Local community structure in social & information networks

Joint with
David F. Gleich,
(Purdue), supported by
NSF CAREER
1149756-CCF

PURDUE UNIVERSITY

Kyle Kloster
Purdue University

Network analysis via Heat Kernel

Uses include
- Local community detection
- Link prediction
- Node centrality

What **is** it?

a graph diffusion
a function of a matrix

$$\exp(\boldsymbol{G}) = \sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{G}^k$$

For **G**, a network's $\left(\begin{array}{ll} \text{random-walk,} & \mathbf{P} \\ \text{adjacency,} & \mathbf{A} \\ \text{Laplacian,} & \mathbf{L} \end{array} \right)$ matrix

Heat Kernel describes node connectivity

$$(\boldsymbol{A}^k)_{ij} = \#\ \text{walks of length k from node } i \text{ to } j$$

$$\exp(\boldsymbol{A})_{ij} = \sum_{k=0}^{\infty} \frac{1}{k!} (\boldsymbol{A}^k)_{ij}$$

"sum up" the walks between *i* and *j*

For a small set of seed nodes, **s**, $\exp(\boldsymbol{A})\,\mathbf{s}$ describes nodes most relevant to **s**

Big data, big slowdown

$$\exp(\boldsymbol{A})\,\mathbf{s} = \sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{A}^k \mathbf{s}$$

each term takes
O(|E|) work!

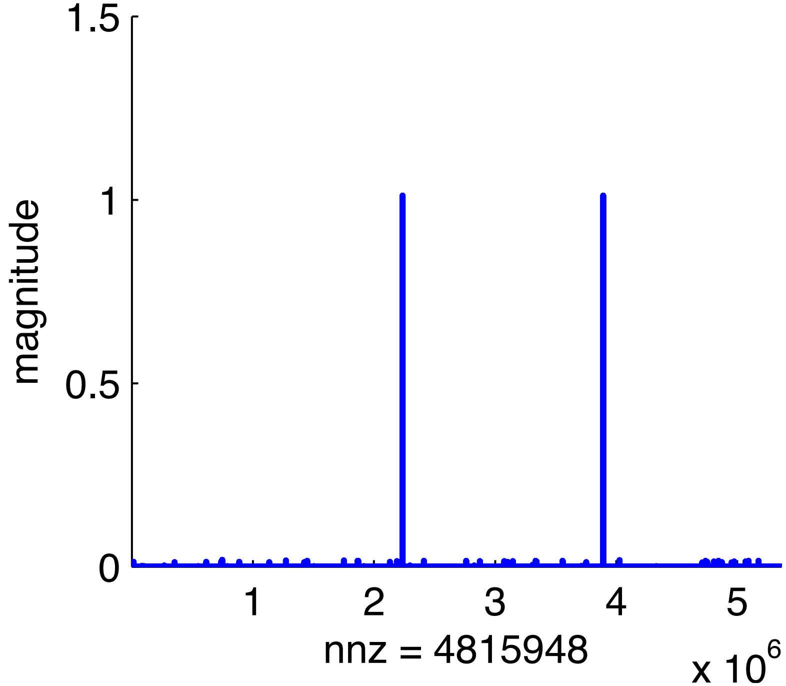
And real-world
networks are

diameter ~ 4
~ O(10^9) #nodes
~ O(10^{10}) #edges = |E|
constantly changing

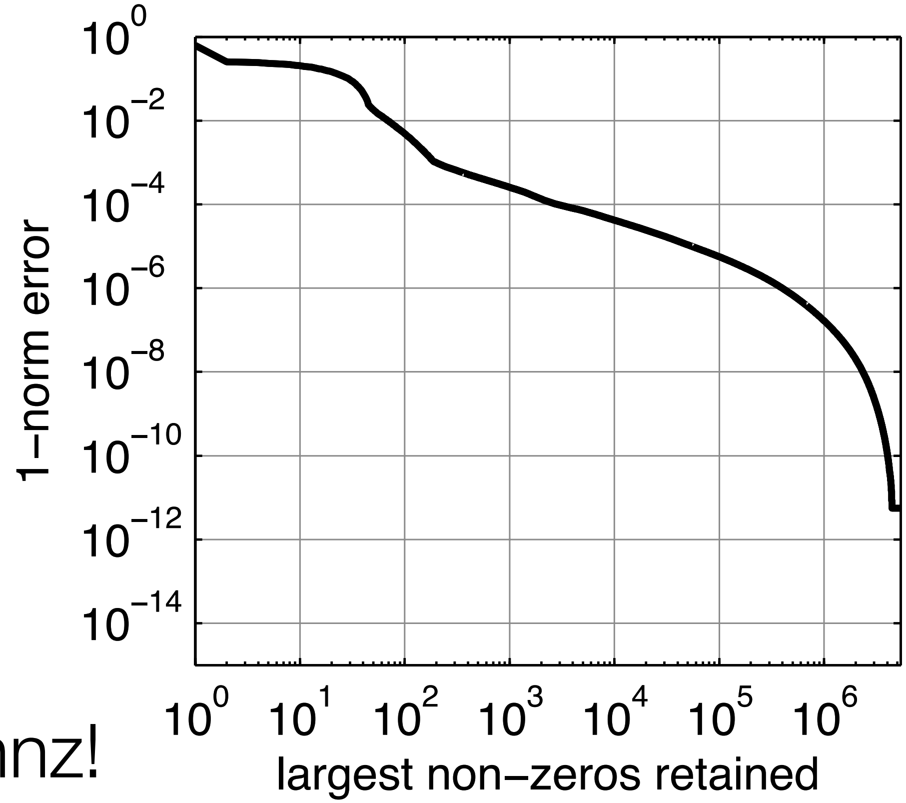
→ **speed** is a priority over accuracy

Local solution, big speedup

Magnitude of entries in solution vector



Accuracy of approximation using only large entries

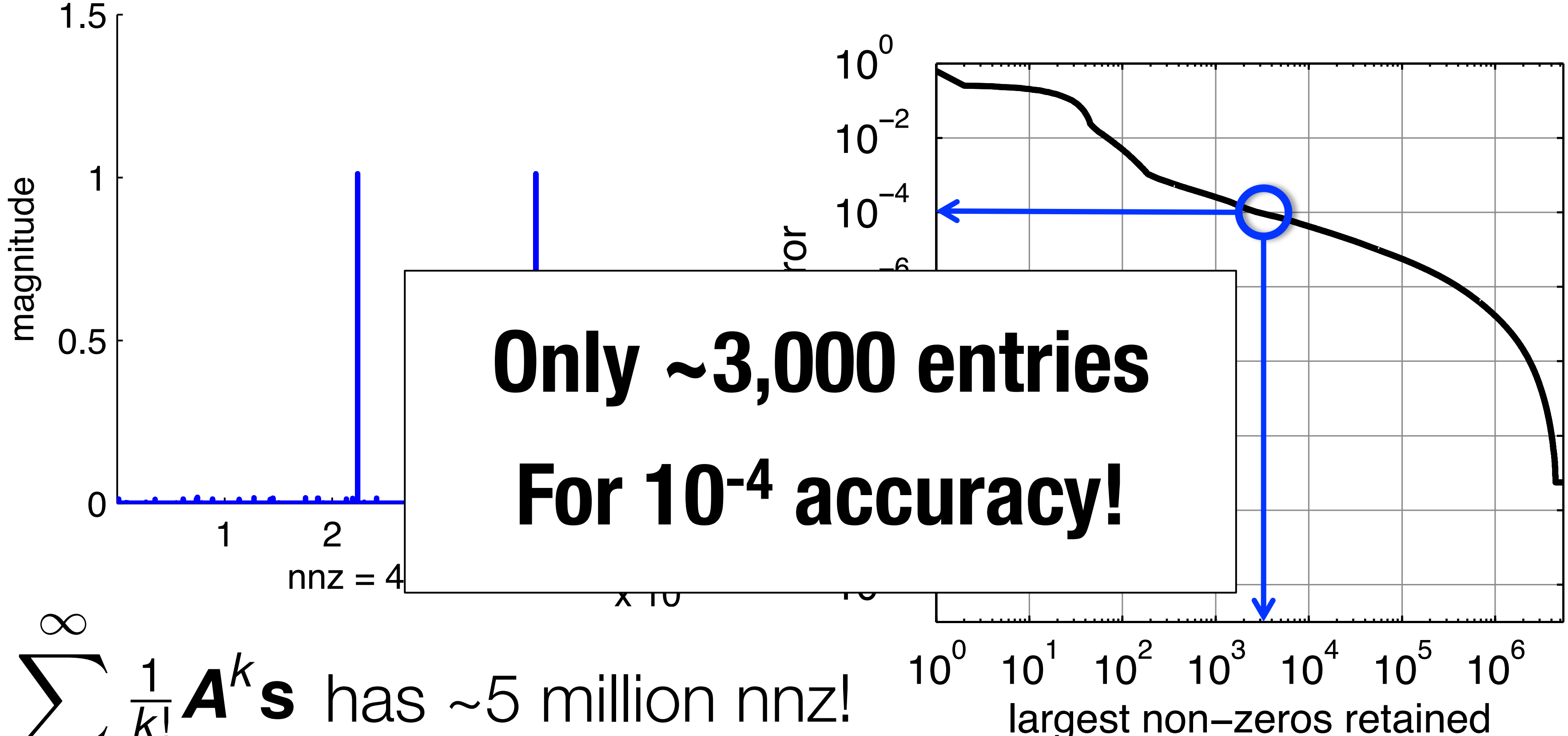


$\sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{A}^k \mathbf{s}$ has ~5 million nnz!

Local solution, big speedup

Magnitude of entries in solution vector

Accuracy of approximation using only large entries



Only ~3,000 entries

For 10^{-4} accuracy!

$$\sum_{k=0}^{\infty} \frac{1}{k!} \boldsymbol{A}^k \mathbf{s}$$ has ~5 million nnz!

Local solution, local algorithm

A **local algorithm** approximates solution
in work proportional to output size (~3,000)
instead of whole graph (~10^9)

For fast heat kernel,
we want local algorithms!

Our algorithms for $\hat{\mathbf{x}} \approx \exp(\boldsymbol{P})\,\mathbf{s}$

Local community detection:

- *constant time* on any graph, $\tilde{O}(\frac{e^1}{\varepsilon})$
- outperforms PageRank
- accuracy: $\|\boldsymbol{D}^{-1}\mathbf{x} - \boldsymbol{D}^{-1}\hat{\mathbf{x}}\|_\infty < \varepsilon$

Link prediction, ranking:

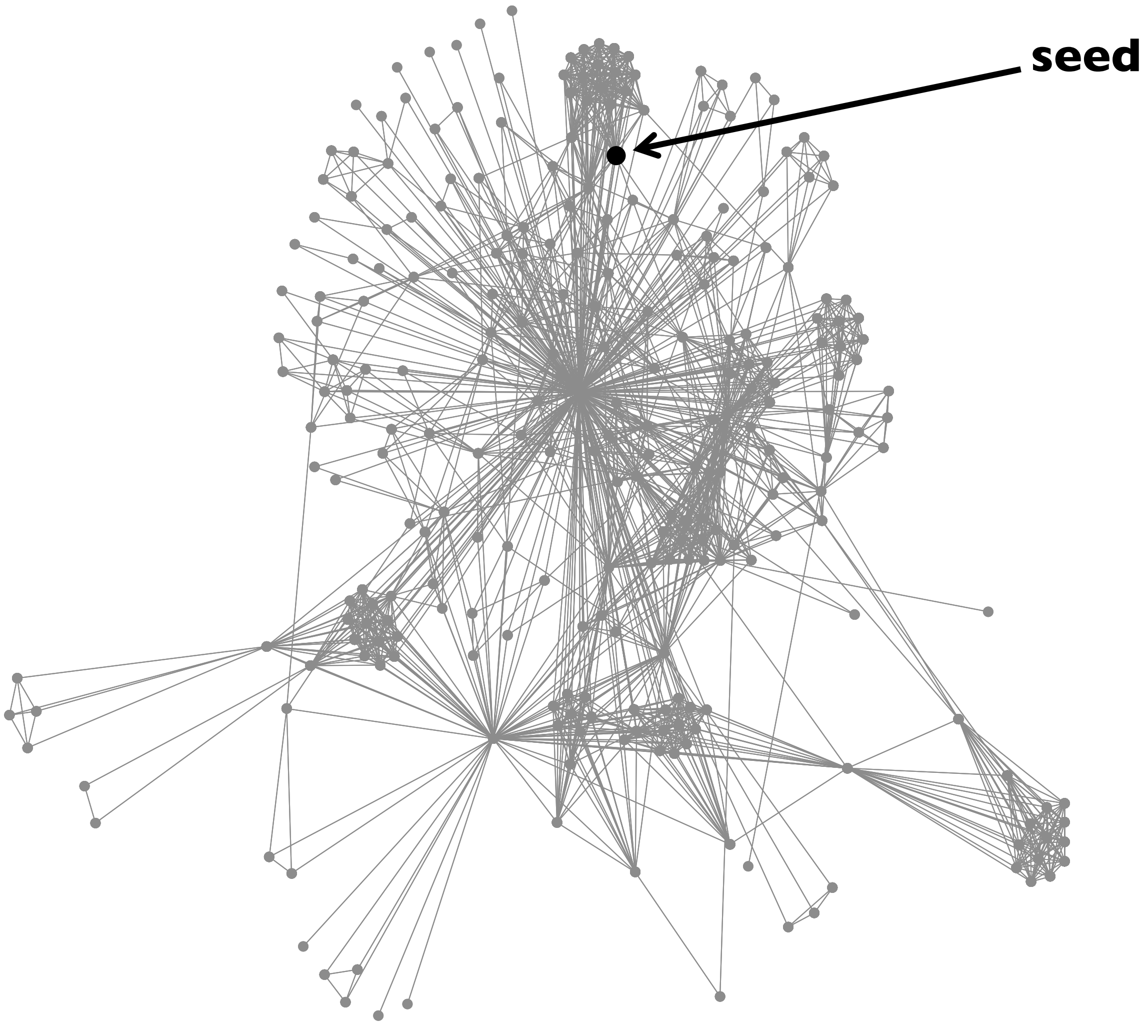
- *sublinear* local method $\tilde{O}(d^2 \log^2 d)$

(on networks with power-law degree dist.)

