Numerical Optimization, Formulations, Serial and Parallel Algorithms for Machine Learning and Network Science

Kimon Fountoulakis, Postdoctoral Fellow and coPI @UCB, ICSI

Outline of the talk

1. My background
2. Parallel methods for convex optimization
3. Variational perspective of local graph clustering
4. Software

My background

1. PhD on numerical optimization. In particular, Newton-type methods for signal/image processing and machine learning. Complexity of Newton-type methods and development of provably efficient preconditioners for iterative solution of linear systems.

2. Three year postdoctoral fellow at University of California Berkeley. Experience on parallelizing optimization methods for ML and Graph analytics (**this talk**). Also worked a lot on variational perspective of local graph clustering (**this talk**).

Parallel methods for convex optimization

Avoiding synchronization in first-order methods for sparse convex optimization, A. Devarakonda, J. Demmel, K. Fountoulakis, M. Mahoney, IPDPS, 2018

Avoiding communication in primal and dual coordinate descent methods, A. Devarakonda, J. Demmel, K. Fountoulakis, M. Mahoney, SIAM Scientific Computing, 2018 (minor revisions)

Optimization problems

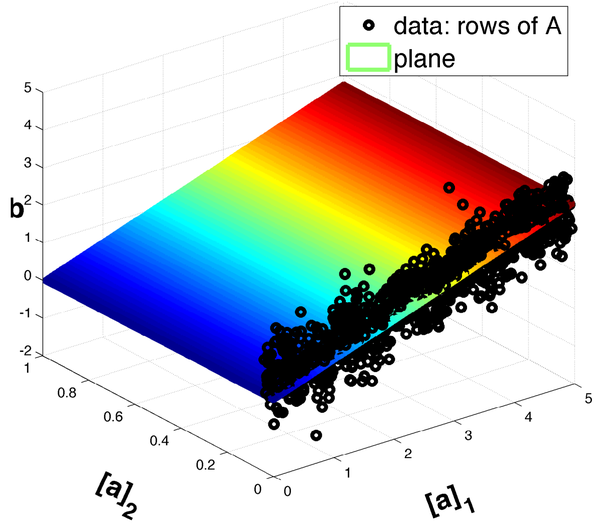
$$\text{minimize } \lambda g(x) + f(x; A, b)$$

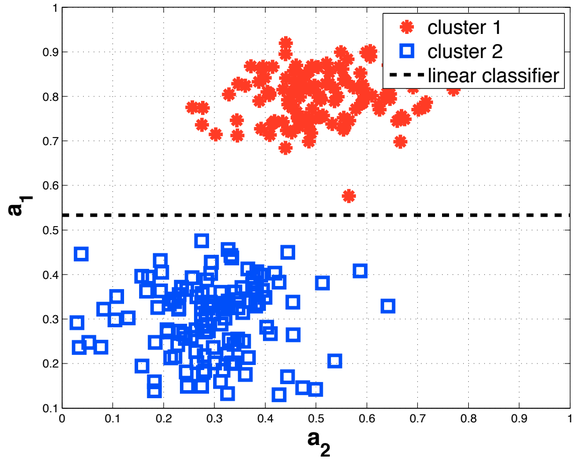
- Sparse regression $g(x) = \|x\|_1 \quad f(x) = \|Ax - b\|_2^2$

- Sparse SVM $g(x) = \|x\|_1 \quad f(x) = \sum_{i=1}^{m} \max(1 - b_i A_i^T x, 0)^2$

