

Hypergraph Partitioning for Computing Matrix Powers

Nicholas Knight

Erin Carson, James Demmel

UC Berkeley, Parallel Computing Lab

CSC 11

Hypergraph Partitioning for Computing Matrix Powers

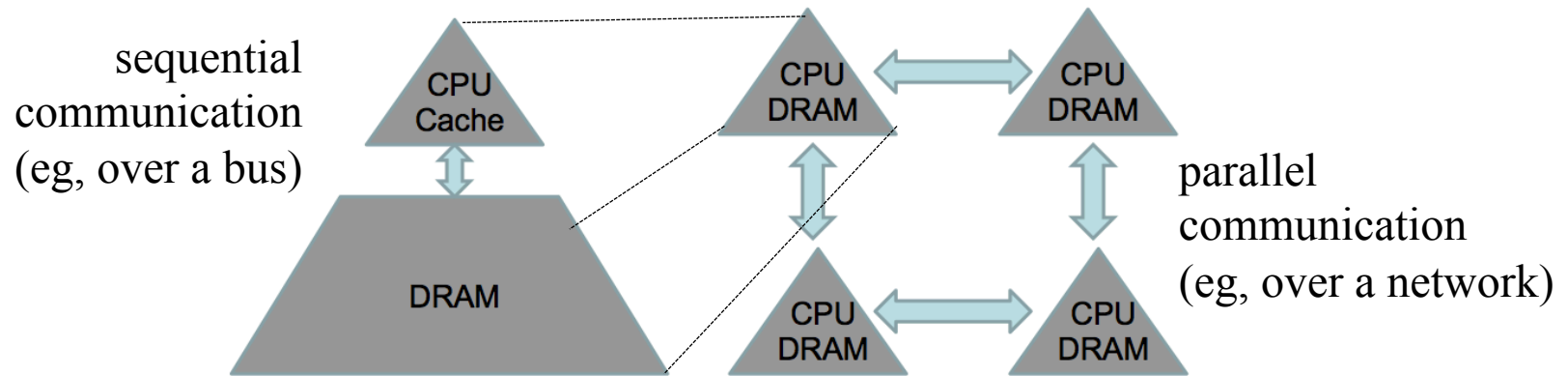
- Motivation
 - Avoiding communication
 - Communication-avoiding Krylov subspace methods
 - The matrix powers kernel
- Hypergraph model for the matrix powers kernel
 - Partitioning (to reduce communication)
- Heuristics involving structure prediction
 - Cohen's reachability estimation algorithm
 - Hypergraph sparsification
- Ongoing work
 - Load-balancing

Hypergraph Partitioning for Computing Matrix Powers

- Motivation
 - Avoiding communication
 - Communication-avoiding Krylov subspace methods
 - The matrix powers kernel
- Hypergraph model for the matrix powers kernel
 - Partitioning (to reduce communication)
- Heuristics involving structure prediction
 - Cohen's reachability estimation algorithm
 - Hypergraph sparsification
- Ongoing work
 - Load-balancing

Communication is Costly, Computation is Cheap

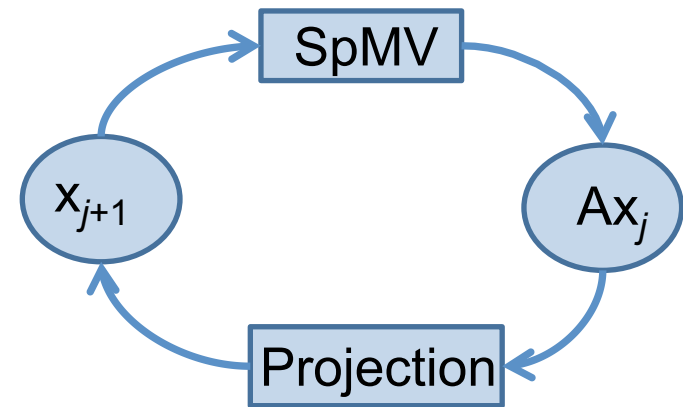
Algorithm runtime = communication + computation



- The **gap** between processing power and communication cost is **increasing exponentially**:
$$\text{Time-per-flop} \ll 1/\text{bandwidth} \ll \text{latency}$$
- Applications must reduce communication to improve efficiency.

Communication-Avoiding Krylov Subspace Methods

- Krylov subspace methods:
 - Large class of iterative methods for solving linear systems, eigenvalue problems, etc.
 - Sequence of projections onto Krylov subspaces: $\text{span}(x, Ax, A^2x, \dots, A^s x)$
- **Problem:** Krylov subspace methods are communication-bound
- **Solution:** Communication-avoiding reformulations
 - Take $s > 1$ iterations with communication cost of 1 iteration:
 - Reduce parallel latency cost by $O(s)$
 - Reduce sequential latency and bandwidth costs by $O(s)$
 - At the cost of extra work and storage

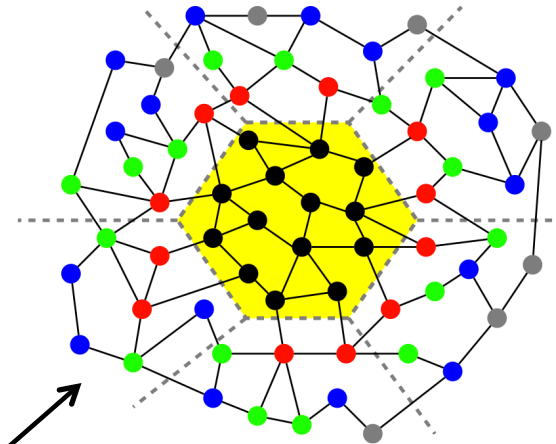


Matrix Powers Kernel:

$$\text{Asx}(A, s, x) = [x, Ax, A^2x, \dots, A^s x]$$

Avoids communication in repeated SpMV:

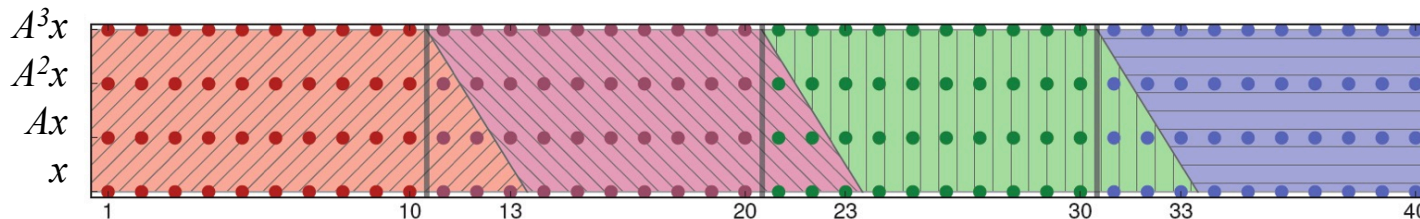
- In serial, by exploiting temporal locality:
 - Reading A
 - Reading $V := [x, Ax, A^2x, \dots, A^s x]$
- In parallel, by doing only 1 ‘expand’ phase (instead of s phases).



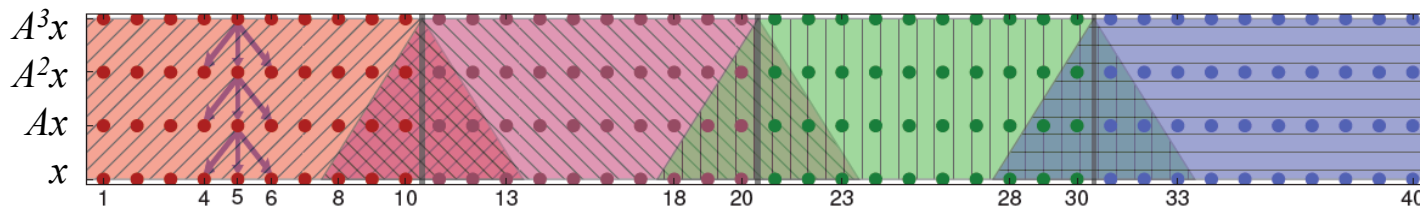
Also works for
general graphs!

black = local elements
red = 1-level dependencies
green = 2-level dependencies
blue = 3-level dependencies

- Example: 40×40 tridiagonal matrix, $s = 3$:



Sequential



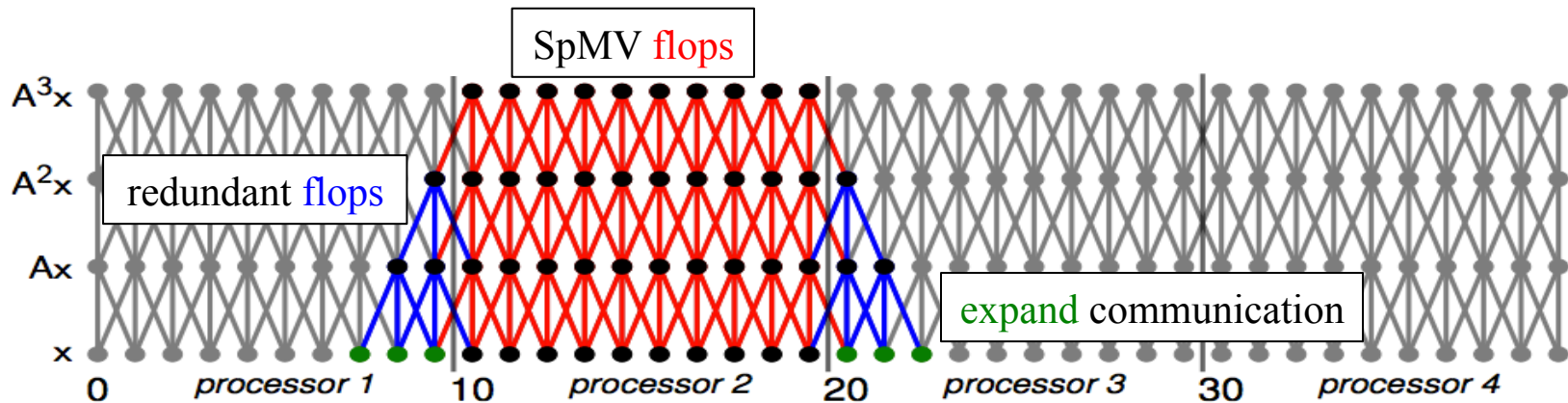
Parallel

Hypergraph Partitioning for Computing Matrix Powers

- Motivation
 - Avoiding communication
 - Communication-avoiding Krylov subspace methods
 - The matrix powers kernel
- Hypergraph model for the matrix powers kernel
 - Partitioning (to reduce communication)
- Heuristics involving structure prediction
 - Cohen's reachability estimation algorithm
 - Hypergraph sparsification
- Ongoing work
 - Load-balancing

