Snowball Sampling a Large Graph

William Cohen

Out March 20, 2013
Due Wed, April 3, 2013 via Blackboard

1 Background

A "snowball sample" of a graph starts with some set of seed nodes of interest, and then repeatedly adds some neighbors of the seed nodes and their incident edges. The idea is to come up with some version of the "local neighborhood" of a node so that one can do analysis of, say, the Facebook friend graph of a small subcommunity. Doing this is unfortunately tricky for a large graph. This assignment uses some of the ideas in a 2006 FOCS paper "Local graph partitioning using PageRank vectors" by Andersen, Chung, and Lang to do a sort of snowball sampling of a large graph—one which you have on disk.

Some notation first.

- G is a graph, V the vertices, E the edges, $n = |V|$, and $m = |E|$.
- I'll use indices i for vertices when convenient, so v_i has index i.
- $d(v)$ is the degree of $v \in V$, and D is a matrix with $D_{i,i} = d(v_i)$.
- χ_v is a unit (row) vector with all weight on vertex v.
- A is an adjacency matrix for G.
- $W = \frac{1}{2}(I + D^{-1}A)$ is a "lazy random walk" matrix, where there is probability 1/2 of staying at vertex v, and probability 1/2 of moving to some other vertex u connected to v.

- We consider a "lazy" version of personalized PageRank, which is the unique solution to

$$pr(\alpha, s) = \alpha s + (1-\alpha) pr(\alpha, s) W \tag{1}$$

  where s is a "seed" distribution (row vector), α is a "teleportation constant".

- It is easy to see that $pr(\alpha, s)$ is a linear function of s

$$pr(\alpha, s) = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t s W^t = s[\alpha \sum_{t=0}^{\infty} (1-\alpha)^t W^t] = s\alpha[I - (1-\alpha)W]^{-1} \tag{2}$$

- It's easy to show that

$$pr(\alpha, s) = \alpha s + (1-\alpha) pr(\alpha, sW) \tag{3}$$

  Note the subtle difference from Eq 1 - this statement is true, but not obvious.

2 Approximating PageRank with "pushes"

The personalized PageRank (row) vector $pr(\alpha, s)$ can be incrementally approximated as the following.

We maintain a pair of vectors p (the current approximation) and r (the "residual"). Initially $r = s$ and p is an all-zeros vector. This guarantee that the following equality is satisfied

$$p + pr(\alpha, r) = pr(\alpha, s) \tag{4}$$

Now we repeatedly apply Eq 3 to move probability mass from r to p, but maintain the equality in Eq 4.

We define a $push(u, p, r)$ operation as

$$p' = p + \alpha r_u$$

$$r' = r - r_u + (1-\alpha) r_u W$$

where u is a node with non-zero weight in r and r_u is a vector which is zero everywhere except with weight $r(u)$ on node u. A push operation move

α of u's weight from r to p, and then distributing the remaining $(1-\alpha)$ weight within r as if a single step of the random walk associated with W were performed. This operation maintains the equality in Eq 4. Notice that to do a "push" on u, we need to know $d(u)$ and the neighbors of u, but we don't need to know anything else about the graph.

Let $apr(\alpha, \epsilon, v_0)$ be an "approximate PageRank" which is the result of performing "pushes" repeatedly, in any order, until there is no vertex u such that $r(u)/d(u) \geq \epsilon$ (and then using p as the approximation). Then you can show that

- Computing $apr(\alpha, v_0)$ takes time $O(\frac{1}{\epsilon\alpha})$
- $\sum_{v:p(v)>0} d(v) \leq \frac{1}{\epsilon\alpha}$

It can also be shown that if there is a small, low-conductance set of vertices that contains v_0, then for an appropriately chosen α and ϵ, the non-zero elements of p will contain that set.

3 Approximating PageRank on a very large graph

This suggests a scheme for approximating PageRank on a very large graph — one too large for even a complete vertex-weight vector to fit in memory. Compute $apr(\alpha, \epsilon, v_0)$ by repeatedly scanning through the adjacency-list of the graph. Whenever you scan past a node u with neighbors $v_1, \ldots, v_k$ in the stream, push u if $r(u)/d(u) > \epsilon$, and otherwise ignode u.