- Elastic net $g(x) = \frac{\eta}{2}\|x\|_2^2 + (1-\eta)\|x\|_1$

- Group lasso $g(x) = \sum_{j=1}^{J} \|x_j\|_{K_j}$



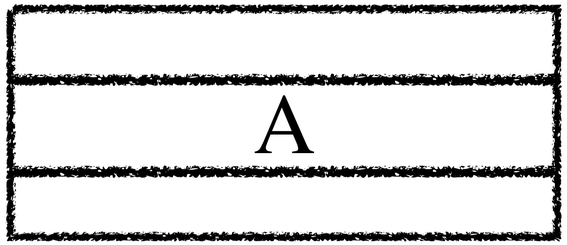


Assumptions for this talk

Ridge Regression

$$\text{minimize } \lambda\|x\|_2^2 + \frac{1}{2}\|Ax - b\|_2^2$$

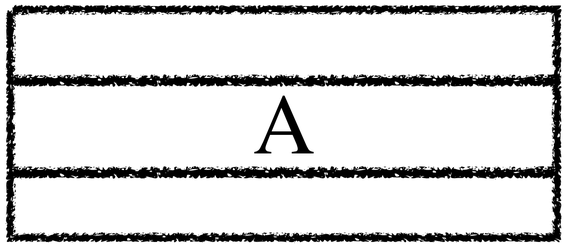
1D Row Partition



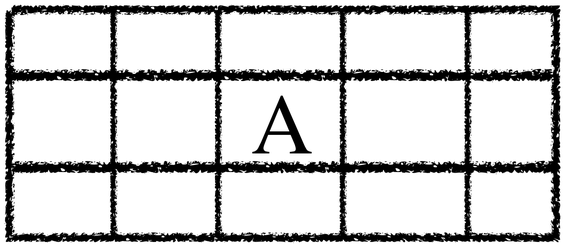
Method: Single coordinate descent

Scalable results for all data layouts

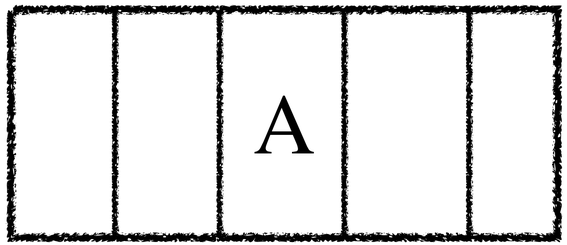
1D Row Partition



2D Block Partition



1D Column Partition



* Best performance depends on dataset and algorithm

Methods that we can speed up

- Block coordinate descent

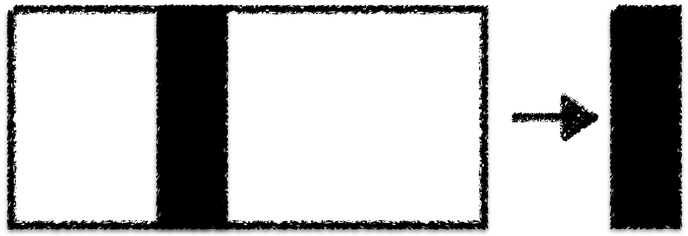
- Accelerated block coordinate descent

- Newton-type (first-order++) block coordinate descent

- Proximal versions of the above

An example: coordinate descent

- Sample a column of data



$\boldsymbol{e_k}$: indicator vector for chosen coordinate at kth iteration

$$\boldsymbol{x_{k+1}} := \boldsymbol{x_k} + \boldsymbol{e_k}\Delta x_k$$

$$\boldsymbol{A_k} := \boldsymbol{A e_k}$$

$$\Delta x_k := \operatorname{argmin}_{\Delta x \in \mathbb{R}} \lambda \|\boldsymbol{x_k} + \boldsymbol{e_k}\Delta x\|_2^2 + \frac{1}{2}\|\boldsymbol{A x_k} + \boldsymbol{A}_k \Delta x - \boldsymbol{b}\|_2^2$$

Or equivalently $$\Delta x_k := -\underbrace{\frac{1}{2\lambda + \boldsymbol{A_k}^T \boldsymbol{A_k}}}_{\text{step-size}} \underbrace{(2\lambda \boldsymbol{x_k}^T \boldsymbol{e_k} + (\boldsymbol{A x_k} - \boldsymbol{b})^T \boldsymbol{A_k})}_{\text{partial derivative}}$$

What are the basic computational primitives that we need to parallelize?

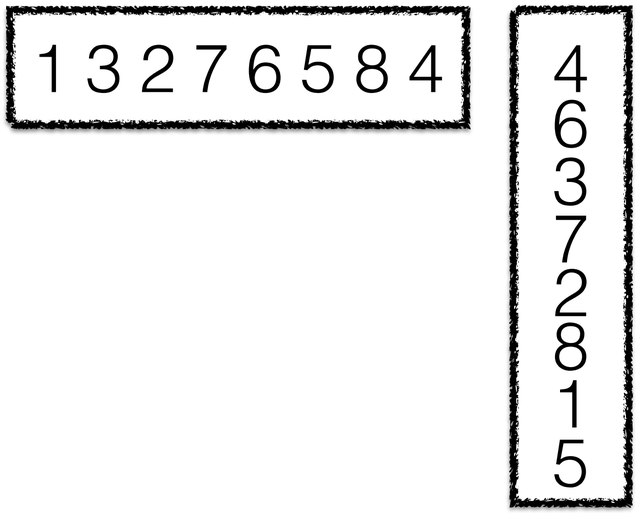
$$\Delta x_k := -\underbrace{\frac{1}{2\lambda + \boldsymbol{A_k}^T \boldsymbol{A_k}}}_{\text{step-size}} \underbrace{(2\lambda \boldsymbol{x_k}^T \boldsymbol{e_k} + (\boldsymbol{A x_k} - \boldsymbol{b})^T \boldsymbol{A_k})}_{\text{partial derivative}}$$