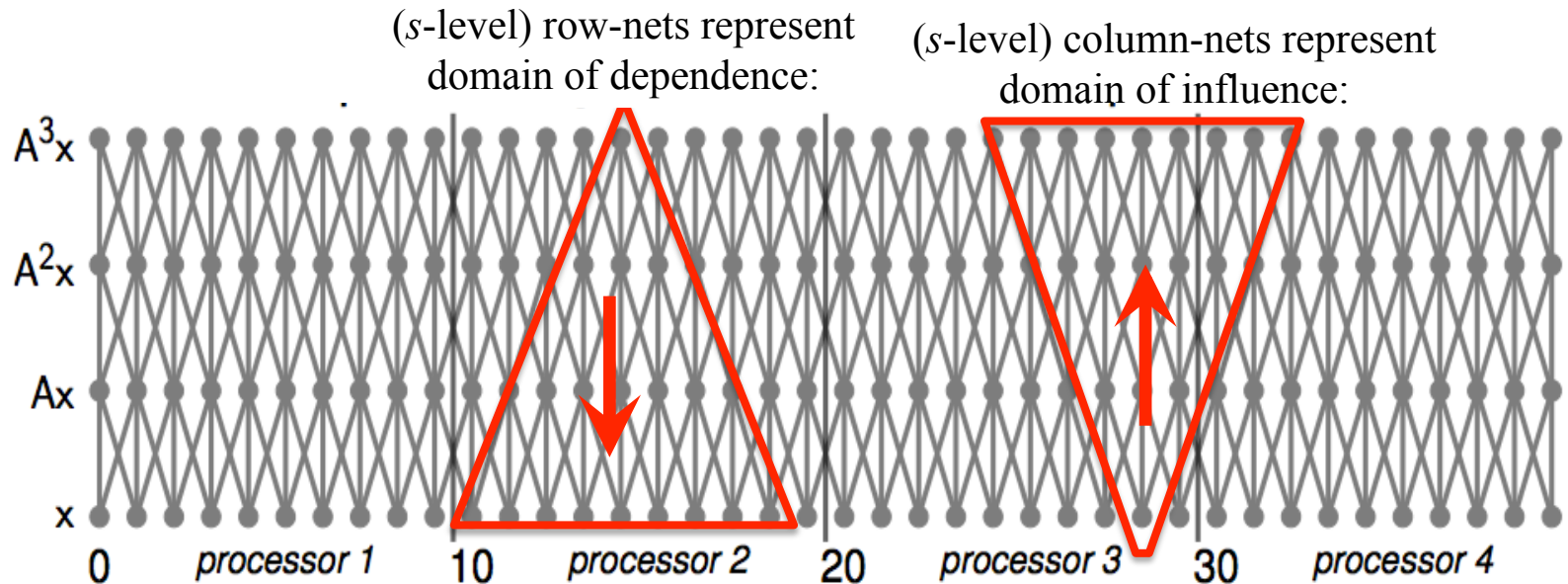
Parallel Partitioning for the Matrix Powers Kernel

- ‘Expand’ communication upfront, no ‘fold’ communication.



- Previous implementation used **graph partitioning** (with $A+A^T$):
 - (a) Doesn't accurately count communication [Catalyurek, 99]
 - (b) Poor approximation for unsymmetric matrices
 - (c) Doesn't account for structure of A^s
- **Hypergraph partitioning** remedies (a,b) for SpMV. How to model (c)?

Hypergraph Partitioning for Matrix Powers



Parallel communication for $A^s x(A, s, x) = [x, Ax, A^2x, \dots, A^s x]$, given overlapping partition of A

=

Parallel communication for $y = A^s x$, given 1D rowwise layout of A^s

(assuming no cancellation and nonzero diagonal)

- Minimizing communication in matrix powers reduces to hypergraph partitioning s -level column-nets.
- **Problem:** Computational and storage cost:
 - $s \times$ Boolean sparse matrix-matrix multiplies!

Do we really need s -level column nets?

- Our experiments showed:
 - For matrices with regular structure (like stencils), no significant benefits vs. 1-level column-nets
 - For many irregular problems, communication volume reduced by up to 20% vs. 1-level nets.
 - Probably not enough to justify the cost of computing, storing, and partitioning s -level nets
 - Idea: sparsify the hypergraph using reachability estimation

Hypergraph Partitioning for Computing Matrix Powers

- Motivation
 - Avoiding communication
 - Communication-avoiding Krylov subspace methods
 - The matrix powers kernel
- Hypergraph model for the matrix powers kernel
 - Partitioning (to reduce communication)
- Heuristics involving structure prediction
 - Cohen's reachability estimation algorithm
 - Hypergraph sparsification
- Ongoing work
 - Load-balancing

Estimating Column-Net Sizes

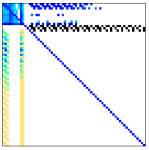
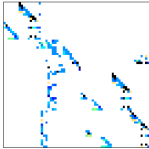
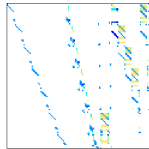
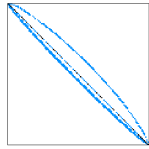
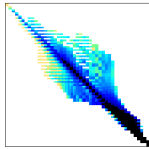
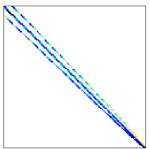
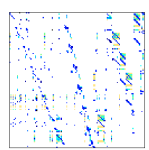
- Reachability estimation [Cohen '94]
 - $O(\text{nnz})$ time randomized algorithm for estimating size of transitive closure.
 - Can be used to estimate nnz-per-column in matrix product A^s in $O(s \cdot \text{nnz})$ time.

Idea: Dynamically Sparsifying the Hypergraph

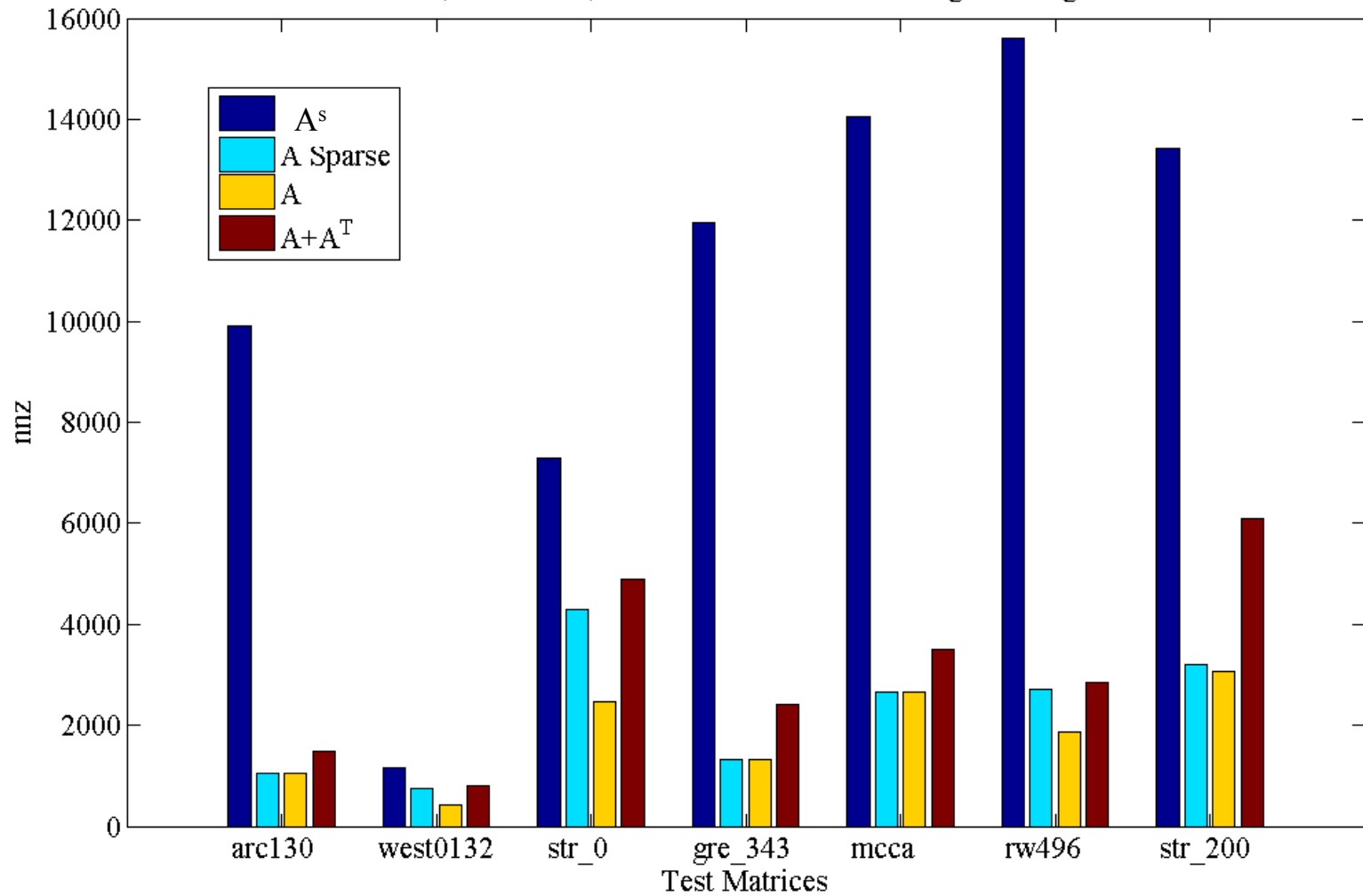
- Run reachability estimation for each column-net, with given tolerance tol :
 - Drop nets whose sizes exceed $n \cdot tol$, use 1-level nets instead.
 - $tol = 0 \rightarrow$ column-nets of A ,
 - $tol = 1 \rightarrow$ column-nets of A^s

Preliminary Experiments

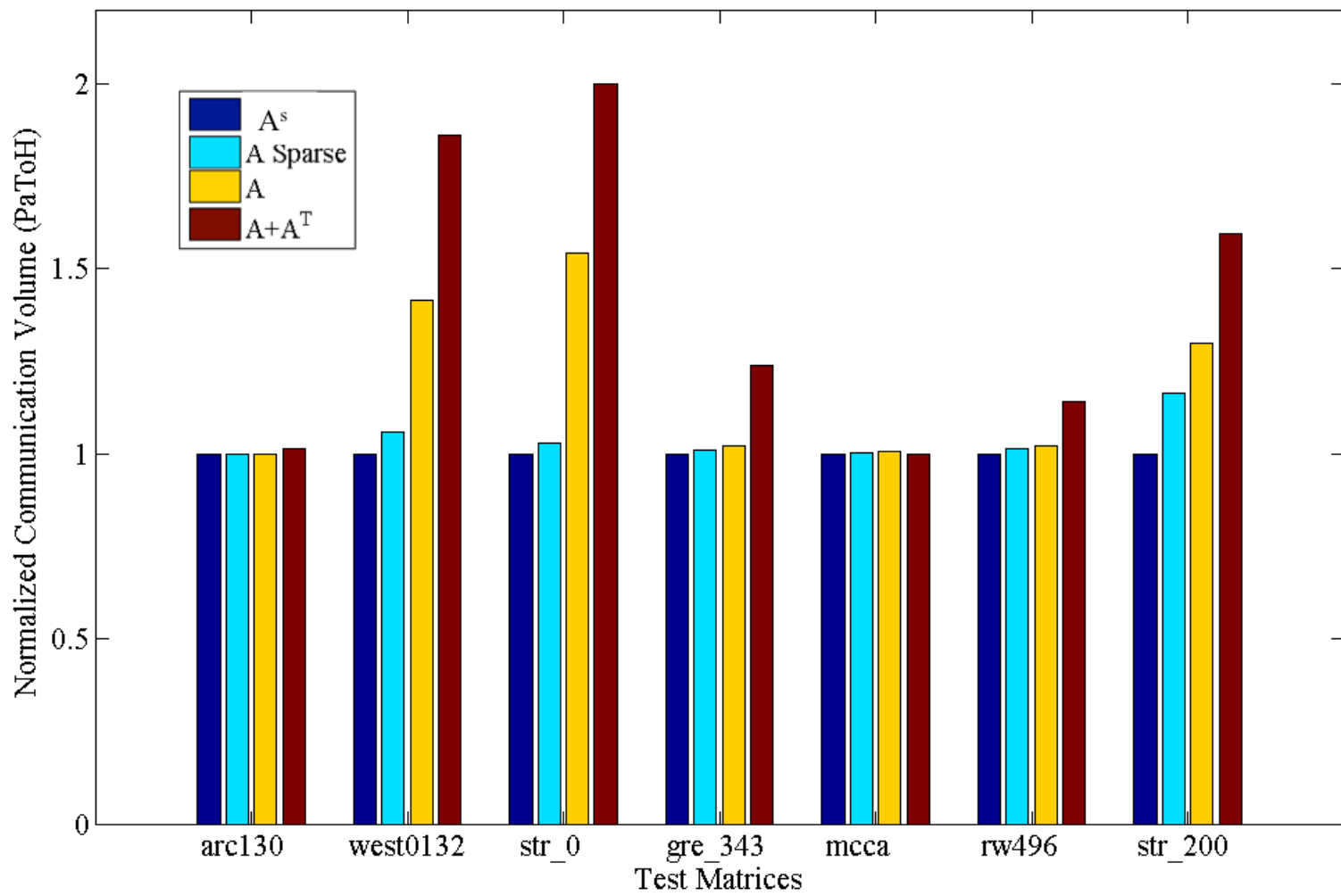
- Set of small test matrices from UFSSMC [Davis '94]
- $tol = 0.5$ (half-dense), 4 parts, $s \in \{2, 3, 4\}$ depending on fill in A^s
- Comparison of hypergraph size and communication volume for four strategies:
 - s -level column nets
 - Sparsified column nets (somewhere between s - and 1-level)
 - 1-level column nets
 - Graph partitioning ($A+A^T$)
- Software: PaToH [Catalyurek, Aykanat, '99] and Metis [Karypis, Kumar '98]

