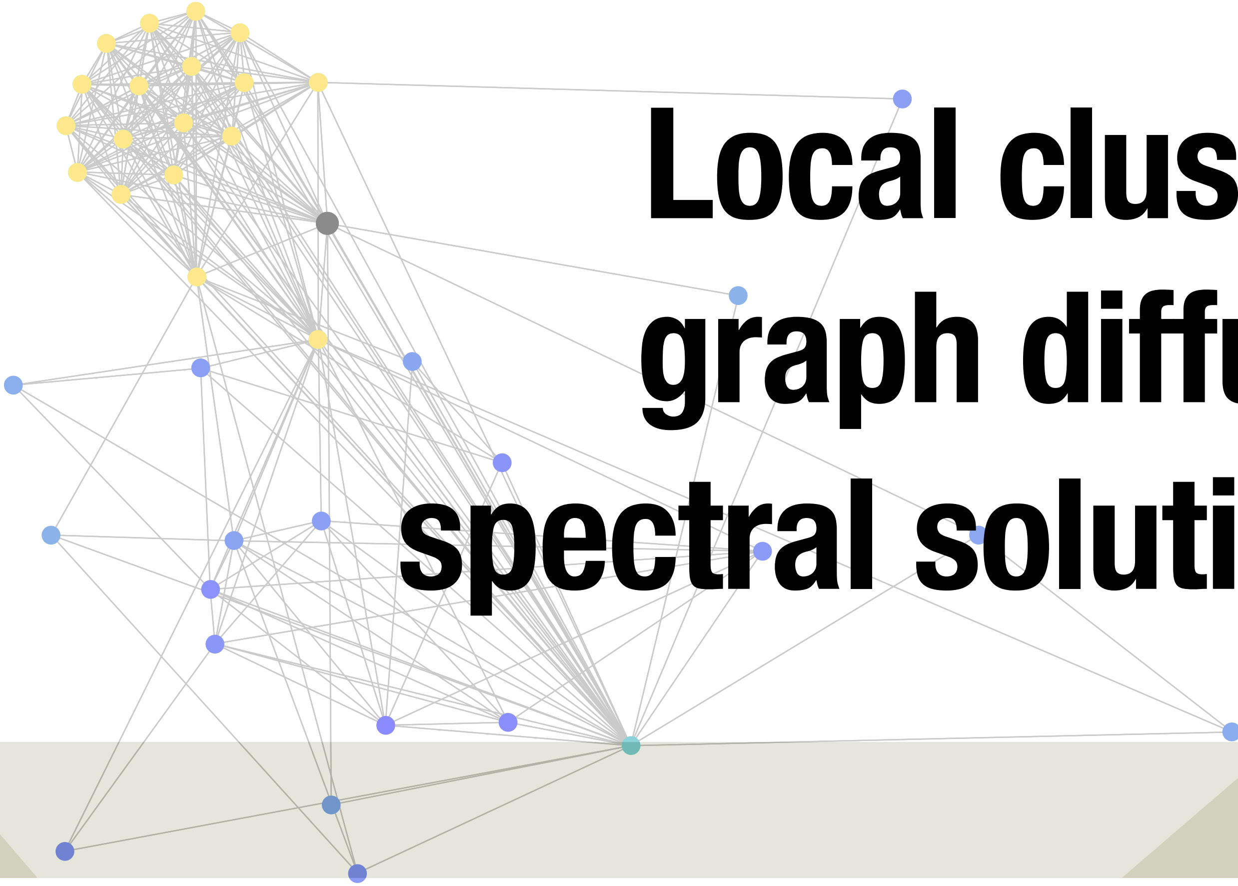
Local clustering with graph diffusions and spectral solution paths



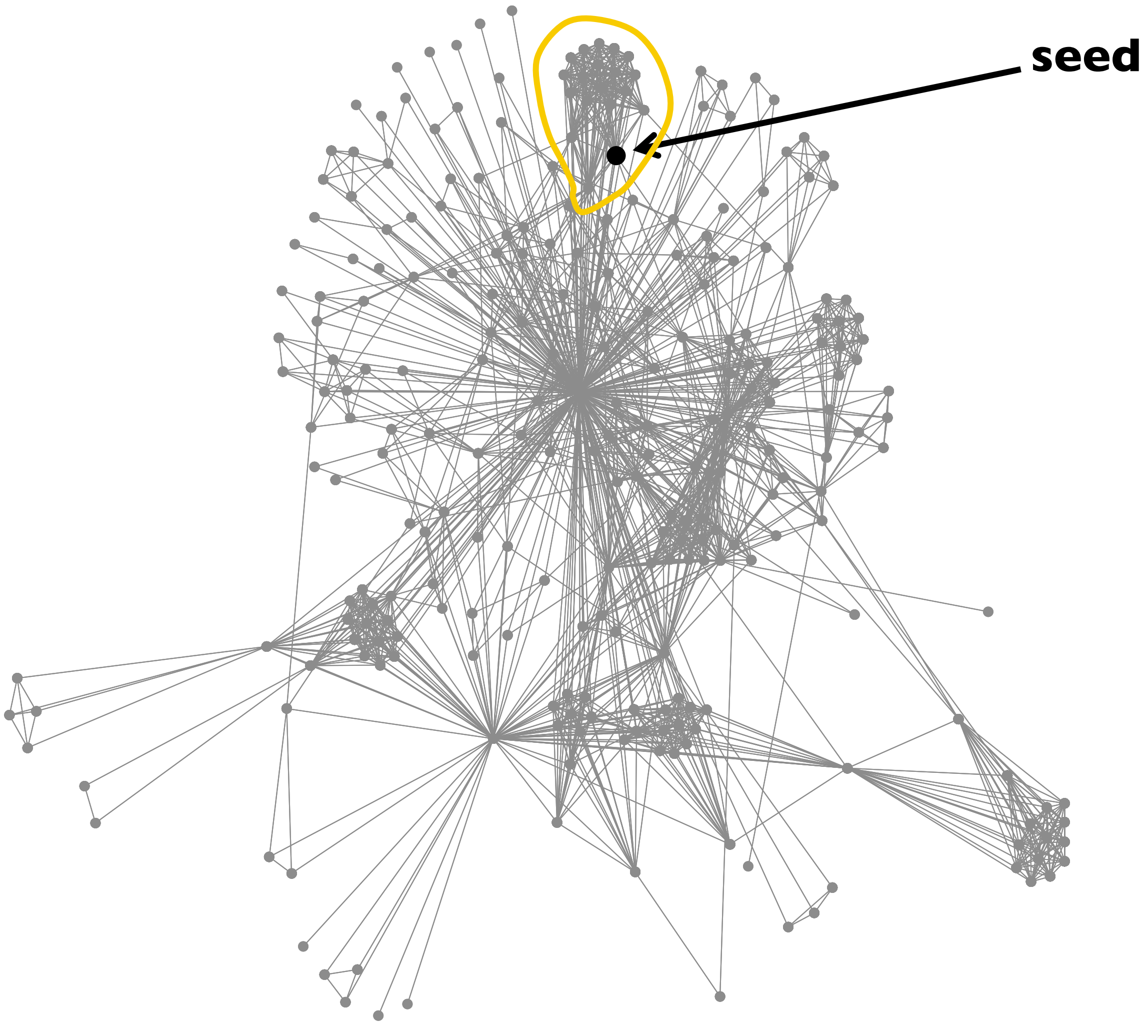
Joint with
David F. Gleich,
(Purdue), supported by
NSF CAREER
1149756-CCF

