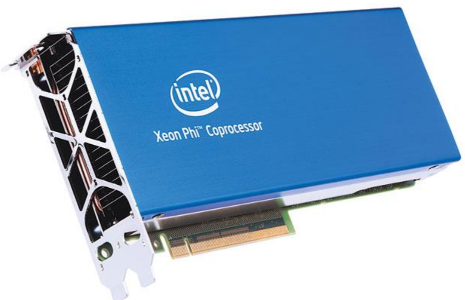
Parallel Graph Algorithms

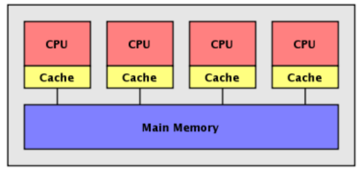
Richard Peng
Georgia Tech

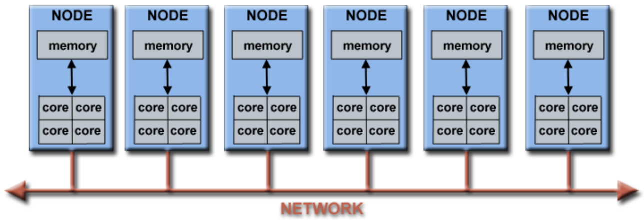
OUTLINE

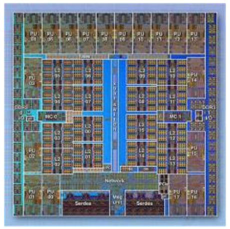
- **Model and problems**
- Graph decompositions
- Randomized clusterings
- Interface with optimization

THE MODEL









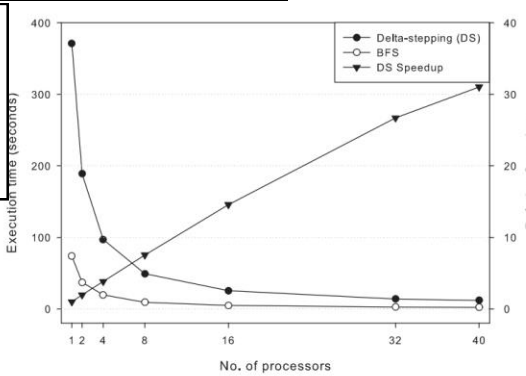
The `scale up' approach:

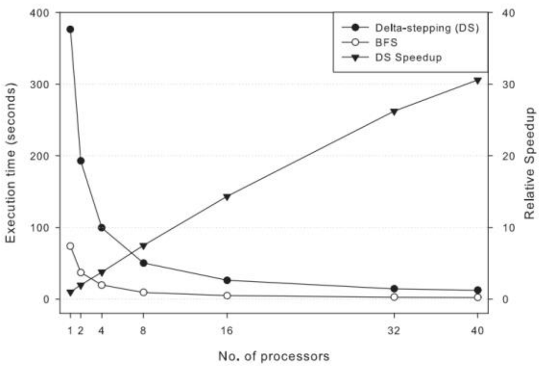
- Have a number of processors
- Shared memory

'Typcial' spec:

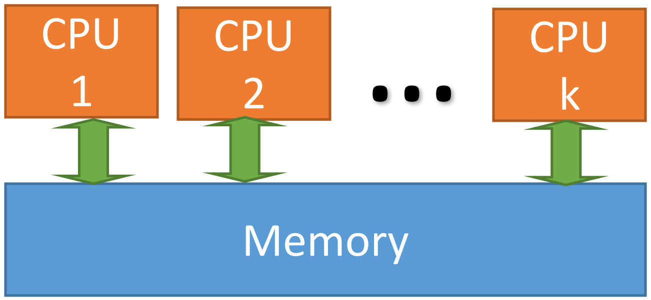
- 256 cores
- 1TB RAM

[Madduri-Bader-Berry-Crobak`07]: shortest path



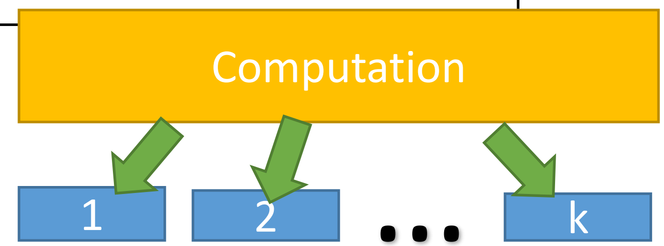


THE MODEL



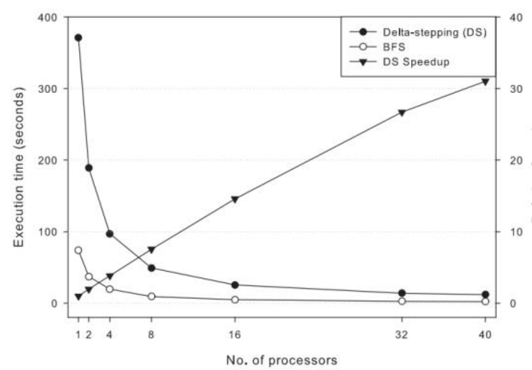
Better upper bounds under dubious(?) assumptions:

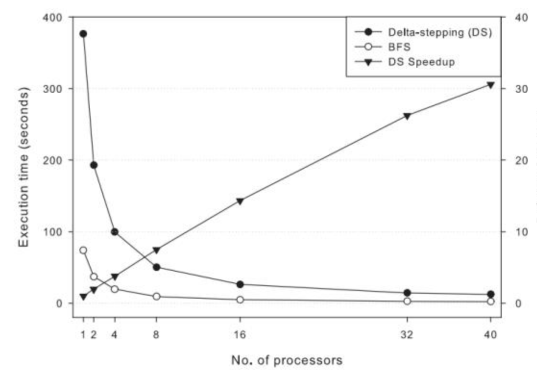
- Concurrent read/write, PRAM
- Unit cost memory