Matrix	Application	n	nnz	Spy plot
arc130	Materials Science	130	1037	
west0132	Chemical Engineering	132	413	
str_0	LP	363	2454	
gre_343	Directed graph	343	1032	
mcca	Astrophysics	180	2659	
rw496	Markov Chain Model	496	1859	
str_200	LP	363	3068	

nnz (workload) for Various Partitioning Strategies



Normalized Communication Volume for Various Partitioning Strategies



Results and Observations

- Sparsified nets lead to comparable partition quality for **significantly** reduced hypergraph size
- Tuning parameter *tol* gives flexibility to trade off:
 - Quality of partition
 - Computation and storage costs

Hypergraph Partitioning for Computing Matrix Powers

- Motivation
 - Avoiding communication
 - Communication-avoiding Krylov subspace methods
 - The matrix powers kernel
- Hypergraph model for the matrix powers kernel
 - Partitioning (to reduce communication)
- Heuristics involving structure prediction
 - Cohen's reachability estimation algorithm
 - Hypergraph sparsification
- Ongoing work
 - Load-balancing

Load Balancing

- Matrix powers kernel workload depends on redundant computation
 - Overlapping ghost zones induced by partition (chicken-and-egg problem)
- Strategy (ongoing work):
 1. Initially partition using 1-level column-nets
 2. Refine partition with iterative swapping, taking net overlap into account
 - Swapping metric: estimate overlap using Cohen's algorithm.
- Related work: [Pinar, Hendrickson '01], used graph model.

Further Ongoing Work

1. Another use of reachability estimation:
 - Estimate maximum s such that A^s can be well-partitioned. Can't expect user to know this s in advance.
2. Extensions to some low-diameter graphs:
 - We can use blocking covers [Leiserson, Rao, Toledo, 97] to handle a few dense rows in A .
 - We have generalized this approach to $A + UV^T$.
3. High-performance implementation
 - Parallel partitioners, larger graphs

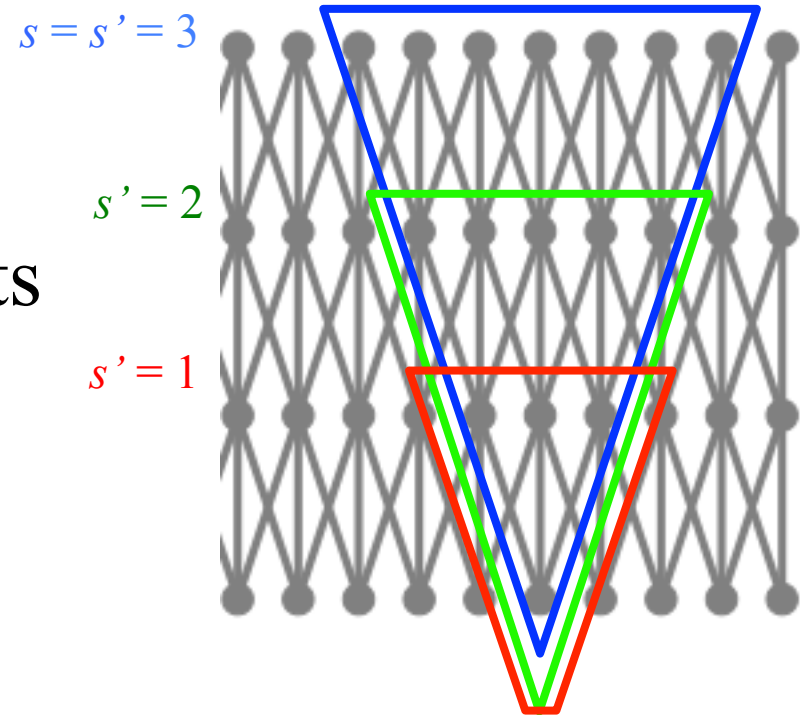
(More) References

- R. H. Bisseling and W. Meesen. Communication balancing in parallel sparse matrix-vector multiplication. *Electronic Transactions on Numerical Analysis*, 21:47–65, 2005.
- E. G. Boman. A nested dissection approach to sparse matrix partitioning. In *Proc. Applied Math. and Mechanics*, volume 7, 2007. Presented at ICIAM07, Zurich, Switzerland, July 2007.
- U. Catalyurek and C. Aykanat. Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. *IEEE Trans. Parallel Dist. Systems*, 10(7):673–693, 1999
- B. Hendrickson and T. G. Kolda. Partitioning rectangular and structurally nonsymmetric sparse matrices for parallel computation. *SIAM Journal on Scientific Computing*, 21(6):2048–2072, 2000
- G. Karypis and V. Kumar. METIS 4.0: Unstructured graph partitioning and sparse matrix ordering system. technical report, Dept. Computer Science, University of Minnesota, 1998. <http://www.cs.umn.edu/~metis>
- B. Uçar and C. Aykanat. Partitioning Sparse Matrices for Parallel Preconditioned Iterative Methods. *SIAM J. Sci. Comp.* 29:4, 2007.
- B.W. Kernighan and S. Lin, An Efficient Heuristic Procedure for Partitioning Graphs, *The Bell System Technical J.*, vol. 49, pp. 291-307, Feb. 1970
- C.M. Fiduccia and R.M. Mattheyses, A Linear-Time Heuristic for Improving Network Partitions, *Proc. 19th ACM/IEEE Design Automation Conf.*, pp. 175-181, 1982
- E. Cohen. Structure prediction and computation of sparse matrix products. *J. Combinatorial Optimization*, 2:307-332, 1999.
- C.E. Leiserson, S. Rao, and S. Toledo, “Efficient Out-of-Core Algorithms for Linear Relaxation Using Blocking Covers,” *J. Computer and System Sciences*, vol. 54, no. 2, pp. 332–344, 1997.
- T. A. Davis. The University of Florida Sparse Matrix Collection, 1994. Matrices found at <http://www.cise.ufl.edu/research/sparse/matrices/>
- J. Demmel, M. Hoemmen, M. Mohiyiddin, and K. Yelick. Avoiding communication in computing Krylov subspaces. Technical Report No. UCB/EECS 2007-123, 2007.

Thank you

Experiment 1: Communication Volume for Approximated s -level Column Nets

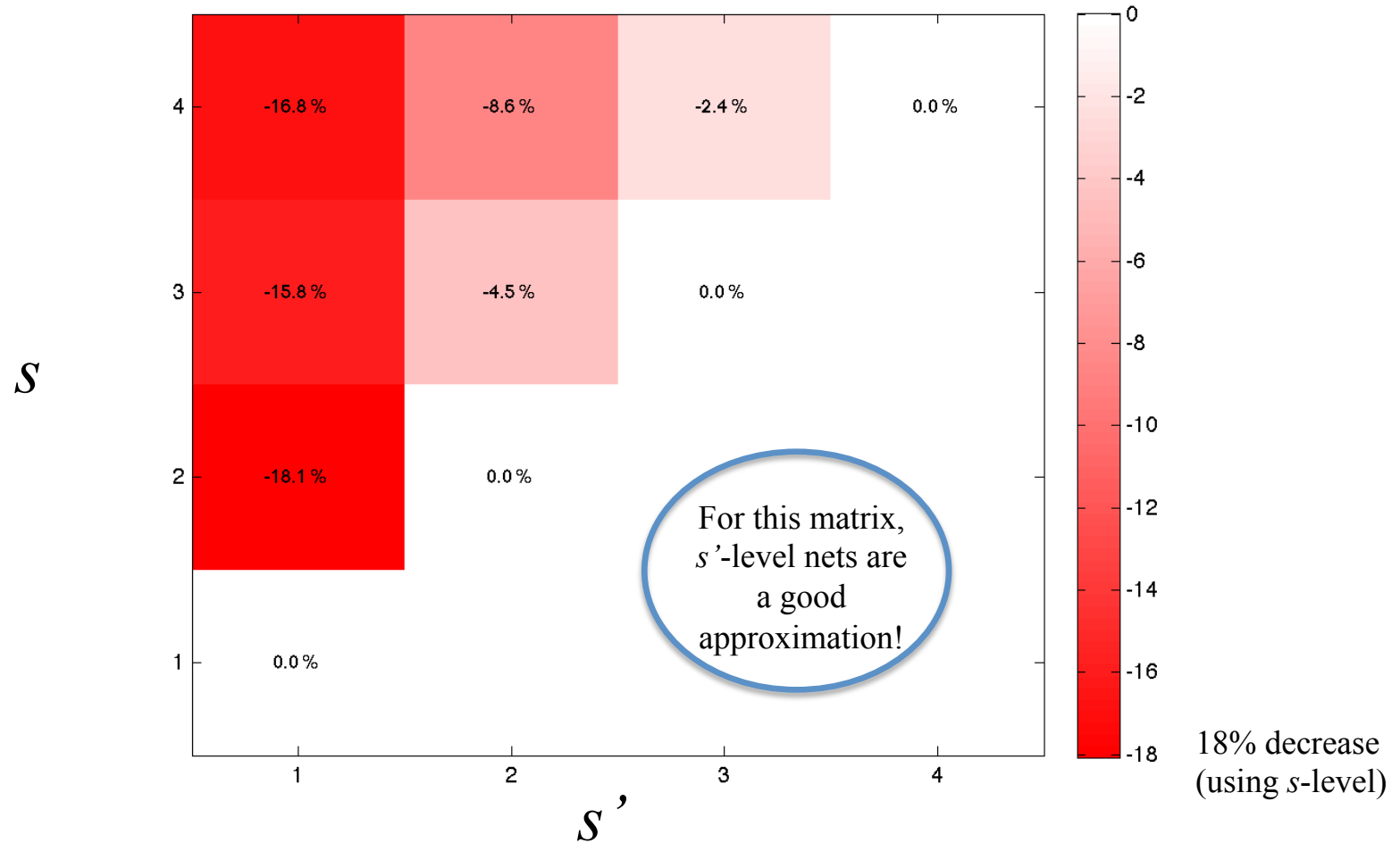
- How good do s' -level nets approximate s -level nets, for $1 \leq s' < s$?
- 8 parts