- accuracy: $\|\mathbf{x} - \hat{\mathbf{x}}\|_1 < \varepsilon$

Local Community Detection

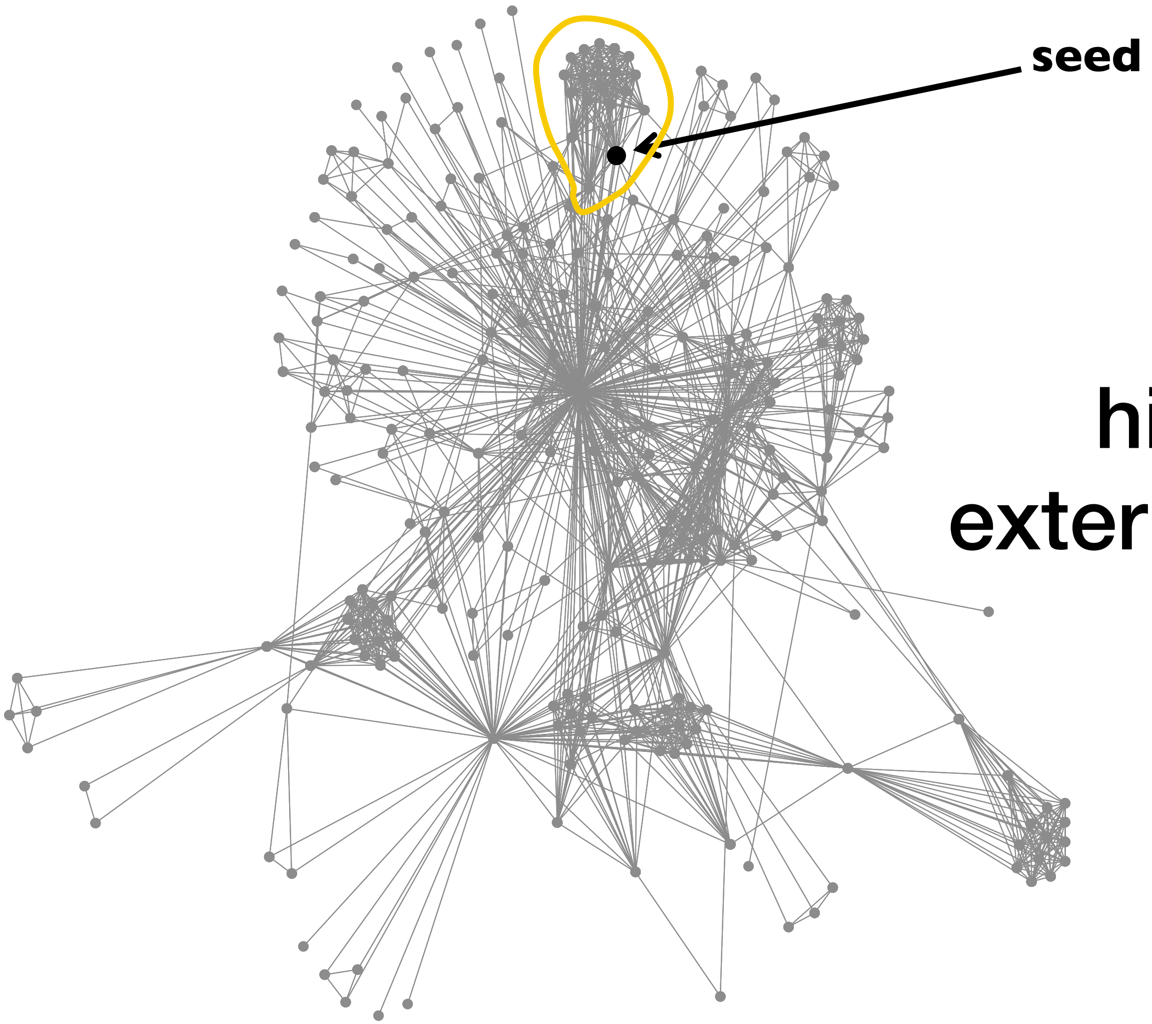
Given seed(s) S in G, find a community that contains S.



"Community" ?

Local Community Detection

Given seed(s) *S* in *G*, find a community that contains *S*.



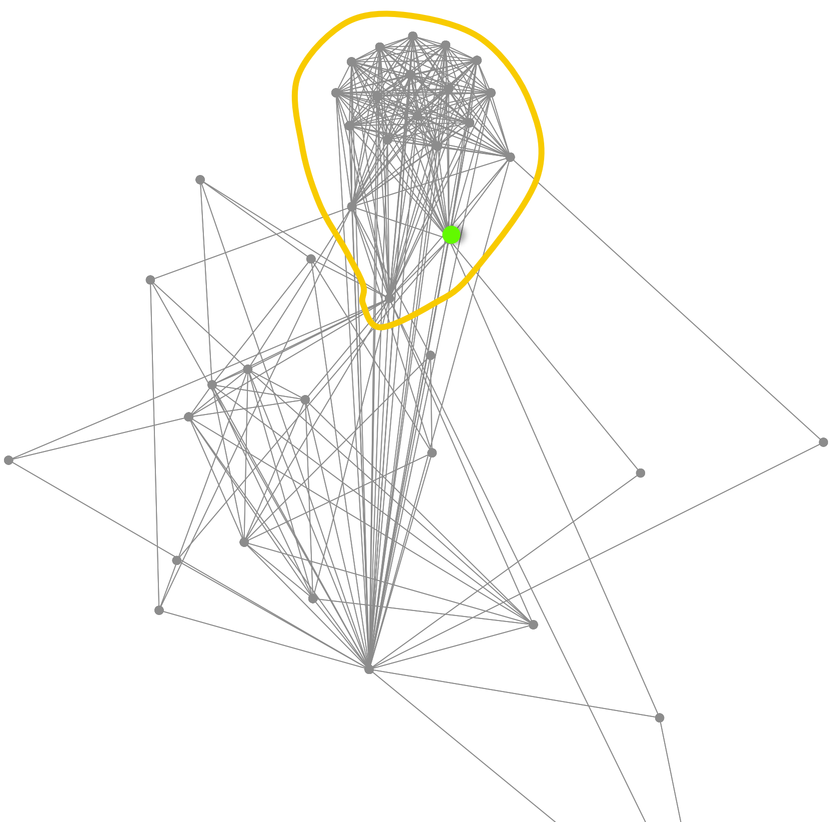
“Community” ?

high internal, low external connectivity

Low-conductance sets are communities

$$\text{conductance}(\ T\ ) = \frac{\#\text{ edges leaving } T}{\#\text{ edge endpoints in } T}$$

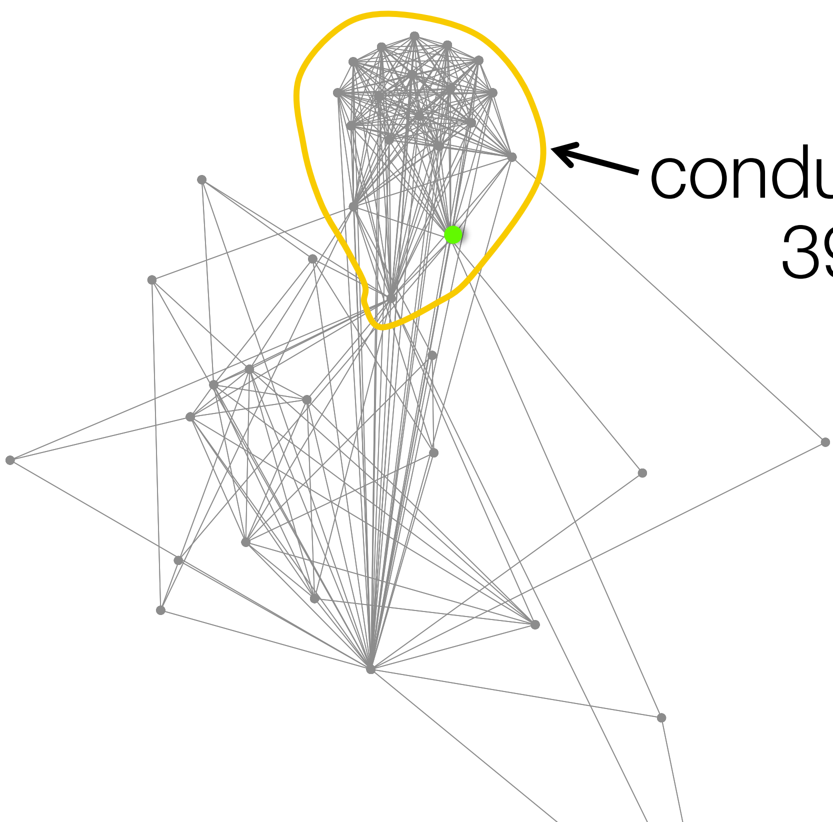
= " chance a random step exits T "



Low-conductance sets are communities

$$\text{conductance}(\ T\ ) = \frac{\#\text{ edges leaving } T}{\#\text{ edge endpoints in } T}$$

$$= \text{“ chance a random step exits } T\text{ ”}$$

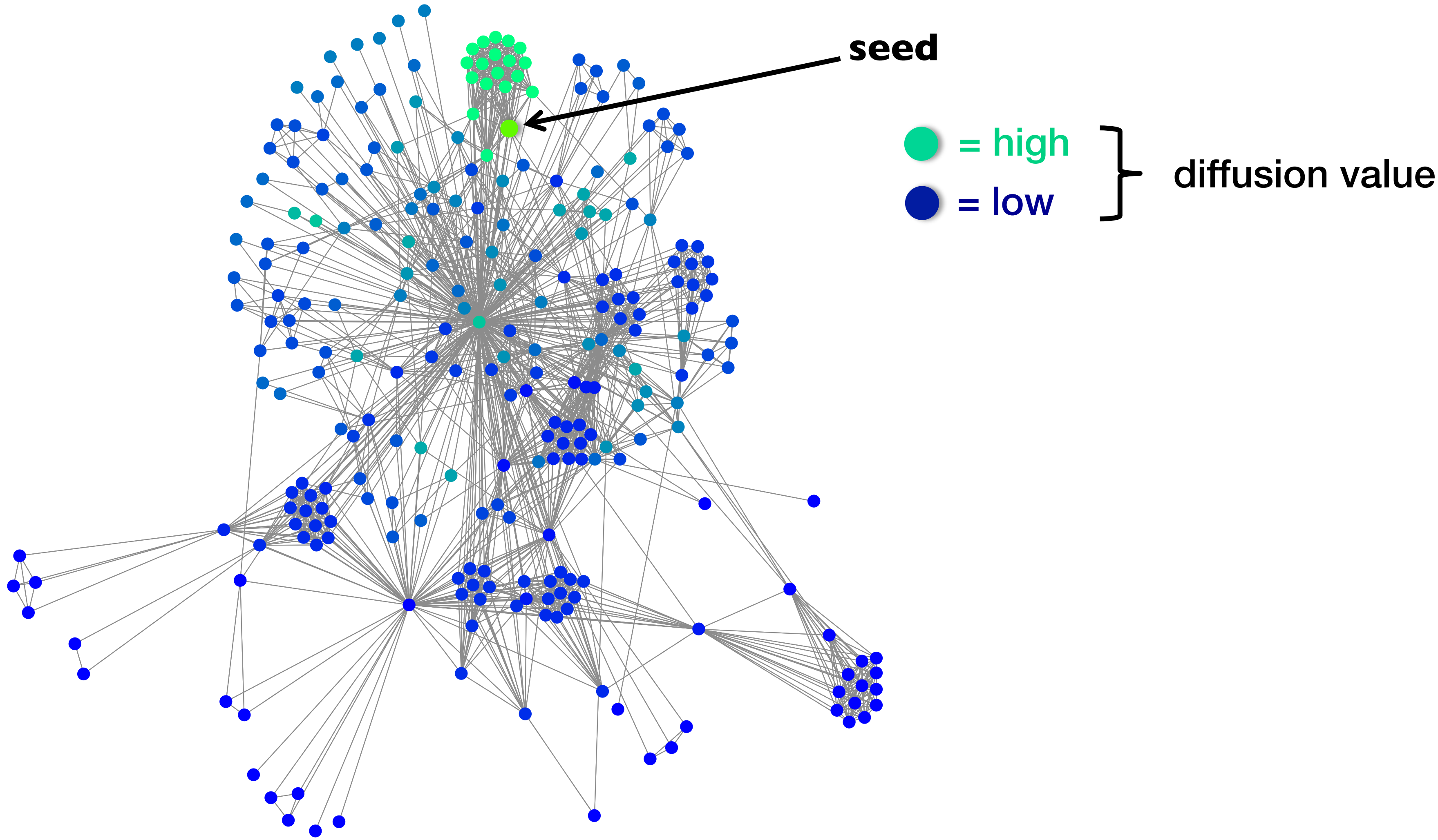


conductance(comm) =
39/381 = .102

How to find these ?

