Overview of this week

- ~~Debugging tips for ML algorithms~~
- ~~Graph algorithms~~
  - ~~A prototypical graph algorithm: PageRank~~
    - ~~In memory~~
    - ~~Putting more and more on disk …~~
  - Sampling from a graph
    - What is a good sample? (~~graph statistics~~)
    - What methods work? (PPR/RWR)
    - HW: PageRank-Nibble method + Gephi

Common statistics for graphs

William Cohen

	network	type	n	m	z	ℓ	α	$C^{(1)}$	$C^{(2)}$
social	film actors	undirected	449 913	25 516 482	113.43	3.48	2.3	0.20	0.78
	company directors	undirected	7 673	55 392	14.44	4.60	–	0.59	0.88
	math c[illegible]	[illegible]directed	253 3[illegible]	496 489	3.92	7.57	–	0.15	0.34
	physics [illegible]	[illegible]ndirected	[illegible]2 909	245 [illegible]00	9.2[illegible]	6.19	–	0.45	0.56
	biology coauthorship	undirected	1 520 251	11 [illegible]03 064	[illegible].53	4.9[illegible]	–	0.0[illegible]	0.60
	telephone call graph	undire[illegible]	47 000 000	[illegible]0 000 000	3.16		2.1		
	email messages	[illegible]ted	59 912	86 30[illegible]	1.44	4.95	1.5/2.0		0.16
	email addres[illegible]	[illegible]d	16 8[illegible]	57 [illegible]29	3.3[illegible]	5.22	–	[illegible]	0.13
	student relat[illegible]	[illegible]cted	[illegible]73	477	[illegible].66	1[illegible]	[illegible]	[illegible]	0.001
	sexual contacts	undirected	2 810				[illegible]		
information	WWW `nd.edu`	directed	269 504	1 497 13[illegible]	5.55	1[illegible]	[illegible]	[illegible]	0.29
	WWW Altavista	directed	203 549 046	2 130 000 [illegible]0	10.46	16[illegible]	[illegible]		
	citation network	direc[illegible]	783 3[illegible]	6 7[illegible]	[illegible]		[illegible]		
	Roget's Thesaurus	[illegible]	[illegible]22	5[illegible]	[illegible]				
	word co-occurrenc[illegible]	[illegible]	[illegible]0 902	[illegible]7 000[illegible]	[illegible]				
technological	Internet	undirected	10 697	31[illegible]	[illegible]				
	power grid	undirected	4 9[illegible]	6[illegible]	[illegible]				
	train routes	undirec[illegible]	[illegible]7	19[illegible]	[illegible]				
	software packages	[illegible]	[illegible]439	1[illegible]	[illegible]				
	software classes	[illegible]	1 377	2[illegible]	[illegible]				
	electronic circuits	undirected	24 097	53[illegible]	[illegible]				
	peer-to-peer network	undirect[illegible]	880	1[illegible]	[illegible]				
biological	metabolic network	undir[illegible]	765	3[illegible]	[illegible]				
	protein interactions	u[illegible]ed	2 115	2[illegible]	[illegible]				
	marine food [illegible]	[illegible]	135	[illegible]	[illegible]				
	freshwater fo[illegible]	[illegible]	92	[illegible]	[illegible]				
	neural netwo[illegible]	[illegible]	307	2[illegible]	[illegible]				

nodes

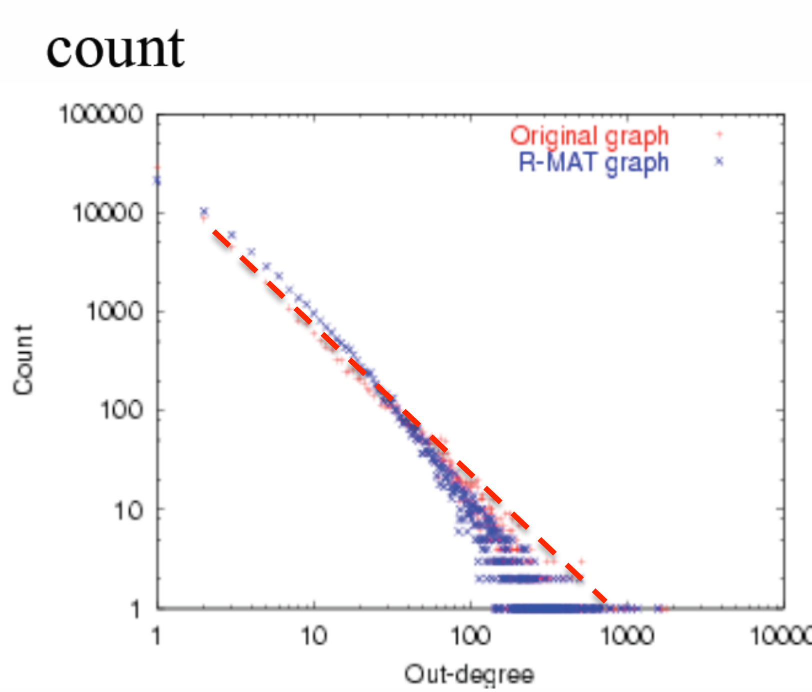
edges

avg degree

diameter

clustering coefficient (homophily)

cooefficient of degree curve



An important question

- How do you explore a dataset?
  - compute statistics (e.g., feature histograms, conditional feature histograms, correlation coefficients, ...)
  - sample and inspect
    - run a bunch of small-scale experiments
- How do you explore a graph?
  - compute statistics (degree distribution, ...)
  - sample and inspect
    - how do you sample?

Overview of this week

- Debugging tips for ML algorithms
- Graph algorithms
  - A prototypical graph algorithm: PageRank
    - In memory
    - Putting more and more on disk …
  - Sampling from a graph
    - What is a good sample? (graph statistics)
    - **What sampling methods work? (PPR/RWR)**
    - HW: PageRank-Nibble method + Gephi

Sampling from Large Graphs

Jure Leskovec
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA
jure@cs.cmu.edu

Christos Faloutsos
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA
christos@cs.cmu.edu

KDD 2006

Brief summary

- Define *goals* of sampling:
  - “scale-down” – find G’<G with similar statistics
  - “back in time”: for a growing G, find G’<G that is similar (statistically) to an earlier version of G
- Experiment on real graphs with plausible sampling methods, such as
  - RN – random nodes, sampled uniformly
  - …
- See how well they perform