PURDUE
UNIVERSITY

Kyle Kloster
Purdue University

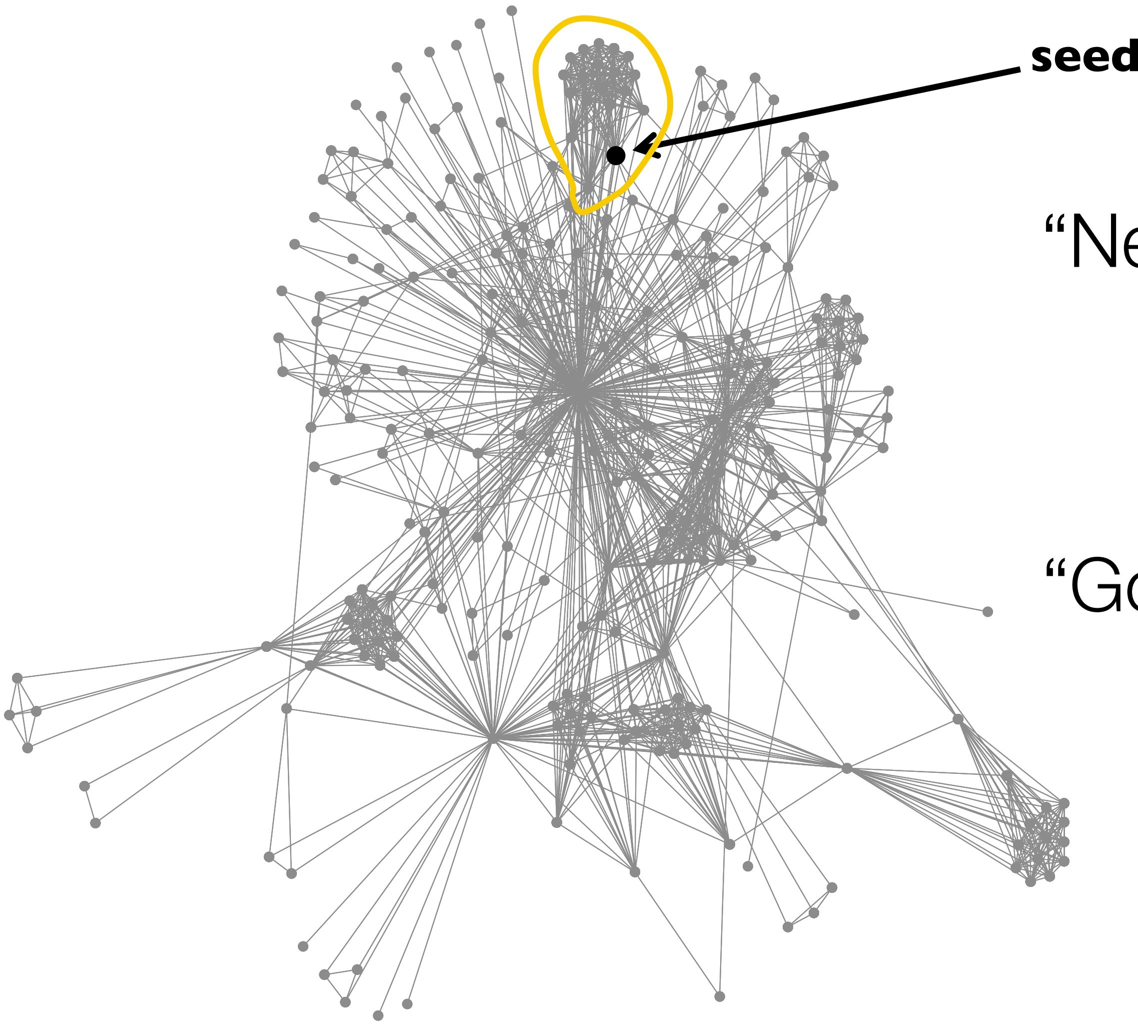
Local Clustering

Given seed(s) *S* in *G*, find a good cluster *near* S



Local Clustering

Given seed(s) *S* in *G*, find a good cluster *near* S



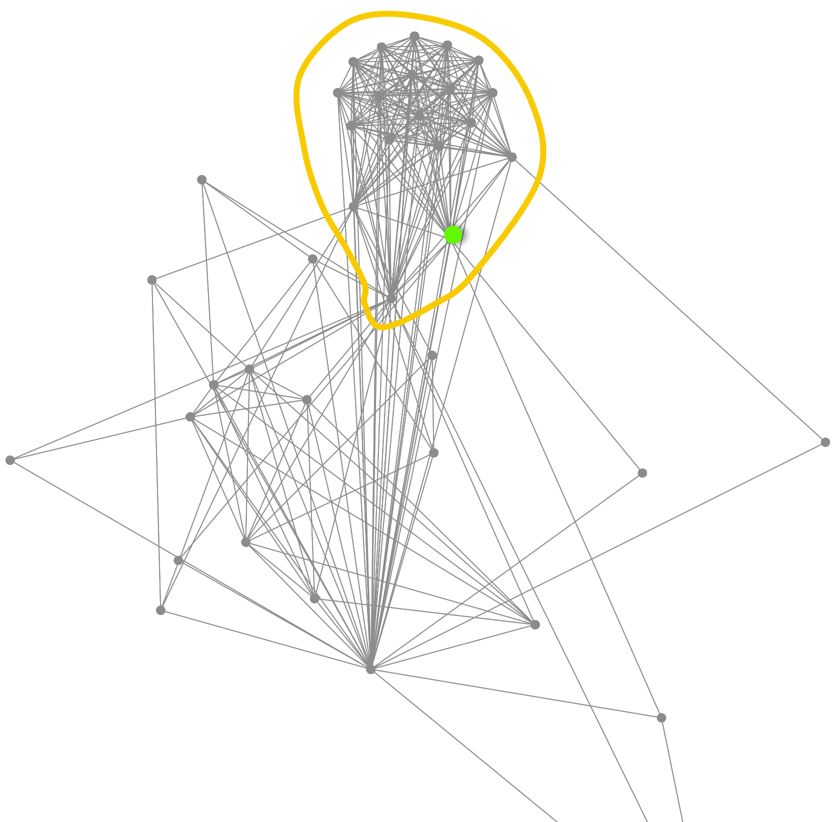
"Near"? ->
local, small
containing S

"Good"? ->
low conductance

Low-conductance sets are clusters

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

(for small sets T,
i.e. vol(T) < vol(G)/2)

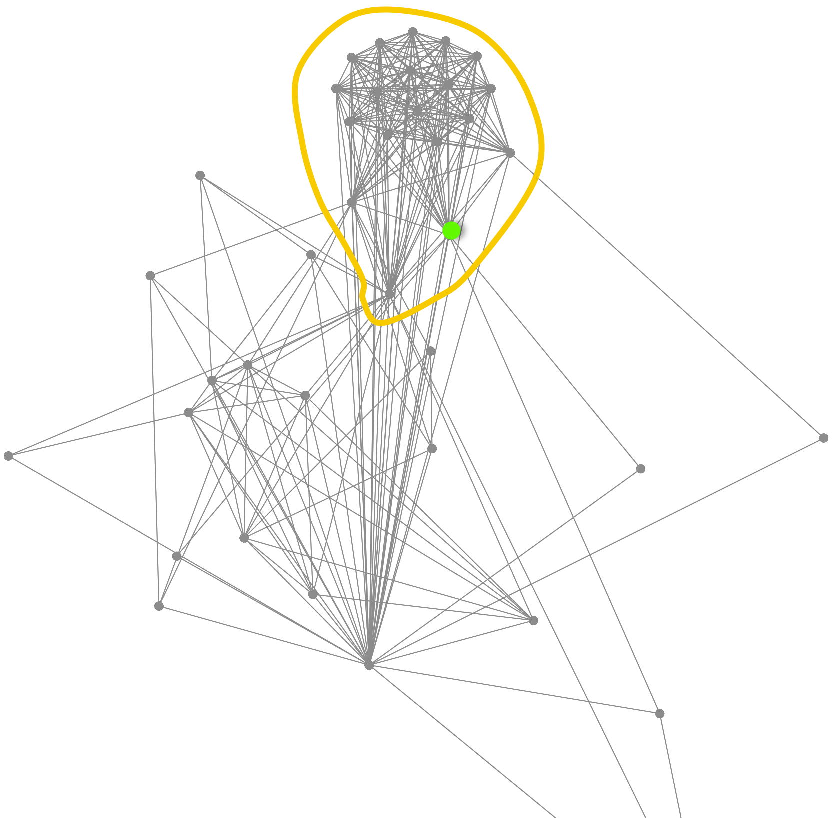


= " chance a random edge
that touches T exits *T* "

Low-conductance sets are clusters

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

(for small sets T,
i.e. vol(T) < vol(G)/2)



For a global cluster,

could use Fiedler…

But we want

a local cluster

Fiedler

Compute Fiedler vector, **v**:

$$\boldsymbol{L}\mathbf{v} = \lambda_2 \boldsymbol{D}\mathbf{v}$$

“Sweep” over **v**:

1. sort:

$$v(1) \geq v(2) \geq \cdots$$

2. for each set $S_k = (1,\ldots,k)$ compute conductance

$$\phi(S_k)$$

3. output best S_k

Fiedler

Compute Fiedler vector, **v**:

$$\boldsymbol{L}\mathbf{v} = \lambda_2 \boldsymbol{D}\mathbf{v}$$

"Sweep" over **v**:

1. sort:

$$v(1) \geq v(2) \geq \cdots$$

2. for each set S_k = (1,…,k) compute conductance

$$\phi(S_k)$$

3. output best S_k

Cheeger Inequality: Fiedler finds a cluster "not too much worse" than global optimal

But we want **local…**

Local Fiedler and diffusions

[Mahoney Orecchia Vishnoi 12]

“A local spectral method…”

$$\boldsymbol{L}\mathbf{v} = \boldsymbol{D}\mathbf{v}[\lambda]$$ **Fiedler**

$$\boldsymbol{L}\mathbf{v} = \boldsymbol{D}\mathbf{v}[\lambda] + \text{“}\mathbf{s}\text{”}$$ **with local bias** (MOV)

(normalized seed vector **s**)

THM: MOV is a scaling of personalized PageRank*!

Local Fiedler and diffusions

Intuition: why MOV ~ PageRank

$$\boldsymbol{L}\mathbf{v} = \boldsymbol{D}\mathbf{v}[\lambda]$$ **Fiedler**

$$\boldsymbol{L}\mathbf{v} = \boldsymbol{D}\mathbf{v}[\lambda] + \text{“}\mathbf{s}\text{”}$$ **with local bias**

$$(\boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2})\hat{\mathbf{v}} = \hat{\mathbf{v}}[\lambda] + \text{“}\mathbf{s}\text{”}$$

$$\boldsymbol{A}\boldsymbol{D}^{-1}\hat{\mathbf{v}} = \hat{\mathbf{v}}[1-\lambda] + \text{“}\mathbf{s}\text{”}$$

$$(\boldsymbol{I} - \alpha\boldsymbol{P})\,\hat{\mathbf{v}} = \text{“}\mathbf{s}\text{”}$$ **PageRank vector, a diffusion**

PageRank and other diffusions

"Personalized" PageRank (PPR)

[Andersen, Chung, Lang 06]: **local** Cheeger inequality and fast algorithm, "Push" procedure

Standard setting

$$(\boldsymbol{I} - \alpha \boldsymbol{P})\,\mathbf{x} = \hat{\mathbf{s}} \longrightarrow$$