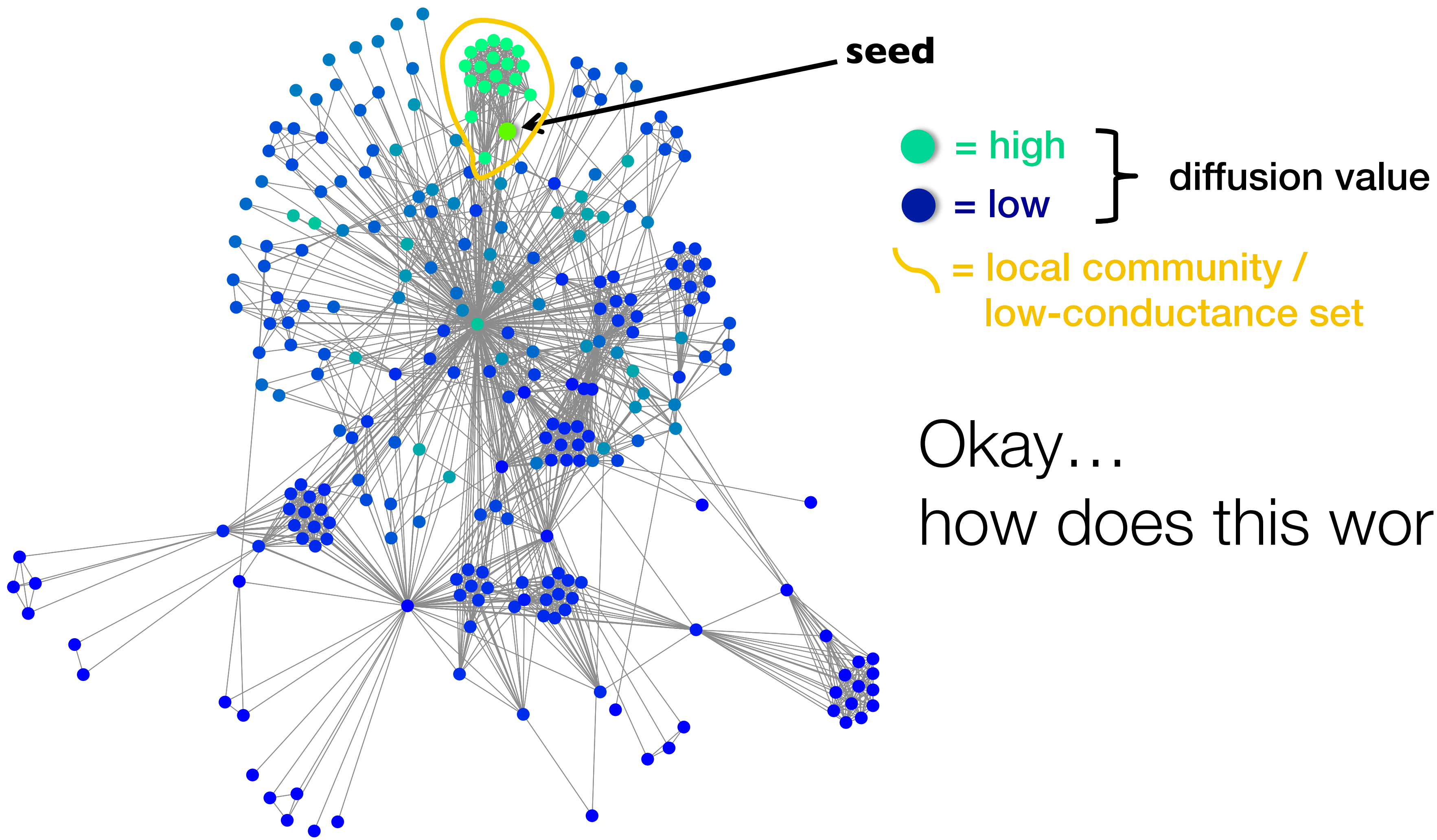
Graph diffusions find low-conductance sets

A diffusion propagates “rank” from a seed across a graph.



Graph diffusions find low-conductance sets

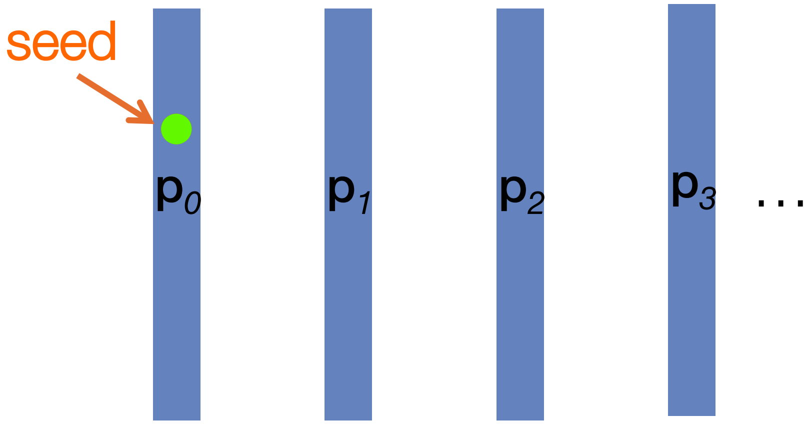
A diffusion propagates “rank” from a seed across a graph.



Okay…
how does this work?

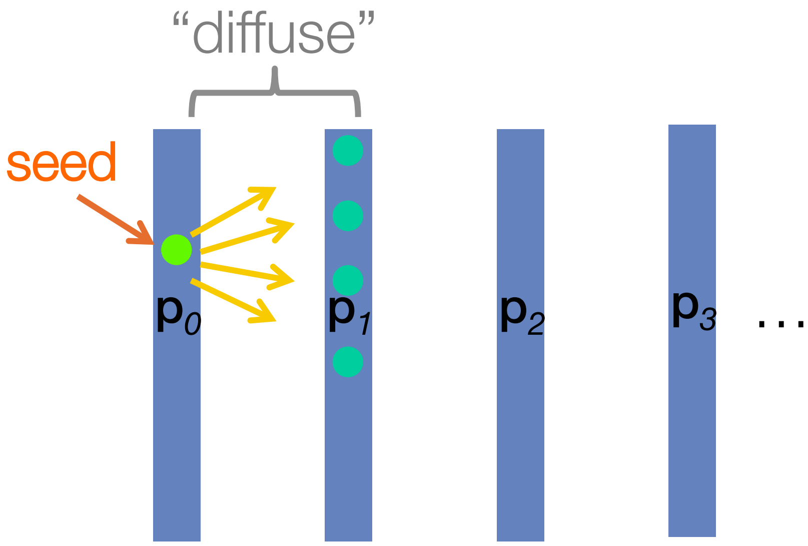
Graph Diffusion

A diffusion models how a mass (green dye, money, popularity) spreads from a seed across a network.



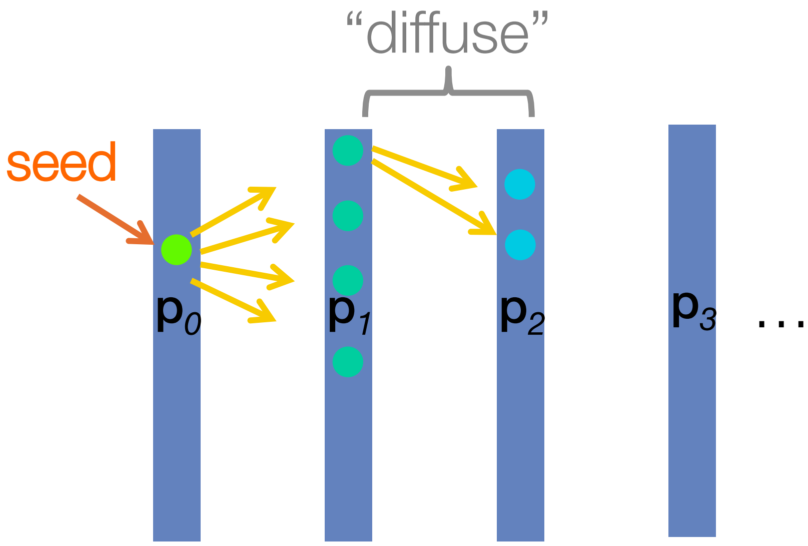
Graph Diffusion

A diffusion models how a mass (green dye, money, popularity) spreads from a seed across a network.



Graph Diffusion

A diffusion models how a mass (green dye, money, popularity) spreads from a seed across a network.



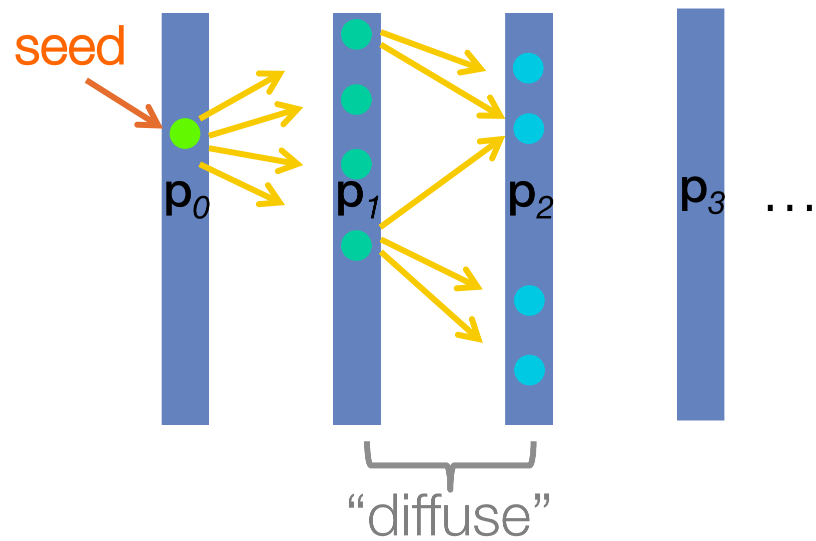
Once mass reaches a node, it propagates to the neighbors, with some decay.

"decay": dye dilutes, money is taxed, popularity fades

Graph Diffusion

A diffusion models how a mass (green dye, money, popularity) spreads from a seed across a network.

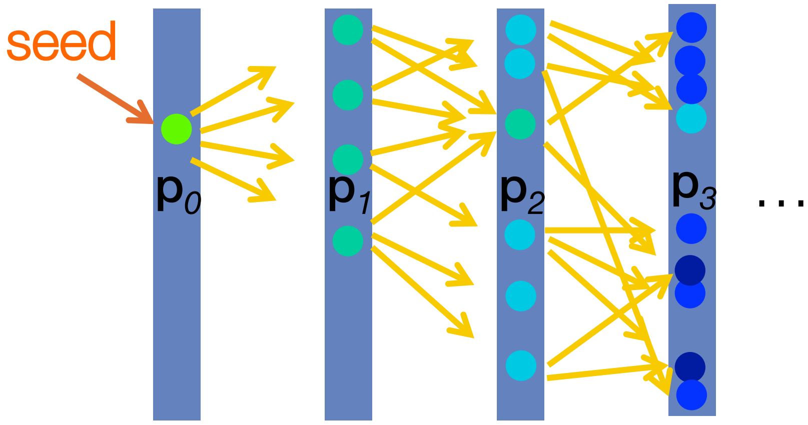
Once mass reaches a node, it propagates to the neighbors, with some decay.



"decay": dye dilutes, money is taxed, popularity fades

Graph Diffusion

A diffusion models how a mass (green dye, money, popularity) spreads from a seed across a network.