Brief summary

- Experiment on real graphs with plausible sampling methods, such as
  - RN – random nodes, sampled uniformly
    - RPN – random nodes, sampled by PageRank
    - RDP – random nodes sampled by in-degree
  - RE – random edges
  - RJ – run PageRank's "random surfer" for *n* steps
  - RW – run RWR's "random surfer" for *n* steps
  - FF – repeatedly pick *r(i)* neighbors of *i* to "burn", and then recursively sample from them

RWR/Personalized PageRank vs PR

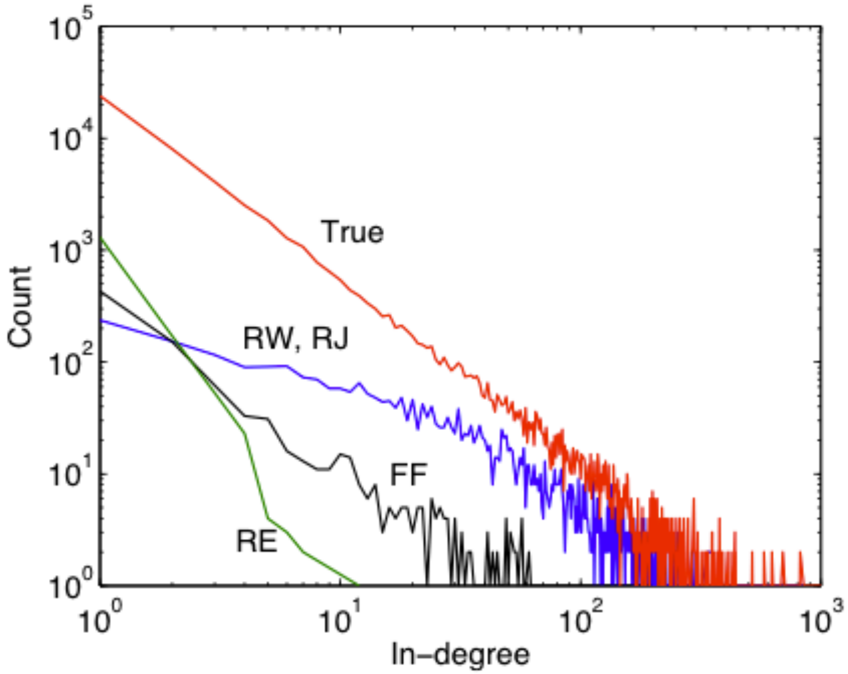
- PageRank update:

$$\text{Let } \mathbf{v}^{t+1} = c\mathbf{u} + (1\text{-}c)\mathbf{W}\mathbf{v}^{t}$$

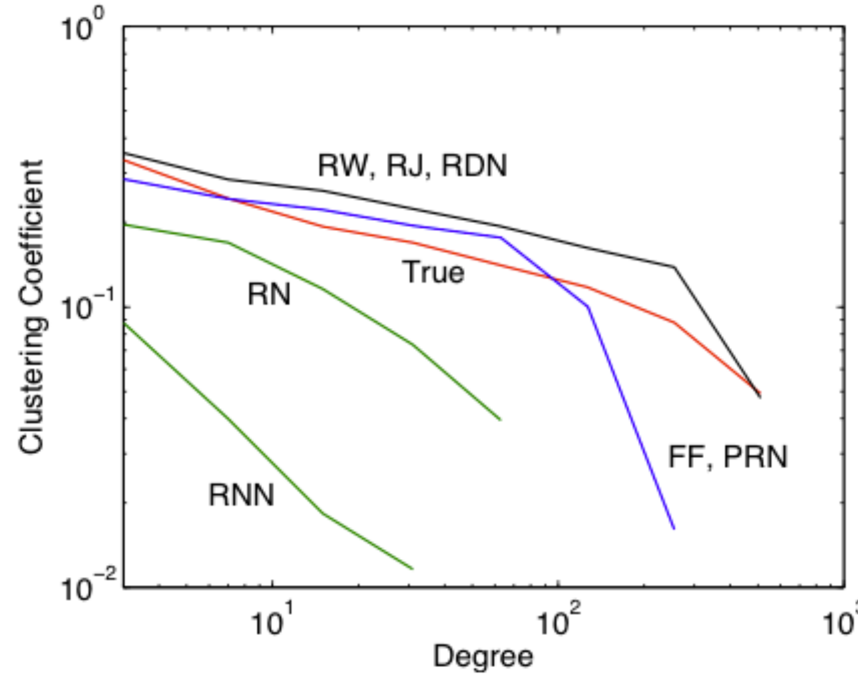
- Personalized PR/RWR update:

$$\text{Let } \mathbf{v}^{t+1} = c\mathbf{s} + (1\text{-}c)\mathbf{W}\mathbf{v}^{t}$$

s is the seed vector or **personalization** vector
in RN it's just a random unit vector

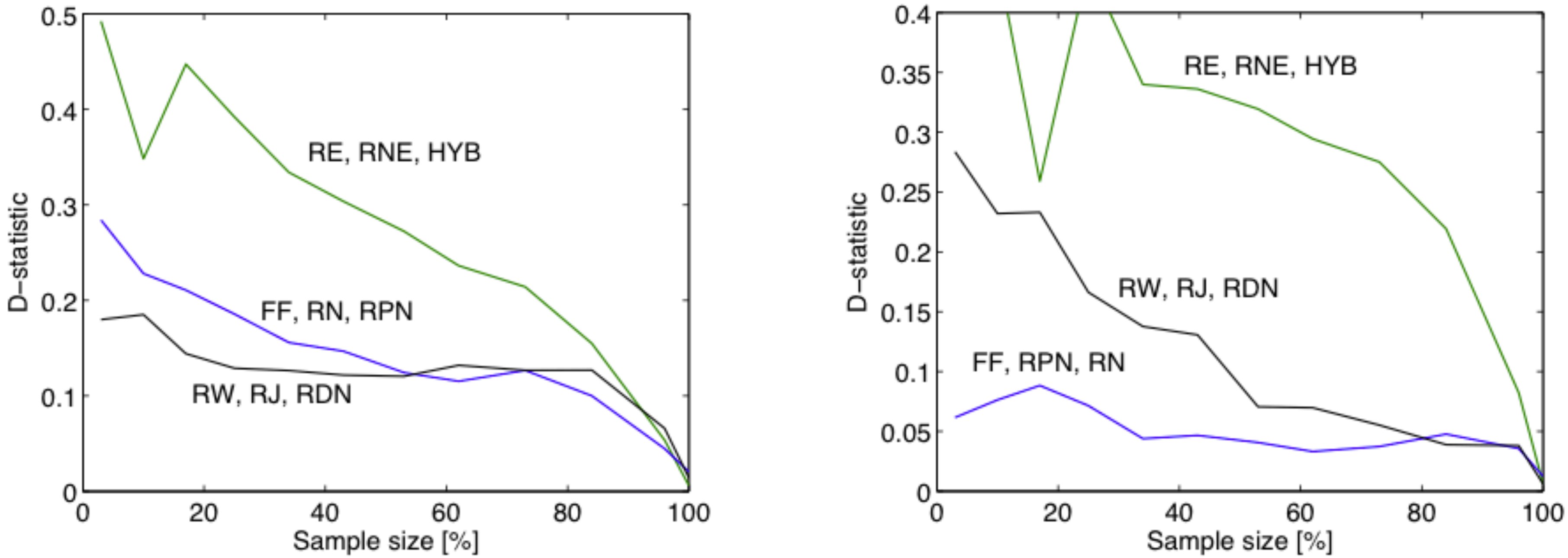


(a) S1: In-degree



(b) S9: Clustering coef.