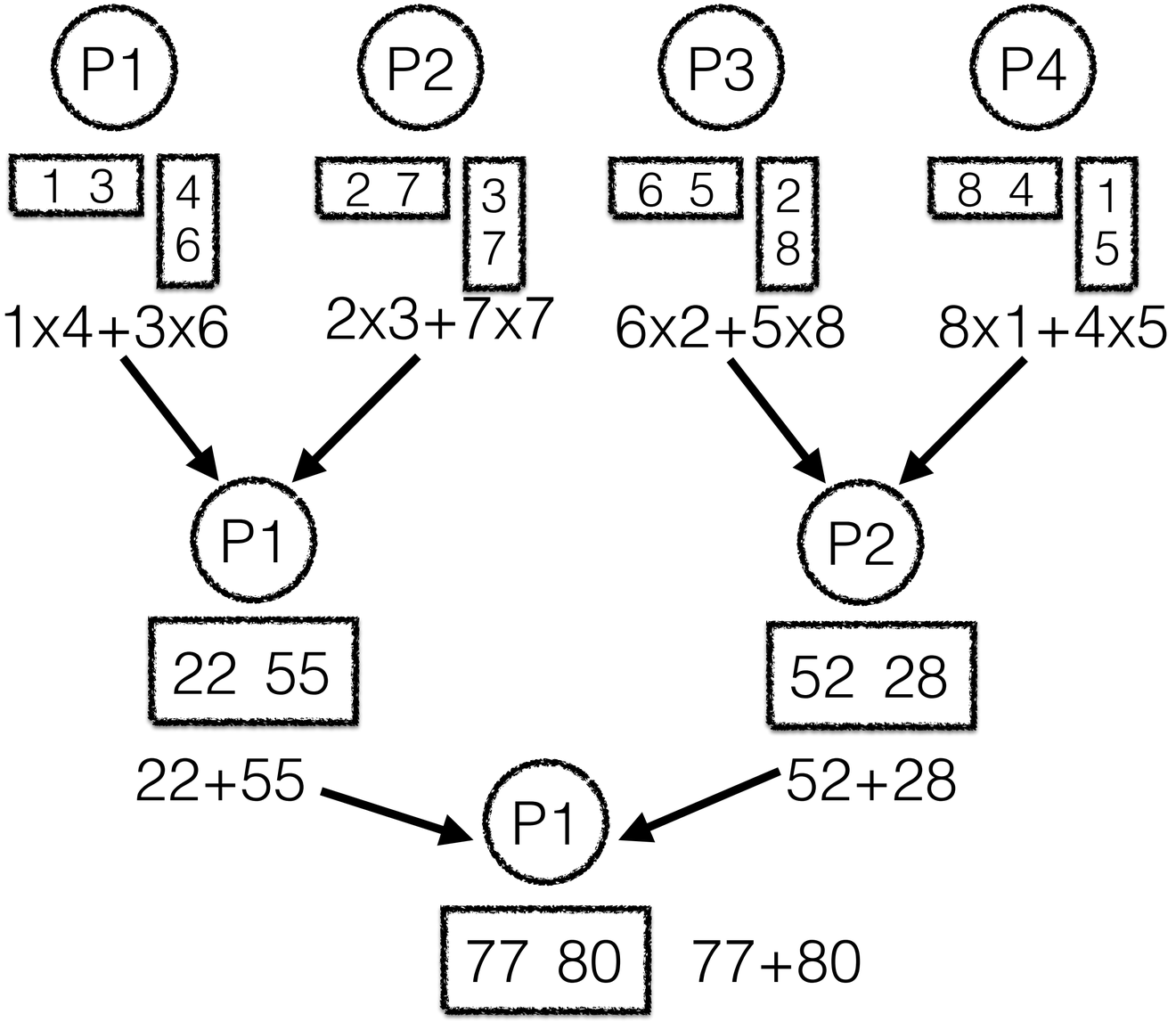
At each iteration we need to compute in parallel the following inner products

$$\boldsymbol{A_{k+s}}^T \boldsymbol{A_{k+s}} \qquad (\boldsymbol{A x_{k+s}} - \boldsymbol{b})^T \boldsymbol{A_{k+s}}$$

How can we compute inner products in parallel?

Distribute this in 4 Processors





How much does it cost to compute an inner product in parallel?

Log P (depth of binary tree) messages in parallel, where P is the number of processors

Each message costs $\alpha + n\beta$ **communication time**

Constant cost: start-up time

Transfer time per byte

Number of bytes to send

One inner product cost (length of vector)/P FLOPS **computational time**

Running time: computational time + communication time

Summary of coordinate descent

Pseudo-code

1 tree reduction per iteration

- Sample a column of data

- Compute partial derivative

- Update solution

- Repeat

A better version of coordinate descent

1 tree reduction per “s” iterations instead of 1 tree reduction at every iteration

“s” is a user defined parameter

However, each node of the tree performs more computations

Summary of results

Decrease start-up time (latency) by a factor of s

No free lunch: increase number of bytes and flops by a factor of s

Flops are distributed across processors

Logarithmic dependence of start-up time **on number of processors**

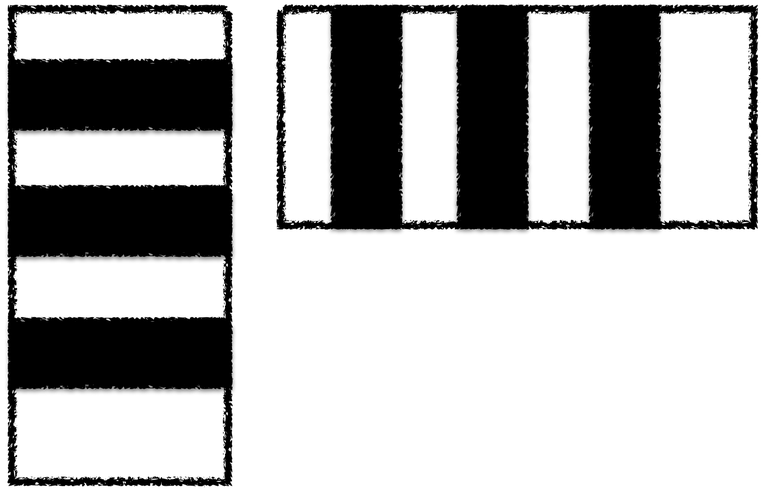
Avoid start-up costs by viewing the algorithm as a recurrence