Diffusion perspective

$$\mathbf{x} = \sum_{k=0} \alpha^k \boldsymbol{P}^k \hat{\mathbf{s}}$$

PageRank and other diffusions

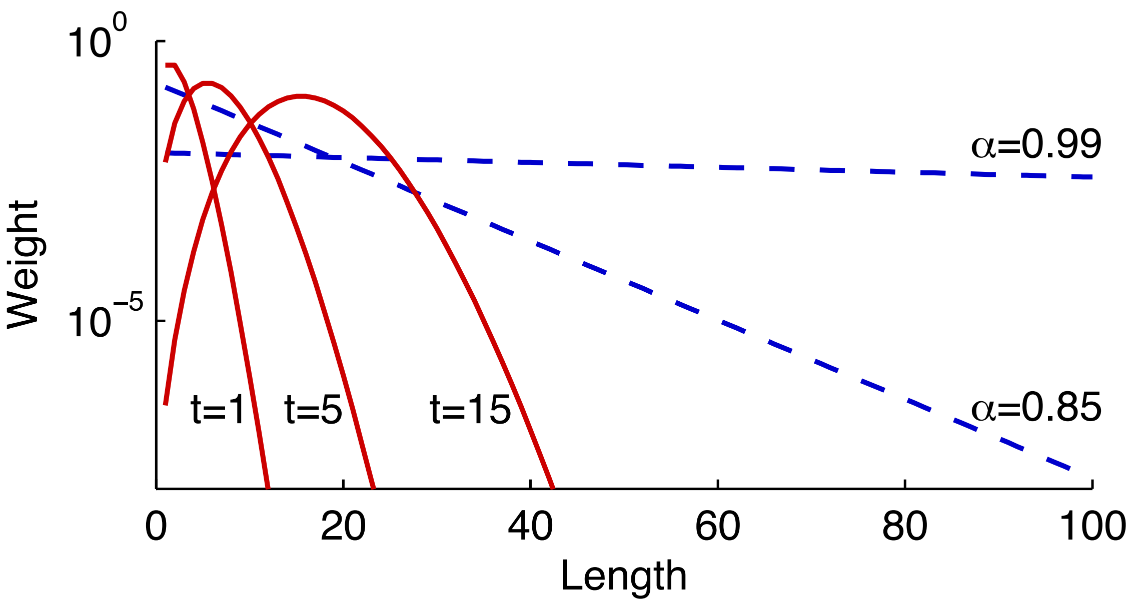
"Personalized" PageRank (PPR)

$$\mathbf{x} = \sum_{k=0} \alpha^k \boldsymbol{P}^k \hat{\mathbf{s}}$$

[Andersen, Chung, Lang 06]: local Cheeger inequality and fast algorithm, "Push" procedure

Heat Kernel diffusion (HK)
(many more!)

$$\mathbf{f} = \sum_{k=0} \frac{t^k}{k!} \boldsymbol{P}^k \hat{\mathbf{s}}$$



Various diffusions explore different aspects of graphs.

Diffusions, theory & practice

	good conductance	fast algorithm
PR	Local Cheeger Inequality	[Andersen Chung Lang 06] "PPR-push" is $O(1/(\varepsilon(1-\alpha)))$
HK	Local Cheeger Inequality [Chung 07]	[K., Gleich 2014] "HK-push" is $O(e^t C/\varepsilon)$
TDPR	Open question	[Avron, Horesh 2015]
Gen Diff	Open question	This talk

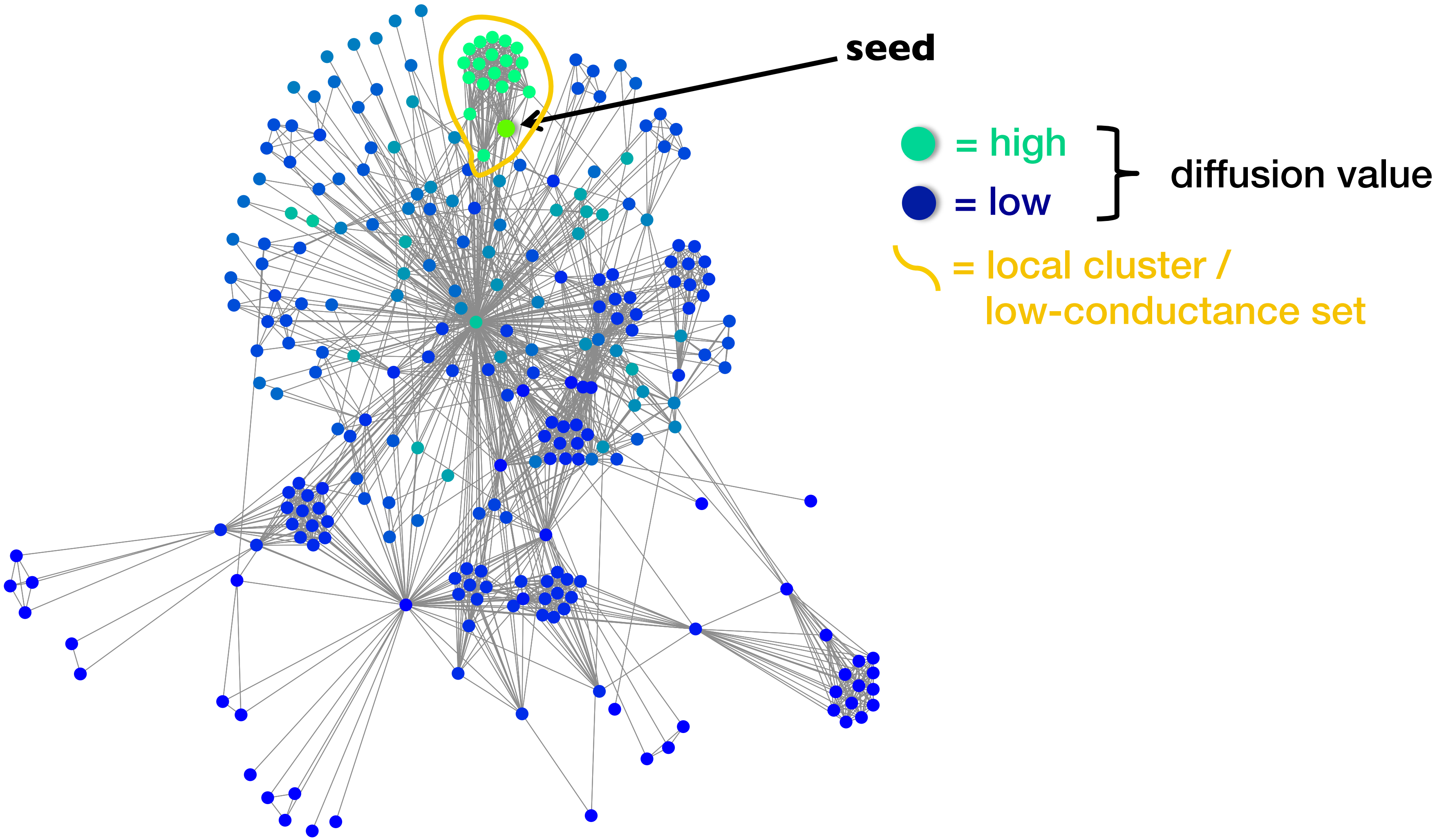
Diffusions, theory & practice

	good conductance	fast algorithm
PR	Local Cheeger Inequality	[Andersen Chung Lang 06] "PPR-push" is $O(1/(\varepsilon(1-\alpha)))$
HK	Local Cheeger Inequality [Chung 07]	[K., Gleich 2014] "HK-push" is $O(e^t C/\varepsilon)$
TDPR	Open question	[Avron, Horesh 2015]
Gen Diff	Open question	This talk

David Gleich and I are working with Olivia Simpson (a student of Fan Chung's)

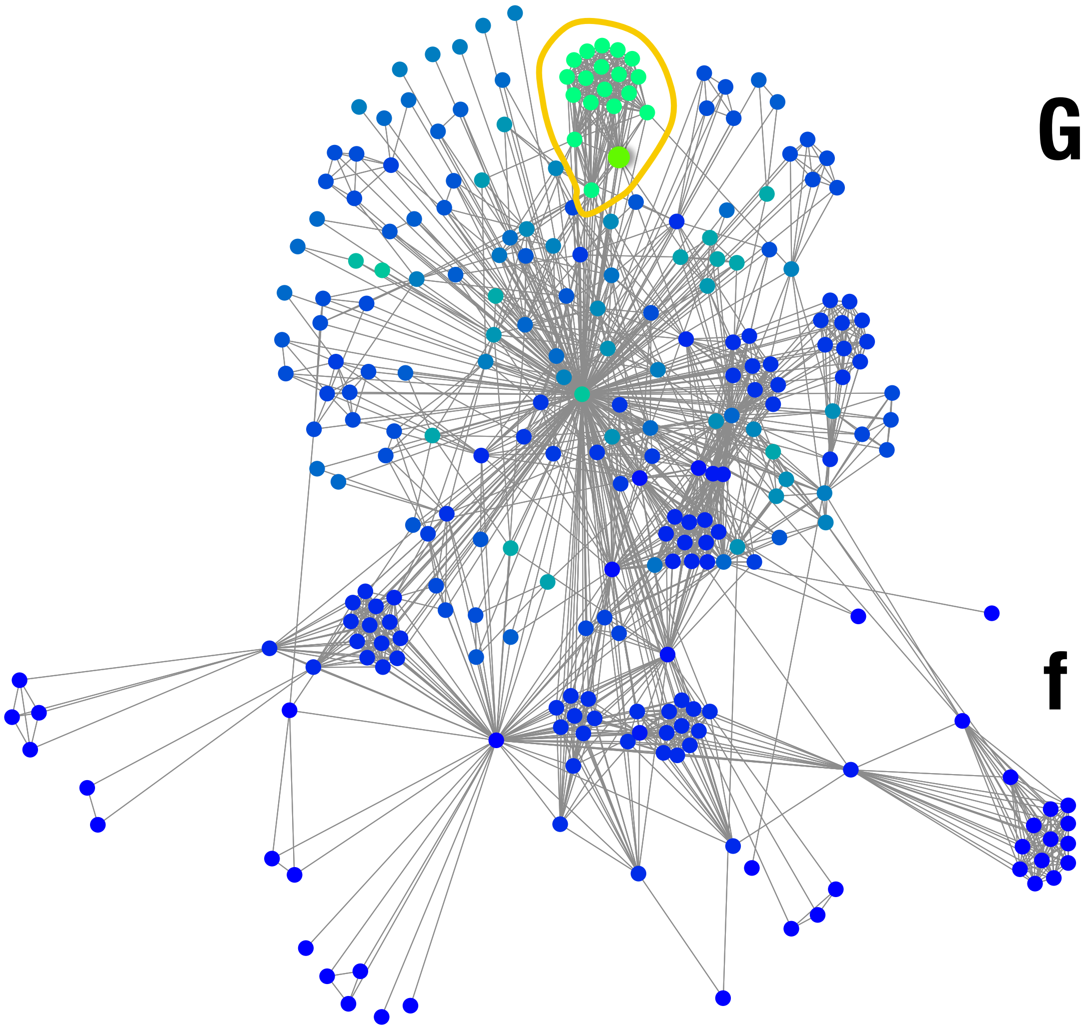
General diffusions: intuition

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



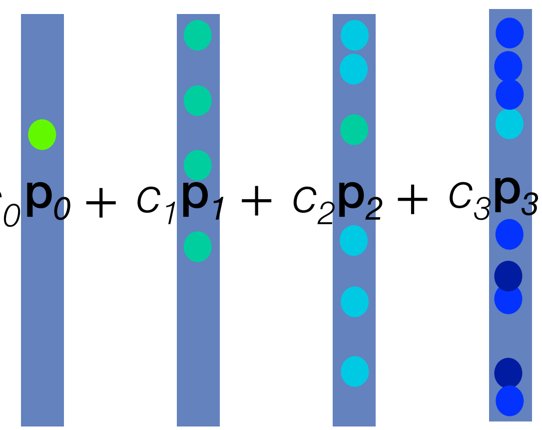
General diffusions

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



General diffusion vector

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



$$\mathbf{f} = c_0 \mathbf{p}_0 + c_1 \mathbf{p}_1 + c_2 \mathbf{p}_2 + c_3 \mathbf{p}_3 + \ldots$$

Sweep over **f**!