10% sample – pooled on five datasets



(a) Scale-down (b) Back-in-time

d-statistic measures agreement between distributions

- D=max{|F(x)-F’(x)|} where F, F’ are cdf’s
- max over nine different statistics

	Static graph patterns							
	in-deg	out-deg	wcc	scc	hops	sng-val	sng-vec	clust
RN	0.084	0.145	0.814	0.193	0.231	0.079	0.112	0.327
RPN	**0.062**	**0.097**	0.792	0.194	**0.200**	0.048	0.081	0.243
RDN	0.110	0.128	0.818	0.193	0.238	0.041	0.048	0.256
RE	0.216	0.305	**0.367**	0.206	0.509	0.169	0.192	0.525
RNE	0.277	0.404	0.390	0.224	0.702	0.255	0.273	0.709
HYB	0.273	0.394	0.386	0.224	0.683	0.240	0.251	0.670
RNN	0.179	0.014	0.581	0.206	0.252	0.060	0.255	0.398
RJ	0.132	0.151	0.771	0.215	0.264	0.076	0.143	**0.235**
RW	0.082	0.131	0.685	0.194	0.243	0.049	**0.033**	**0.243**
FF	0.082	0.105	0.664	0.194	**0.203**	**0.038**	0.092	**0.244**

Goal

- An efficient way of running RWR on a large graph
  - can use only "random access"
    - you can ask about the neighbors of a node, you can't scan thru the graph
    - common situation with APIs
  - leads to a plausible sampling strategy
    - Jure & Christos's experiments
    - some formal results that justify it….

Local Graph Partitioning using PageRank Vectors

Reid Andersen
University of California, San Diego

Fan Chung
University of California, San Diego

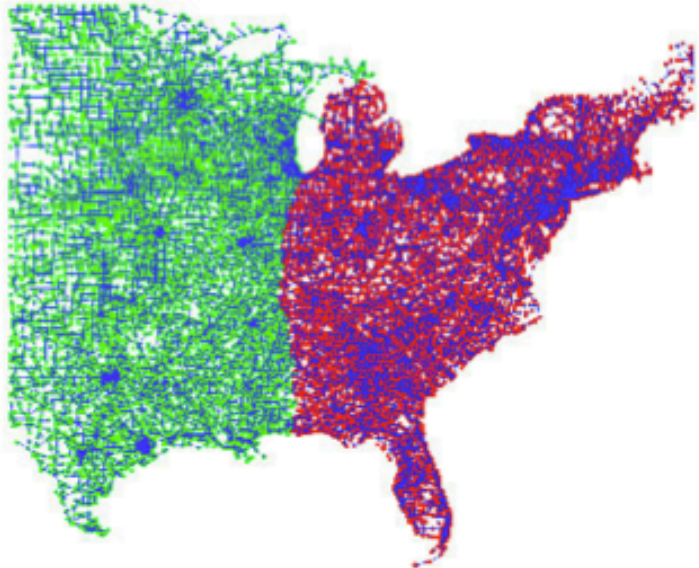
Kevin Lang
Yahoo! Research

FOCS 2006

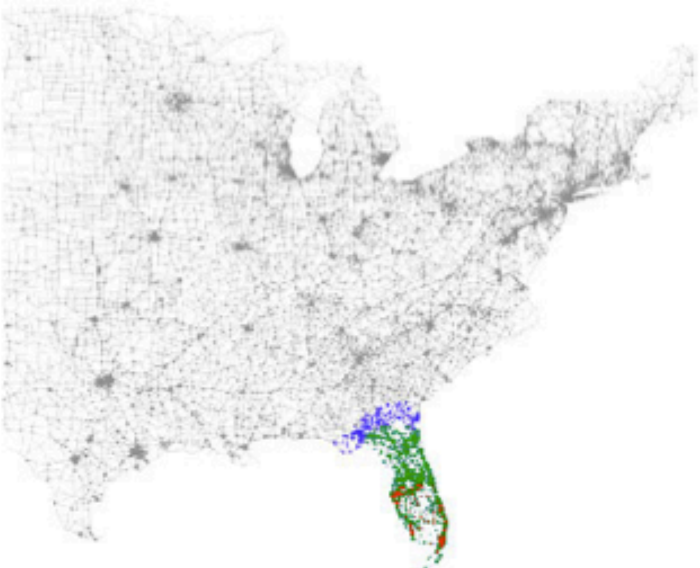
What is Local Graph Partitioning?

A local graph partitioning algorithm finds a small cut near the given seed(s) with running time depending only on the size of the output.

Global

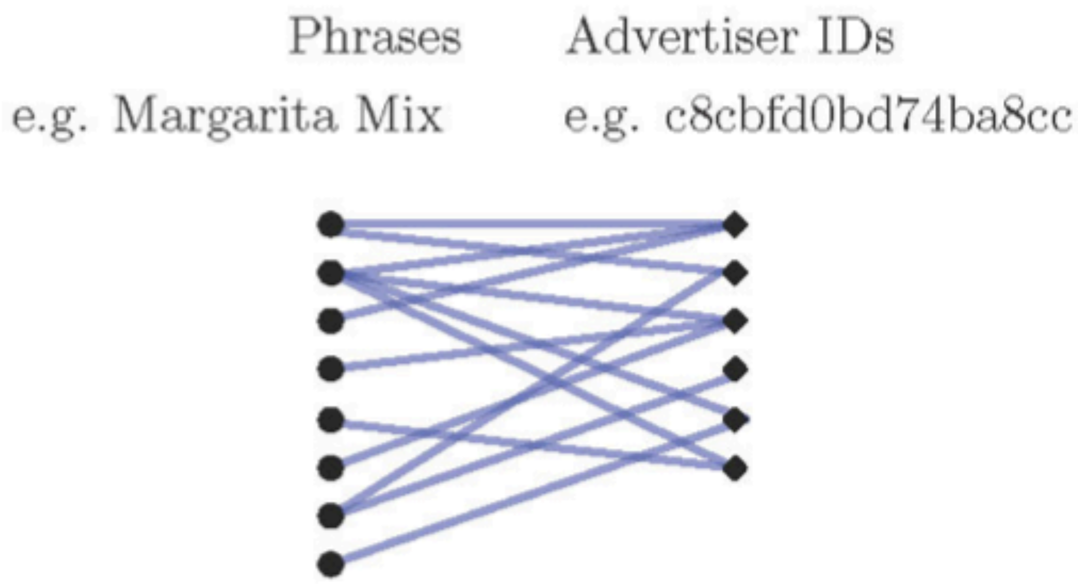


Local



What is Local Graph Partitioning?