Split computation up across processors which then process them in parallel

Goal: prove that k processors → factor k speedups

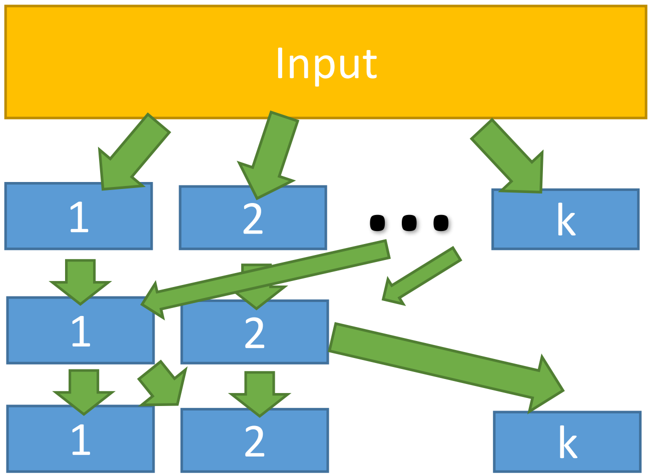




MEASURING COST

Multi-processor:

Taking time time t(n) with p(n) processors



Work/depth:

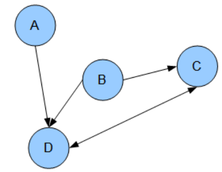
- Work: total operations performed
- Depth: max chain of dependencies

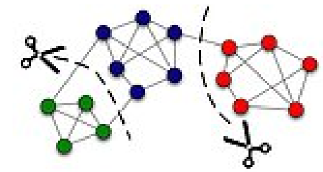
Work efficient: work should be close to sequential algorithms

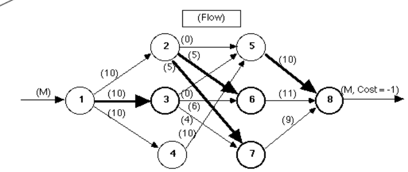
- $d(n) = t(n)$, $w(n) \leq t(n) \times p(n)$,
- Many other models for parallelism are inter-reducible with polylog(n) overhead

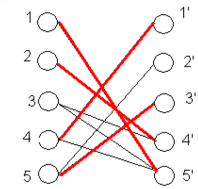
GRAPH PROBLEMS

- Connectivity / Reachability
- Shortest paths
- Optimization: flows, matchings



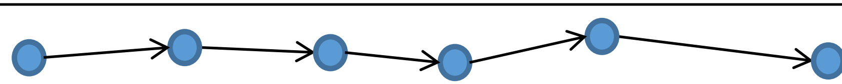






P-complete under polylog depth reductions

Open: on k processors, provably obtain factor O(k) speedups of directed s → t reachability over DFS



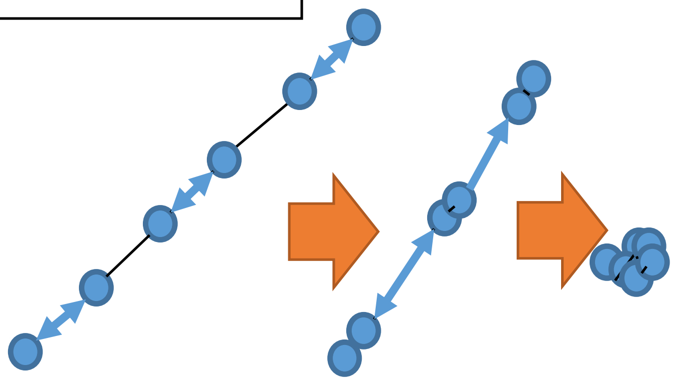
Formally: $O(m\log^{O(1)}n)$ work, $O(m\log^{O(1)}n/k)$ depth

UNDIRECTED REACHABILITY

In an undirected graph G, is s connected to t?

Repeatedly:

- Each vertex pick a neighbor
- Contract edges



# of vertices halves per round

O(logn) rounds

(omitting details on contract):

$O(\log^2 n)$ depth, O(mlogn) work

- More powerful operation: contract graph
- Communication no longer on original edge, closer to CLIQUE than CONGEST

SHORTEST PATH VIA. MATRICES

Graph, undirected or directed

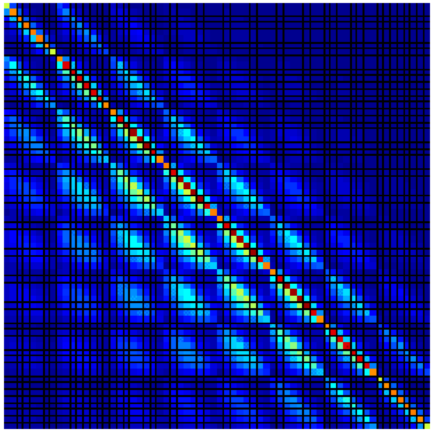
Find shortest path from s to t

Min-plus matrix multiplication:

$d(u, v) = \min_w d(u, w) + d(w, v)$

Depth: $\log^{O(1)} n$

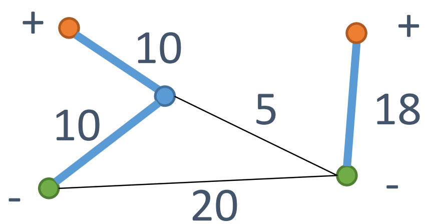
Work: $O(n^3 \log n)$



Open: better work-depth tradeoffs: e.g. $(1+ \varepsilon)$-approximation in $O(m\log^2 n \varepsilon^{-2})$ work, $O(n^{0.7})$ depth