General algorithm

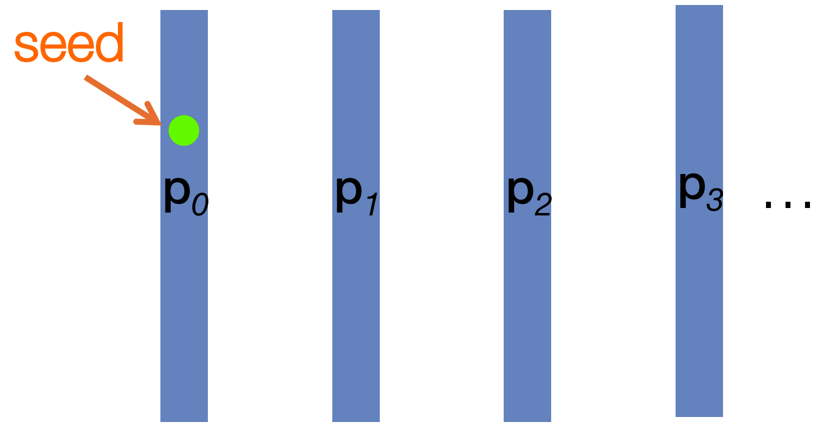
1. Approximate **f** so $\|\boldsymbol{D}^{-1}(\mathbf{f} - \hat{\mathbf{f}})\|_\infty \leq \epsilon$
2. Scale, $\boldsymbol{D}^{-1}\hat{\mathbf{f}}$
3. Then sweep!

How to do this efficiently?

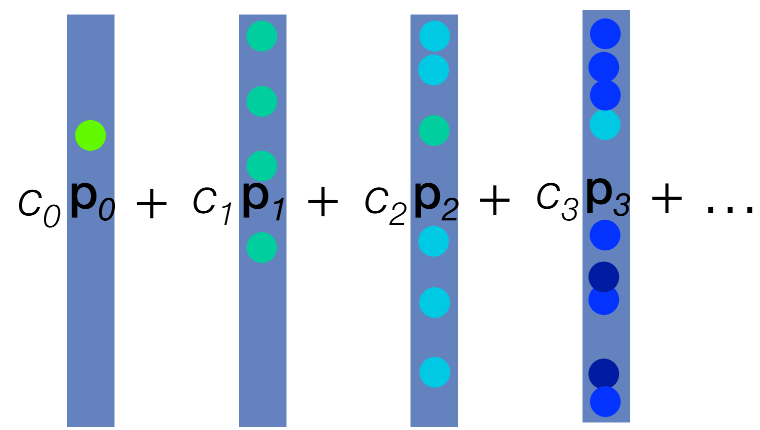
Algorithm Intuition

From parameters c_k, ε, seed $\mathbf{s}$ …

Starting from here…



How to end up here?

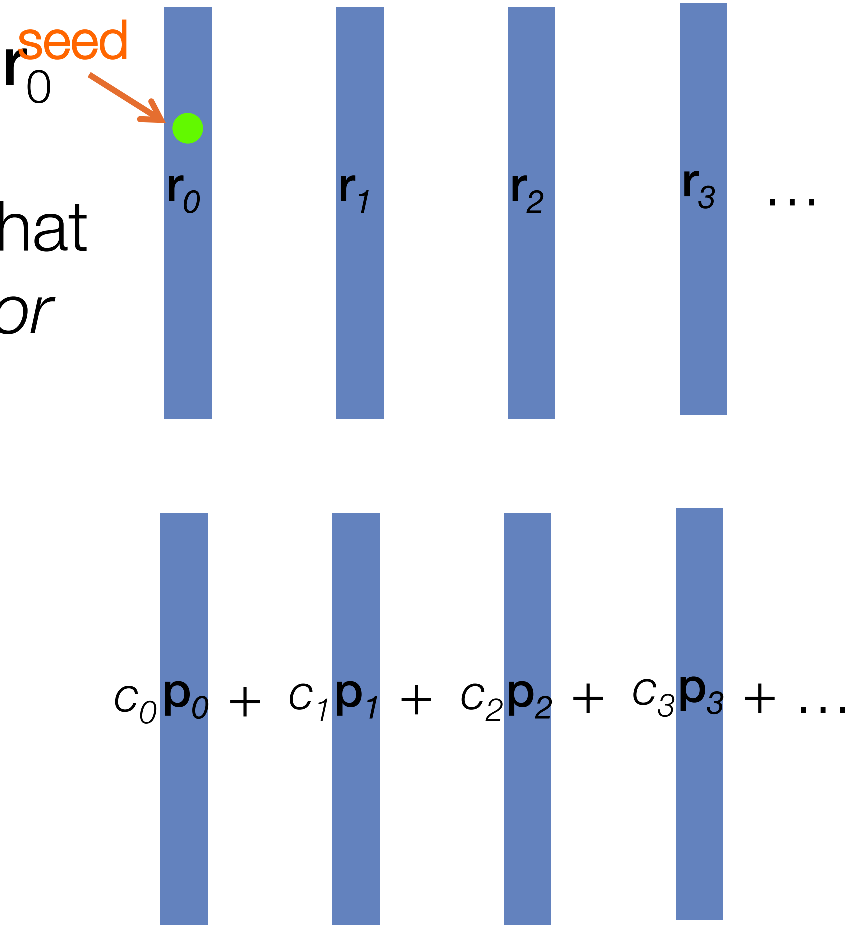


Algorithm Intuition

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

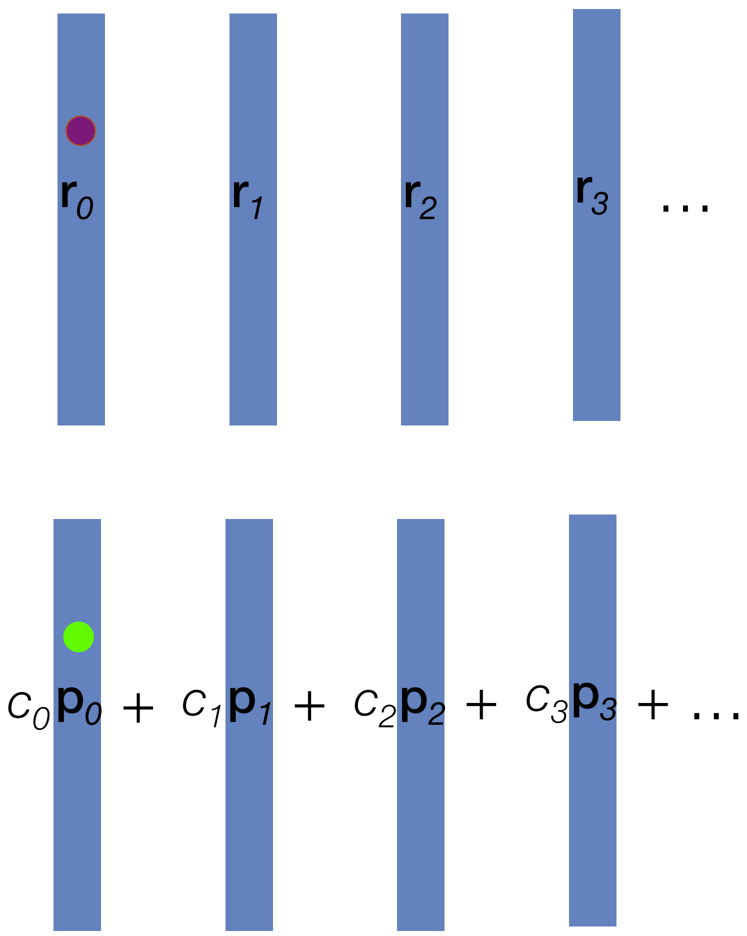
The residuals $\mathbf{r}_k$ hold mass that is unprocessed – it's like *error*

Idea: "push" any entry $r_k(j)/\ d_j >$ (some threshold)