A bidding graph from Yahoo sponsored search



On the left are search phrases, on the right are advertisers. Each edge represents a bid by an advertiser on a phrase.

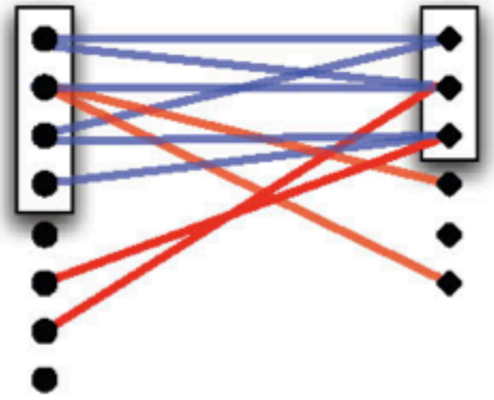
$400K$ phrases, $200K$ advertisers, and 2 million edges.

What is Local Graph Partitioning?

Submarkets in bidding graph

The bidding graph has submarkets, sets of bidders and phrases that interact mostly with each other.

Phrases about margarita mix   Purveyors of margarita mix

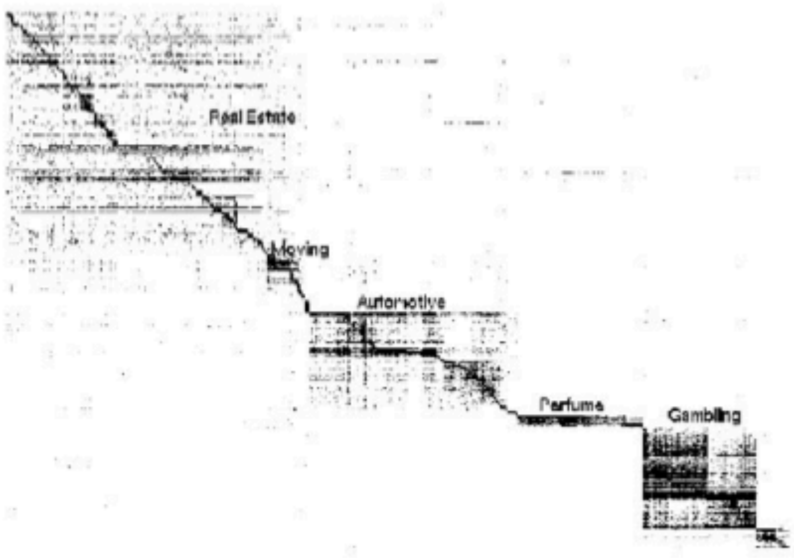


These sets of vertices (containing both advertisers and phrases) have small conductance.

What is Local Graph Partitioning?

Submarkets in the bigging graph

The bidding graph has numerous submarkets, related to real estate, flower delivery, hotels, gambling, ...



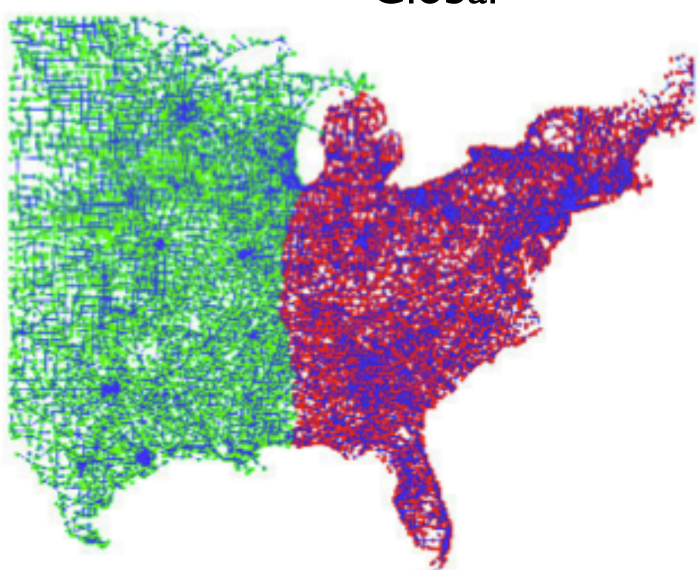
It is useful to identify these submarkets.

- Find groups of related phrases to suggest to advertisers.
- Find small submarkets for testing and experimentation.

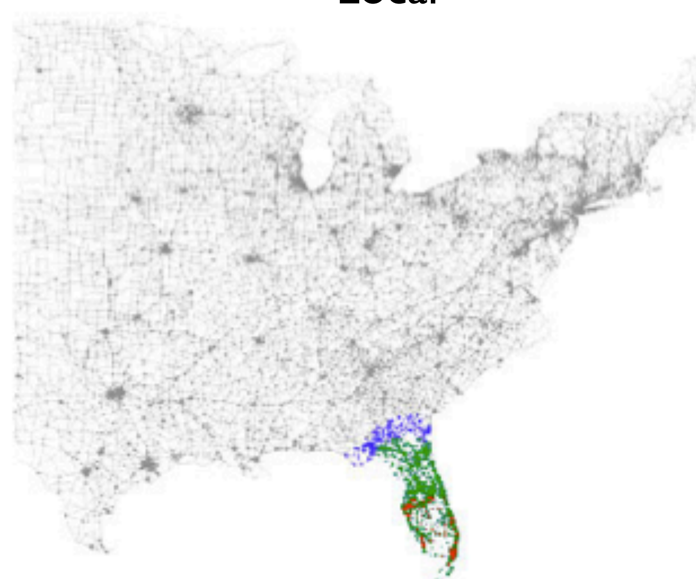
What is Local Graph Partitioning?

A local graph partitioning algorithm finds a small cut near the given seed(s) with running time depending only on the size of the output.