Instead we can unroll

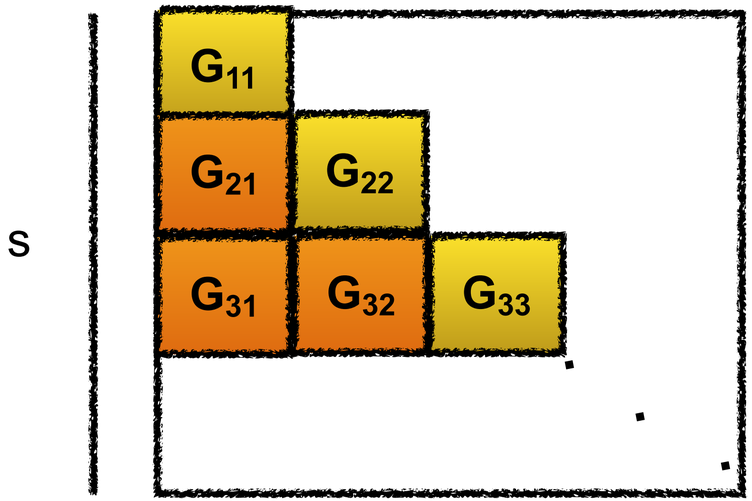
$$\Delta x_{k+s} := -\underbrace{\frac{1}{2\lambda + \boldsymbol{A_{k+s}}^T \boldsymbol{A_{k+s}}}}_{\text{step-size}} \underbrace{(2\lambda \boldsymbol{x_{k+s}}^T \boldsymbol{e_{k+s}} + (\boldsymbol{A x_{k+s}} - \boldsymbol{b})^T \boldsymbol{A_{k+s}})}_{\text{partial derivative}}$$

"s" iterations back and notice that all we need to compute the direction is a bunch of inner products which can be performed in **parallel**.

"s" chosen columns



Store all pairs of inner products in an s x s covar. matrix

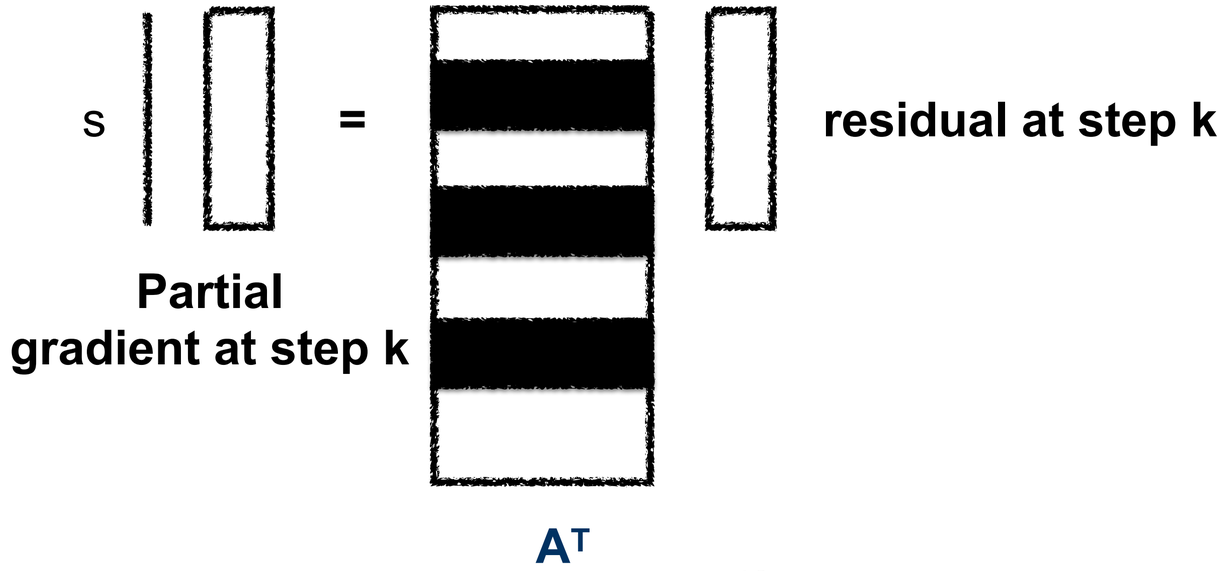


Avoid start-up costs by viewing the algorithm as a recurrence

Instead we can unroll

$$\Delta x_{k+s} := -\underbrace{\frac{1}{2\lambda + \boldsymbol{A_{k+s}}^T \boldsymbol{A_{k+s}}}}_{\text{step-size}} \underbrace{(2\lambda \boldsymbol{x_{k+s}}^T \boldsymbol{e_{k+s}} + (\boldsymbol{A x_{k+s}} - \boldsymbol{b})^T \boldsymbol{A_{k+s}})}_{\text{partial derivative}}$$

“s” iterations back and notice that all we need to compute the direction is a bunch of inner products which can be performed in **parallel**.