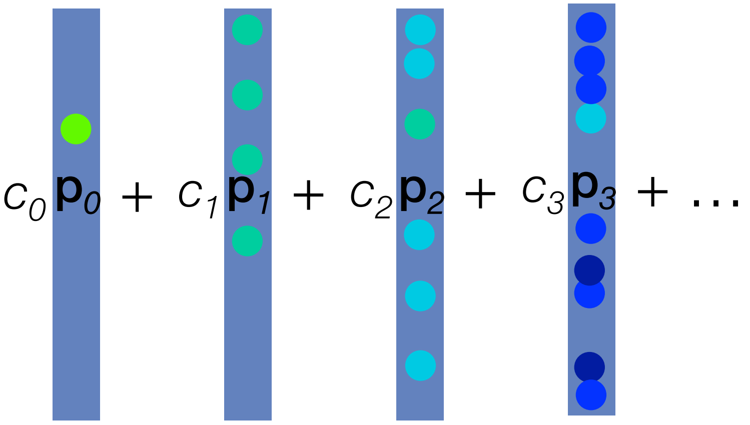


Once mass reaches a node, it propagates to the neighbors, with some decay.

"decay": dye dilutes, money is taxed, popularity fades

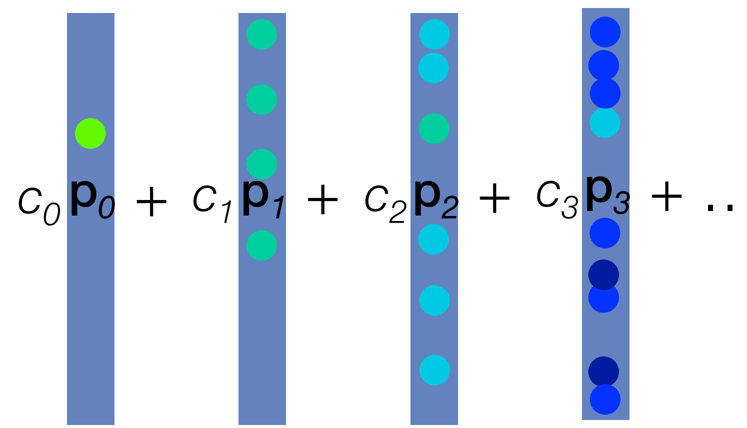
Diffusion score

"diffusion score" of a node = weighted sum of the mass at that node during different stages.



Diffusion score

“diffusion score” of a node = weighted sum of the mass at that node during different stages.



diffusion score vector = **f**

$$\mathbf{f} = \sum_{k=0}^{\infty} c_k \boldsymbol{P}^k \mathbf{s}$$

$\boldsymbol{P} =$ random-walk transition matrix

$\mathbf{s} =$ normalized seed vector

$c_k =$ weight on stage *k*

Heat Kernel vs. PageRank Diffusions

Heat Kernel uses $t^k/k!$

Our work is new analysis for this diffusion.

$$\frac{t^0}{0!}\mathbf{p}_0 + \frac{t^1}{1!}\mathbf{p}_1 + \frac{t^2}{2!}\mathbf{p}_2 + \frac{t^3}{3!}\mathbf{p}_3 + \dots$$

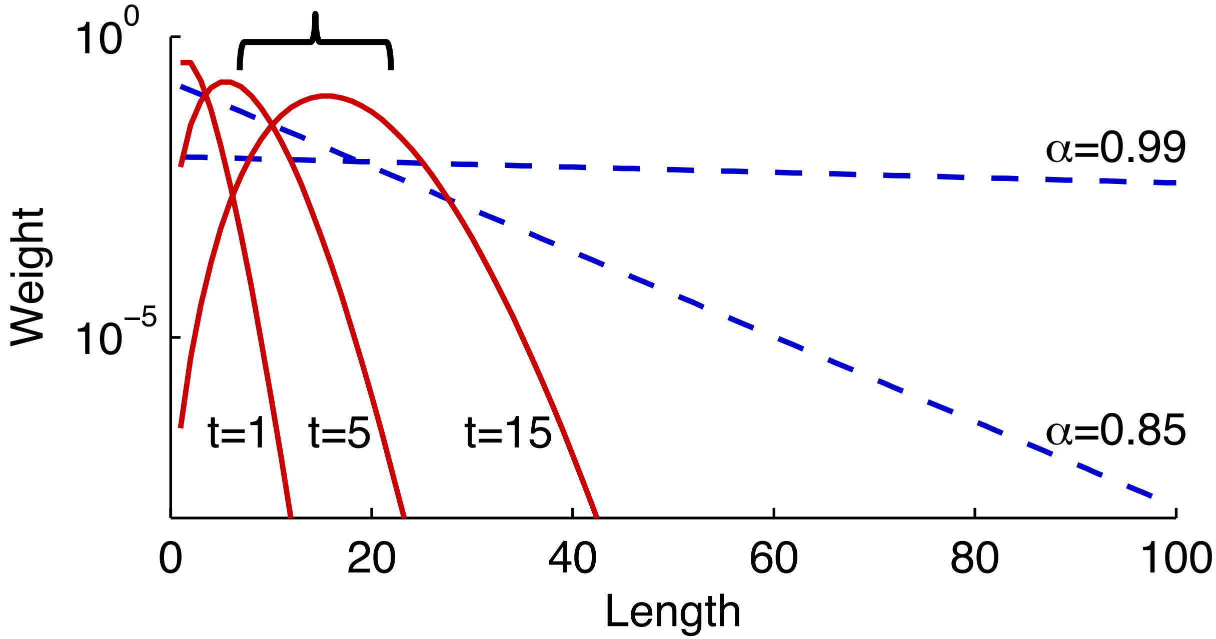
PageRank uses α^k at stage k.

Standard, widely-used diffusion we use for comparison.

$$\alpha^0\mathbf{p}_0 + \alpha^1\mathbf{p}_1 + \alpha^2\mathbf{p}_2 + \alpha^3\mathbf{p}_3 + \dots$$

Heat Kernel vs. PageRank Behavior

HK emphasizes **earlier stages** of diffusion.



PR, α^k

HK, $t^k/k!$

→ involve **shorter walks** from seed,

→ so HK looks at **smaller sets** than PR

Heat Kernel vs. PageRank Theory

	good conductance	fast algorithm
PR	Local Cheeger Inequality: "PR finds set of near-optimal conductance"	"PPR-push" is $O(1/(\varepsilon(1-\alpha)))$ in theory, fast in practice [Andersen Chung Lang 06]
HK		

Heat Kernel vs. PageRank Theory

	good conductance	fast algorithm
PR	Local Cheeger Inequality: “PR finds set of near-optimal conductance”	“PPR-push” is $O(1/(\varepsilon(1-\alpha)))$ in theory, fast in practice [Andersen Chung Lang 06]
HK	Local Cheeger Inequality [Chung 07]	

Heat Kernel vs. PageRank Theory

	good conductance	fast algorithm
PR	Local Cheeger Inequality: "PR finds set of near-optimal conductance"	"PPR-push" is $O(1/(\varepsilon(1-\alpha)))$ in theory, fast in practice [Andersen Chung Lang 06]
HK	Local Cheeger Inequality [Chung 07]	**Our work**

Our work on Heat Kernel: theory

THEOREM Our algorithm for a relative ε-accuracy in a degree-weighted norm has

runtime <= $O(\, e^t(\log(1/\varepsilon) + \log(t)) \,/\, \varepsilon)$

(which is constant, regardless of graph size)

Our work on Heat Kernel: theory

THEOREM Our algorithm for a relative ε-accuracy in a degree-weighted norm has

runtime <= $O(\ e^t(\log(1/\varepsilon) + \log(t)) \ /\ \varepsilon)$

(which is constant, regardless of graph size)

COROLLARY HK is local!

(O(1) runtime → diffusion vector has O(1) entries)

Our work on Heat Kernel: results

First efficient, deterministic HK algorithm. Deterministic is important to be able to compare the behaviors of HK and PR *experimentally*:

Our key findings

- **HK more accurately describes ground-truth communities in real-world networks**
- identifies smaller sets → better precision
- speed & conductance comparable with PR

Twitter graph

41.6 M nodes

2.4 B edges

Python demo

un-optimized Python code

on a laptop

Available for download:

`https://gist.github.com/dgleich/cf170a226aa848240cf4`

Algorithm Outline

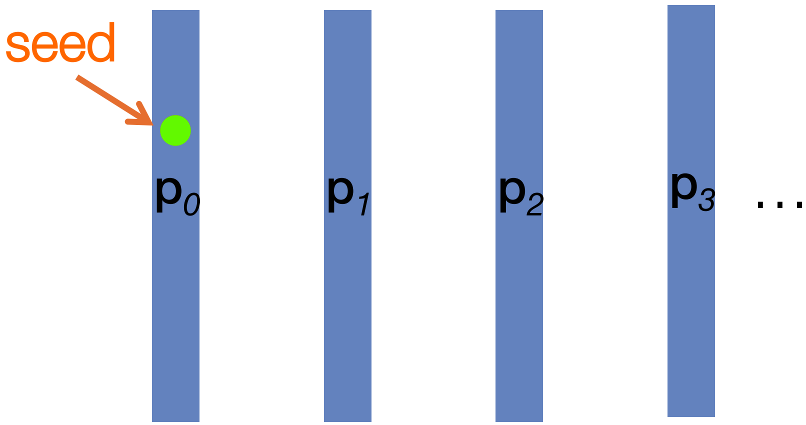
Computing HK

1. Pre-compute "`push`" thresholds
2. Do "`push`" on all entries above threshold

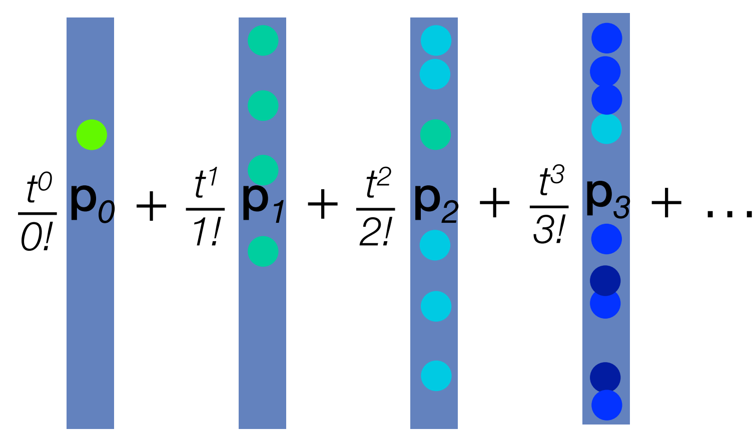
Algorithm Intuition

Computing HK given parameters t, ε, seed $\mathbf{s}$

Starting from here…



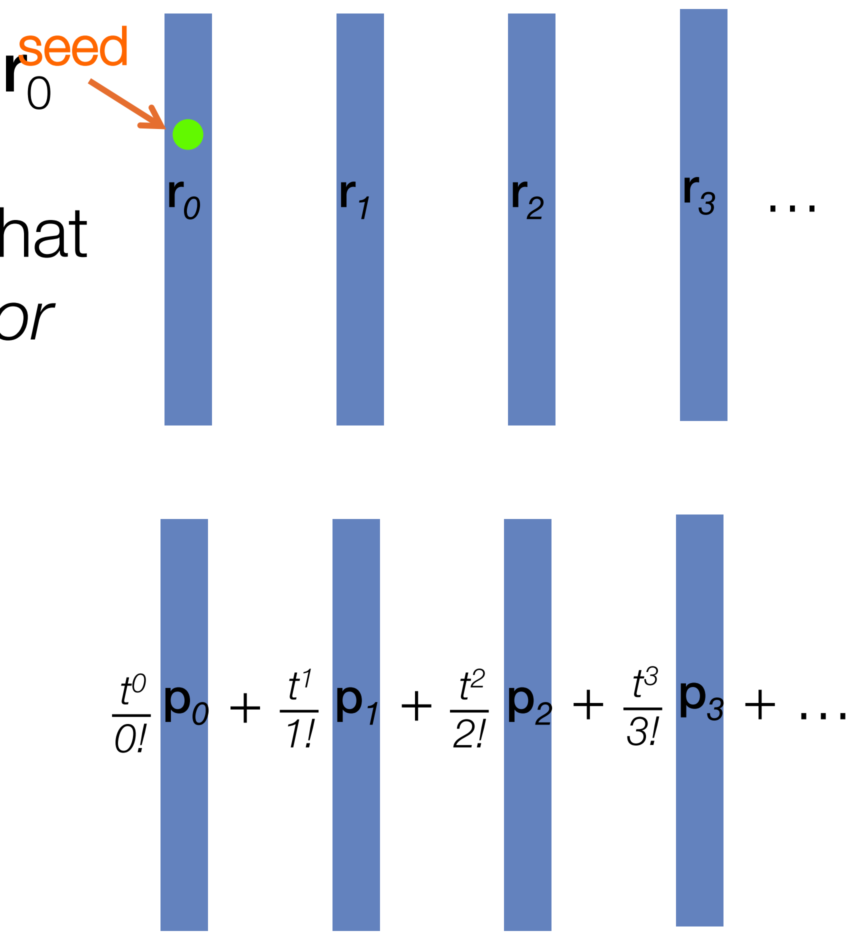
How to end up here?