Global

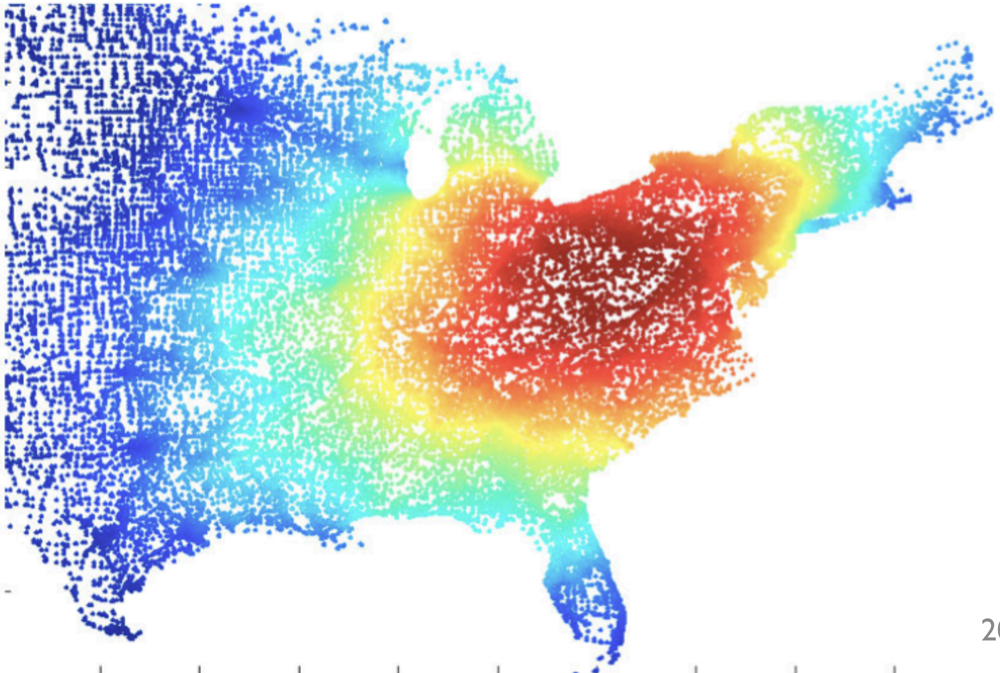


Local



Key idea: a "sweep"

- Order all vertices in some way $v_{i,1}$, $v_{i,2}$, ….
  - Say, by personalized PageRank from a seed
- Pick a prefix $v_{i,1}$, $v_{i,2}$, …. $v_{i,k}$ that is "best"
  - ….



What is a “good” subgraph?

$$\partial(S) = \{\{x, y\} \in E \mid x \in S, y \notin S\}$$

the edges leaving S

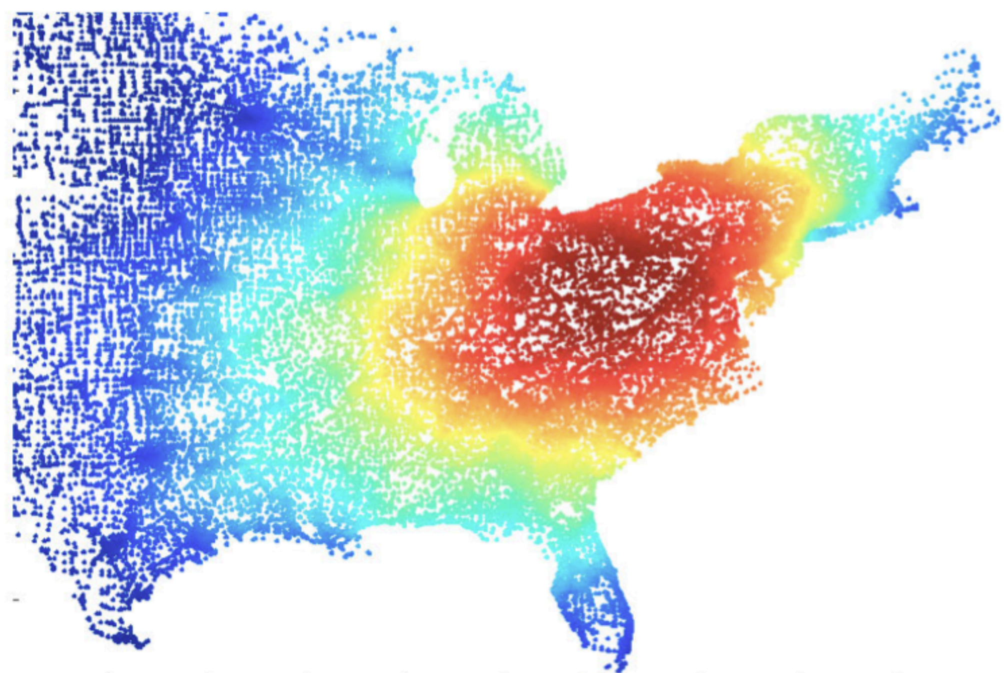
$$\Phi(S) = \frac{|\partial(S)|}{\min\left(\mathrm{vol}(S), 2m - \mathrm{vol}(S)\right)}.$$

- vol(S) is sum of deg(x) for x in S
- for small S: Prob(random *edge* leaves S)

Key idea: a "sweep"

- Order all vertices in some way $v_{i,1}$, $v_{i,2}$, ...
  - Say, by personalized PageRank from a seed
- Pick a prefix S={ $v_{i,1}$, $v_{i,2}$, ... $v_{i,k}$ } that is "best"
  - Minimal "conductance" $\phi(S)$

You can re-compute conductance incrementally as you add a new vertex so the sweep is fast



Main results of the paper

1. An *approximate* personalized PageRank computation that only touches nodes "near" the seed
   – but has small error relative to the true PageRank vector
2. A proof that a sweep over the approximate PageRank vector finds a cut with conductance sqrt(α ln m)
   – unless no good cut exists
     - no subset S contains significantly more pass in the approximate PageRank than in a uniform distribution

	Static graph patterns							
	in-deg	out-deg	wcc	scc	hops	sng-val	sng-vec	clust
RN	0.084	0.145	0.814	0.193	0.231	0.079	0.112	0.327
RPN	**0.062**	**0.097**	0.792	0.194	**0.200**	0.048	0.081	0.243
RDN	0.110	0.128	0.818	0.193	0.238	0.041	0.048	0.256
RE	0.216	0.305	**0.367**	0.206	0.509	0.169	0.192	0.525
RNE	0.277	0.404	0.390	0.224	0.702	0.255	0.273	0.709
HYB	0.273	0.394	0.386	0.224	0.683	0.240	0.251	0.670
RNN	0.179	0.014	0.581	0.206	0.252	0.060	0.255	0.398
RJ	0.132	0.151	0.771	0.215	0.264	0.076	0.143	**0.235**
RW	0.082	0.131	0.685	0.194	0.243	0.049	**0.033**	**0.243**
FF	0.082	0.105	0.664	0.194	**0.203**	**0.038**	0.092	**0.244**