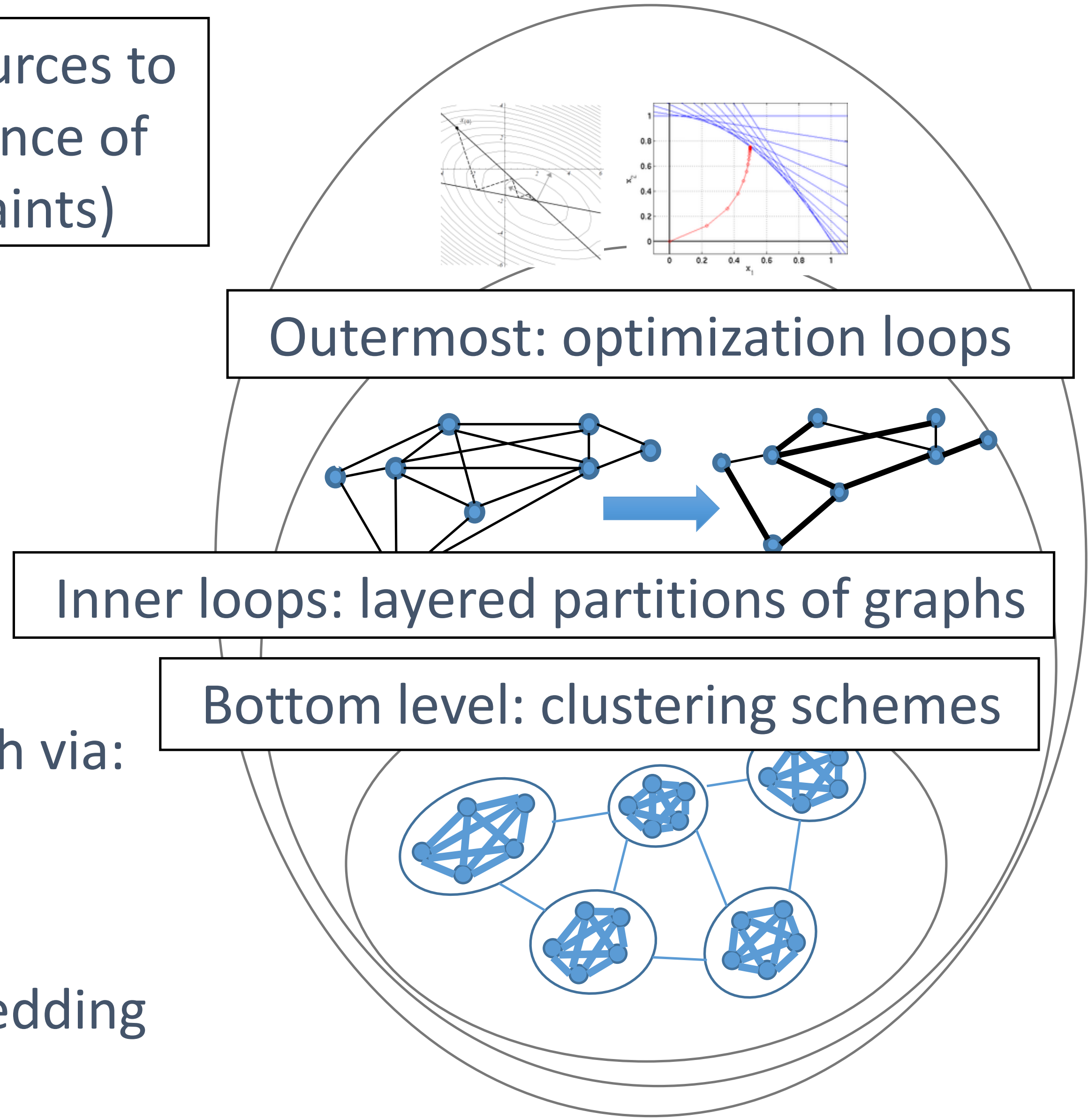
A COMPLETE CALL STACK

Transshipment: match sources to sinks, minimize total distance of paths (no capacity constraints)



[Sherman `16]
[Becker- Karrenbauer -Krinninger-Lenzen `16]:
$O(m^{1+a})$ work, $O(m^a)$ depth via:

- Gradient descent
- Divide-and-conquer
- Graph clustering/embedding

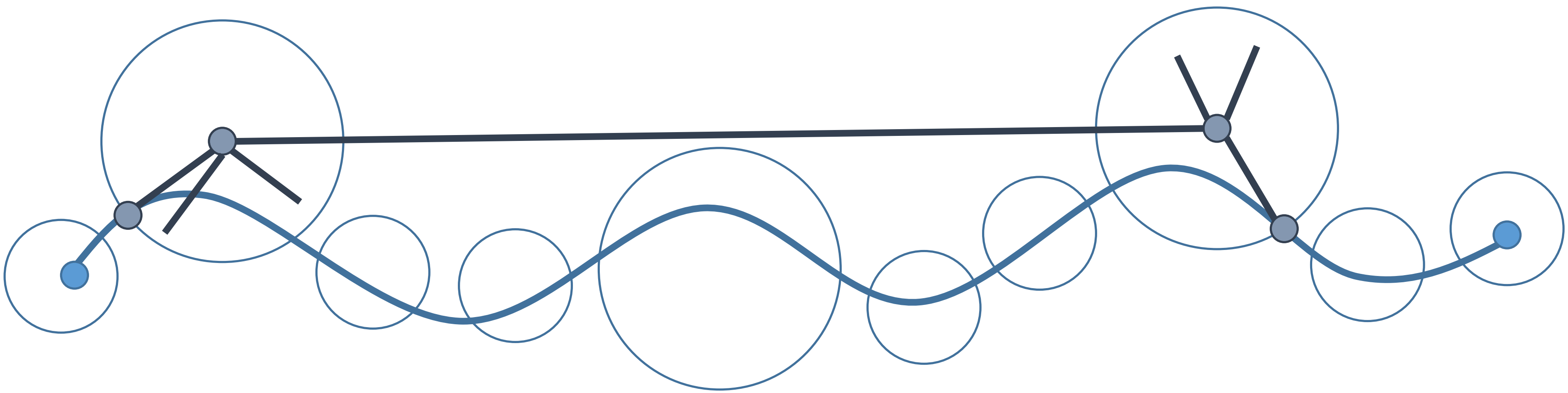


OUTLINE

- Model and problems
- **Graph decompositions**
- Randomized clusterings
- Interface with optimization

HOPSETS

[Klein-Subramanian`93][???]: scaling methods give exact h-hop distances in O(h) depth, O(mlogn) work

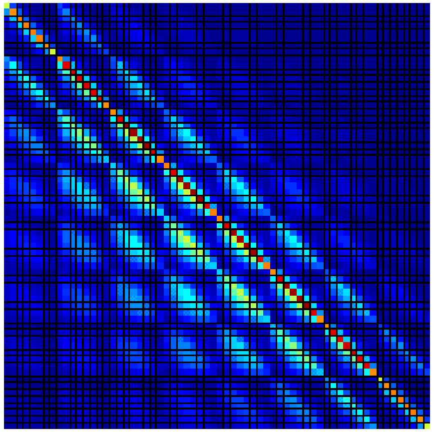


Hopset: add `short cut' edges so that shortest path lengths are approximated by ones with few hops

MATRIX SQUARING AS A HOPSET

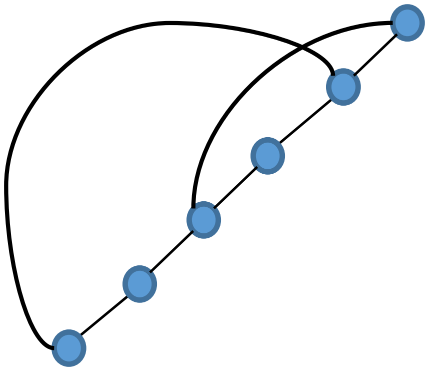
Min-plus matrix multiplication:

$d(u, v) = \min_w d(u, w) + d(w, v)$

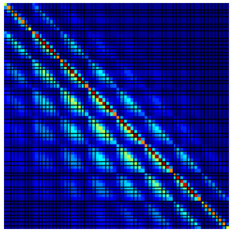


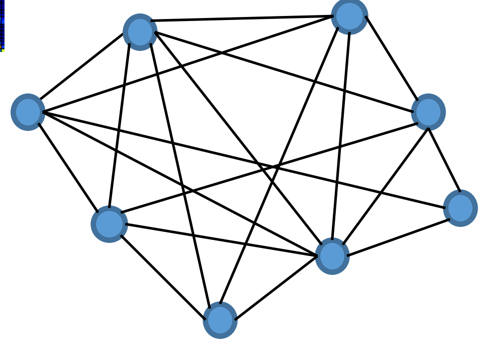
$\mathrm{Dist}^k(u, v)$

- Length of shortest k-hop u-v path
- Can view as a u-v shortcut

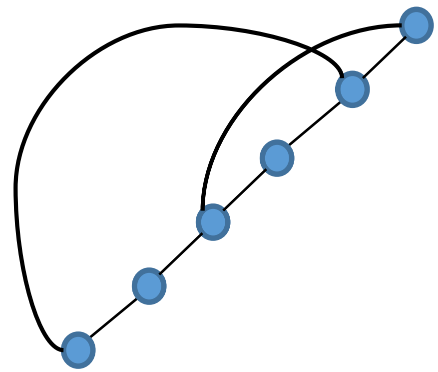


DIFFICULTIES IN FINDING GOOD HOPSETS





Highly connected, O(1)-hop graph is dense, expensive

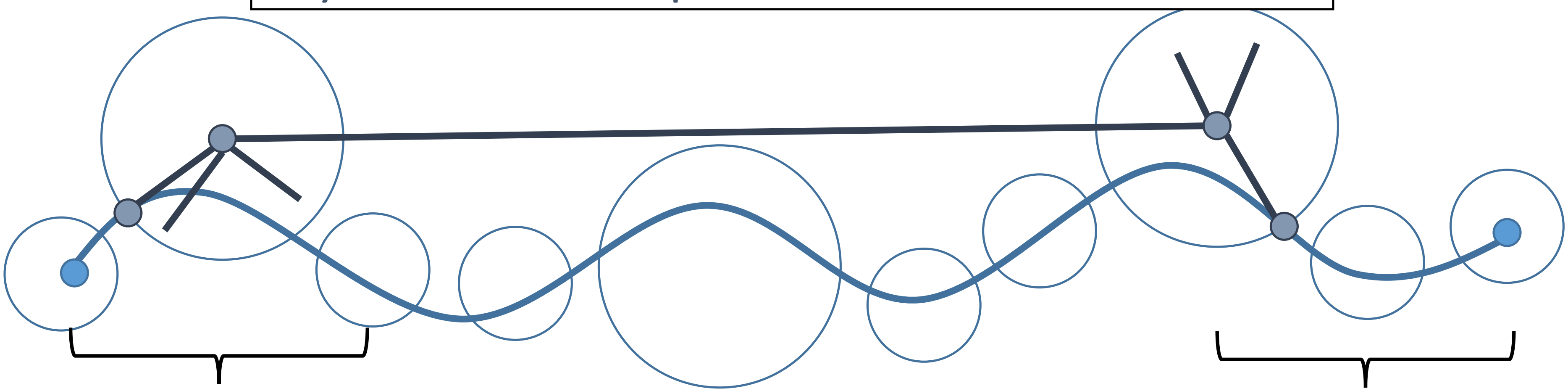


Long paths / tree, need many hop edges

Challenge: avoid paying O(n) steps, each taking O(n)

EXISTENCE OF GOOD HOPSETS

[Miller-Xu][folklore?] connect n random pairs by exact shortest path distances between them



$n^{1/2}$ vertices

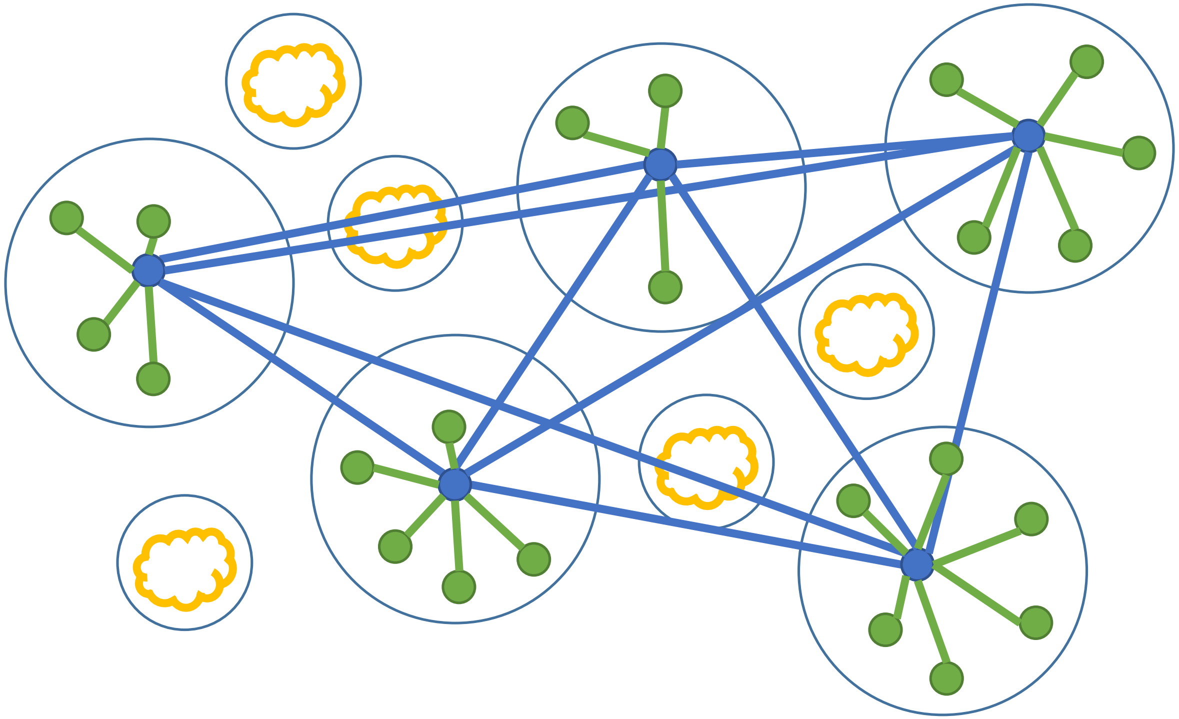
$n^{1/2} \times n^{1/2}$ pairs, one hop expected