In more detail, let the graph be stored in a file where each line contains

$$u, d(u), v_1, \ldots, v_k$$

where the v_i's are the neighbors of u. The algorithm is then

- Let $p = 0$ and $r = \chi_{v_0}$.
- Repeat the following until no pushes are made in a complete scan:
  - For each line in the graph file
    - If $r(u)/d(u) > \epsilon$ then let $p, r = push(u, p, r)$

Finally, take the nodes that have non-zero weight in p, and include all the edges that are incident on these nodes. Since both p and r are sparse, they can be kept in memory.

4 Building a low-conductance subgraph

Some more notation:

- The "volume" of a set S is the number of edges incident on S, i.e.

$$volume(S) = \sum_{u \in S} d(u)$$

- The "boundary" of a set S are the edges from a node $u \in S$ to a node $v \notin S$.

$$boundary(S) \equiv \{(u, v) \in E : u \in S, v \notin S\}$$

- The "conductance of S" for a small set S is the fraction of edges in S that are on the boundary.

$$\Phi(S) = \frac{|boundary(S)|}{volume(S)}$$

  More generally

$$\Phi(S) = \frac{|boundary(S)|}{\min(volume(S), |E| - volume(S))}$$

Intuitively, if a node u is in a low-conductance set S that contains a seed node v_0, then it's plausible that u would have a high score in $pr(\alpha, \chi_{v_0})$. If that's true one way to find such a set would be the following.

- Let $S = \{v_0\}$ and let $S^* = S$
- For all nodes $u \neq v_0$, in decreasing order of the personalized PageRank score $p(u)$:
  - Add u to S.
  - If $\Phi(S) < \Phi(S*)$, then let $S^* = S$.
- Return S^*.

Andersen, Chung and Lang call this is operation "sweep", and show that it will find a small, low-conductance set S if one exists. Note that $boundary(S)$, and hence $\Phi(S)$, can be computed incrementally: $boundary(S + \{u\})$ is the edges in $boundary(S)$, after removing the set of edges that enter u, and adding the edges from u to any node $v \notin S + \{u\}$.

5 Data

An adjacent matrix of wikipedia concepts is available at `/afs/cs.cmu.edu/project/bigML/wikiGraph`. The file `outlink.adj` contains an adjacent matrix of wikipedia graph. Each line is a tab-separated list of wikipedia pages, where the first page has links to each one of the following pages. For debug purpose we provide another file `test.adj`, which is an subset of wikipedia graph.

6 Assignment

In this assignment we are going to implement the snowball algorithm and visualize the result of a few seeds.

A visualization software (Gephi) is available for download [1]. You may use other visualization tool as you like. Figure **??** gives an example result by using "Machine_learning" as seed node. You are expected to produce similar graph for this seed and other seeds.

Hint 1: to load data into gephi I find GDF format is the easiest[2]

Hint 2: Gephi's UI is a little bit confusing. I find 'ForceAtlas 2' with scaling=500 and gravity=500 gives reasonably good result. You can export figures from the preview tab.

Overall your program may have the following 4 steps:

- run approximated PageRank with a seed page s, and parameters α, ϵ.
- create a subgraph that involves nodes in $pr(\alpha, s)$
- do the "sweep" operation and find a small, low-conductance subgraph
- convert the low-conductance subgraph into the GDF format required by Gephi. For a node v use $max(1, \log(p(v)/\epsilon))$ as its node size.

7 Deliverables

Submit a compressed archive (zip, tar, etc) of your code. With it, include a makefile so that

[1] https://gephi.org/users/download

[2] https://gephi.org/users/supported-graph-formats/gdf-format

- The command *make pagerank* generates PageRank vector p for `test.adj`. Print out two tab-separated columns— nodes with non-zero weights in p, and their weights. Here we use seed="A", $\alpha = 0.3$ and $\epsilon = 10^{-5}$. Don't include the data file in your archive.

- The command *make snowball* generates the snowball subgraph for `test.adj`. Print out two tab-separated columns– nodes in the corresponding low-conductance subgraph, and their p values.

In addition, please include a pdf document with the output of your "make demo" command. Also include pdf documents containing a visualizations for

1. the seed "Machine_learning" with $\alpha = 0.3$ and $\epsilon = 10^{-5}$

2. two other seeds of your choice, with α and ϵ of your choice.

BONUS question: how sensitive is the result graph to parameters α and ϵ?

8 Marking breakdown

- Code correctness and makefile functionality [**70 points**].

- Question 1, 2 [**10+20 points**]

- BONUS question [**10 points**]