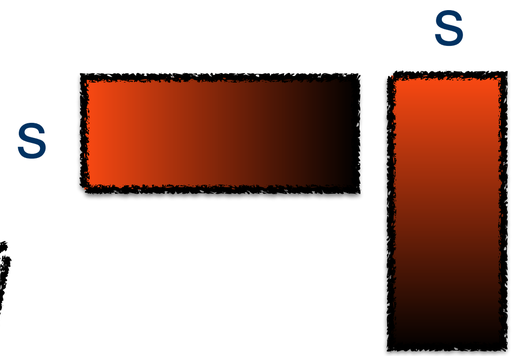


A better algorithm: communication avoiding coordinate descent

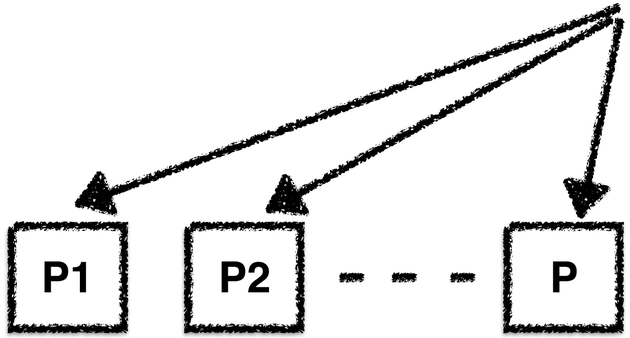
Pseudo-code

1 tree reduction per s iterations

- Compute in parallel anticipated computations for the next “s” iterations



- Redundantly store the result in all processors



- Each processor independently computes the next “s” iterations

- Repeat

Running time in Big-O for coordinate descent

No free lunch: Exchange FLOPS and bandwidth for latency

$$\gamma \times \left(s\frac{Hfm}{P} \right) + \alpha \times \left(\frac{H \log P}{s} \right) + \beta \times sH \log P$$

s=1: coordinate descent s>1: communication avoiding coordinate descent

α Time per message

β Time per word

γ Time per FLOP

P Number of cores

$f = nnz(A)/mn$

H Number of iterations

m Features

n Samples

Optimal parameter “s”