Algorithm Intuition

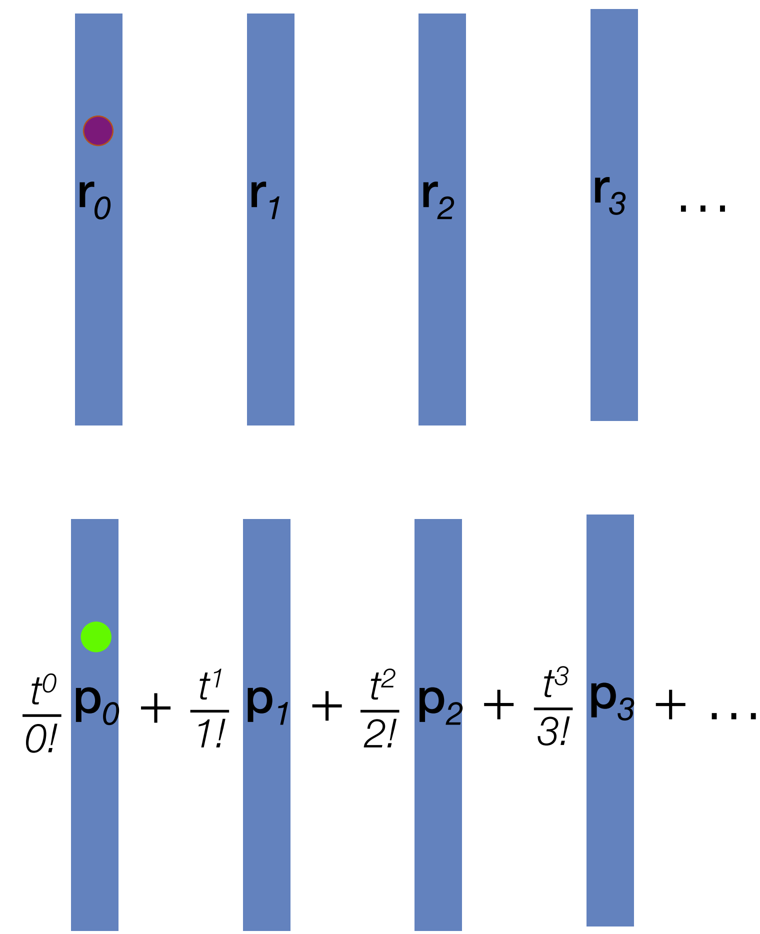
Begin with mass at seed(s)
in a “residual” staging area, $\mathbf{r}_0$

The residuals $\mathbf{r}_k$ hold mass that
is unprocessed – it’s like *error*



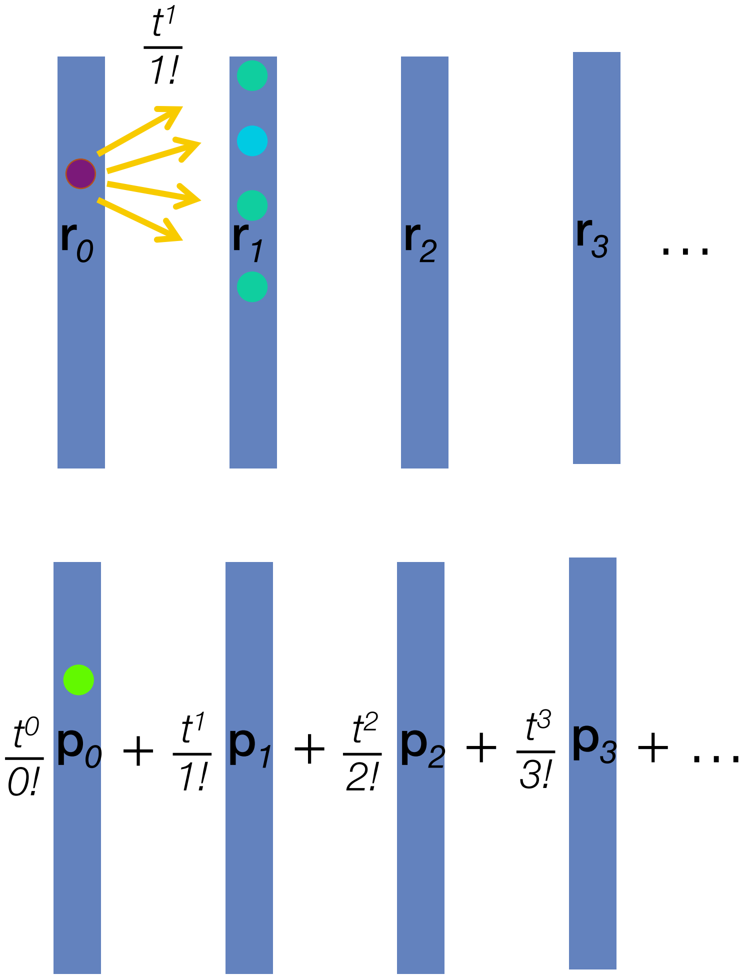
Push Operation

push – (1) remove entry in $\mathbf{r}_k$,
(2) put in **p**,



Push Operation

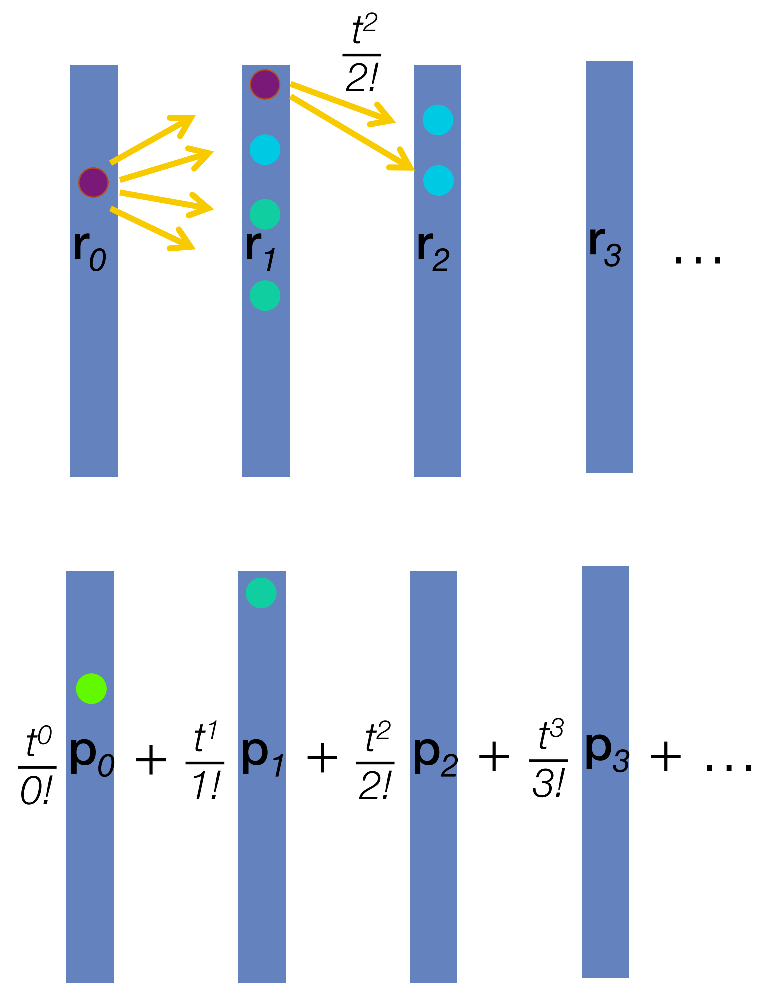
push – (1) remove entry in $\mathbf{r}_k$,
(2) put in $\mathbf{p}$,
(3) then scale and spread to neighbors in next $\mathbf{r}$



Push Operation

push – (1) remove entry in $\mathbf{r}_k$,
(2) put in $\mathbf{p}$,
(3) then scale and spread to neighbors in next $\mathbf{r}$

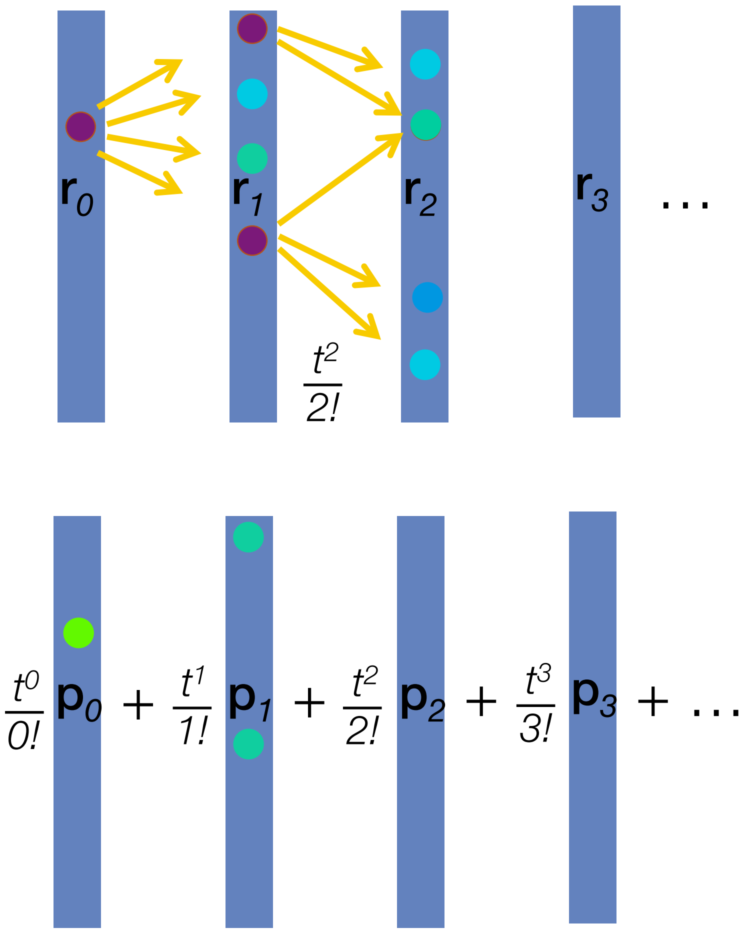
(repeat)



Push Operation

push – (1) remove entry in $\mathbf{r}_k$,
(2) put in **p**,
(3) then scale and
spread to neighbors
in next **r**

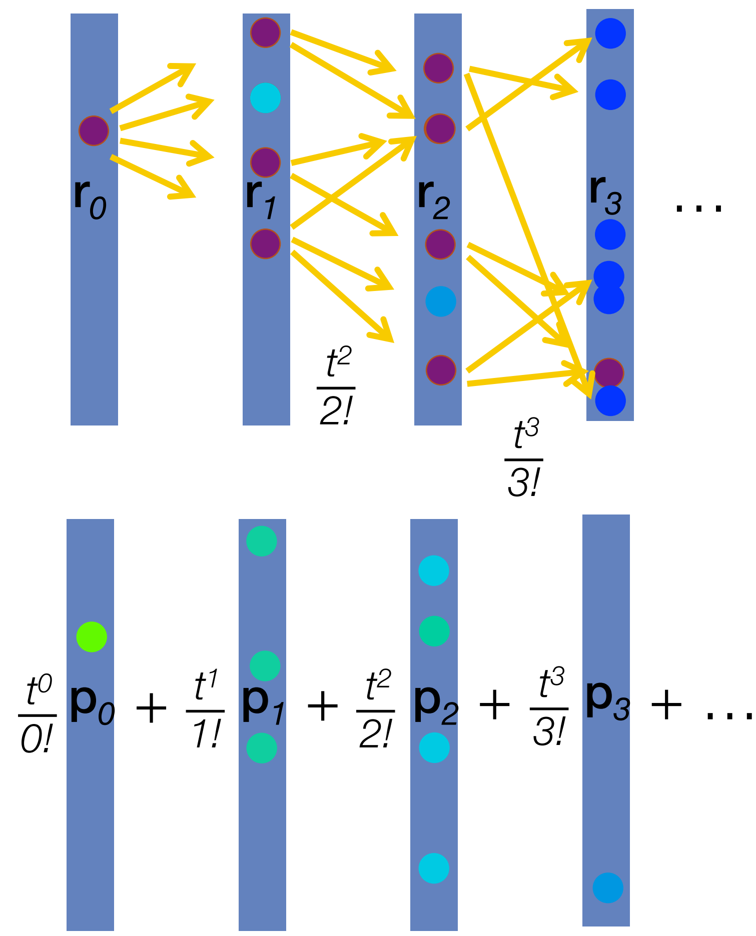
(repeat)