red = 1-level column-net
green = 2-level column-net
blue = 3-level column-net

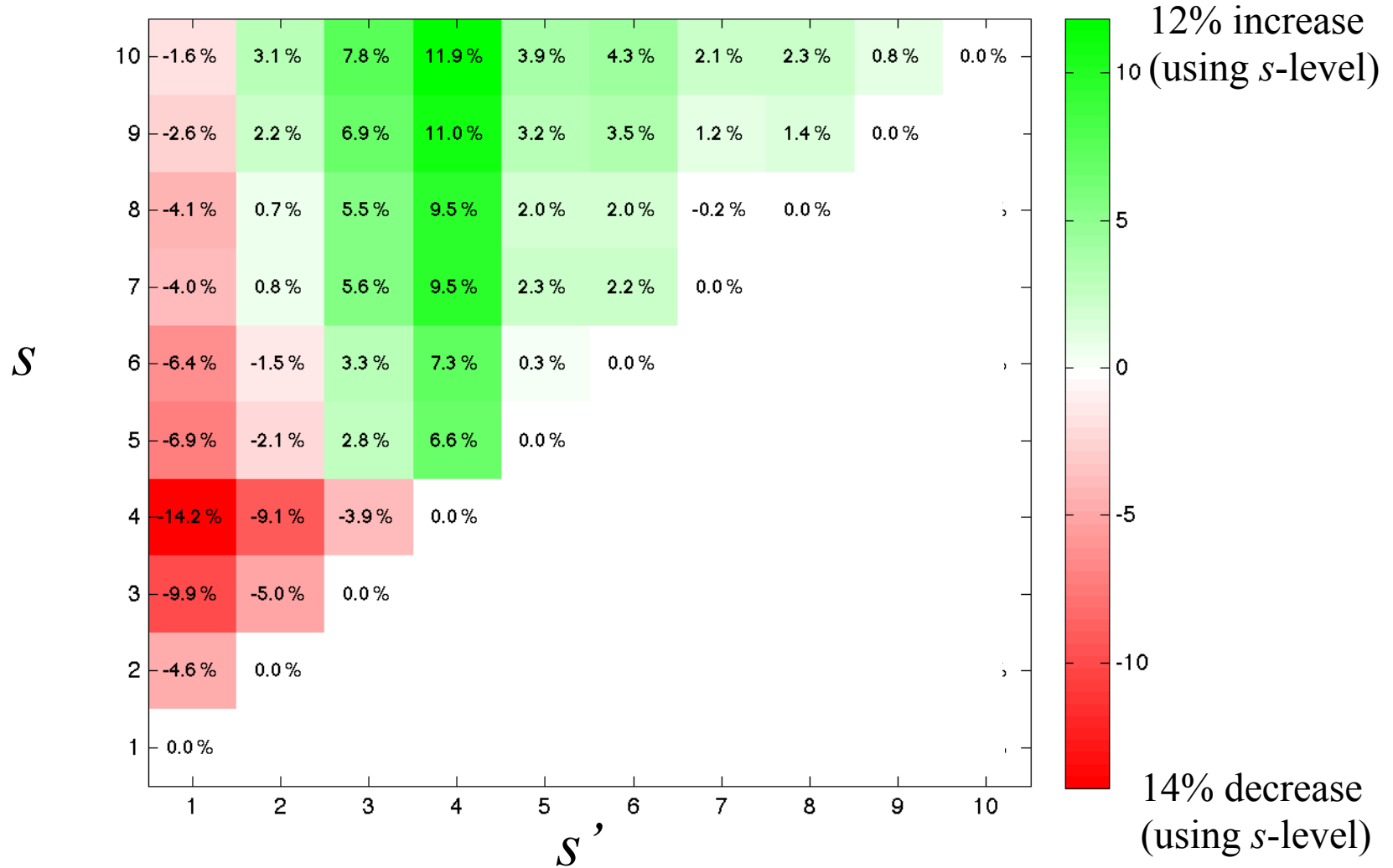
Example: Reduction in Total Communication Volume Using s' -level nets to Approximate Communication in $A^{s \times}$

1hr71c: $n \sim 70\text{K}$, $\text{nnz} \sim 1.5\text{M}$ (chem. process sim.)



Example: Reduction in Total Communication Volume Using s' -level nets to Approximate Communication in $A^{s \times}$

shyy161: $n \sim 75\text{K}$, $\text{nnz} \sim 330\text{K}$ (CFD)