$$s = \sqrt{\frac{\alpha P \log P}{\gamma f m + \beta P \log P}}$$

α Time per message

β Time per word

γ Time per FLOP

P Number of cores

$f = nnz(A)/mn$

H Number of iterations

m Features

n Samples

-Let’s assume A is very sparse, i.e., γfm is very small, then

$$s \approx \sqrt{\frac{\alpha}{\beta}}$$

Cori or Edison

$$= \mathcal{O}\left(\sqrt{\frac{10^{-6}}{10^{-10}}}\right) = 100$$

Preliminary experiments

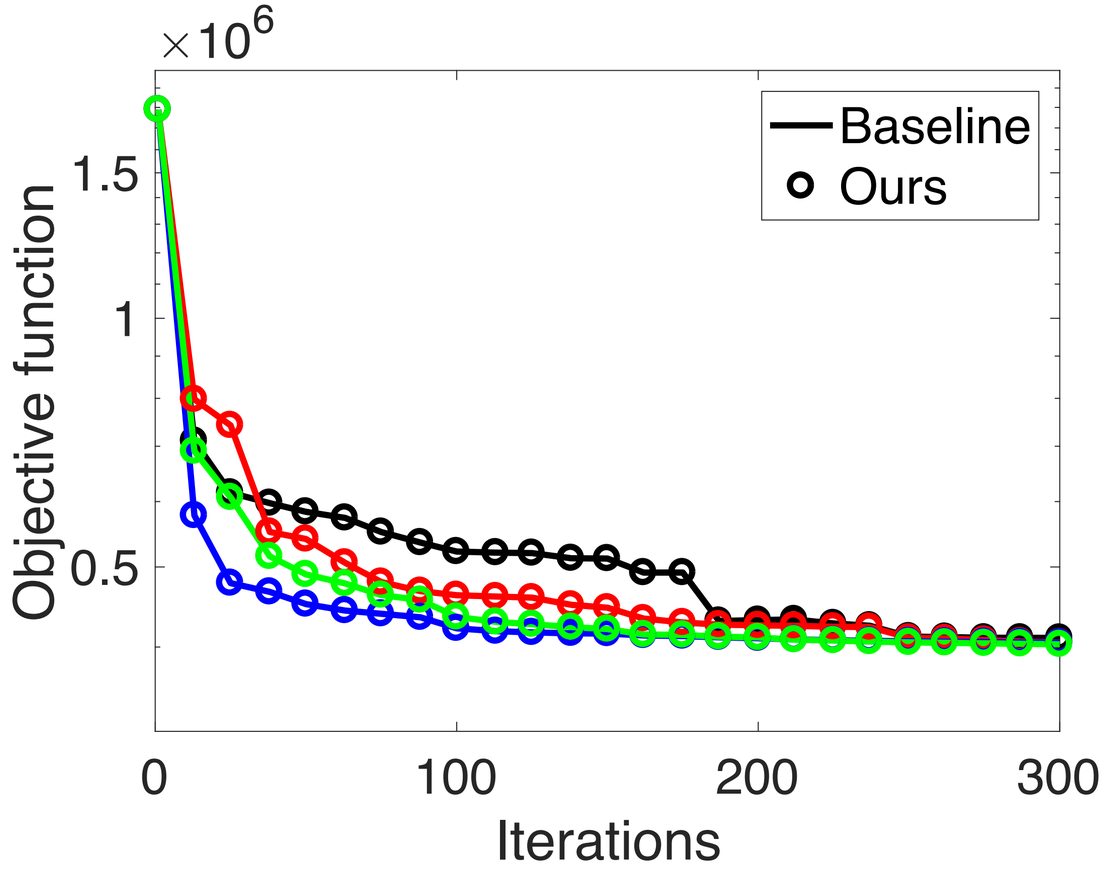
Summary of (LIBSVM) datasets

Name	#Features	#Data points	Density of non-zeros
url	3,231,961	2,396,130	0.0036%
epsilon	2,000	400,000	100%
news20	62,021	15,935	0.13%
covtype	54	581,012	22%

C++ using the Message Passing Interface (MPI). Intel MKL library for sparse and dense BLAS routines. All methods were tested on a Cray XC30.

Convergence of re-organized algorithms

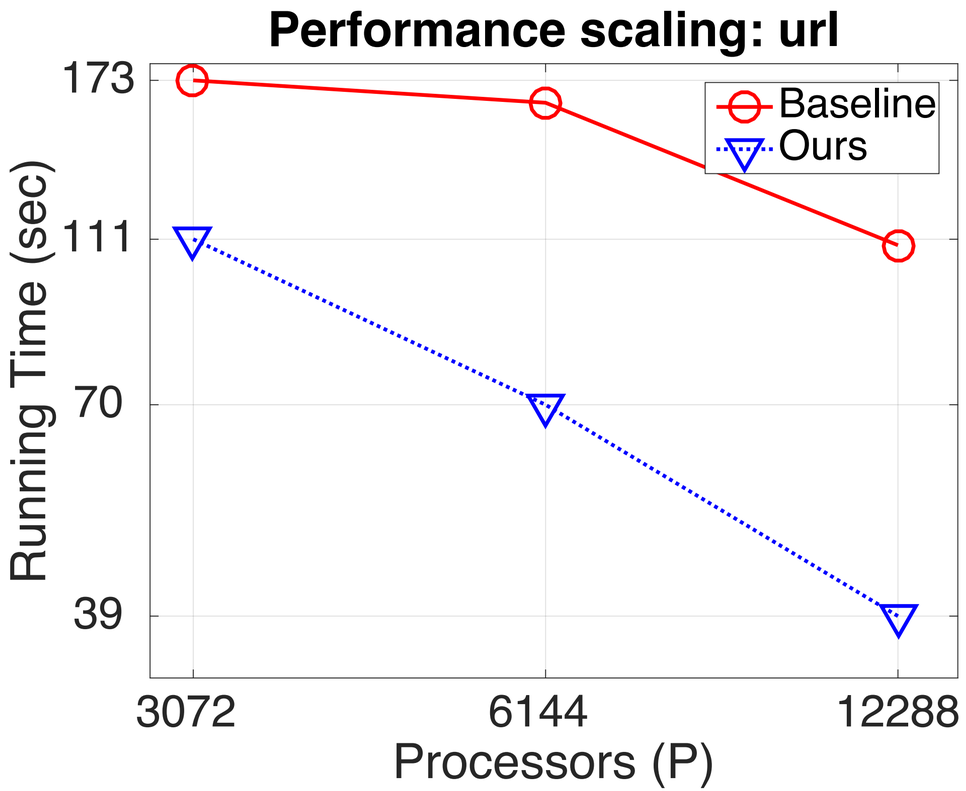
Convergence rate remains the same in exact arithmetic
Empirically stable convergence: no divergence between methods