$n^{1/2}$ vertices

Implies $O(m\log n)$ work, $O(n^{1/2}\log^2 n)$ depth shortest path algorithm

CONSTRUCTING HOPSETS

- ε-net like: partition into clusters, connect centers
- (In undirected case) recurse on smaller clusters



SOME PREVIOUS WORKS ON HOPSETS

Open:
- exact hopsets in nearly-linear work
- Polylog hopcount + size + work

Hop count H	Size	Work	Depth	Note
$\boxed{\tilde{O}(n^{1/2})}$	$\tilde{O}(n)$	$\boxed{\tilde{O}(mn^{1/2})}$	$\tilde{O}(H)$	[UY91, KS97] (directed)
$O(\text{poly}\log n)$	$\boxed{O(n^{1+\alpha})}$	$\tilde{O}(mn^{\alpha})$	$\tilde{O}(H)$	[Coh00]
$(\log n)^{O((\log\log n)^2)}$	$O(n^{1+O(\frac{1}{\log\log n})})$	$\tilde{O}(mn^{O(\frac{1}{\log\log n})})$	$\tilde{O}(H)$	[Coh00]
$\boxed{O(n^{\frac{4+\alpha}{4+2\alpha}})}$	$O(n)$	$O(m\log^{3+\alpha} n)$	$\tilde{O}(H)$	[MPVX`15]
O(1)	$O(n^{1+1/H})$	$\boxed{O(m^2)}$	polylog(n)	[EN `16]

OUTLINE

- Model and problems
- Graph decompositions
- **Randomized clusterings**
- Interface with optimization

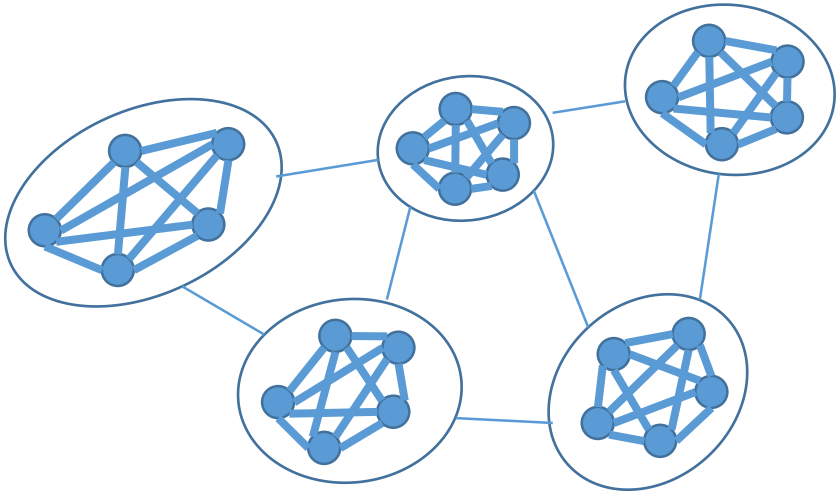
KEY TOOL IN DECOMPOSING GRPAHS

Low Diameter Decompositions

- Partition of V into clusters $S_1, S_2, \dots, S_k$ s.t.
- The diameter of each S_i is at most d.
- βm edges between clusters.

Typically parameters:

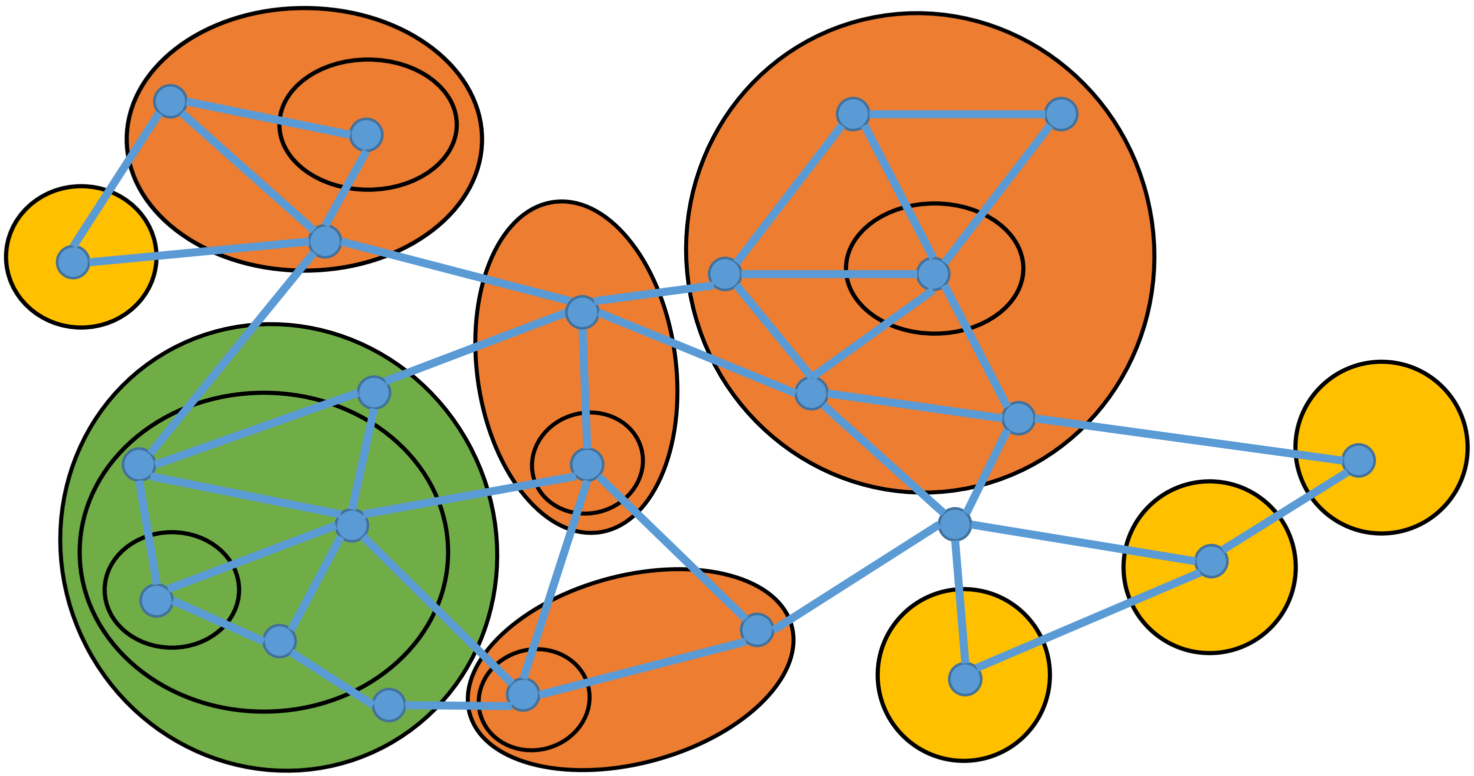
- $\beta = \log^{-O(1)} n$,
- $d = O(\log n / \beta)$