Push Operation

push – (1) remove entry in $\mathbf{r}_k$,
(2) put in **p**,
(3) then scale and spread to neighbors in next **r**

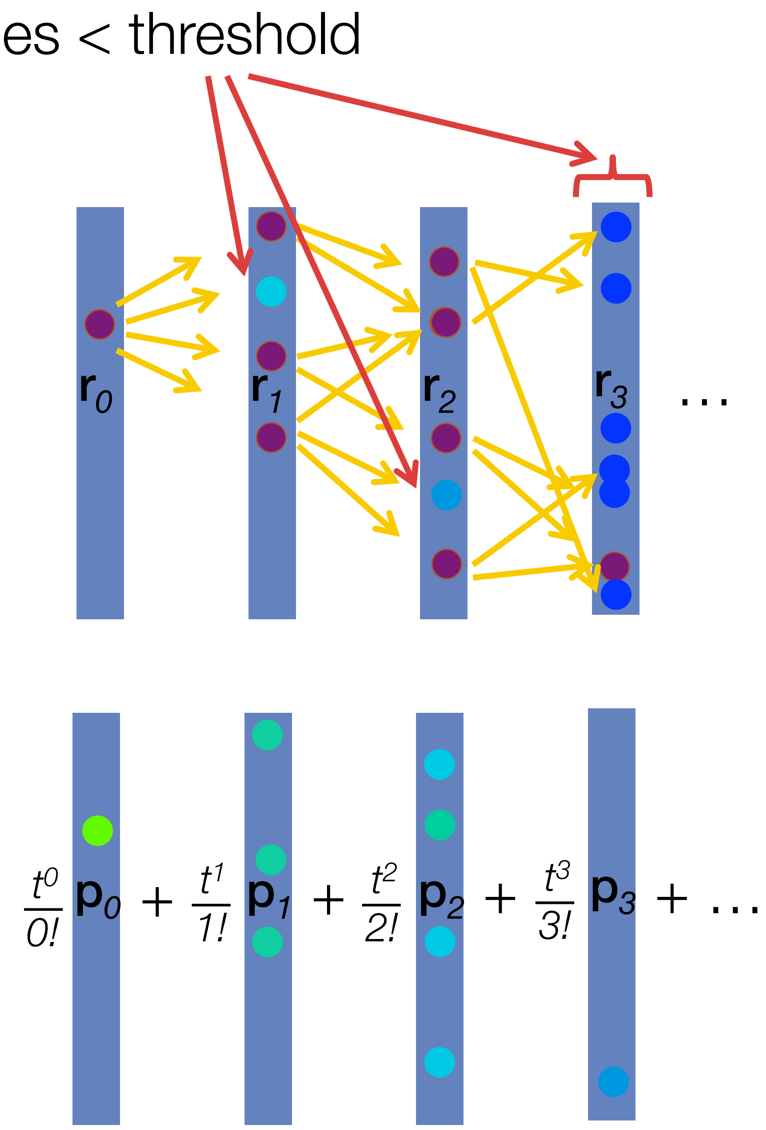
(repeat)



Thresholds

ERROR equals weighted sum of entries left in $\mathbf{r}_k$

→ Set threshold so "leftovers" sum to $< \varepsilon$



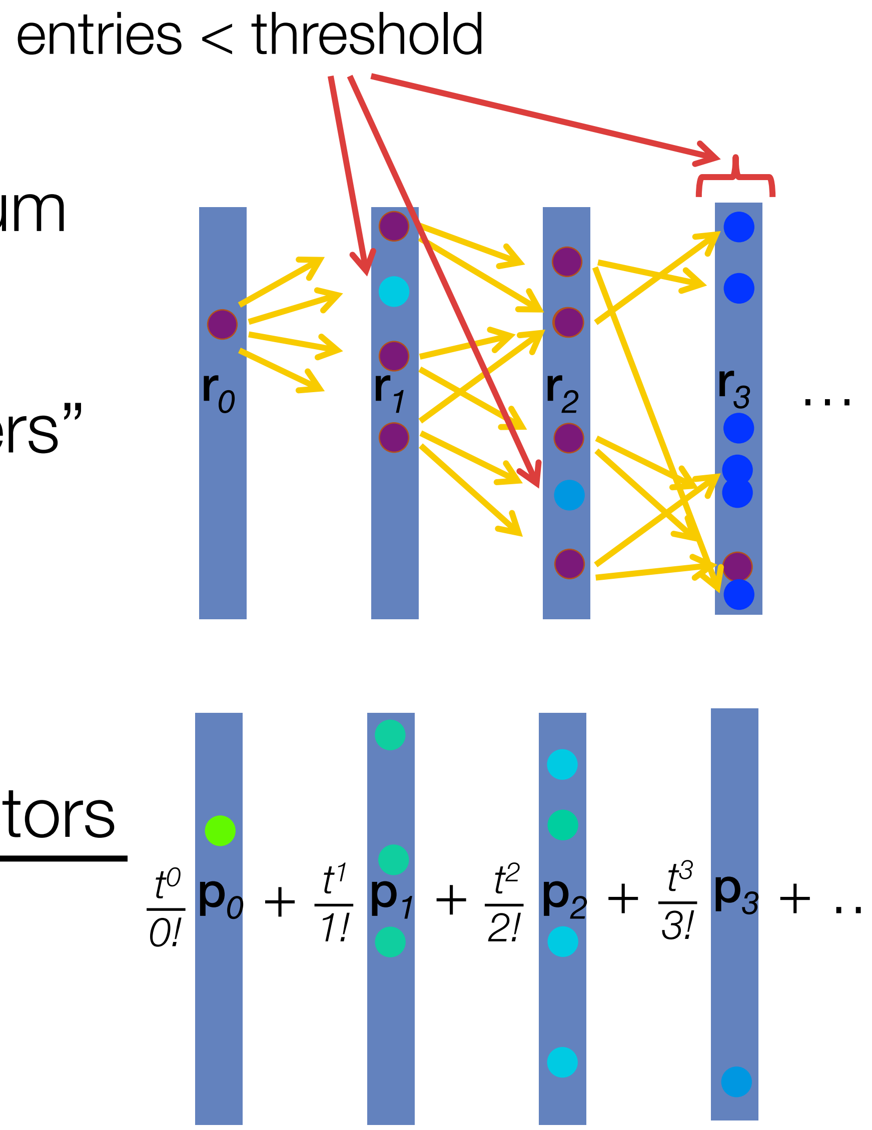
Thresholds

ERROR equals weighted sum of entries left in $\mathbf{r}_k$

→ Set threshold so "leftovers" sum to $< \varepsilon$

Threshold for stage $\mathbf{r}_k$ is

$$\frac{\text{sum of remaining scale factors}}{\text{prior scaling factor}}$$



Algorithm Outline

Computing HK

1. Pre-compute "`push`" thresholds
2. Do "`push`" on all entries above threshold

Once no more entries are > threshold: convergence!

Communities in Real-world Networks

Given a seed in an unidentified real-world community, how well can HK and PR describe that community?

Measure quality using F_1-measure.

Graph	$\|V\|$	$\|E\|$
amazon	330 K	930 K
dblp	320 K	1 M
youtube	1.1 M	3 M
lj	4 M	35 M
orkut	3.1 M	120 M
friendster	66 M	1.8 B

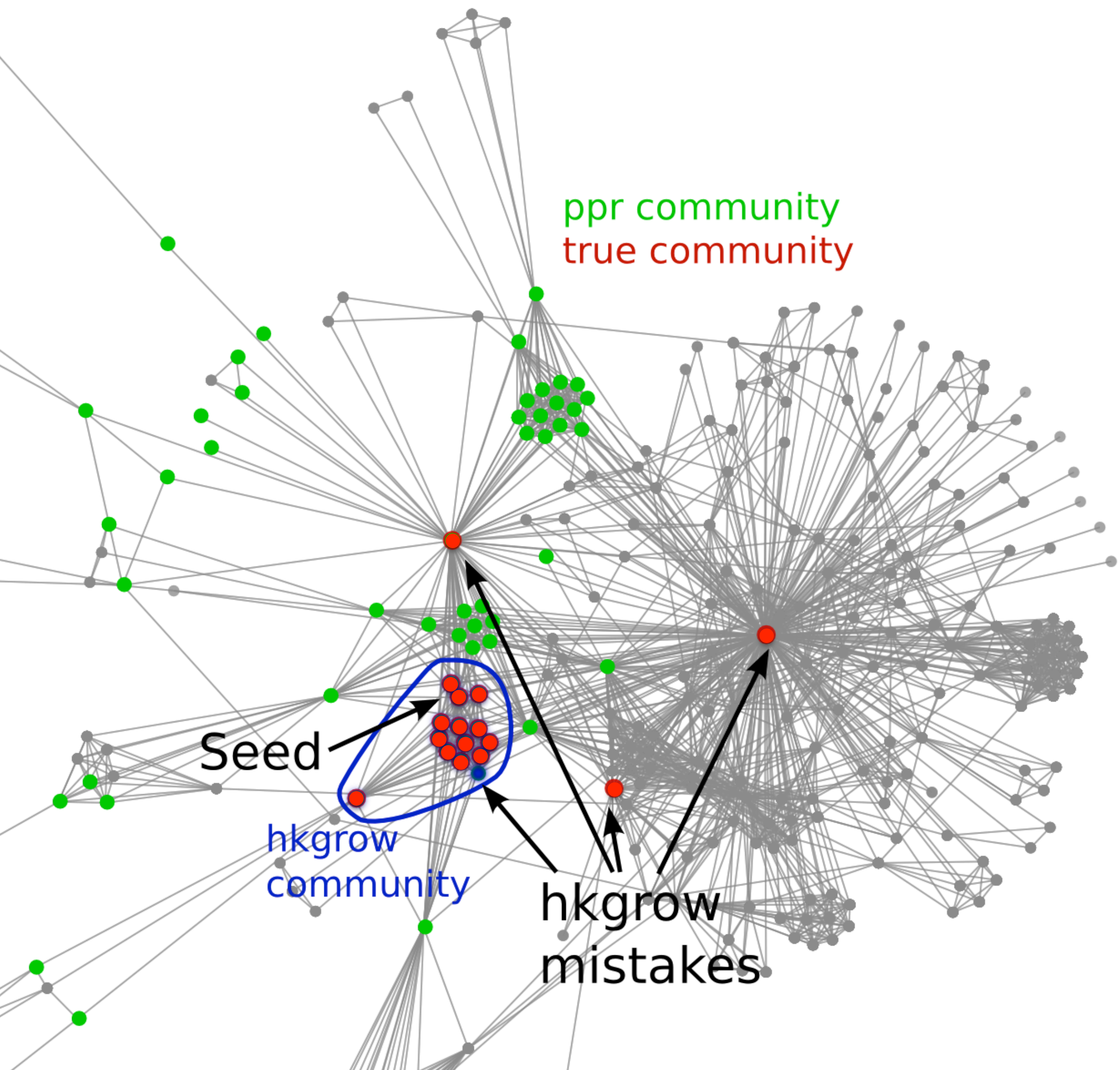
Datasets from SNAP collection [Leskovec]