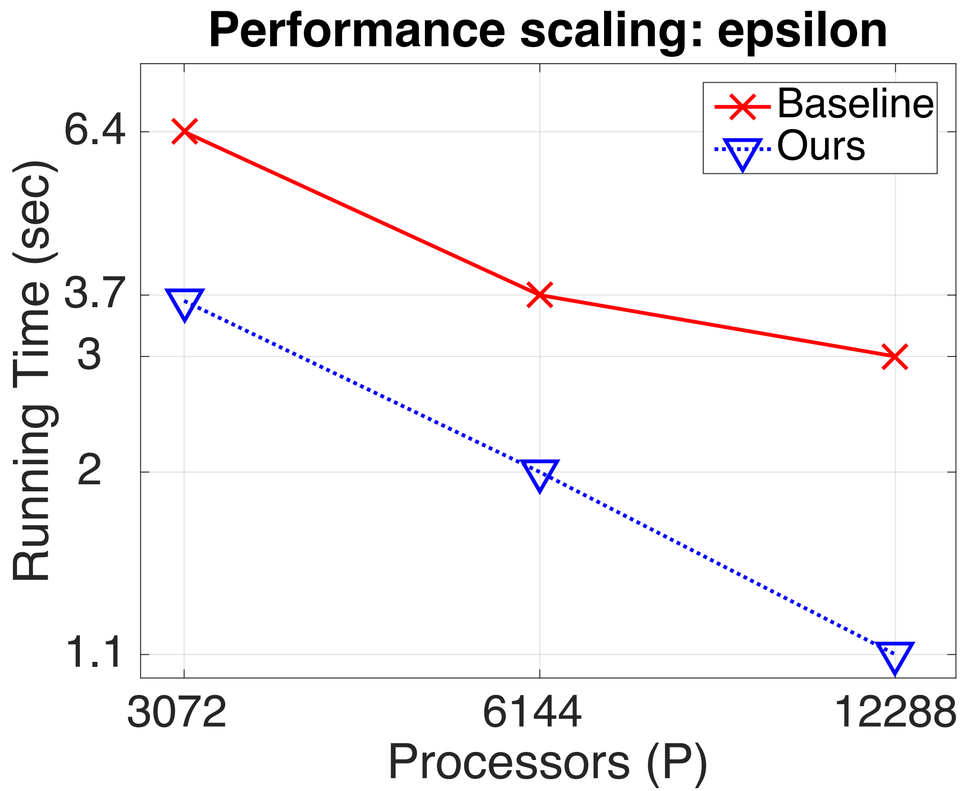


Scalability performance

“The more processors the better”