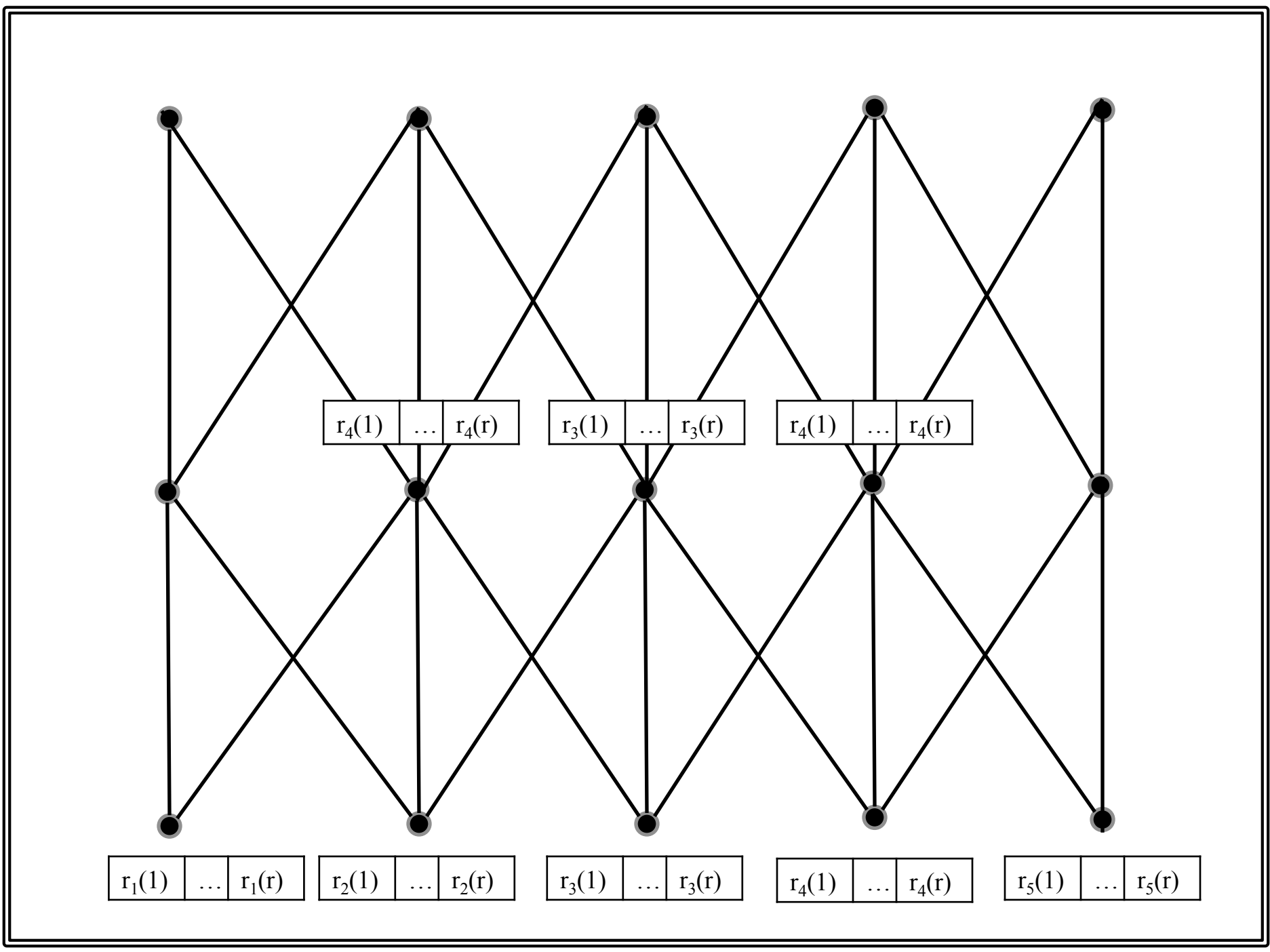


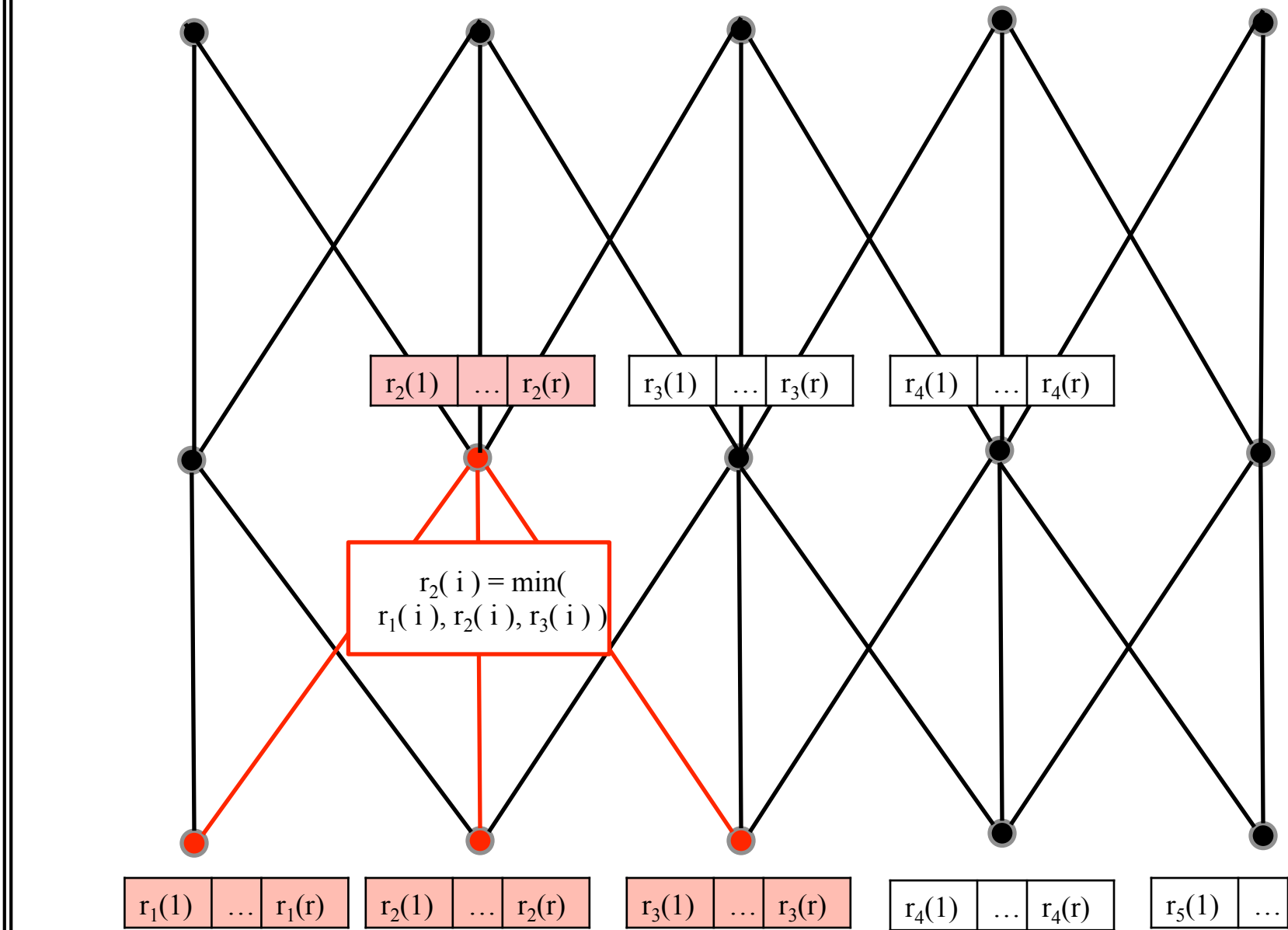
Algorithm Overview

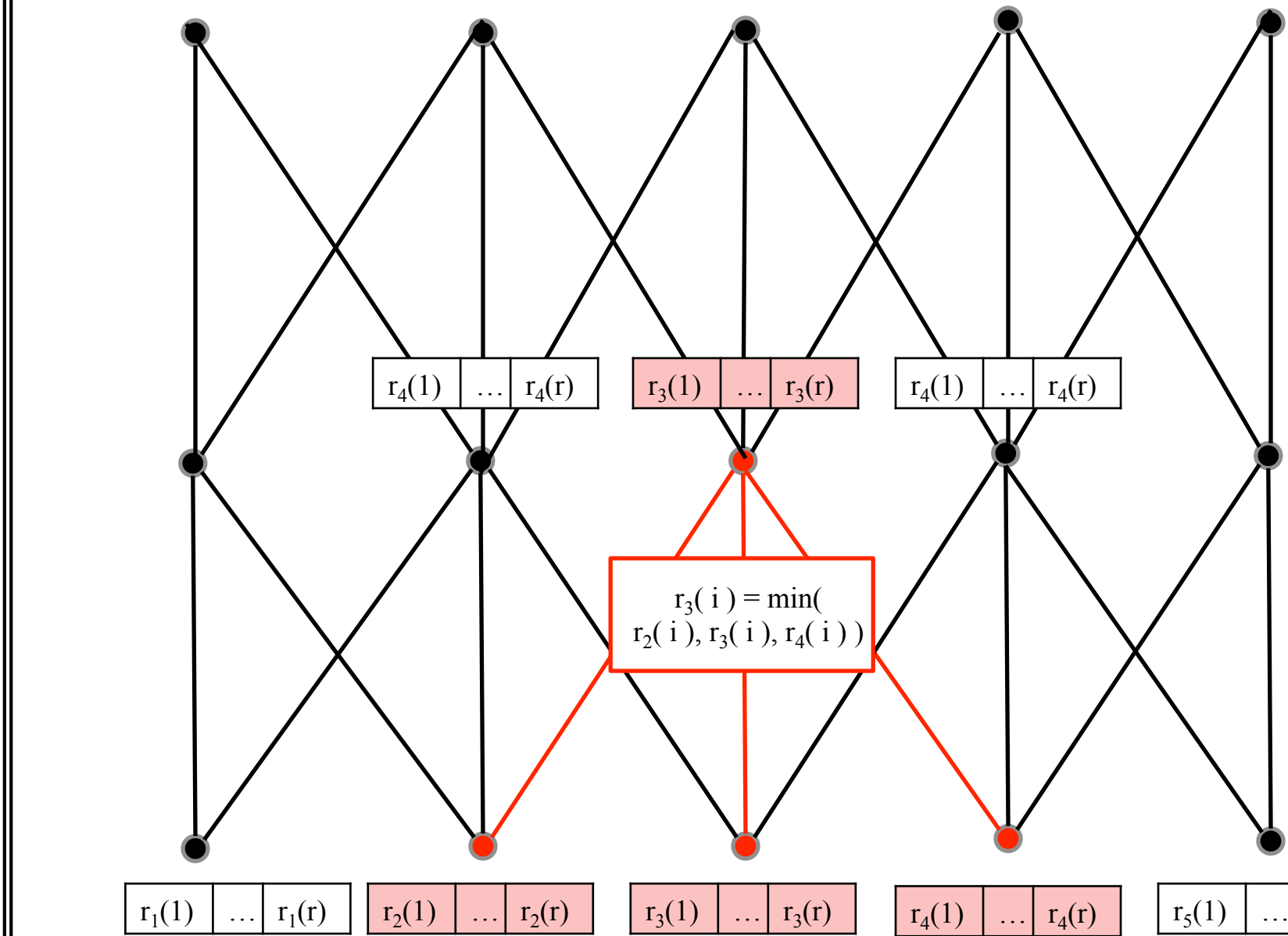
- Initially assign r -vector of rankings (a_1, \dots, a_r) , (sampled from exponential R.V., $\lambda = 1$) to each vertex v
- In each iteration (up to s), for each vertex v , take the coordinate-wise minima of the r -vectors reachable from v (denoted $S(v)$, non-zeros in column of A corresponding to v)
- Apply estimator:
$$\frac{r-1}{\sum_{i=1}^r a_i}$$
- Intuition: lowest-ranked node in $S(v)$ is highly correlated with $|S(v)|$
 - Example: If $S(v)$ contains half the nodes, we expect the lowest rank of nodes in $S(v)$ is very small.

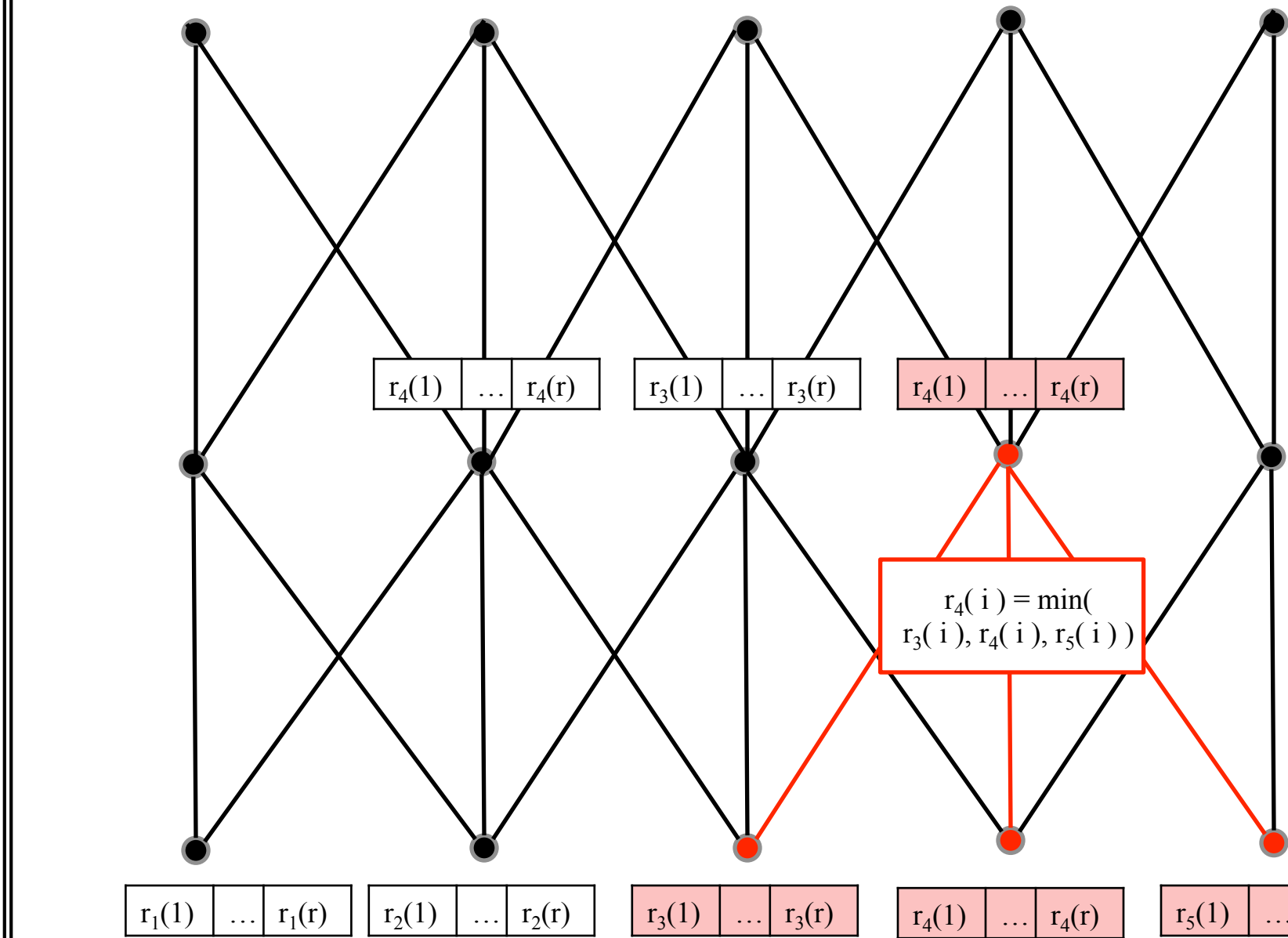
$$\text{Prob}(|\hat{T} - T| \geq \epsilon T) = O\left(\frac{1}{\epsilon\sqrt{r}}\right)$$

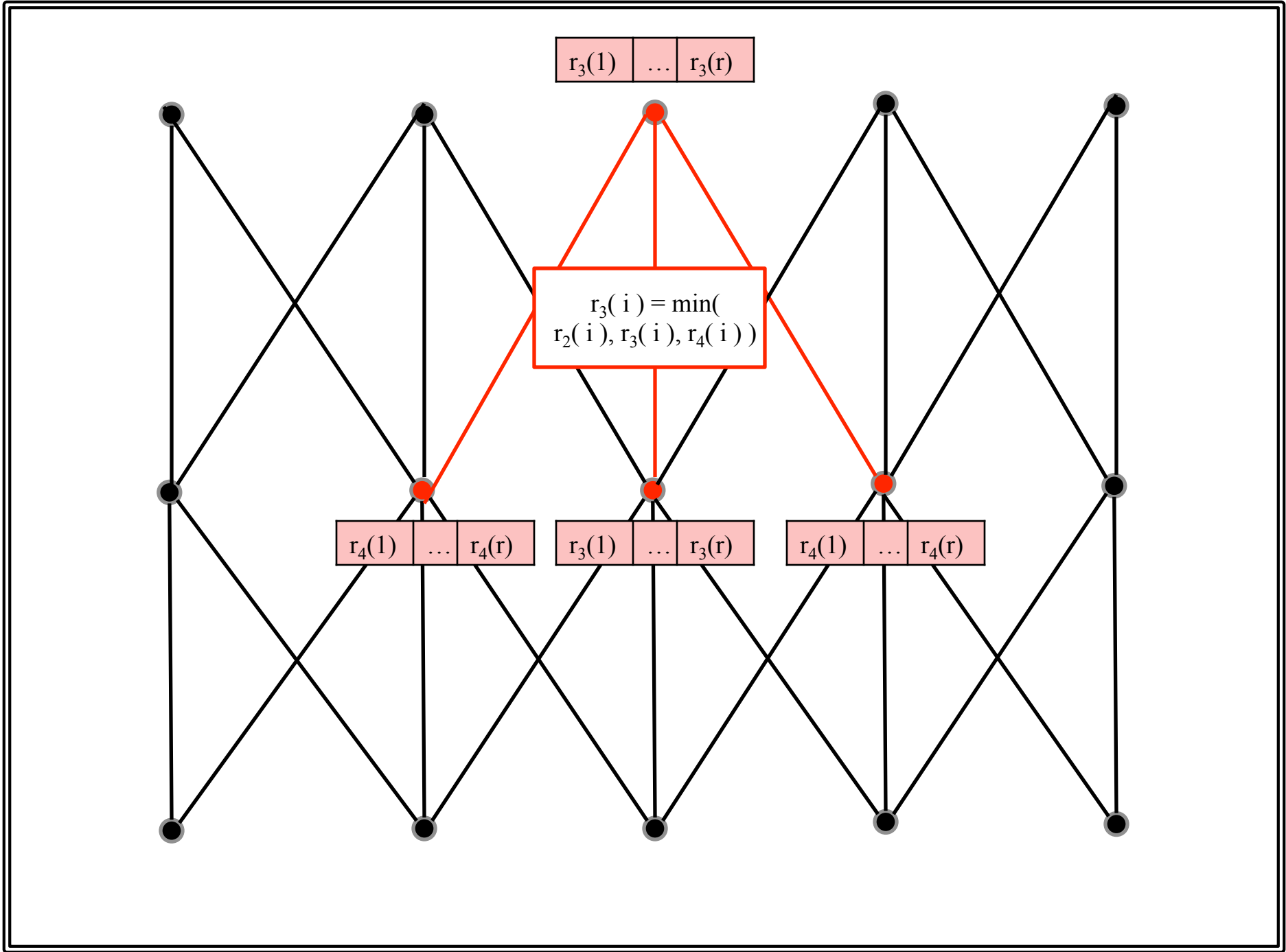
where T is the actual size of the transitive closure, r is the number of randomized rankings per vector