EXP START TIME CLUSTERING

Parallel variant of a clustering scheme in [Bartal `96]

- Each vertex u starts unit speed BFS at time -Exp(β)
- BFS stops at 'owned' v, owns any 'sleeping' v reached.



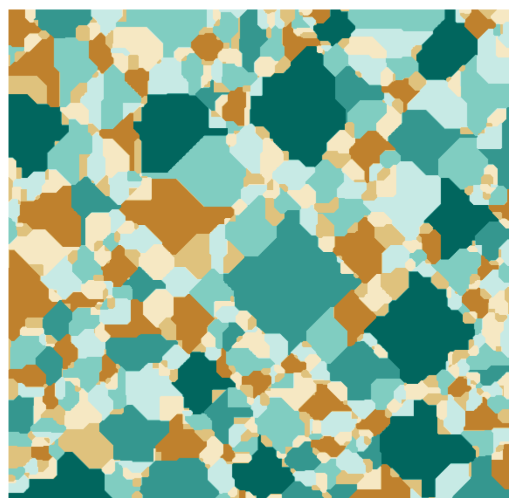
EXP START TIME CLUSTERING ON GRID

β=0.002

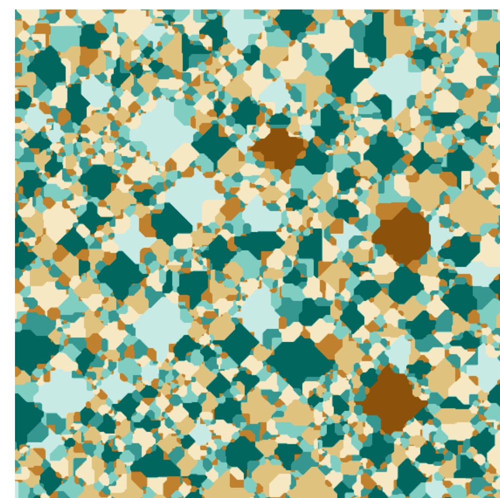
β=0.005

β=0.01

β=0.02

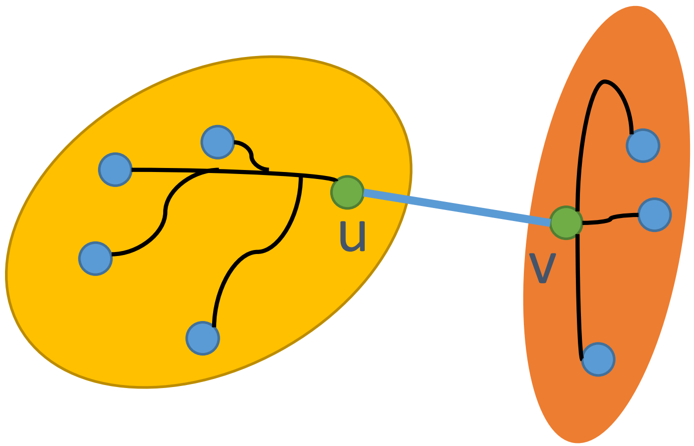


β=0.05



β=0.1

ANALYSIS ON UNDIRECTED GRAPHS

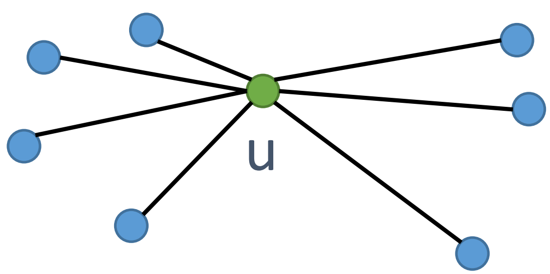


More global view:

- Each vertex picks δ_u = -Exp(β)
- v assigned to $\boxed{\text{argmin}_u \text{ dist}(u) + \delta_u}$

Diameter: w.h.p. $\min_u \delta_u \approx -O(\log n / \beta)$

e = uv 'cut' only if first two BFSs reach u within O(1)

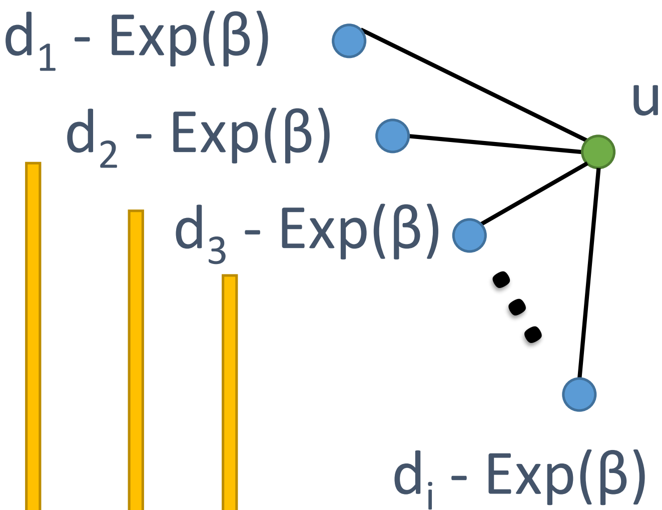


'Backward' analysis, view from u:

- Only dist(v, w) and δ_v affect the way things reach u
- Equivalent to star centered at u