F_1-measure is the harmonic mean of

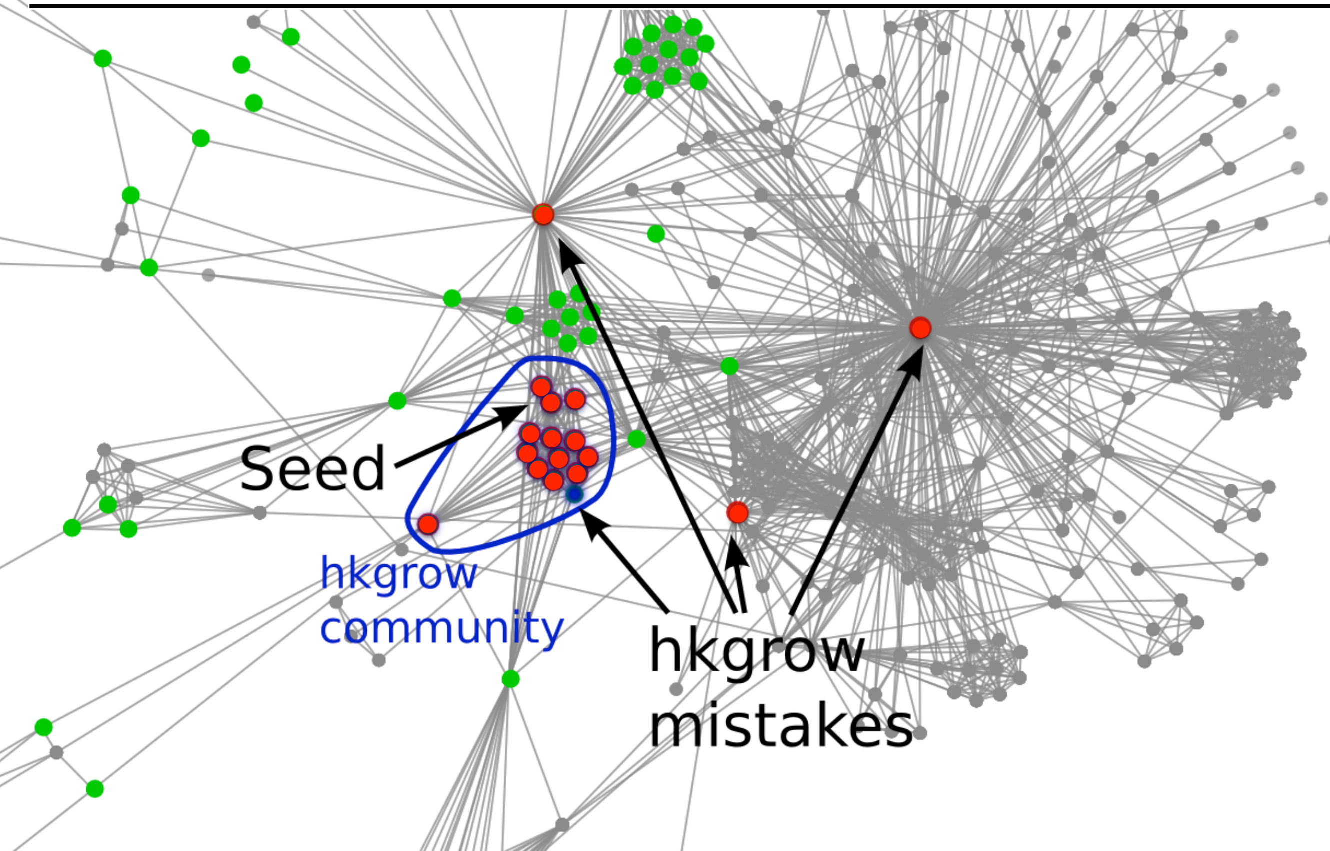
$$\textbf{precision} = \frac{\text{\# correct guesses}}{\text{\# total guesses}}$$

and

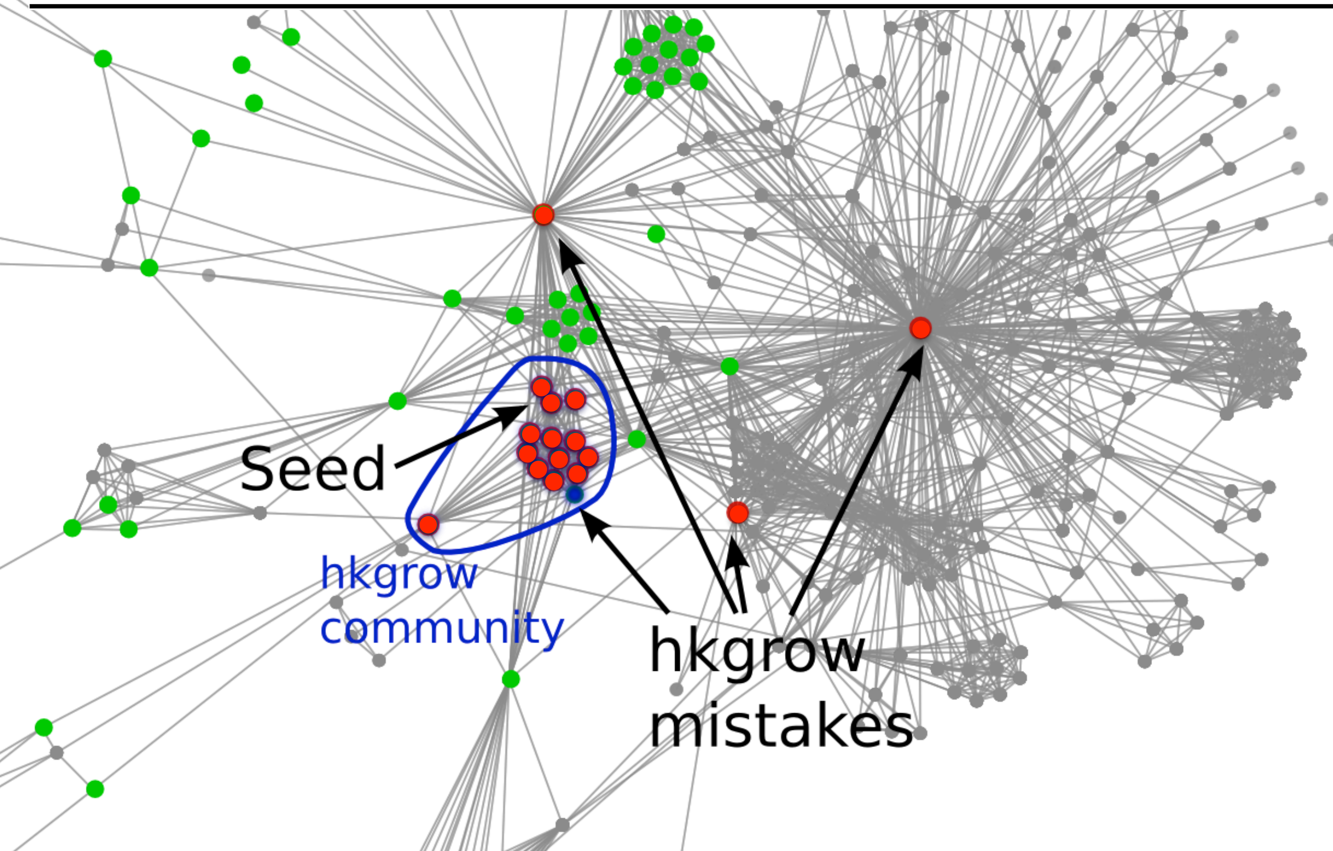
$$\textbf{recall} = \frac{\text{\# answers you get}}{\text{\# answers there are}}$$


ppr community
true community
Seed
hkgrow
community
hkgrow
mistakes

data	F_1		precision		set size		comm
	HK	PR	HK	PR	HK	PR	size
amazon	0.325	0.140	0.244	0.107	193	15293	495



data	F_1 HK	F_1 PR	precision HK	precision PR	set size HK	set size PR	comm size
amazon	0.325	0.140	0.244	0.107	193	15293	495
dblp	0.257	0.115	0.208	0.081	44	16026	1429
youtube	0.177	0.136	0.135	0.098	1010	6079	1615
lj	0.131	0.107	0.102	0.086	283	738	662
orkut	0.055	0.044	0.036	0.031	537	1989	4526
friendster	0.078	0.090	0.066	0.075	229	333	724



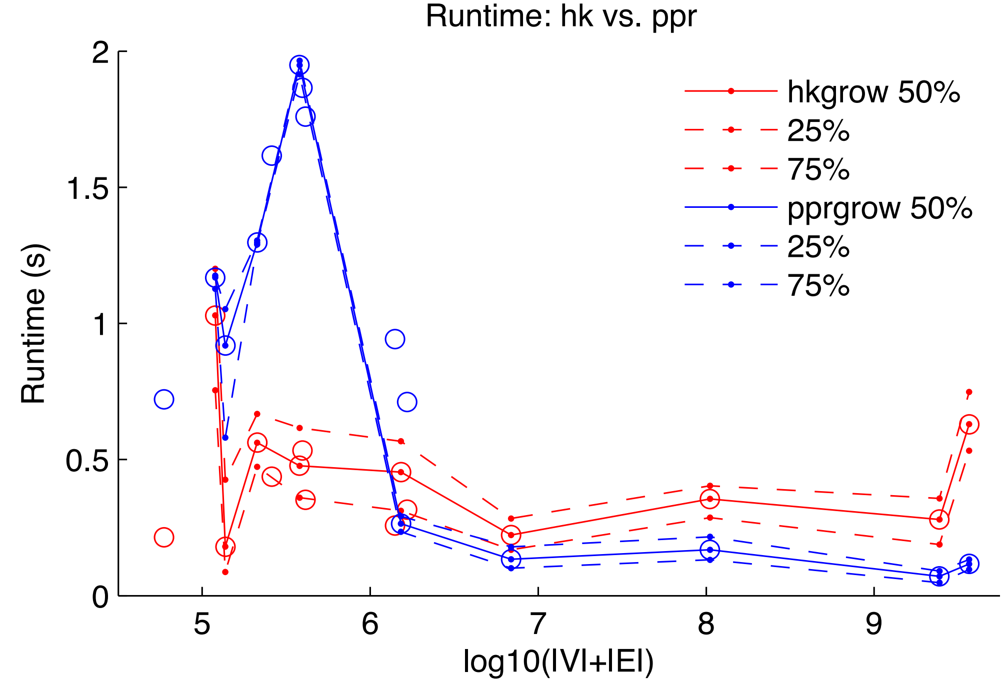
PR achieves high recall by "guessing" a huge set

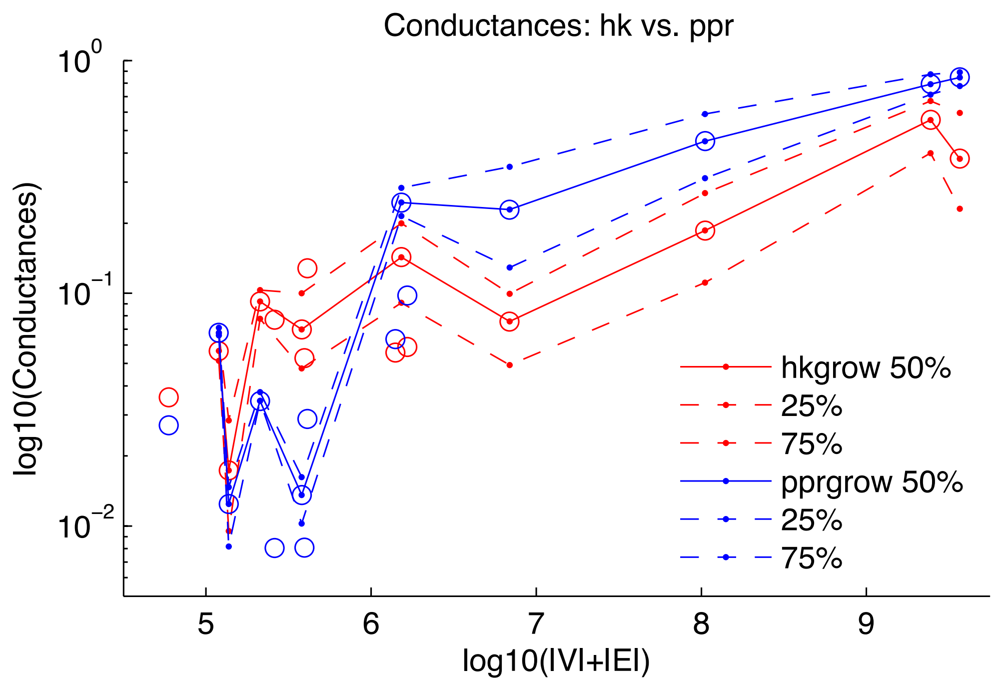
HK identifies a tighter cluster, so attains better precision

Runtime & Conductance

HK is comparable in runtime and conductance.

As graphs scale,
the diffusions' performance becomes even more similar.