Delayed low-rank updates (blocking)

For square matrices A and B and vector x , $\{c_j\}$ is a sequence of vectors:

$$c_j = (A + B)^j x, \quad 0 \leq j \leq s$$

This recurrence generates the same sequence:

$$c_j = A^j x + \sum_{k=1}^j A^{j-k} B c_{j-k}$$

An algorithm that computes this sequence:

1. Compute and store
 1. $[A^0, \dots, A^{s-1}] \cdot B$
 2. $[A^0, \dots, A^s] \cdot x$
2. Compute $[c_0, c_1, \dots, c_s]$ recursively

Write $B = UV^T$, where U, V are $n \times r$ matrices ...

Algorithm 1:

1. Compute and store

$$[A^0, \dots, A^{s-1}] \cdot U$$

$$[A^0, \dots, A^s] \cdot x$$

2. Compute and store

$$V^T \cdot [A^0, \dots, A^{s-2}] \cdot U$$

$$V^T \cdot [A^0, \dots, A^{s-1}] \cdot x$$

3. Compute c_j for $j=1:s$ by:

$$V^T c_j = V^T A^j x +$$

$$\sum_{k=1}^j V^T A^{k-1} U \cdot V^T c_{j-k}$$

$$c_j = A^j x + \sum_{k=1}^j A^{k-1} U \cdot V^T c_{j-k}$$

Algorithm 0:

Compute c_j for $j=1:s$ by

$$c_j = A \cdot c_{j-1} + U \cdot V^T \cdot c_{j-1}$$

Algorithm 2:

1. Compute and store

$$V^T \cdot [A^0, \dots, A^{s-2}] \cdot U$$

$$V^T \cdot [A^0, \dots, A^{s-1}] \cdot x$$

2. Compute c_j for $j=1:s$ by:

$$V^T c_j = V^T A^j x +$$

$$\sum_{k=1}^j V^T A^{k-1} U \cdot V^T c_{j-k}$$

$$c_j = A \cdot c_{j-1} + U \cdot V^T c_{j-1}$$

Flop counts, assuming dense U and V

	Alg. 0	Alg. 1	Alg. 2
SpMV's with A	s	s	2s-1
Multiplies by V^T	s	1	1
Dense flops	4nrs	$nrs(s+3) + O(ns+r^2s^2)$	$4nrs + O(ns + r^2s^2)$
Memory footprint (Neglect A, U, V)	$n(s+1)$	$n((s+1)+sr)+O(r^2s)$	$n(s+1)+O(r^2s)$
Offline SpMMs [U, A·U, ..., A ^{s-1} ·U]	0	$(s-1) \times r$	$(s-1) \times r$
Offline dense flops $V^T \cdot [U, A \cdot U, \dots, A^{s-1} \cdot U]$	0	$2nr^2 (s-1)$	$2nr^2 (s-1)$

Note on 'SpMM' –our matrix powers kernel accepts multiple source vectors. This is a beneficial SpMV optimization when you are memory bound.

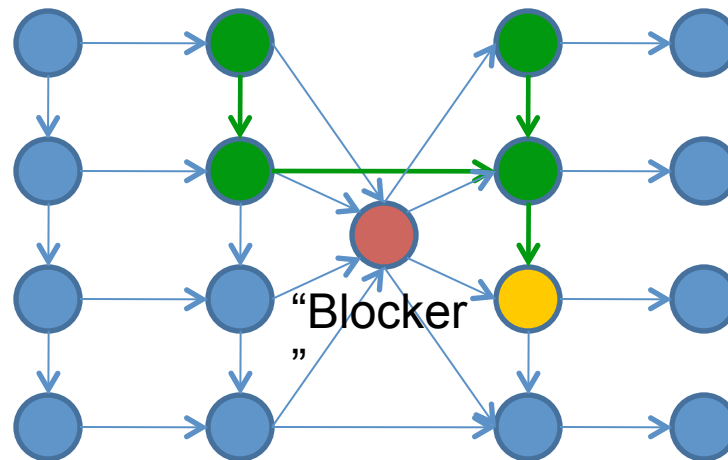
Previous work: blocking covers

“...Linear relaxation using blocking covers”

[Leiserson, Rao, Toledo '97]

Linear relaxation on graph G

- Cut outgoing edges from some vertices
- Equivalently, zero selected matrix rows of the matrix
 - $A = G - UV^T$, $B = UV^T$,
 - $U =$ identity columns, $V^T =$ selected matrix rows
- Considered “Alg. 2” variant

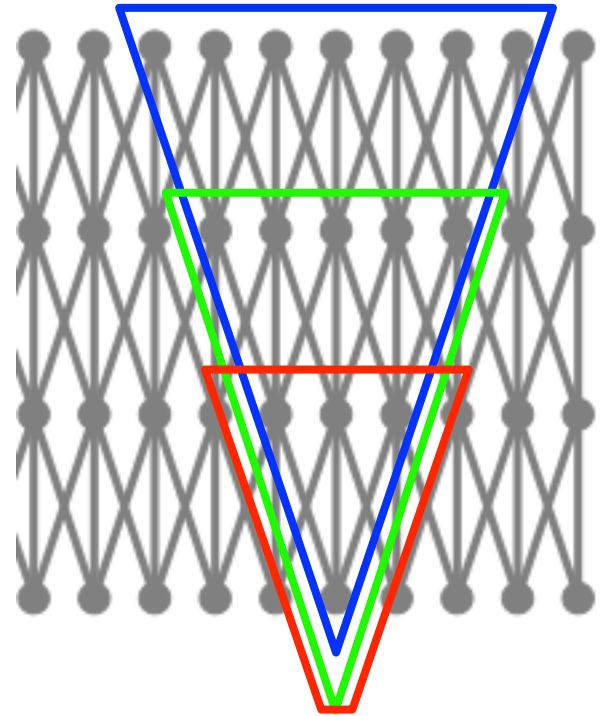


Blocking Covers

- Motivation: Hong and Kung's "Red-Blue Pebble Game": we can not avoid data movement if we can't find a good cover for a graph
 - Need to have subgraphs with high diameter
 - Low diameter indicates that information travels too quickly – we can not increase temporal locality
 - Equivalent to saying the matrix must be "well-partitioned"
- Blocking Covers Idea: [Leiserson, Toledo, Rao, 1995]
 - Blocking Covers: artificially restrict information from passing through some vertices by treating their state variables symbolically
 - We can maintain dependencies among the symbolic variables as a matrix
 - Good technique when connections are locally dense but globally sparse.
 - Main idea: if we remove some subset of vertices from the graph, then we *can* find a good cover.
- **Can we use the idea of blocking covers in the Matrix Powers Kernel to handle matrices which are not well-partitioned? (Future work)**

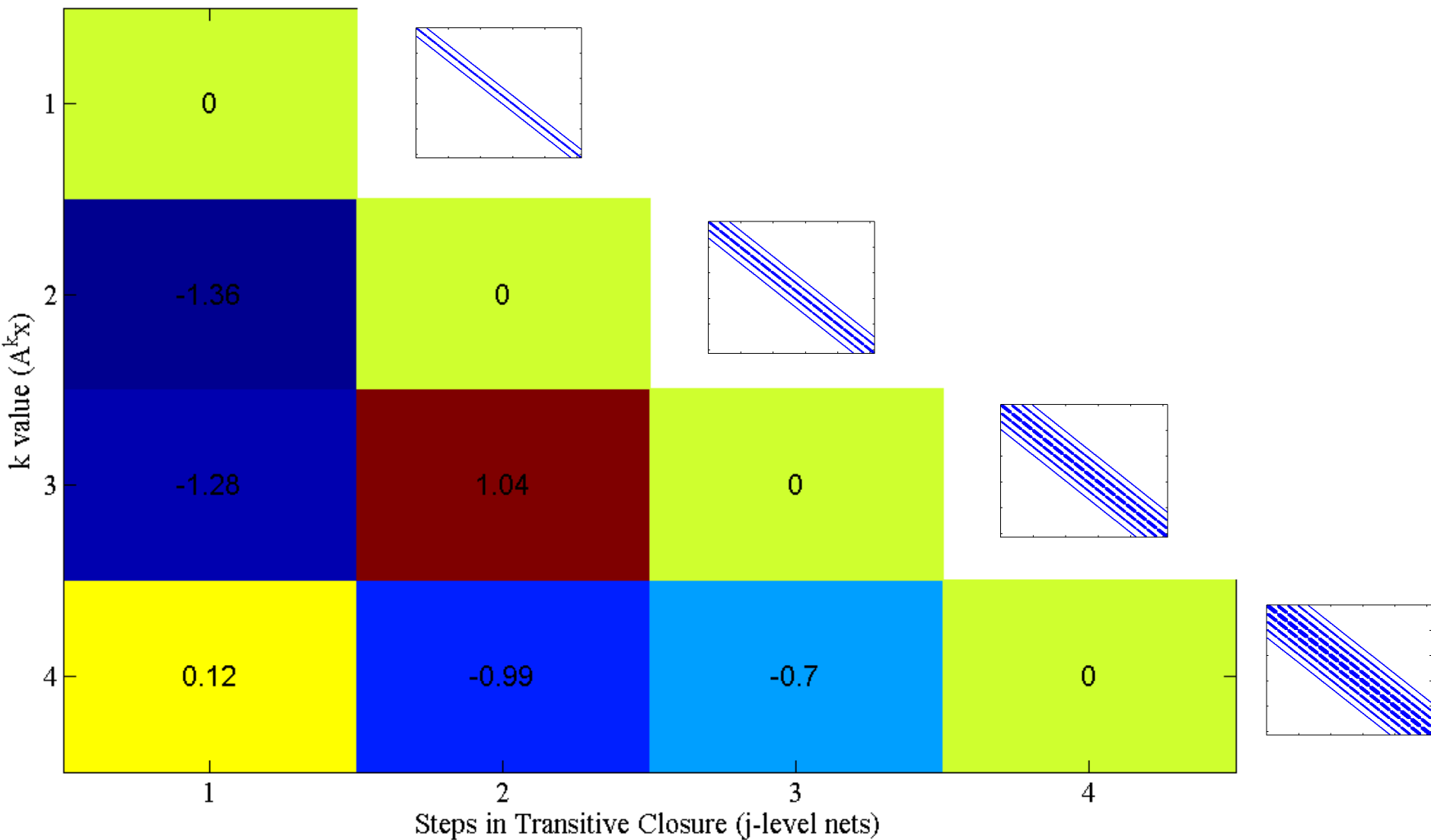
Experiment 1: Communication volume

- How good do j -level nets approximate s -level nets, for $1 \leq j < s$?
- Test 2 matrices:
 - 16×16 mesh, 5-pt. stencil (symmetric)
 - West0479 (unsymmetric)
- 4 parts
- Average over 3 runs
- Vertex weights set to row-net sizes of \tilde{A} (nnz-per-row).