Result 2 explains Jure & Christos's experimental results with RW sampling:

- RW approximately picks up a *random subcommunity* (maybe with some extra nodes)
- Features like clustering coefficient, degree should be representative of the graph as a whole…
  - which is roughly a mixture of subcommunities

Main results of the paper

1. An *approximate* personalized PageRank computation that only touches nodes “near” the seed
   - but has small error relative to the true PageRank vector

This is a very useful technique to know about…

Random Walks

G : a graph

P : transition probability matrix

$$P(u,v) = \begin{cases} \frac{1}{d_u} & \text{if } u : v, \\ 0 & \text{otherwise.} \end{cases}$$

d_u := the degree of u.

A lazy walk: $$W = \frac{I+P}{2}$$

avoids messy "dead ends"….

Random Walks: PageRank

A (bored) surfer

- either surf a random webpage

  with probability α

- or surf a linked webpage

  with probability $1 - \alpha$

α : the jumping constant

$$p = \alpha\left(\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n}\right) + (1-\alpha)\,pW$$

Random Walks: PageRank

Two equivalent ways to define PageRank $p=pr(\alpha,s)$

(1) $$p = \alpha s + (1-\alpha) pW$$

(2) $$p = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t (sW^t)$$

$s = (\frac{1}{n}, \frac{1}{n}, \ldots., \frac{1}{n})$ ⟹ the (original) PageRank

$s =$ some "seed", e.g., $(1, 0, \ldots., 0)$

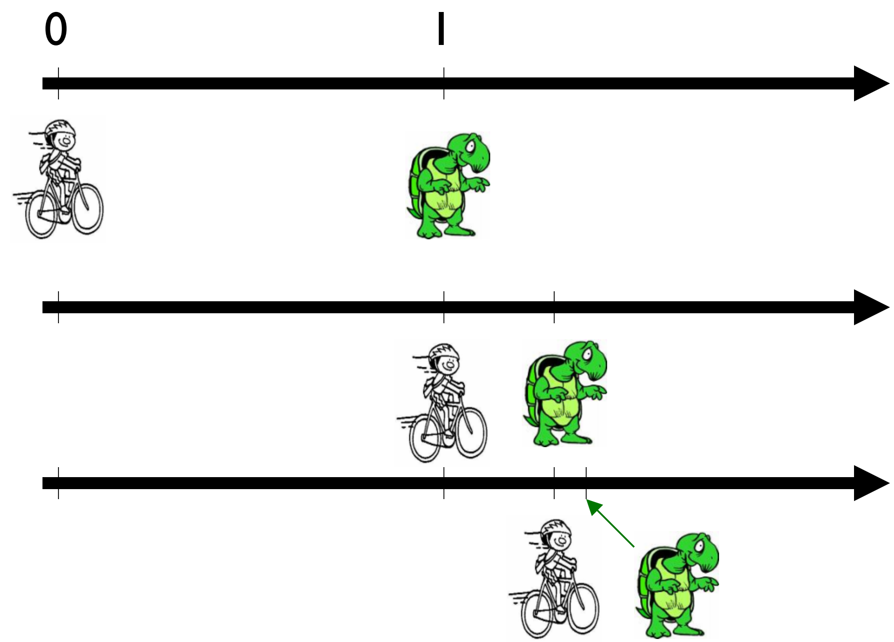
⟹ personalized PageRank

Flashback: Zeno's paradox

- Lance Armstrong and the tortoise have a race
- Lance is 10x faster
- Tortoise has a 1m head start at time 0

- So, when Lance gets to 1m the tortoise is at 1.1m
- So, when Lance gets to 1.1m the tortoise is at 1.11m …
- So, when Lance gets to 1.11m the tortoise is at 1.111m … and Lance will *never* catch up -?

1+0.1+0.01+0.001+0.0001+… = ?



unresolved until calculus was invented

Zeno: powned by telescoping sums

Let *x* be less than 1. Then

$$y = 1 + x + x^2 + x^3 + \ldots + x^n$$

$$y(1-x) = (1 + x + x^2 + x^3 + \ldots + x^n)(1-x)$$

$$y(1-x) = (1 - \cancel{x}) + (\cancel{x} - \cancel{x^2}) + (\cancel{x^2} - x^3) + \ldots + (x^n - x^{n+1})$$

$$y(1-x) = 1 - x^{n+1}$$

$$y = \frac{1 - x^{n+1}}{(1-x)}$$

$$y \approx (1-x)^{-1}$$

Example: x=0.1, and 1+0.1+0.01+0.001+…. = 1.11111 = 10/9.

Graph = Matrix

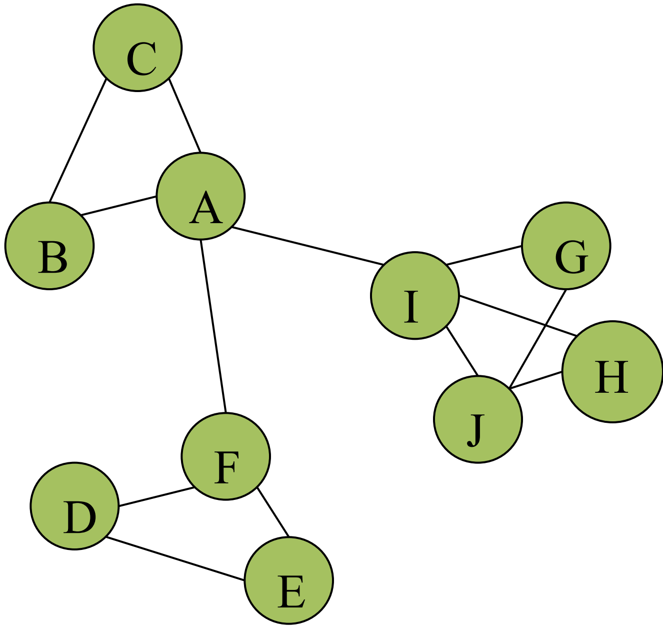
Vector = Node → Weight

M

	A	B	C	D	E	F	G	H	I	J
A	_	1	1			1				
B	1	_	1							
C	1	1	_							
D				_	1	1				
E				1	_	1				
F	1			1	1	_				
G							_		1	1
H								_	1	1
I							1	1	_	1
J							1	1	1	_

v

	A
A	3
B	2
C	3
D	
E	
F	
G	
H	
I	
J	



M

Racing through a graph?