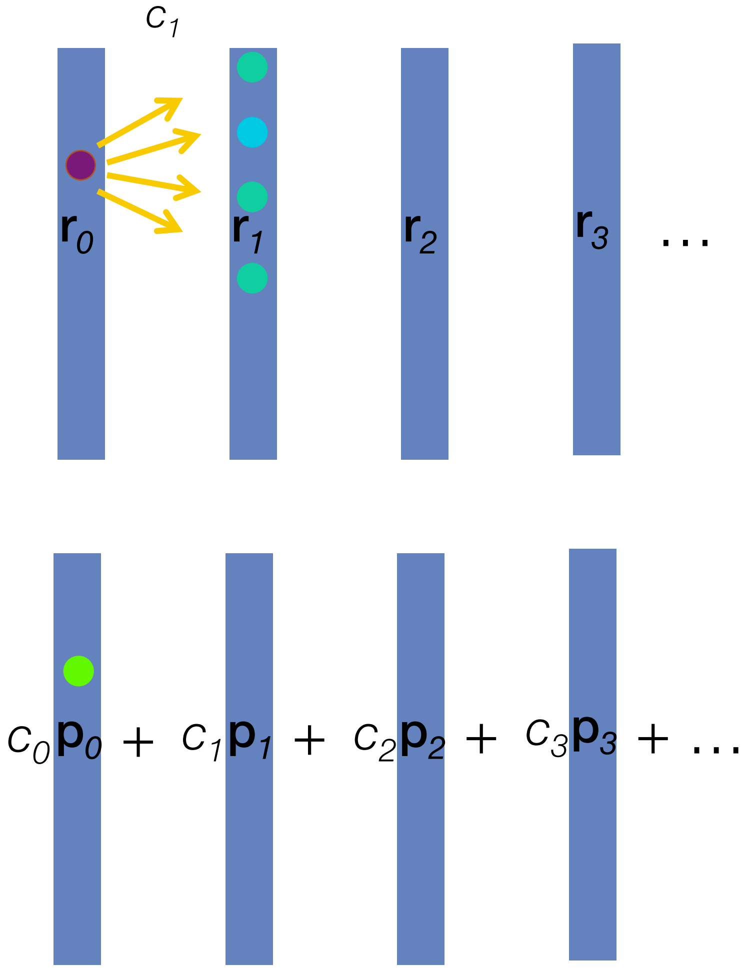
Push Operation

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



Push Operation

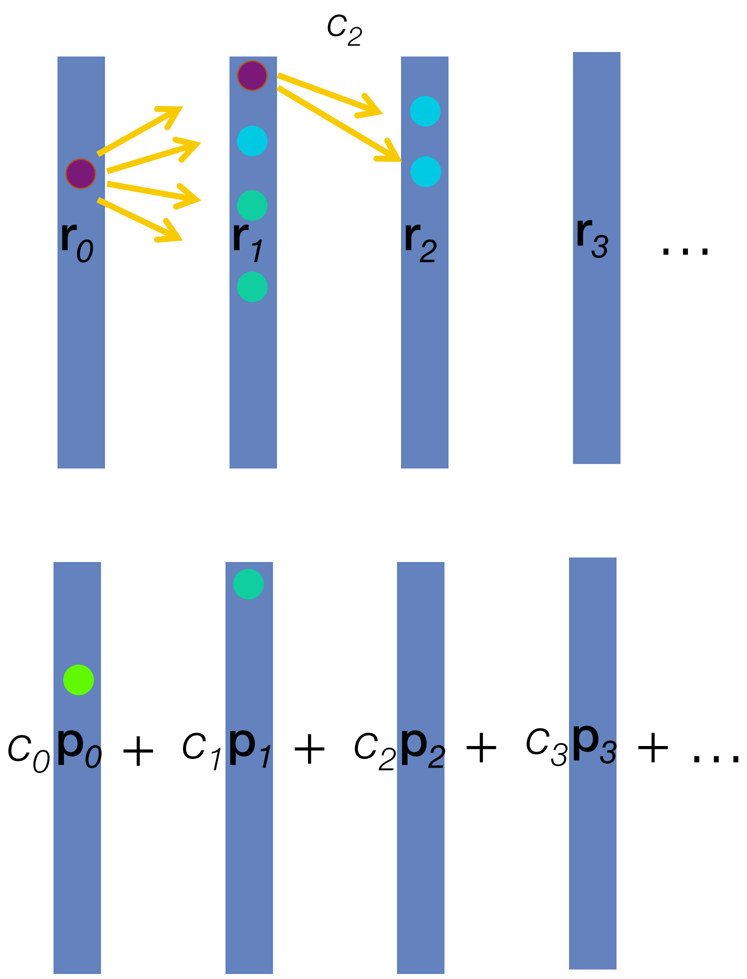
push – (1) remove entry in $\mathbf{r}_k$,
(2) put in **f**,
(3) then scale and spread to neighbors in next **r**



Push Operation

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

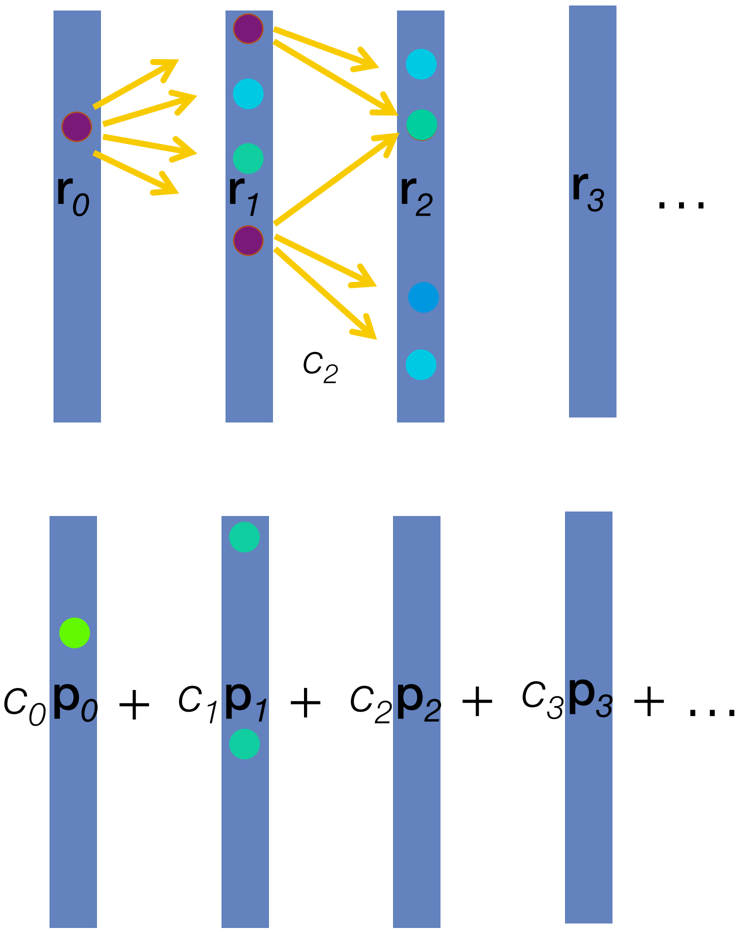
(repeat)



Push Operation

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

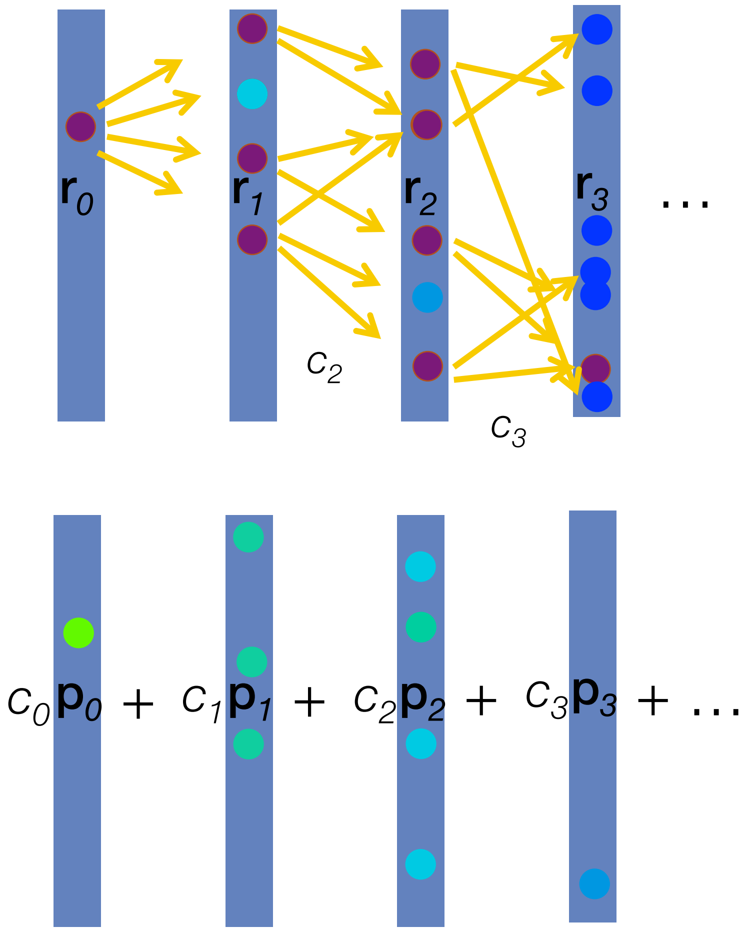
(repeat)



Push Operation

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

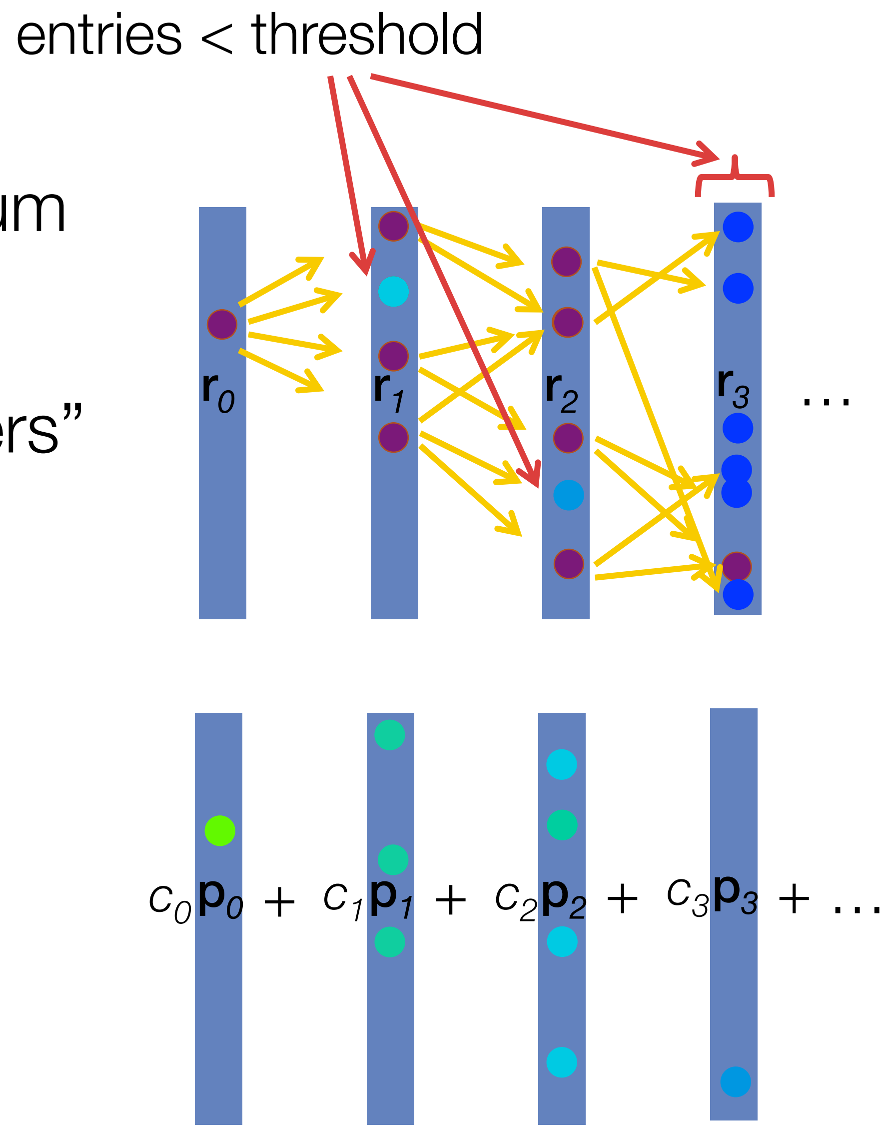
(repeat)