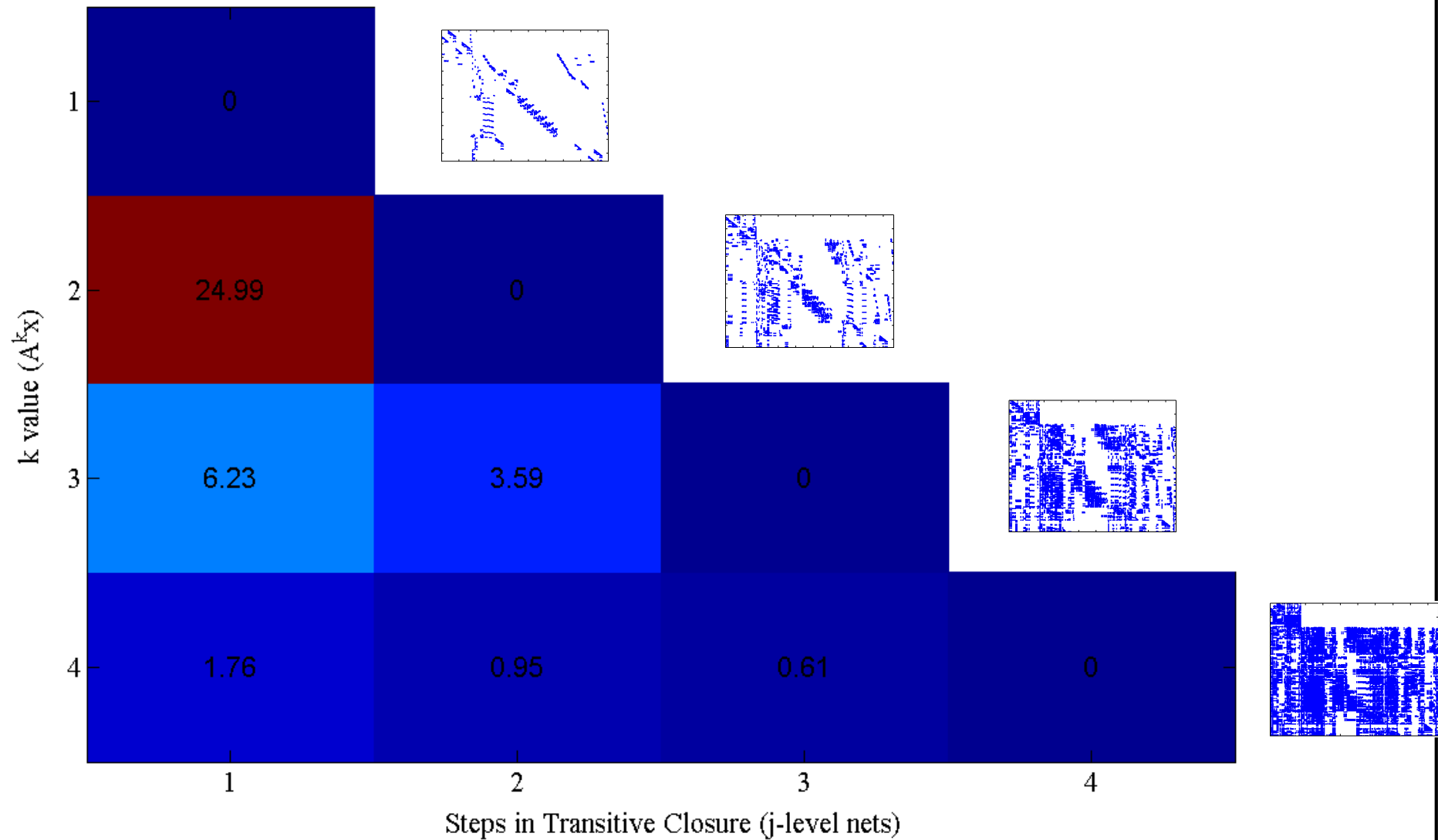


red = 1-level column-net
green = 2-level column-net
blue = 3-level column-net

Total Communication Volume, 16x16 5pt stencil, 4 parts



Total Communication Volume, west0479 matrix, 4 parts



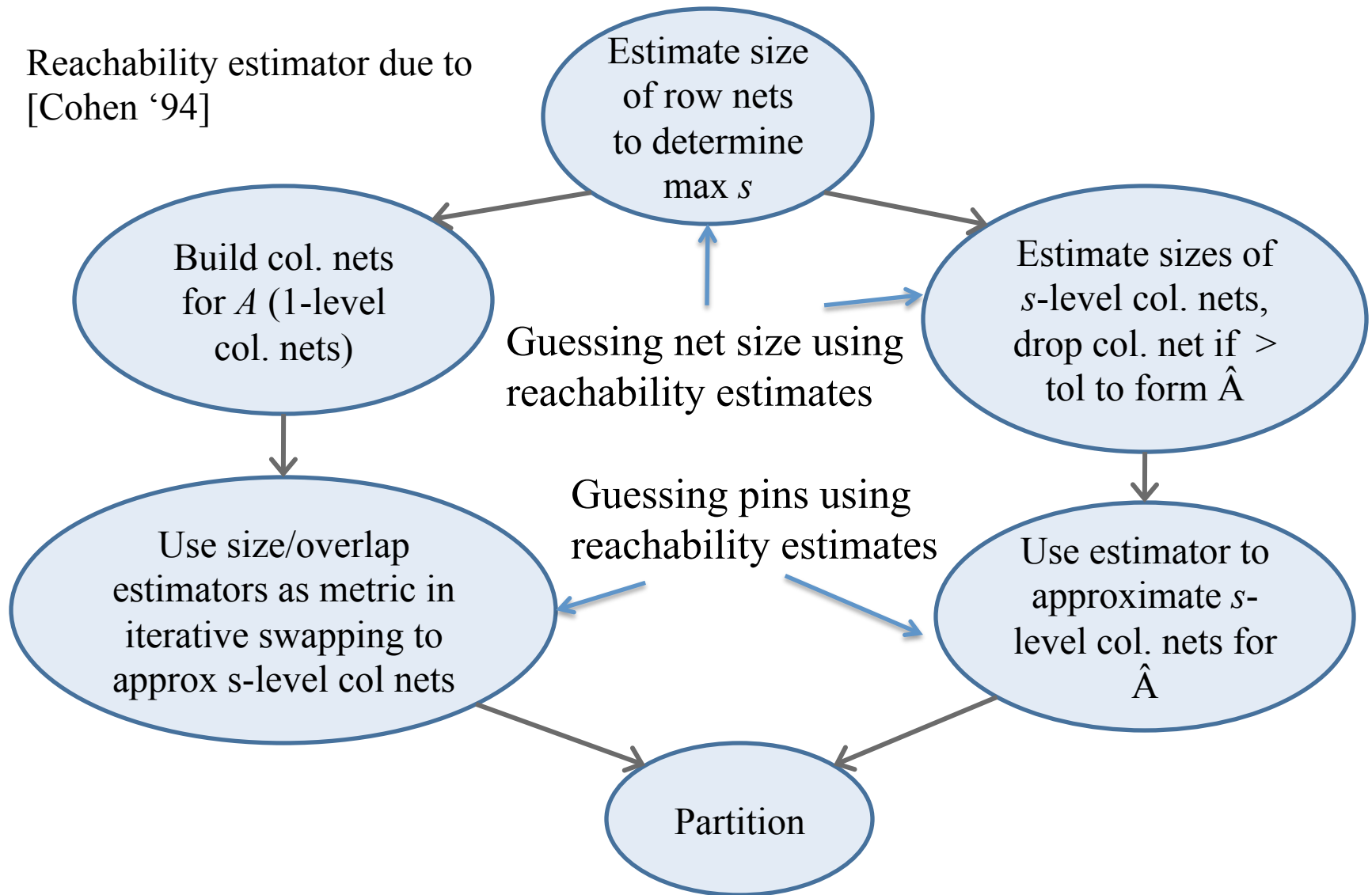
Experiment 1: Results

Communication volume

- For minimizing communication volume, s -level column nets always best (within noise)
 - biggest improvement for highly unsymmetric matrices.
 - but some levels $j < s$ were close.
 - Justifies approximating $|A|^j \approx |A|^s$ for partitioning.
- Partition quality degrades as matrix fills.
 - At some power s , $|A|^s$ cannot be “well partitioned” anyway. Think of a dense row.
 - How can we determine cutoff s ?

Reachability estimates and hypergraph partitioning

Reachability estimator due to
[Cohen '94]



Load Balancing

- SpMV (row-wise partition):
 - weight vertex i proportional to nnz in row i
 - ie: $w_i = 2 * nnz_i - 1$ (flops)
- Akx:
 - flops, thus vertex weights, are partition-dependent.
 - our original proposal needed n constraints per vertex, and this formulation still was an approximation. (PaToH broke...)
 - Assign vertex weights to column-net hypergraph...
 - # of deps in (column) level-nets
 - col level-net = multiset, the union of the $1:k$ level column nets
 - Same, but use row level-nets
 - Weight by nnzs in columns/rows (to count flops)

Previous Akx load-balancing:

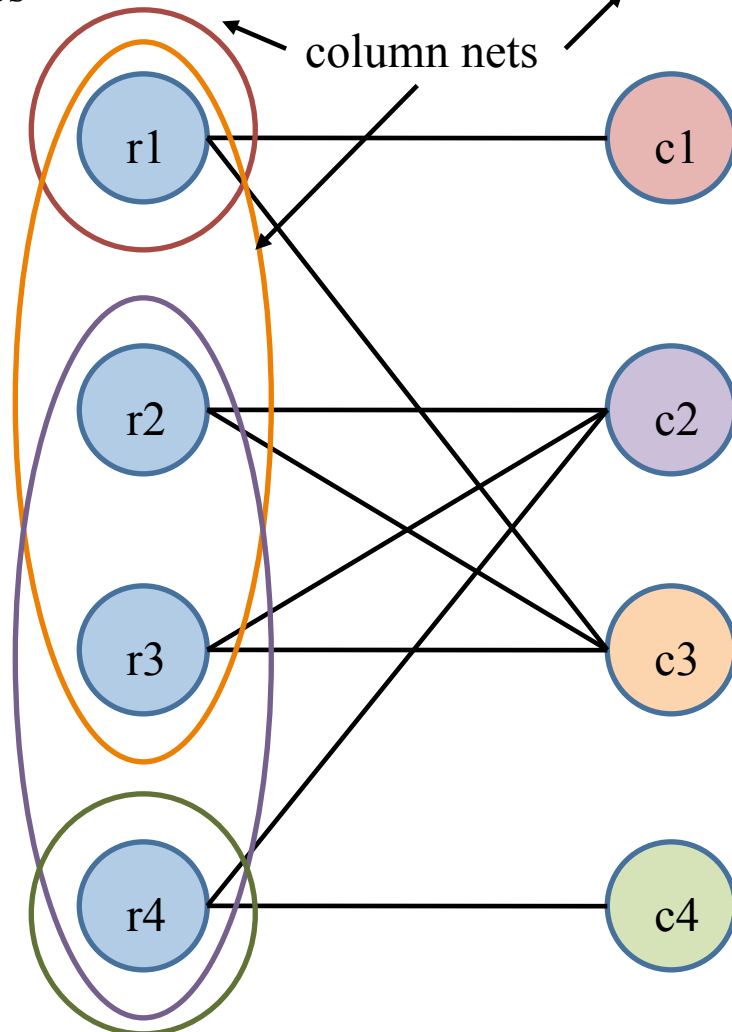
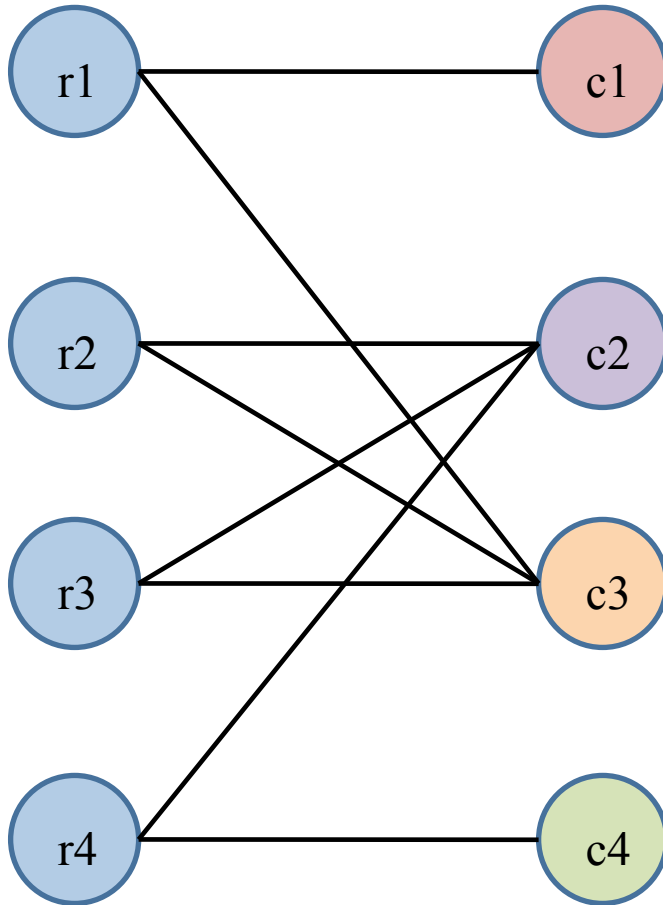
(from Marghoob)

- Default partitioner:
 1. METIS k-way, where $k = \# \text{threads}$
 2. Manually load-balance:
 1. Isolate largest+smallest parts,
 2. set target part_weights based on imbalance ϵ
 3. Bipartition
 3. Recursively bisect each part (thread-block) into cache-blocks

Graph/hypergraph of a matrix

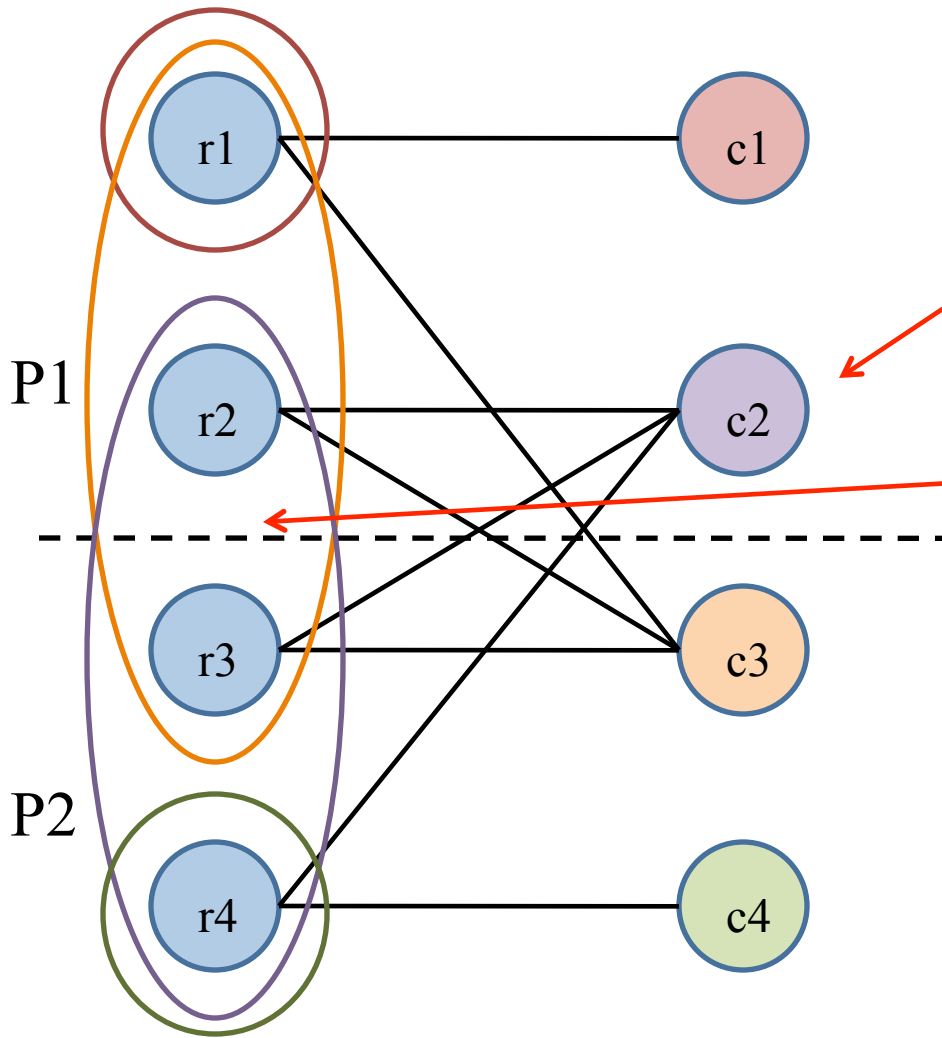
row vertices

column vertices


$$\begin{bmatrix} \times & 0 & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & 0 & \times \end{bmatrix}$$

Now, given a partition $P1 = \{r1, r2\}$, $P2 = \{r3, r4\}$, let's look at the dependencies and the values which must be communicated:

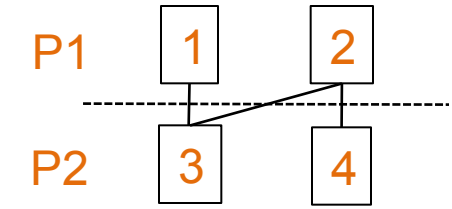
$$\begin{bmatrix} \times & 0 & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & \times & 0 \\ 0 & \times & 0 & \times \end{bmatrix}$$



Source vector entries corresponding to $c2$ and $c3$ are needed by both partitions – so total volume of communication is 2

This corresponds to the hyperedge cut

But graph cut is 3!



⇒ Cut size of graph partition may not accurately count communication volume

Hypergraph Partitioning for Matrix Powers

Reducing Partitioning Time using
Estimated Size of the Transitive Closure

Estimating Reachability

- Fastest algorithms for computing transitive closure are $O(nm)$, where $m = nnz * n$ (Dijkstra)
- Edith Cohen's Reachability Estimation ('94)
 - $O(n * nnz)$ time algorithm for estimating size of transitive closure (the size of each hyperedge in A^k)
 - Previous best was $O(n * \sqrt{m})$ (Lipton and Naughton)
 - Adapted to compute col/row sizes in matrix product (for k repeated SpMV's, equivalent to k steps of the transitive closure)
- Motivation: DB-query size estimations, data mining, optimal matmul ordering, efficient memory allocation
 - **New motivation: Reducing hypergraph partitioning time (partitioning time and building the column nets) for computing matrix powers**

Algorithm Overview

- Initially assign r -vector of rankings (sampled from exponential R.V., $\lambda = 1$) to each vertex v
- In each iteration (up to k), for each vertex v , take the coordinate-wise minima of the r -vectors reachable from v (denoted $S(v)$, non-zeros in column of A corresponding to v)
- Intuition: lowest-ranked node in $S(v)$ is highly correlated with $|S(v)|$
 - Example: If $S(v)$ contains half the nodes, we expect the lowest rank of nodes in $S(v)$ is very small.

$$\text{Prob}(|\hat{T} - T| \geq \epsilon T) = O\left(\frac{1}{\epsilon\sqrt{r}}\right)$$

where T is the actual size of the transitive closure, r is the number of randomized ranks per vector

Estimating Size of the Transitive Closure

3.1. Estimation algorithm for column-sizes

1. Assign for each node in V_1 an r -vector that consists of random samples from the Exponential distribution with parameter $\lambda = 1$ (that is, density function $f(x) = \lambda e^{-\lambda x}$ and distribution function $F(x) = 1 - e^{-\lambda x}$.)
2. For layers $i = 2, \dots, k + 1$ do as follows:
 - For each node $v \in V_i$:
Assign to v the r -vector that equals the coordinate-wise minima of the r -vectors of all the V_{i-1} neighbors of v .
3. At this point, every node in the graph has an r -vector associated with it. To estimate the number of nonzeros of a column in \mathbf{A}'_j , apply an *estimator* to the r -vector of the corresponding node from layer $j + 1$. For a vector (a_1, \dots, a_r) , we apply the estimator

$$\frac{r - 1}{\sum_{i=1}^r a_i}$$

($r - 1$ divided by the sum of the coordinates.)

Preliminary Experiments

- 1) Use estimator to determine maximum k value
 - Given user-specified desired k value, compute powers of A for $s = 1:k+1$, applying estimator to current sums for each row
 - When estimated nnz per row is $> tol * n$, stop and return s (tol is chosen based on our definition of “well-partitioned”)
- 2) For this value s , run column-size estimator on A^s
 - If estimated size of column $j > (nprocs - 1)/nprocs$, zero out this column of the matrix (except diagonal entry)
 - Intuition: if size of column is greater than $(nprocs - 1)/nprocs$, all partitions will probably need element j from the source vector anyway
 - $r \sim \log(n)$
 - Note: not yet using estimator to estimate k -level col nets for sparsified A , explicitly computing powers right now
 - This is an ok alternative if $nnz(\text{sparsified } A^k) \sim nnz(A)$

Further Extensions to Hypergraph Partitioning for Matrix Powers

- Future Work
 - Computing the transitive closure (which vertices are in k -level column nets)
 - Is this estimate useful? We already need to compute the row-nets in $A^k x$ preprocessing so we get the column-nets for free. Could we replace the dependency computation in $A^k x$ preprocessing with this method? Or is an estimate not good enough?
 - computes transitive closure in $O(n^2 \log n)$ time WHP
 - Computing similarity (shared vertices) between k -level column nets
 - Could be used as a metric for KL iterative swapping in 1-level column nets to approximate k -level column nets