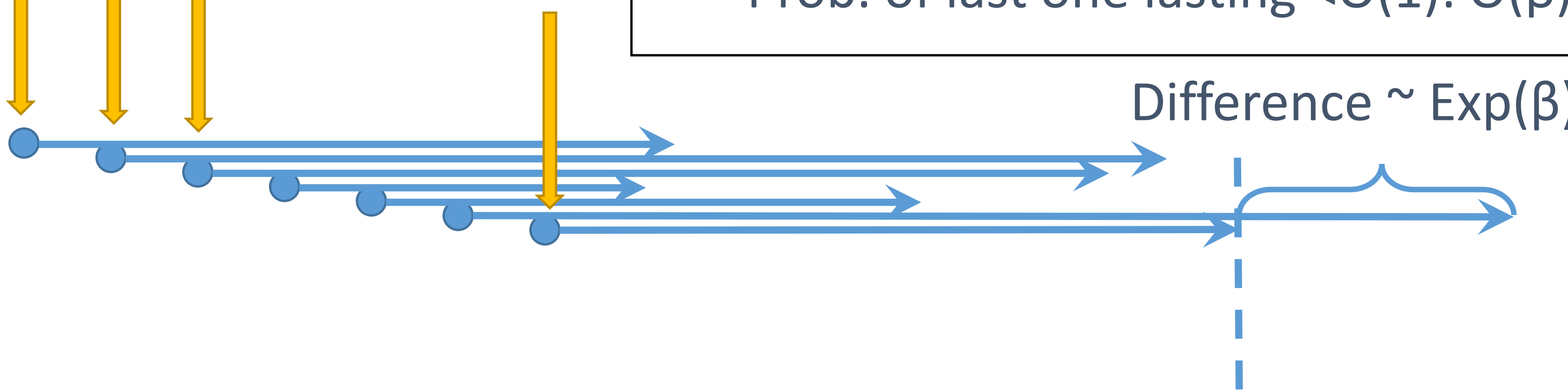
ANALYSIS: VIEW GRAPH FROM MIDPOINT

First two BFSs reach O(1) apart
⇔ max and 2nd max of k copies
of shifted Exp(β) are within O(1)



Exp(β) can be viewd as particle decay:

- Start from 2nd last particle decayed
- Prob. of last one lasting <O(1): O(β)



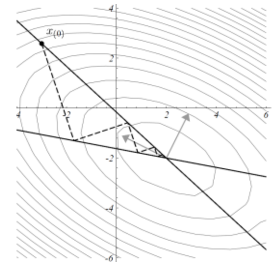
OUTLINE

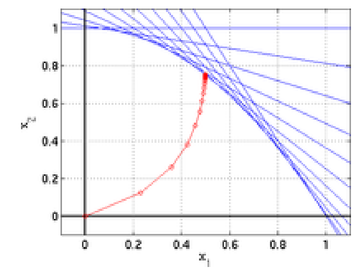
- Model and problems
- Graph decompositions
- Randomized clusterings
- **Interface with optimization**

ITERATIVE METHODS

Gradual convergence to solution

- Gradient descent
- Newton steps
- Mirror descent



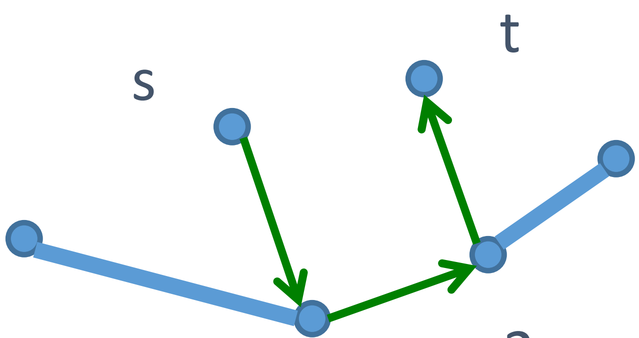


Preconditioning: solve problem in **A** by solving several problems in some **B** ≈ **A**, 'error removal'

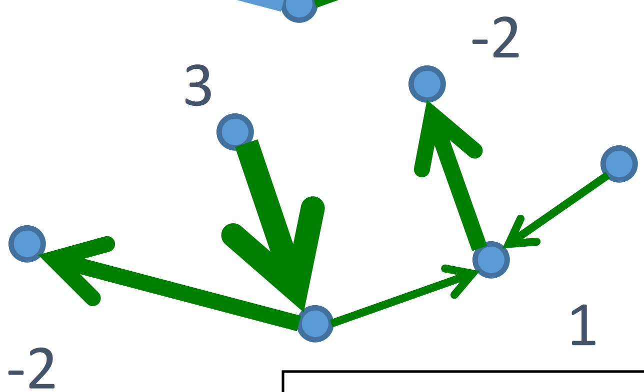
[Sherman `13] [Kelner-Lee-Orecchia-Sidford `14]: given operator that α-approximates maxflow for ANY demand **d**, can compute $(1 + \varepsilon)$-approx maxflow in $O(\alpha^2 \log n \varepsilon^{-2})$ calls

[Madry `10] [KLOS `13]: $\alpha = O(m^\theta)$ in $O(m^{1+\theta})$ time

ALGORITHM AS OPERATOR



Tree: unique s-t path, maxflow = demand / bottleneck edge

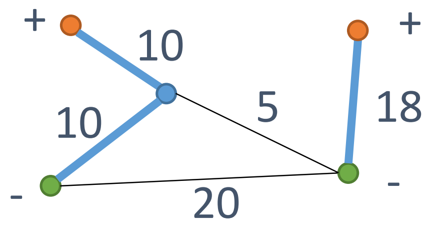


Multiple (exact) demands: flows along each edge determined via linear mapping, O(n) time

[Racke `01][Racke-Shah-Taubig `14]: ANY undirected graph has a tree that's an $O(\log^c n)$ approximator

RECENT: TRANSSHIPMENT PROBLEM

Generalization of shortest paths: match sources to sinks, minimize total distance of paths (no capacity constraints)



Open:

- $O(m\log^{O(1)}n)$ work in $O(\log^{O(1)}n)$ depth?
- Implications for shortest paths?

\`precondition' using L_1 embeddings:

- [Sherman \`16]: $O(m^{1+a})$ work, $O(m^a)$ depth
- [Becker-Karrenbauer-Krinninger-Lenzen \`16]: $O(\varepsilon^{-1}\text{polylog}(n))$ distrbibuted rounds

THE OTHER END: THE LAPLACIAN PARADIGM

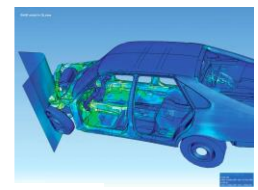
Open: distributed Laplacian solvers?