Thresholds

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

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



Thresholds

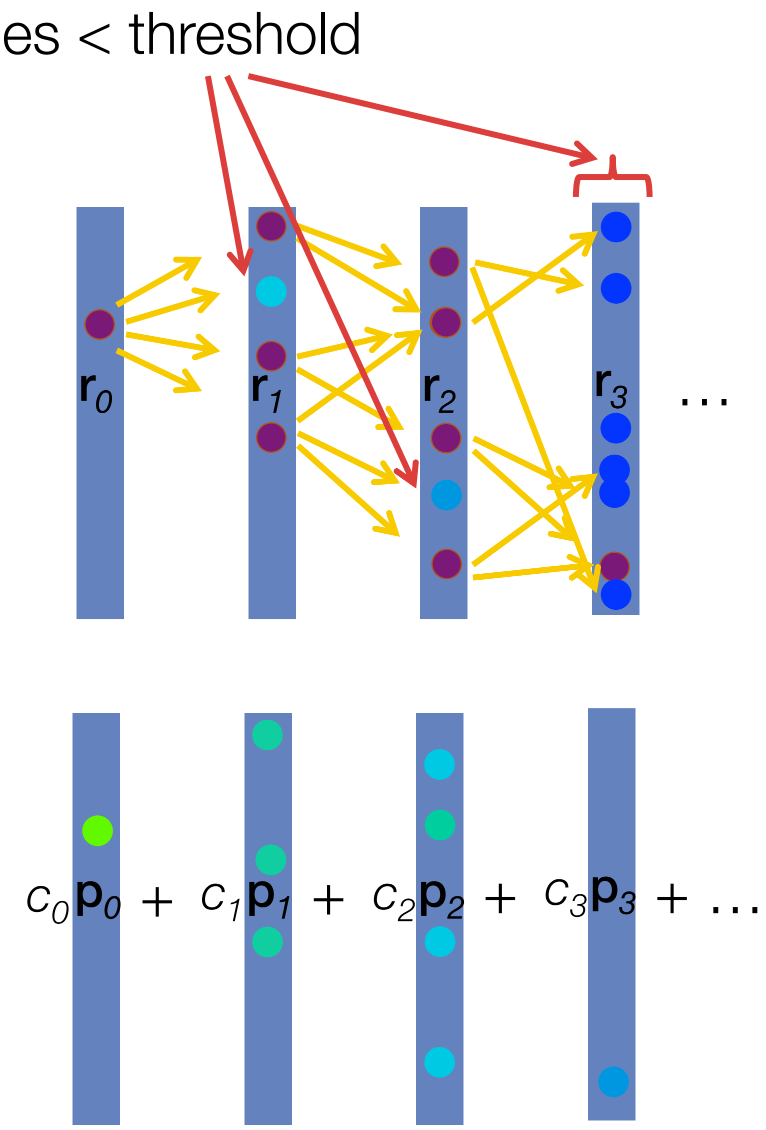
ERROR equals weighted sum of entries left in $\mathbf{r}_k$

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

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

$$\epsilon / \left(\sum_{j=k+1}^{\infty} c_j \right)$$

Then $\|\boldsymbol{D}^{-1}(\mathbf{f} - \hat{\mathbf{f}})\|_\infty \leq \epsilon$



Another perspective

$$\boldsymbol{L}\mathbf{v} = \boldsymbol{D}\mathbf{v}[\lambda]$$ Fiedler

$$\boldsymbol{L}\mathbf{v} = \boldsymbol{D}\mathbf{v}[\lambda] + \text{“}\mathbf{s}\text{”}$$ with local bias

$$(\boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2})\hat{\mathbf{v}} = \hat{\mathbf{v}}[\lambda] + \text{“}\mathbf{s}\text{”}$$

$$\boldsymbol{A}\boldsymbol{D}^{-1}\hat{\mathbf{v}} = \hat{\mathbf{v}}[1 - \lambda] + \text{“}\mathbf{s}\text{”}$$

$$(\boldsymbol{I} - \alpha \boldsymbol{P})\,\hat{\mathbf{v}} = \text{“}\mathbf{s}\text{”}$$ PageRank vector, a diffusion

Another perspective

$$\boldsymbol{L}\mathbf{V}_k = \boldsymbol{D}\mathbf{V}_k\Lambda_k$$ **Fiedler**

$$\boldsymbol{L}\mathbf{V}_k = \boldsymbol{D}\mathbf{V}_k\Lambda_k + \mathbf{S}$$ **with local bias**

$$(\boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2})\hat{\mathbf{V}}_k = \hat{\mathbf{V}}_k\Lambda_k + \hat{\mathbf{S}}$$

$$\boldsymbol{A}\boldsymbol{D}^{-1}\hat{\mathbf{V}}_k = \hat{\mathbf{V}}_k(\mathbf{I} - \Lambda_k) + \hat{\mathbf{S}}$$

Another perspective

$$\boldsymbol{L}\mathbf{V}_k = \boldsymbol{D}\mathbf{V}_k\Lambda_k$$ **Fiedler**

$$\boldsymbol{L}\mathbf{V}_k = \boldsymbol{D}\mathbf{V}_k\Lambda_k + \mathbf{S}$$ **with local bias**

$$(\boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2})\hat{\mathbf{V}}_k = \hat{\mathbf{V}}_k\Lambda_k + \hat{\mathbf{S}}$$

$$\boldsymbol{A}\boldsymbol{D}^{-1}\hat{\mathbf{V}}_k = \hat{\mathbf{V}}_k(\mathbf{I} - \Lambda_k) + \hat{\mathbf{S}}$$

$$\boldsymbol{P}\hat{\mathbf{V}}_k\Gamma = \hat{\mathbf{V}}_k + \bar{\mathbf{S}}$$

Mix-product property
For Kronecker product

Another perspective

$$\boldsymbol{L}\mathbf{V}_k = \boldsymbol{D}\mathbf{V}_k\Lambda_k$$ **Fiedler**

$$\boldsymbol{L}\mathbf{V}_k = \boldsymbol{D}\mathbf{V}_k\Lambda_k + \mathbf{S}$$ **with local bias**

$$(\boldsymbol{I} - \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2})\hat{\mathbf{V}}_k = \hat{\mathbf{V}}_k\Lambda_k + \hat{\mathbf{S}}$$

$$\boldsymbol{A}\boldsymbol{D}^{-1}\hat{\mathbf{V}}_k = \hat{\mathbf{V}}_k(\mathbf{I} - \Lambda_k) + \hat{\mathbf{S}}$$

$$\boldsymbol{P}\hat{\mathbf{V}}_k\Gamma = \hat{\mathbf{V}}_k + \bar{\mathbf{S}}$$

Mix-product property
For Kronecker product

$$(\boldsymbol{I} - \Gamma^T \otimes \boldsymbol{P})\text{vec}(\hat{\mathbf{V}}_k) = \text{vec}(\tilde{\mathbf{S}})$$

Another perspective

$$\left.\begin{array}{c}(\boldsymbol{I} - \Gamma^T \otimes \boldsymbol{P})\mathrm{vec}(\hat{\mathbf{V}}_k) = \mathrm{vec}(\tilde{\mathbf{S}}) \\ (\boldsymbol{I} - \alpha \boldsymbol{P})\,\hat{\mathbf{v}} = \tilde{\mathbf{s}}\end{array}\right\}$$

- generalizes PageRank to "matrix teleportation parameter"

Standard spectral approach: $\Gamma = (\boldsymbol{I} - \boldsymbol{\Lambda}_k)^{-1}$

Another perspective

$$\left.\begin{aligned}(\boldsymbol{I} - \Gamma^T \otimes \boldsymbol{P})\mathrm{vec}(\hat{\mathbf{V}}_k) &= \mathrm{vec}(\tilde{\mathbf{S}}) \\ (\boldsymbol{I} - \alpha \boldsymbol{P})\,\hat{\mathbf{v}} &= \tilde{\mathbf{s}}\end{aligned}\right\}$$

- generalizes PageRank to "matrix teleportation parameter"

Our framework is equivalent to:

$$\Gamma = \begin{bmatrix} 0 & \tilde{c}_0 & & \\ & 0 & \ddots & \\ & & \ddots & \tilde{c}_N \\ & & & 0 \end{bmatrix}$$

(Details in [K., Gleich KDD 14])

General diffusions: conclusion

THM: For diffusion coefficients $c_k >= 0$ satisfying

$$\sum_{k=0}^{\infty} c_k = 1 \quad \text{and} \quad \left.\sum_{k=0}^{N} c_k \leq \epsilon/2 \right\} \text{“rate of decay”}$$

“generalized push” approximates the diffusion **f**

on a symmetric graph so that $\|D^{-1}(\mathbf{f} - \hat{\mathbf{f}})\|_\infty \leq \epsilon$

in work bounded by $O(2N^2/\epsilon)$

Constant for any inputs!
(If diffusion decays fast)