Algorithms for $\hat{\mathbf{x}} \approx \exp(\boldsymbol{P})\,\mathbf{s}$

Constant time local community detection ✓

Link prediction, ranking:

- *sublinear* local method $\tilde{O}(d^2 \log^2 d)$ (on networks with power-law degree dist.)
- accuracy: $\|\mathbf{x} - \hat{\mathbf{x}}\|_1 < \varepsilon$

Algorithm 2 outline $\hat{\mathbf{x}} \approx \exp(\boldsymbol{P})\,\mathbf{s}$

(1) Approximate with a polynomial

(2) Convert to linear system

(3) Solve with sparse linear solver

(Details in paper)

Algorithm 2 outline $\quad \hat{\mathbf{x}} \approx \exp(\boldsymbol{P})\,\mathbf{s}$

(1) Approximate with a polynomial

(2) Convert to linear system

(3) Solve with sparse linear solver

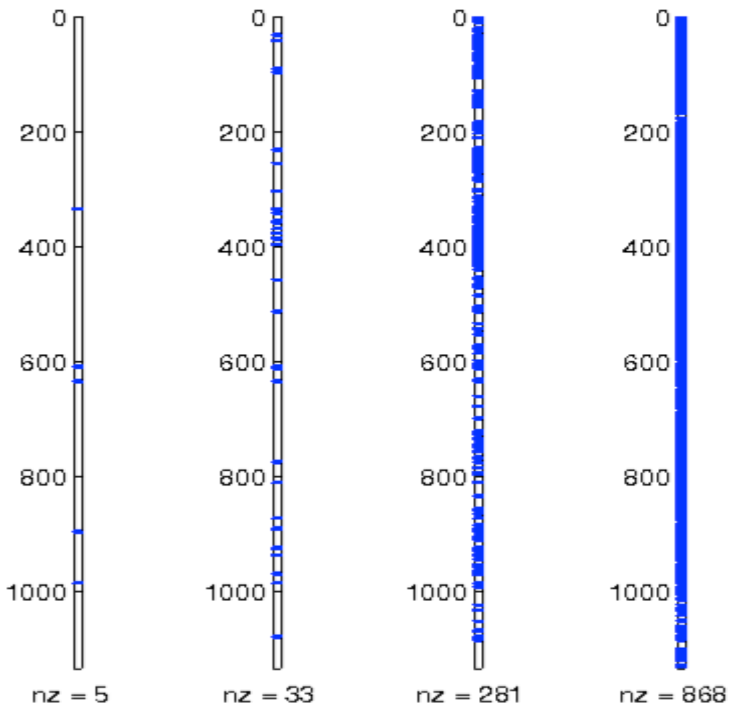
(Details in paper)

Key: We avoid doing these full matrix-vector products

$$\exp(\boldsymbol{P})\,\mathbf{s} \approx \sum_{k=0}^{N} \frac{1}{k!} \boldsymbol{P}^{k} \mathbf{s}$$

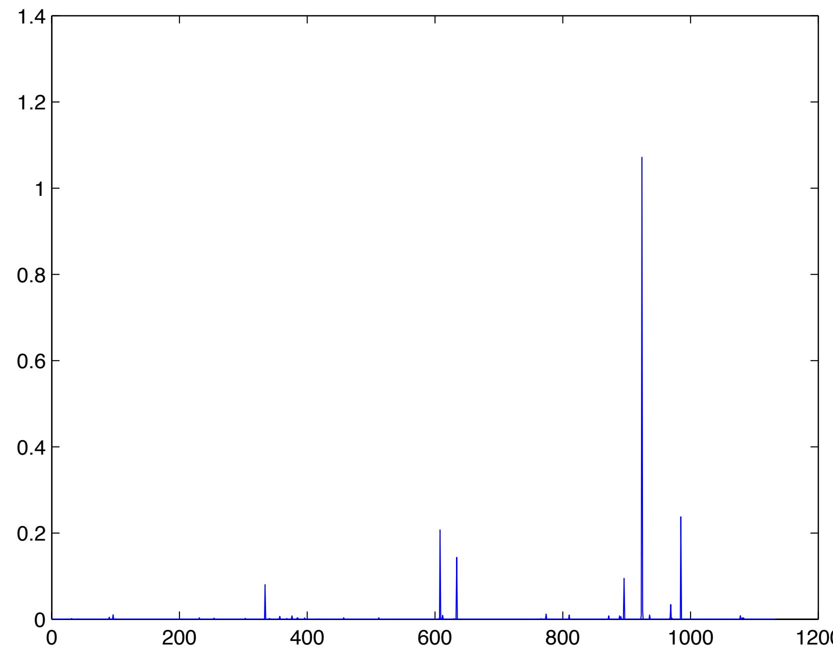
Fill-in vs. local solution

Nonzeros in the terms



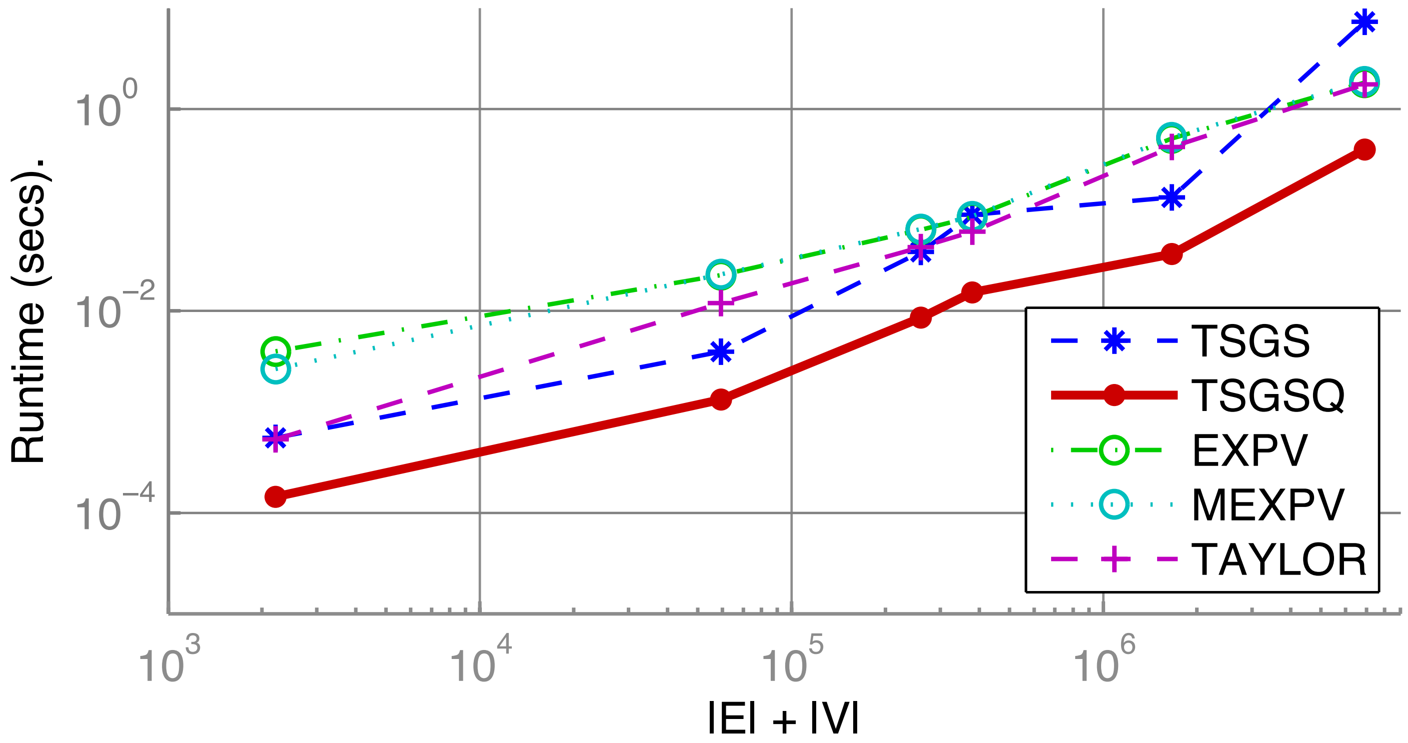
$A^1 s \quad A^2 s \quad A^3 s \quad A^4 s$

Magnitude of entries in solution vector



$$\sum_{k=0}^{\infty} \frac{1}{k!} A^k s$$

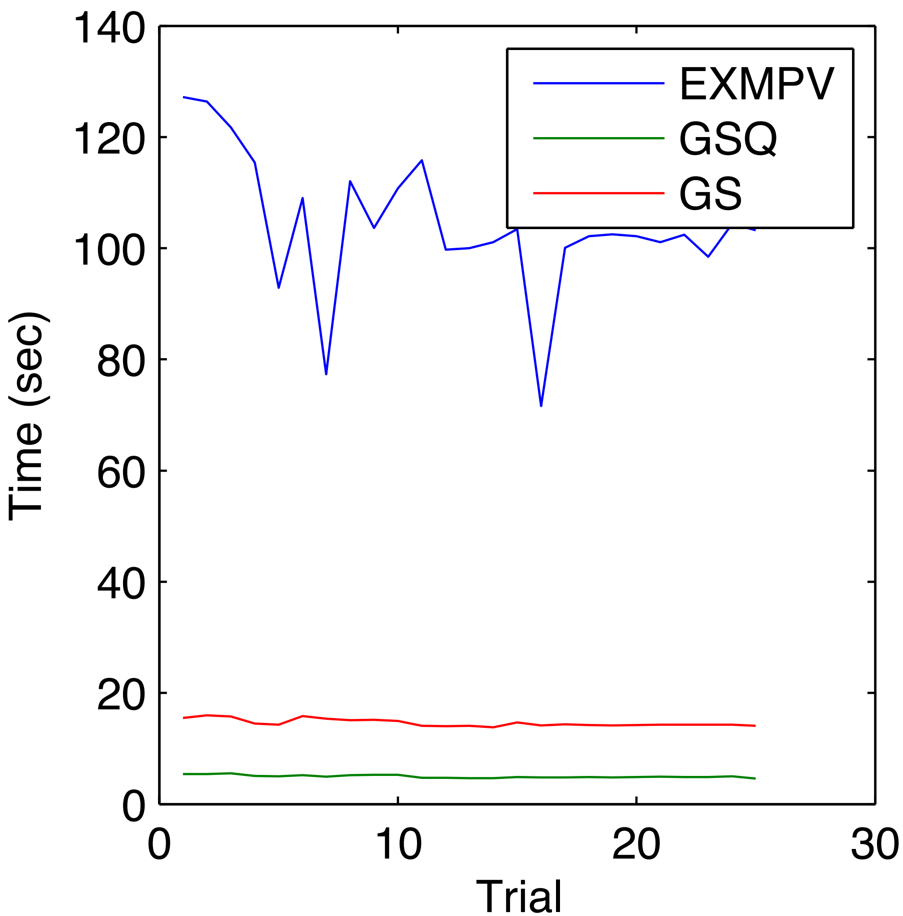
Runtime



TSGSQ is ours, EXPV is a state-of-the-art MatLab function

Runtime on the web-graph

A particularly sparse graph benefits us best

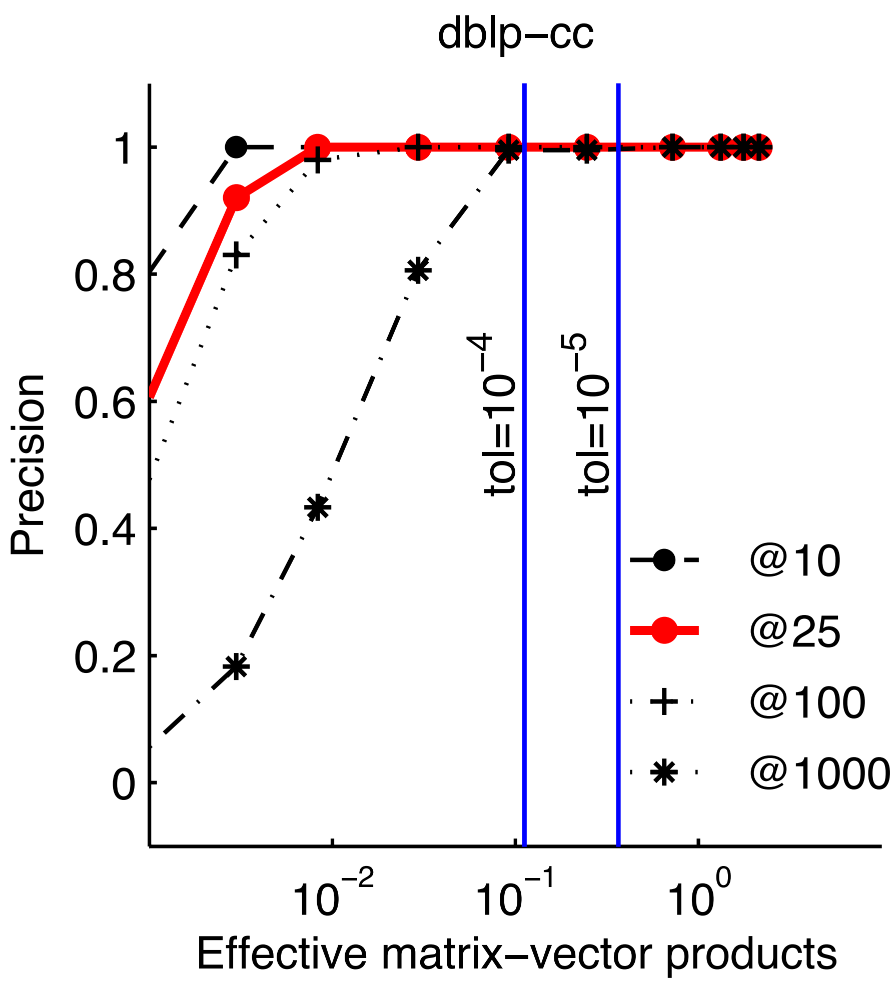


$|V| = O(10^8)$
$|E| = O(10^9)$

GSQ is our method
EXPMV is MatLab

Precision vs. work

We accurately identify large entries!



Code, references, future work

Local clustering via heat kernel code available at

`http://www.cs.purdue.edu/homes/dgleich/codes/hkgrow`

Global heat kernel code available at

`http://www.cs.purdue.edu/homes/dgleich/codes/nexpokit/`

Ongoing work

- generalizing to other diffusions
- simultaneously compute multiple diffusions

Questions or suggestions? Email Kyle Kloster at `kkloste-at-purdue-dot-edu`