Let W[*i*,*j*] **be Pr(walk to *j* from *i*)and let *α* be less than 1. Then:**

$$\mathbf{Y} = \mathbf{I} + \alpha\mathbf{W} + (\alpha\mathbf{W})^2 + (\alpha\mathbf{W})^3 + \ldots(\alpha\mathbf{W})^n$$

$$\mathbf{Y}(\mathbf{I} - \alpha\mathbf{W}) = (\mathbf{I} + \alpha\mathbf{W} + (\alpha\mathbf{W})^2 + (\alpha\mathbf{W})^3 + \ldots)(\mathbf{I} - \alpha\mathbf{W})$$

$$\mathbf{Y}(\mathbf{I} - \alpha\mathbf{W}) = (\mathbf{I} - \cancel{\alpha\mathbf{W}}) + (\cancel{\alpha\mathbf{W}} - \cancel{(\alpha\mathbf{W})^2} + \ldots)(\mathbf{I} - \alpha\mathbf{W})$$

$$\mathbf{Y}(\mathbf{I} - \alpha\mathbf{W}) = \mathbf{I} - (\alpha\mathbf{W})^{n+1}$$

$$\mathbf{Y} \approx (\mathbf{I} - \alpha\mathbf{W})^{-1} \qquad \mathbf{Y}[i, j] = \frac{1}{Z}\Pr(j \mid i)$$

The matrix **(I- *α*W)** is the *Laplacian* of *α***W.**

Generally the Laplacian is **(D- A)** where **D**[*i*,*i*] is the degree of *i* in the adjacency matrix **A.**

Random Walks: PageRank

Two equivalent ways to define PageRank $p = pr(\alpha, s)$

(1) $$p = \alpha s + (1-\alpha) p W$$

(2) $$p = \alpha \sum_{t=0}^{\infty} (1-\alpha)^t (s W^t)$$

$s = (\frac{1}{n}, \frac{1}{n}, \ldots, \frac{1}{n})$ ⟹ the (original) PageRank

$s =$ some "seed", e.g., $(1, 0, \ldots, 0)$

⟹ personalized PageRank

Approximate PageRank: Key Idea

By definition PageRank is fixed point of:

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

Claim:

$$\mathrm{pr}(\alpha, s) = \alpha s + (1-\alpha)\mathrm{pr}(\alpha, sW).$$

Proof:

define a matrix for the pr operator: $R_\alpha s = \mathrm{pr}(\alpha, s)$

$$\begin{aligned} R_\alpha &= \alpha \sum_{t=0}^{\infty} (1-\alpha)^t W^t \\ &= \alpha \left(I + \sum_{u=1}^{\infty} (1-\alpha)^u W^u \right) \\ &= \alpha I + (1-\alpha) W \sum_{t=0}^{\infty} (1-\alpha)^t W^t \\ &= \alpha I + (1-\alpha) W R_\alpha \end{aligned}$$

Approximate PageRank: Key Idea

By definition PageRank is fixed point of:

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

Claim:

$$\mathrm{pr}(\alpha, s) = \alpha s + (1-\alpha)\mathrm{pr}(\alpha, sW).$$

Proof:

$$\begin{aligned} R_\alpha &= \alpha \sum_{t=0}^{\infty} (1-\alpha)^t W^t \\ &= \alpha I + (1-\alpha) W R_\alpha. \end{aligned}$$

$$\begin{aligned} \mathrm{pr}(\alpha, s) &= s R_\alpha \\ &= \alpha s + (1-\alpha) \underline{sW} R_\alpha \\ &= \alpha s + (1-\alpha)\mathrm{pr}(\alpha, sW). \end{aligned}$$

Approximate PageRank: Key Idea

By definition PageRank is fixed point of:

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

Claim:

$$\text{pr}(\alpha, s) = \alpha s + (1 - \alpha)\text{pr}(\alpha, sW).$$

Recursively compute PageRank of "neighbors of *s*" *(=sW)*, then adjust

Key idea in apr:

- do this "recursive step" repeatedly
- focus on nodes where finding PageRank from neighbors will be useful

Approximate PageRank: Key Idea

$$\mathrm{pr}(\alpha, s) = \alpha s + (1-\alpha)\mathrm{pr}(\alpha, sW). \quad W = \frac{I+P}{2}$$

$\mathrm{push}_u(p, r)$:

1. Let $p' = p$ and $r' = r$, except for the following changes:
   (a) $p'(u) = p(u) + \alpha r(u)$.
   (b) $r'(u) = (1-\alpha)r(u)/2$.
   (c) For each v such that $(u, v) \in E$: $r'(v) = r(v) + (1-\alpha)r(u)/(2d(u))$.
2. Return (p', r').

- *p* is current approximation (start at **0**)
- *r* is set of "recursive calls to make"
  - residual error
  - start with all mass on *s*
- *u* is the node picked for the next call

Analysis

Lemma 1. *Let p' and r' be the result of the operation* $\mathbf{push}_u$ *on p and r. Then*

$$p' + \mathrm{pr}(\alpha, r') = p + \mathrm{pr}(\alpha, r).$$

Proof of Lemma 1. After the push operation, we have

$$\begin{aligned} p' &= p + \alpha r(u)\chi_u. \\ r' &= r - r(u)\chi_u + (1-\alpha)r(u)\chi_u W. \end{aligned}$$

Using equation (5),

linearity

$$\begin{aligned} p + \mathrm{pr}(\alpha, r) &= p + \mathrm{pr}(\alpha, r - r(u)\chi_u) + \mathrm{pr}(\alpha, r(u)\chi_u) \\ &= p + \mathrm{pr}(\alpha, r - r(u)\chi_u) + [\alpha r(u)\chi_u + (1-\alpha)\mathrm{pr}(\alpha, r(u)\chi_u W)] \\ &= [p + \alpha r(u)\chi_u] + \mathrm{pr}(\alpha, [r - r(u)\chi_u + (1-\alpha)r(u)\chi_u W]) \\ &= p' + \mathrm{pr}(\alpha, r'). \end{aligned}$$

re-group & linearity

$$pr(\alpha, r - r(u)\chi_u) + (1-\alpha)\, pr(\alpha, r(u)\chi_u W) = pr(\alpha, r - r(u)\chi_u + (1-\alpha)\ r(u)\chi_u W)$$

$$\mathrm{pr}(\alpha, s) = \alpha s + (1-\alpha)\mathrm{pr}(\alpha, sW). \quad (5)$$

Approximate PageRank: Algorithm

`ApproximatePageRank` (v, α, ϵ):

1. Let $p = \vec{0}$, and $r = \chi_v$.
2. While $\max_{u \in V} \frac{r(u)}{d(u)} \geq \epsilon$:
   (a) Choose any vertex u where $\frac{r(u)}{d(u)} \geq \epsilon$.
   (b) Apply $\mathbf{push}_u$ at vertex u, updating p and r.
3. Return p, which satisfies $p = \text{apr}(\alpha, \chi_v, r)$ with $\max_{u \in V} \frac{r(u)}{d(u)} < \epsilon$.