Proof sketch

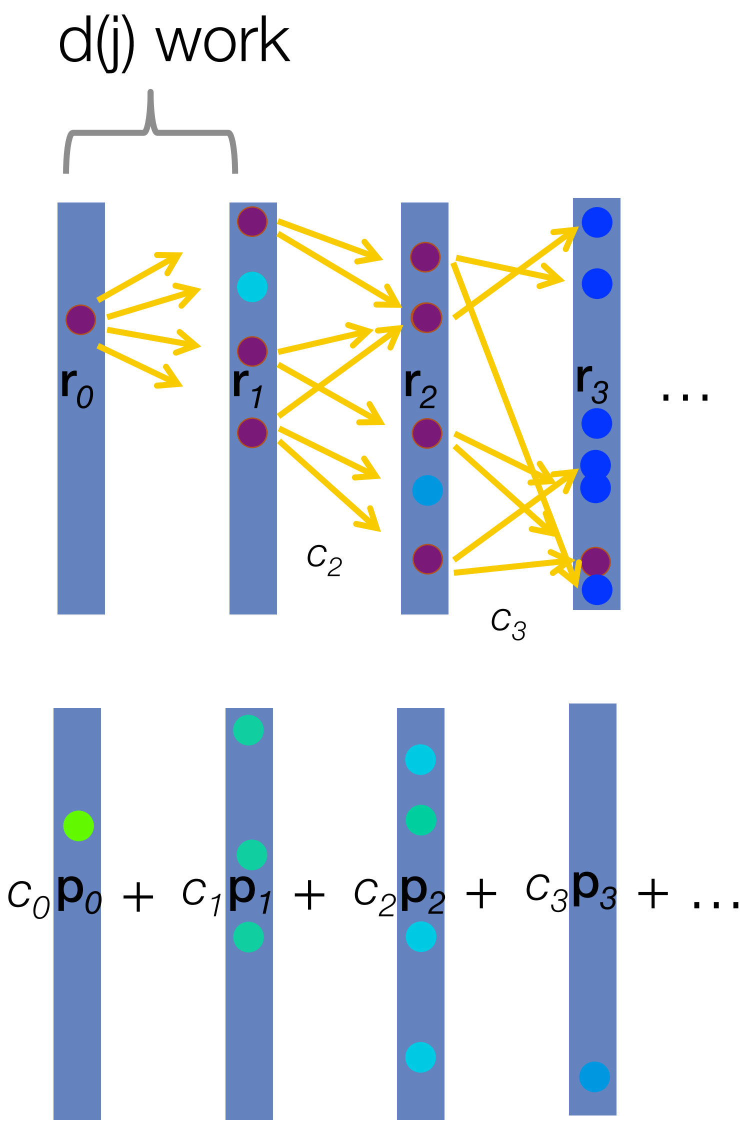
1. Stop pushing after N terms. $\sum_{k=0}^{N} c_k \leq \epsilon/2$

2. Push residual entries in first N terms if $r_k(j) \geq d(j)\epsilon/(2N)$

3. Total work is # pushes: $\sum_{k=0}^{N-1} \sum_{t=1}^{m_k} d(j_t)$

Push Recap

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



Proof sketch

1. Stop pushing after N terms. $\sum_{k=0}^{N} c_k \leq \epsilon/2$

2. Push residual entries in first N terms if $r_k(j) \geq d(j)\epsilon/(2N)$

3. Total work is # pushes: $\sum_{k=0}^{N-1} \sum_{t=1}^{m_k} d(j_t)$

Proof sketch

1. Stop pushing after N terms. $\sum_{k=0}^{N} c_k \leq \epsilon/2$

2. Push residual entries in first N terms if $r_k(j) \geq d(j)\epsilon/(2N)$

3. Total work is # pushes: $\sum_{k=0}^{N-1} \sum_{t=1}^{m_k} d(j_t) \leq \sum_{k=0}^{N-1} \sum_{t=1}^{m_k} r_k(j_t)(2N)/\epsilon$

Proof sketch

1. Stop pushing after N terms. $\sum_{k=0}^{N} c_k \leq \epsilon/2$

2. Push residual entries in first N terms if $r_k(j) \geq d(j)\epsilon/(2N)$

3. Total work is # pushes: $\sum_{k=0}^{N-1} \sum_{t=1}^{m_k} d(j_t) \leq \sum_{k=0}^{N-1} \sum_{t=1}^{m_k} r_k(j_t)(2N)/\epsilon$

4. Each r_k sums to <= 1
(each push is added to **f**, which sums to 1) $\sum_{t=1}^{m_k} r_k(j_t) \leq 1$

$$O(2N^2/\epsilon)$$

Solutions Paths

Benefit of these “push” diffusions?

A direct decomposition is a black box:
Feed in input, get output.

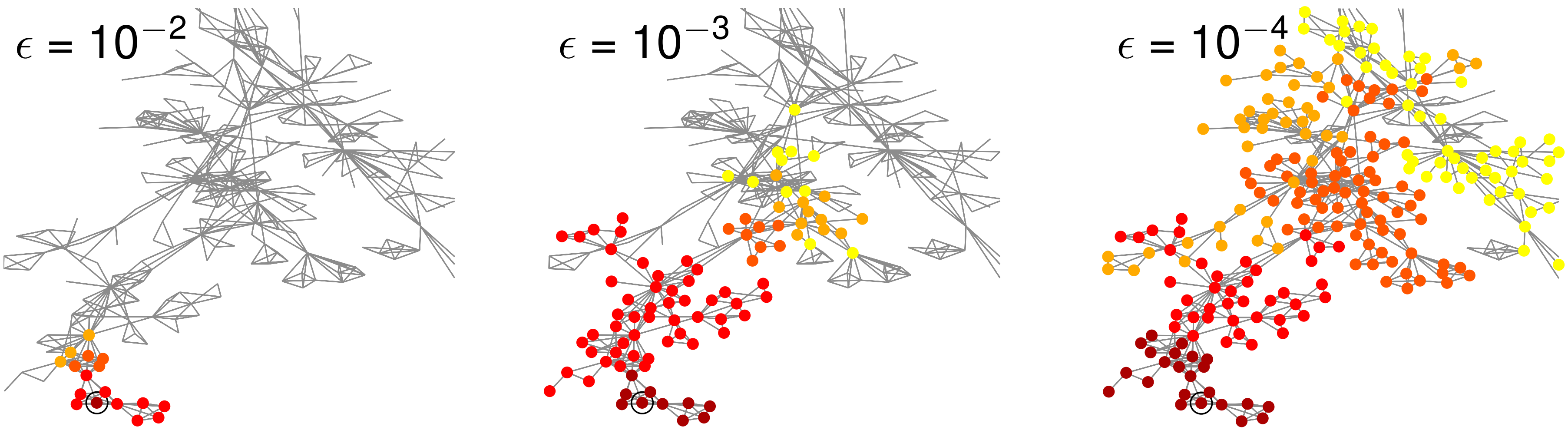
In contrast, the iterative nature of “push” means running the algorithm is essentially “watching” the diffusion process occur.

Solutions Paths

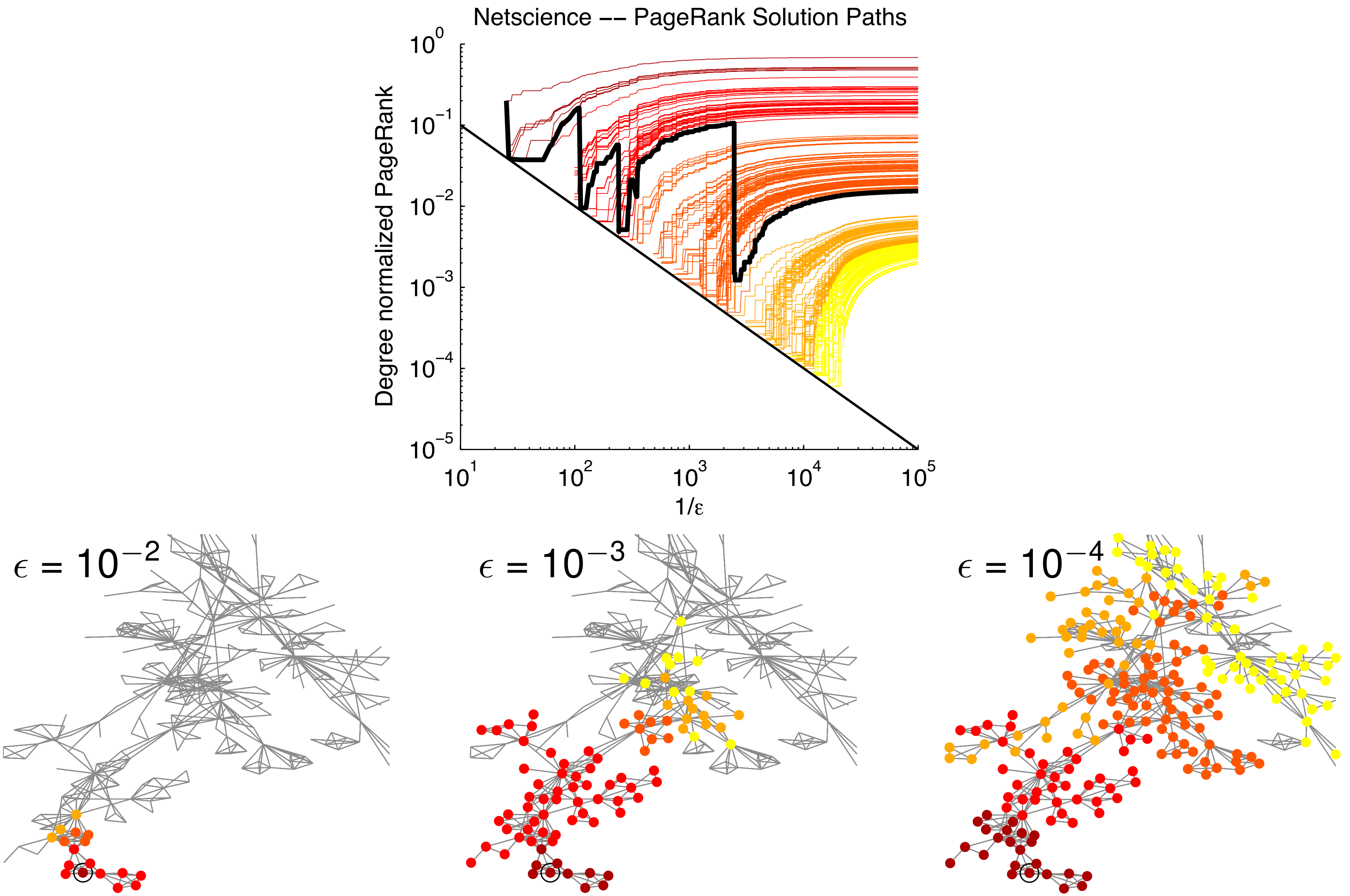
Benefit of these “push” diffusions?

