Kumar
Google
Data

“640k ought to be enough for anybody.”
Graph mining challenges

• Can be implicitly defined
  • Similarities

• Nodes/edges can change
  • Social connections

• Can have special properties
  • Heavy-tailed, small-world, bipartite, ...

• Can be noisy
  • Some edges missing, some spurious
Why are graphs hard?

- Poor **locality** of memory access
  - Neighbors of a node can be arbitrarily located in memory

- Degree of **parallelism** change during execution
  - Can depend on sub-graph structures

- Nodes by themselves do not do much work
  - Edge interactions form the bulk of many graph algorithms
Graph stream

• Graph arrives as an edge stream
  • No random access to graph
  • Can be new edge or updates to existing edges

• Typically single CPU

• Very limited amount of RAM
  • Some cases, only Mb even for Tb+ data
  • May not be able to store any portion of the graph in memory
  • Graph size may be infinite/unknown in advance

• Ideally, make a single pass over the graph
  • In some cases, can take multiple rounds
Graph clustering

- How to solve large-scale clustering problems on graphs?
- Many flavors of clustering definitions
  - k-means, k-median, densest subgraphs, correlation clustering, ..
- Focus on algorithms
  - with provable guarantees
  - that run in a small number of rounds
1. Correlation clustering (CC)

- Given a complete graph where each edge is +1 or -1, partition the nodes to minimize the total number of mistakes
  [Bansal, A. Blum, Chawla]
- Number of clusters not specified a priori
- Often, missing edges are interpreted -1
- Machine learning / data mining applications
Eg: 0 mistakes
Eg: 1 mistake
The Pivot algorithm

A simple iterative algorithm

Pick a node $p$ uniformly at random
Create a cluster around $p$ by including all nodes connected to $p$ by a $+1$ edge
Delete the nodes in this cluster
Repeat with the remaining graph

[Ailon, Charikar, Newman]
Eg: Pivot Algorithm
Eg: Pivot Algorithm
Eg: Pivot Algorithm
Eg: Pivot Algorithm
Eg: Pivot Algorithm
Eg: Pivot Algorithm
Eg: Pivot Algorithm
A bad triangle is a triple of nodes with two positive edges and one negative edge.
Properties

Claim [ACN]. Pivot gives (in expectation) a 3-approximation to minimizing the number of mistakes

Proof focuses on bad triangles and uses LP duality

The algorithm is inherently sequential
A bad example

Pivot takes $\Omega(n)$ rounds
Parallel Pivot

• A parallel version of Pivot Algorithm
  • runs in $O(\log^2 n)$ rounds
  • obtains a $3+\varepsilon$ approximation

[Chierichetti, Dalvi, Kumar]

• Easily implemented in streaming (also Map-Reduce, Pregel, ...)

Parallel Pivot Algorithm

While the graph is not empty

Let $D^+$ be the current maximum positive degree

Activate each node independently with probability $\varepsilon/D^+$

Deactivate nodes connected to other active nodes by +1 edges

The remaining nodes are pivots

Create cluster around each pivot as before

Remove the clusters
Example
Example

\[ D^+ = 2 \]
Example
Example
Example
Example
Example
Example
Example
Properties

Claim. Parallel Pivot halves the maximum degree $D^+$ after $(1/\varepsilon) \log n$ rounds

Algorithm terminates in $(1/\varepsilon) (\log n) (\log D^+)$ rounds

Claim. Induces a close to uniform marginal distribution of the pivots

Can extend the LP dual-based proof of [ACN] to show $3 + \varepsilon$ approximation
Halving max degree

- **Event** $e(v)$: exactly one positive neighbor $w$ of node $v$ gets activated and no positive neighbor of $w$ gets activated
  - $w$ becomes a pivot and hence $v$ is removed

- **Key property**: $\Pr[e(v)] > \frac{\varepsilon}{8}$ if $\deg^+(v) > \frac{D^+}{2}$

- After logarithmic number of rounds, either $v$’s positive degree halves or $v$ will end up in a cluster
Other natural sampling methods can produce a non-constant approximation

- node $u$ is activated with probability $\deg(u)$
- Eg, star of degree $n$
- node $u$ is activated with probability $1/\deg(u)$ [Luby]
- Eg, a clique matched to an independent set of nodes
Eg. inverse degree
Algorithm vs Optimum

VS
Other uniform sampling approaches might require more rounds

- node $u$ is activated $wp << \frac{1}{D^+}$
  - few active nodes, few pivots, many rounds
- node $u$ is activated $wp >> \frac{1}{D^+}$
  - many active nodes, few pivots, many rounds
  - pivots far from uniform distribution
Twitter Dataset

41M nodes, 2.5B positive edges
2. Densest subgraph (DSG)

- Find densest subgraph in undirected graphs
  - **Density** of a subgraph is the ratio of the number of edges to the number of nodes
  - Motivation: Community finding
  - c-approximation = when density is at most c times worse then the best density

Density(●) = 5/4 = 1.2
Complexity of DSG

- DSG can be computed in polynomial time
  - Using parametric flows or LP relaxation

- Natural variants of DSG are hard
  - k-DSG, subgraph with exactly k nodes

- Charikar’s 2-approximation algorithm
  - Iteratively remove the lowest degree node until the graph becomes empty
  - One of the intermediate graphs is a 2-approx.

- These algorithms are hard to scale
A simple iterative algorithm

- Compute the average degree
- Delete all nodes whose degree is \((1+\varepsilon)\) below the average
- Keep track of the density at each step
- Output the densest graph seen during the iteration

[Bahmani, Kumar, Vassilvitskii]
DSG: Example

density = 16/11 = 1.45; average degree = 2*density = 2.90
Best density = 1.45
DSG: Example (contd)

density = 16/11 = 1.45; average degree = 2*density = 2.90
Best density = 1.45
DSG: Example (contd)

density = \frac{9}{5} = 1.8; \text{ average degree} = 2 \times \text{density} = 3.6

Best density = 1.8
DSG: Example (contd)

density = 9/5 = 1.8; average degree = 2*density = 3.6
Best density = 1.8
DSG: Example (contd)

density = \frac{3}{3} = 1; \text{ average degree} = 2 \times \text{density} = 2

Best density = 1.8
DSG: Example (contd)

Best density = 1.8
IM: Remaining graph vs iterations

Remaining nodes

Iterations

ε=0
ε=1
ε=2
**Properties**

**Claim.** Algorithm makes $O(\log_{1+\varepsilon} n)$ passes and uses $O(n)$ memory

Use an averaging argument

**Claim.** Output is a $(2+\varepsilon)$-approx.

$V^* = \text{optimal induced subgraph, } p^* = \text{density}(V^*)$

Each node in $V^*$ has degree at least $p^*$ (optimality)

$V' = \text{first subgraph where we are about to remove a node in } V^*$
Concluding thoughts

- Non-traditional computational models are key to managing big graphs
  - Novel algorithmic ideas
  - New programming paradigms
- Round complexity is important
  - One-pass 2-approximation algorithm for DSG [Bhattacharya, Henzinger, Nanongkai, Tsourakakis]
  - Correlation clustering?
  - k-means++?
- Managing heavy tail, data skew, asynchrony, communication, …
Thank you!

Questions/Comments

ravi.k53 @ gmail