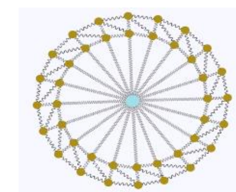
Evidence in favor: [Ghaffari-Karrenbauer-Kuhn-Lenzen-Patt-Shamir`15] distributed undirected maxflow

$Lx=b$

Directly related:

Elliptic systems

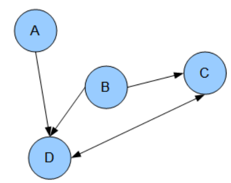


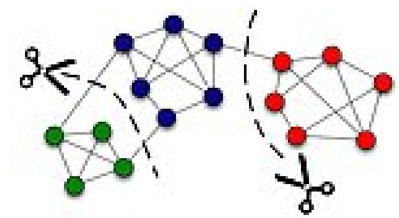


Few iterations:

Eigenvectors,

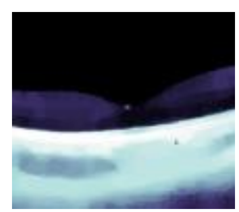
Heat kernels

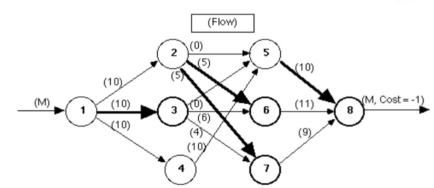


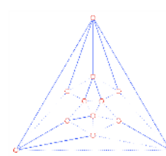


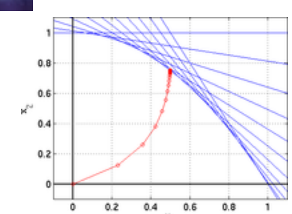
Many iterations / modify algorithm

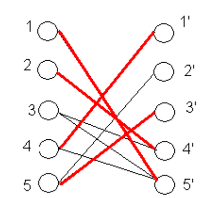
Graph problems, Image processing











THE OTHER END: SPARSIFIED SQUARING

$\mathbf{I} - \mathbf{A}_1 \approx_\varepsilon \mathbf{I} - \mathbf{A}^2$

$\mathbf{I} - \mathbf{A}_2 \approx_\varepsilon \mathbf{I} - \mathbf{A}_1{}^2$

…

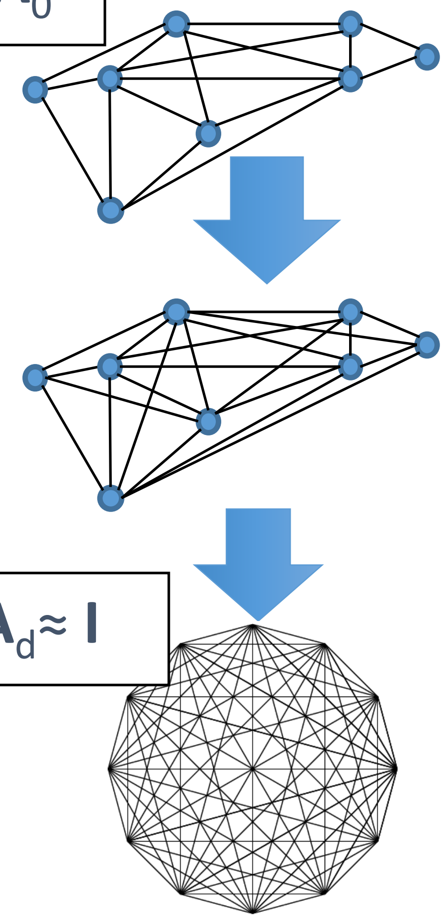
$\mathbf{I} - \mathbf{A}_i \approx_\varepsilon \mathbf{I} - \mathbf{A}_{i-1}{}^2$

$\mathbf{I} - \mathbf{A}_d \approx \mathbf{I}$

Open: more graph algorithms via sparsified squaring?

≈ : approximations of graphs

$\mathbf{I} - \mathbf{A}_0$



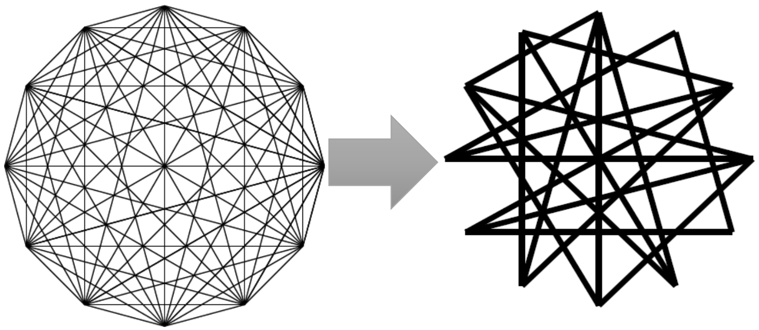
$\mathbf{I} - \mathbf{A}_d \approx \mathbf{I}$

Algorithms involving repeated squaring

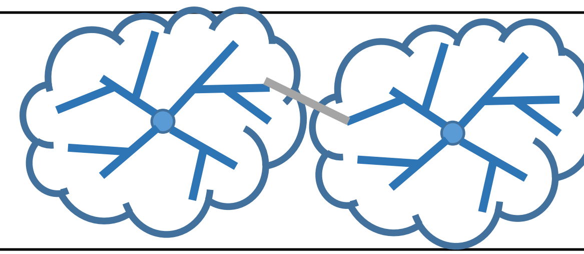
- NC algorithm for shortest path
- [Reingold `05] Logspace connectivity
- Multiscale methods
- [P-Spielman `14] Solving **Lx** = **b**

MAKING SQUARING FAST: SPARSIFICATION

Approximate a dense graph by a sparse one



[Koutis`14]: build and remove O(logn) spanners, repeat with random half of what's left



- [Abraham-Durfee-Koutis-Krinninger-P`16]: can make dynamic
- [Miller-P-Vladu-Xu `15]: spanners via exp. start time clustering

Question: dynamic exponential time clustering and applications?

QUESTIONS

- Connections between PRAM and other models?
- Speeding up directd s→t reachability
- Tight(er) bounds on undirected hopsets?
- Translating PRAM algorithms to data structures?
- Faster transshipment?
- Distributed Laplacian solver / sparsified squaring?
- Numerical approach to more graph problems?