A direct decomposition is a black box:
Feed in input, get output.

In contrast, the iterative nature of “push” means running the algorithm is essentially “watching” the diffusion process occur.



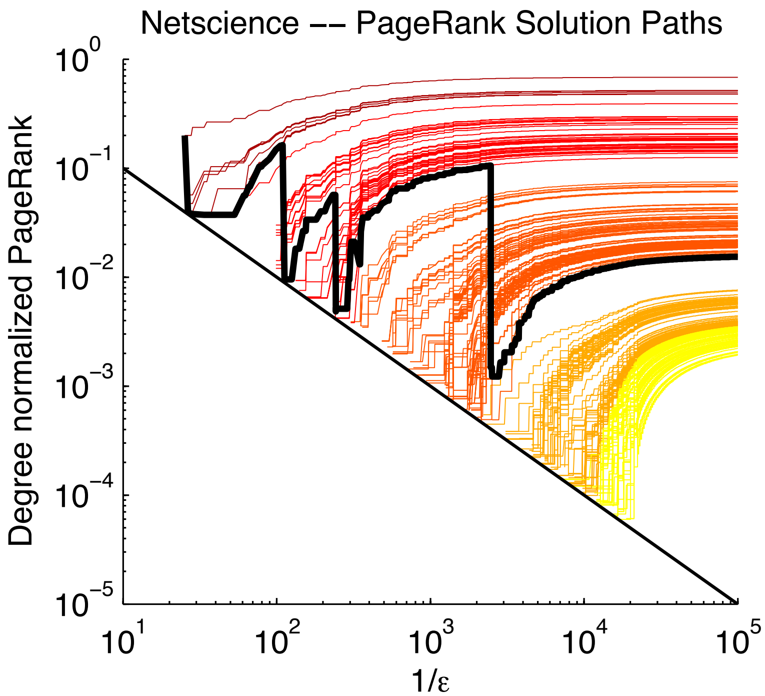
Solutions Paths

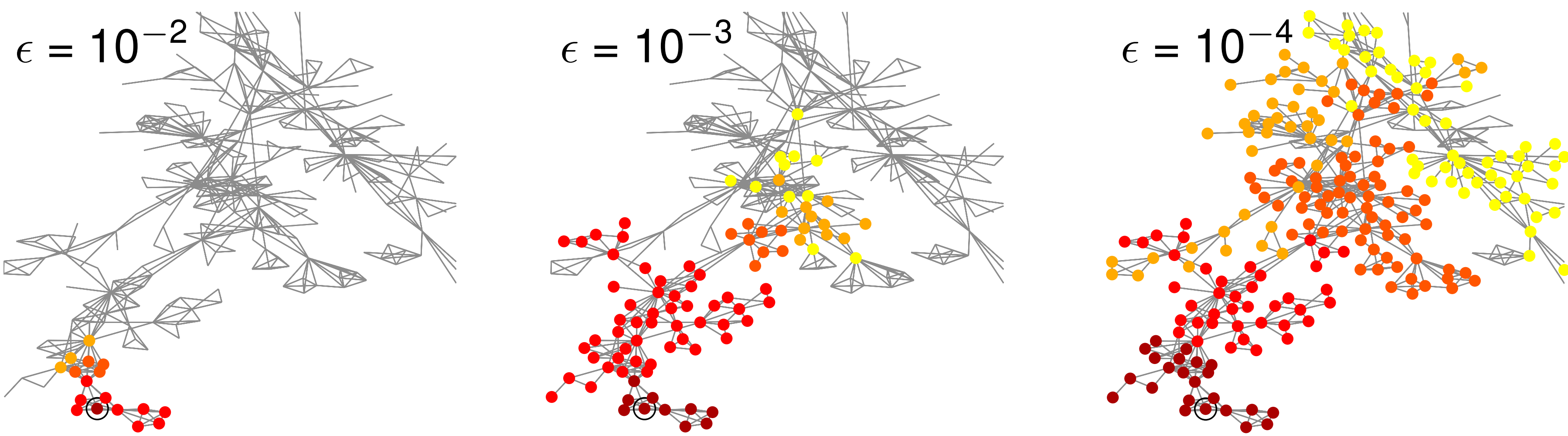


Solutions Paths

Each curve is a node. Its value increases as ε goes to 0.

Thick black line shows set of best conductance.

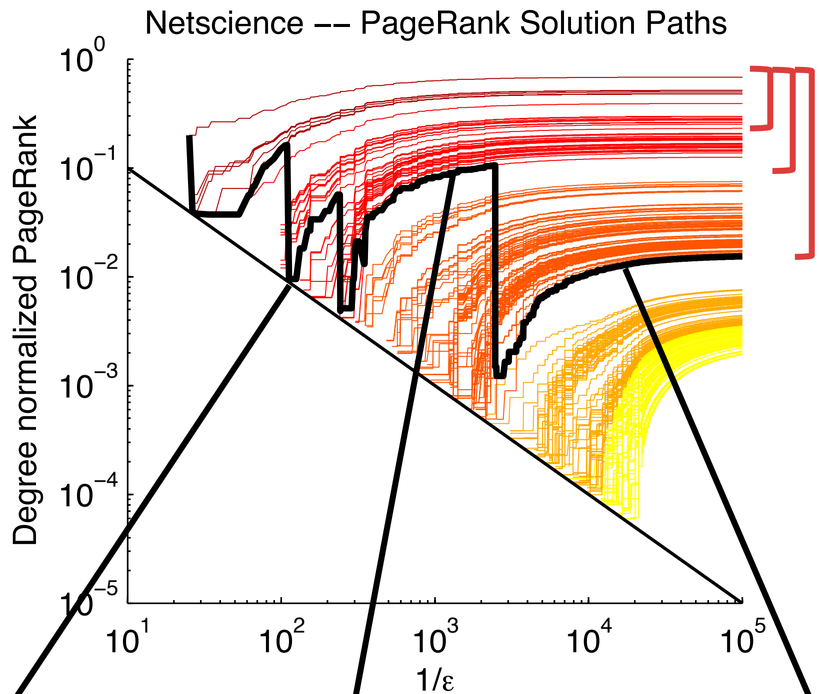




Solutions Paths

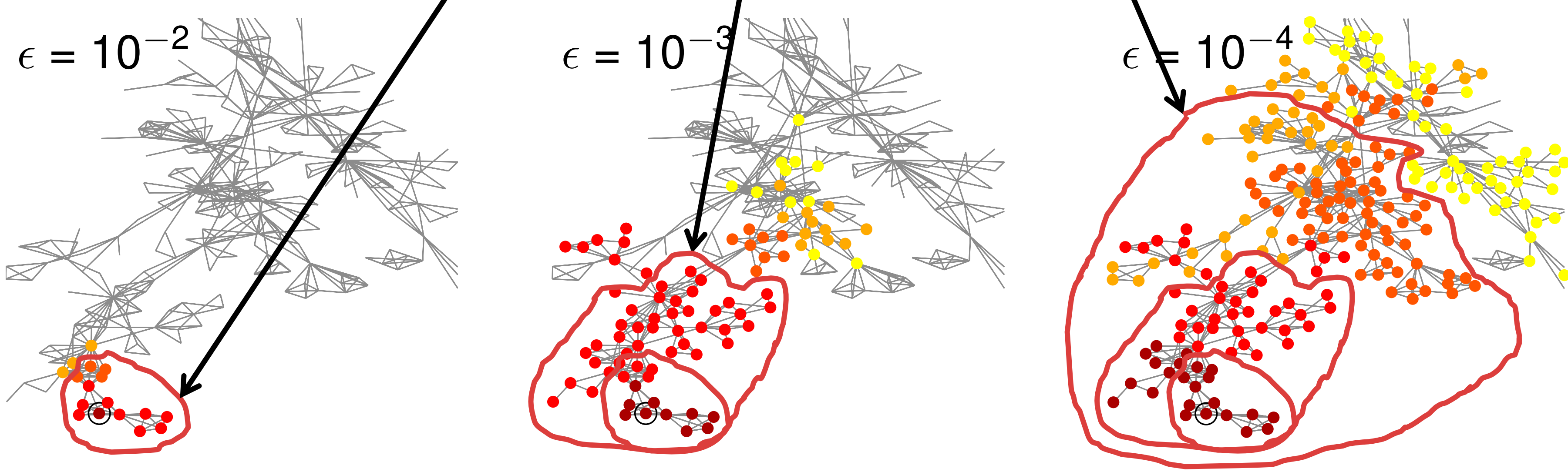
Each curve is a node. Its value increases as ε goes to 0.

Thick black line shows set of best conductance.



Bundles of curves are good clusters

Paths identify *nested* clusters



Solutions Paths

Locate nested, good-conductance sets that a single diffusion + sweep could miss.

Can be done efficiently because the constant-time approach to computing diffusions enables efficient storage and analysis of the push process

Total Paths work (for PageRank): $o\left(\frac{1}{\epsilon(1-\alpha)}\right)^2$

Still efficient!

Thank you

Heat kernel code available at

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

Solution paths: `http://arxiv.org/abs/1503.00322`

(Solution paths, generalized diffusion code soon)

Ongoing work

- Generalized local Cheeger Inequality
  for broader class of diffusions

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