$\mathbf{push}_u(p, r)$:

1. Let $p' = p$ and $r' = r$, except for the following changes:
   (a) $p'(u) = p(u) + \alpha r(u)$.
   (b) $r'(u) = (1 - \alpha) r(u)/2$.
   (c) For each v such that $(u, v) \in E$: $r'(v) = r(v) + (1 - \alpha) r(u)/(2d(u))$.
2. Return (p', r').

Analysis

Lemma 1. *Let p' and r' be the result of the operation* $\mathbf{push}_u$ *on p and r. Then*

$$p' + \mathrm{pr}(\alpha, r') = p + \mathrm{pr}(\alpha, r).$$

So, at every point in the *apr* algorithm:

$$p + \mathrm{pr}(\alpha, r) = \mathrm{pr}(\alpha, \chi_v),$$

Also, at each point, $|r|_1$ **decreases** by $\alpha * \varepsilon$ *degree(*u*), so: after T push operations where degree(i-th u)=d_i, we know

$$\sum_i d_i \cdot \alpha\varepsilon \leq 1 \quad \Rightarrow \quad \sum_{i=1}^{T} d_i \leq \frac{1}{\epsilon\alpha}.$$

which bounds the size of *r* and *p*

Analysis

Theorem 1. `ApproximatePageRank`(v, α, ϵ) *runs in time* $O(\frac{1}{\epsilon\alpha})$, *and computes an approximate PageRank vector* $p = \mathrm{apr}(\alpha, \chi_v, r)$ *such that the residual vector* r *satisfies* $\max_{u \in V} \frac{r(u)}{d(u)} < \epsilon$, *and such that* $\mathrm{vol}(\mathrm{Supp}(p)) \leq \frac{1}{\epsilon\alpha}$.

With the invariant: $$p + \mathrm{pr}(\alpha, r) = \mathrm{pr}(\alpha, \chi_v),$$

This bounds the error of p relative to the PageRank vector.

Comments – API

ApproximatePageRank (v, α, ϵ):

1. Let $p = \vec{0}$, and $r = \chi_v$.
2. While $\max_{u \in V} \frac{r(u)}{d(u)} \geq \epsilon$:
   (a) Choose any vertex u where $\frac{r(u)}{d(u)} \geq \epsilon$.
   (b) Apply $\mathbf{push}_u$ at vertex u, updating p and r.
3. Return p, which satisfies $p = \mathrm{apr}(\alpha, \chi_v, r)$ with $\max_{u \in V} \frac{r(u)}{d(u)} < \epsilon$.

p,r are hash tables – they are small (1/εα)

Could implement with API:

- List<Node> neighbor(Node *u*)
- int degree(Node *u*)

$\mathbf{push}_u(p, r)$:

push just needs *p, r,* and neighbors of *u*

1. Let $p' = p$ and $r' = r$, except for the following changes:
   (a) $p'(u) = p(u) + \alpha r(u)$.
   (b) $r'(u) = (1 - \alpha)r(u)/2$.
   (c) For each v such that $(u, v) \in E$: $r'(v) = r(v) + (1 - \alpha)r(u)/(2d(u))$.
2. Return (p', r').

d(v) = api.degree(*v*)

Comments - Ordering

ApproximatePageRank (v, α, ϵ):

1. Let $p = \vec{0}$, and $r = \chi_v$.
2. While $\max_{u \in V} \frac{r(u)}{d(u)} \geq \epsilon$:
   (a) Choose any vertex u where $\frac{r(u)}{d(u)} \geq \epsilon$.
   (b) Apply $\mathbf{push}_u$ at vertex u, updating p and r.
3. Return p, which satisfies $p = \mathrm{apr}(\alpha, \chi_v, r)$ with $\max_{u \in V} \frac{r(u)}{d(u)} < \epsilon$.

might pick the **largest** *r(u)/d(u)* … ***or…***

push$_u(p, r)$:

1. Let $p' = p$ and $r' = r$, except for the following changes:
   (a) $p'(u) = p(u) + \alpha r(u)$.
   (b) $r'(u) = (1 - \alpha) r(u)/2$.
   (c) For each v such that $(u, v) \in E$: $r'(v) = r(v) + (1 - \alpha) r(u)/(2d(u))$.
2. Return (p', r').

Comments – Ordering for Scanning

ApproximatePageRank (v, α, ϵ):

1. Let $p = \vec{0}$, and $r = \chi_v$.
2. While $\max_{u \in V} \frac{r(u)}{d(u)} \geq \epsilon$:

Scan repeatedly through an adjacency-list encoding of the graph

For every line you read $u, v_1, \ldots, v_{d(u)}$ such that $r(u)/d(u) > \varepsilon$:

(b) Apply $\mathbf{push}_u$ at vertex u, updating p and r.

3. Return p, which satisfies $p = \text{apr}(\alpha, \chi_v, r)$ with $\max_{u \in V} \frac{r(u)}{d(u)} < \epsilon$.

benefit: storage is $O(1/\varepsilon\alpha)$ for the hash tables, avoids any *seeking*

Possible optimizations?

- Much faster than doing random access the first few scans, but then slower the last few
  - …there will be only a few ‘pushes’ per scan
- Optimizations you might imagine:
  - Parallelize?
  - Hybrid seek/scan:
    - Index the nodes in the graph on the first scan
    - Start seeking when you expect too few pushes to justify a scan
      - Say, less than one push/megabyte of scanning
  - Hotspots:
    - Save adjacency-list representation for nodes with a large $r(u)/d(u)$ in a separate file of “hot spots” as you scan
    - Then rescan that smaller list of “hot spots” until their score drops below threshold.

Putting this together

- Given a graph
  - that's too big for memory, and/or
  - that's only accessible via API
- …we can extract a *sample* in an interesting area
  - Run the apr/rwr from a seed node
  - Sweep to find a low-conductance subset
- Then
  - compute statistics
  - test out some ideas
  - visualize it…