The gap between CA and non-CA increases w.r.t. #processors

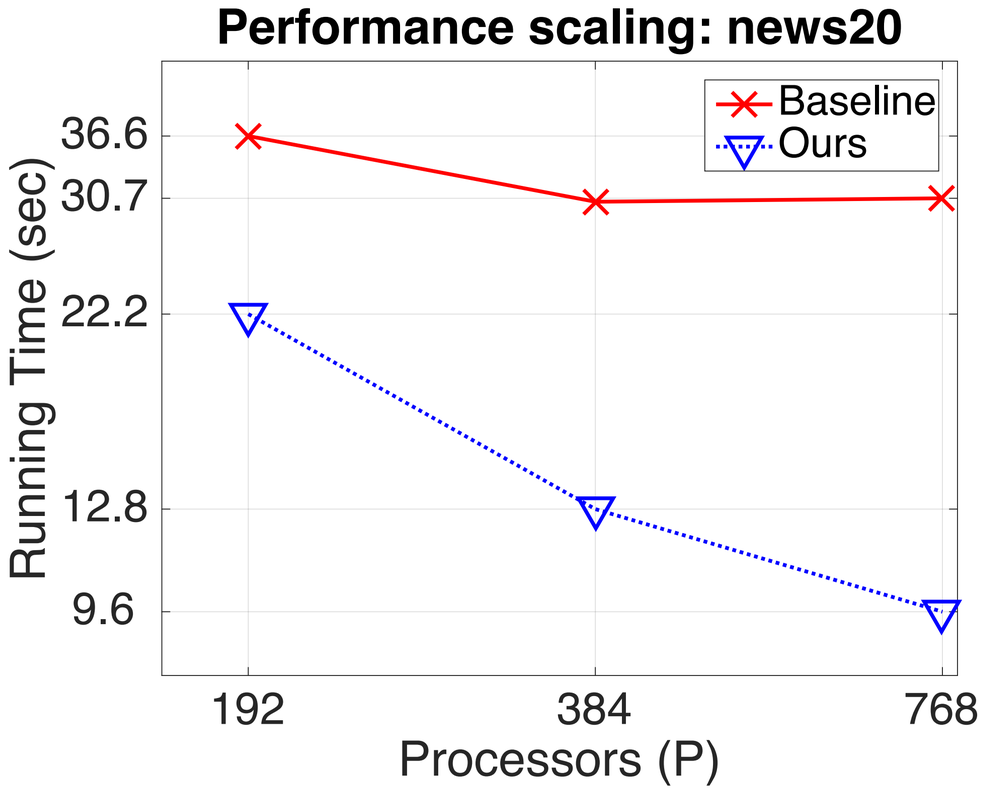


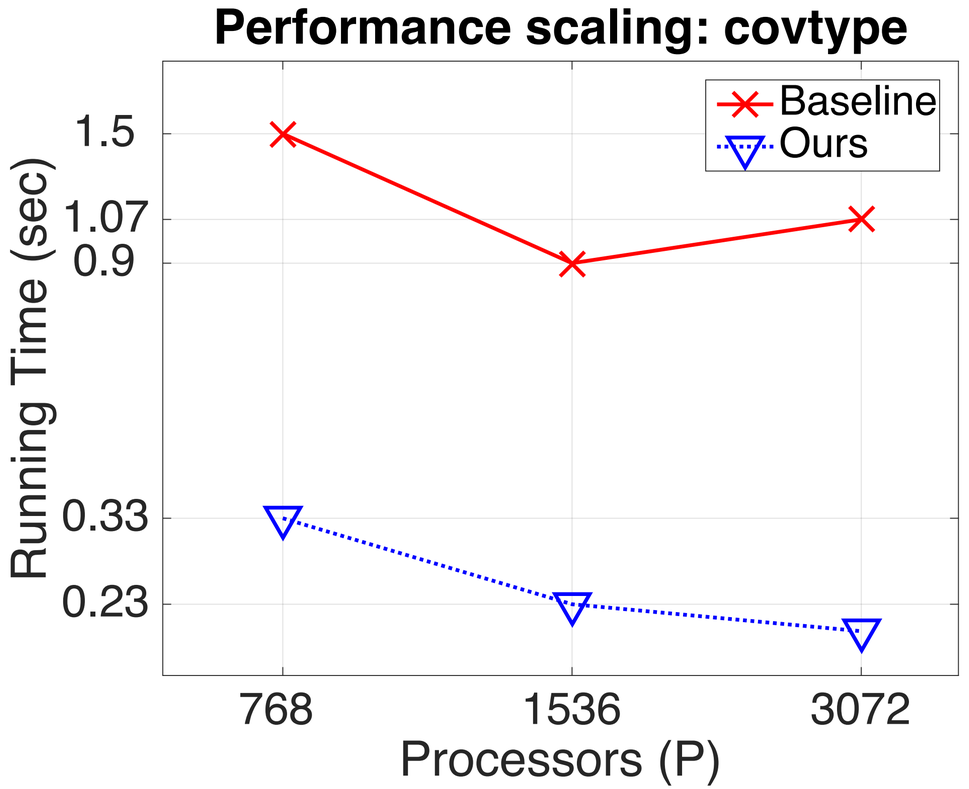


Scalability performance

“The more processors the better”

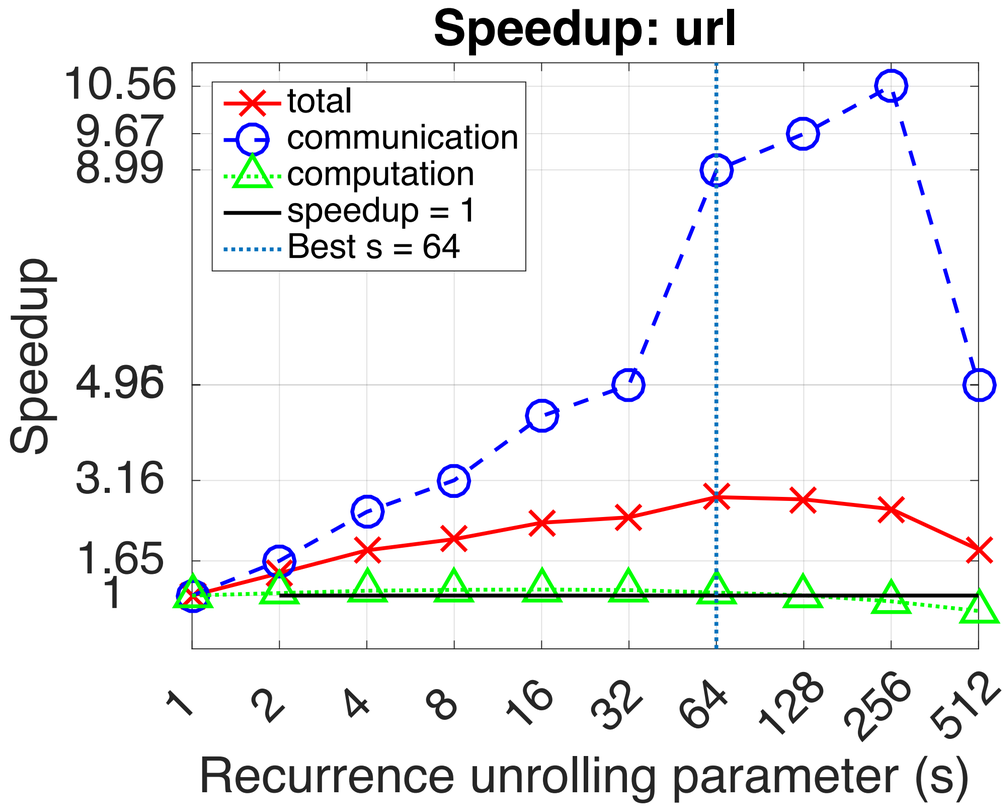
The gap between CA and non-CA increases w.r.t. #processors

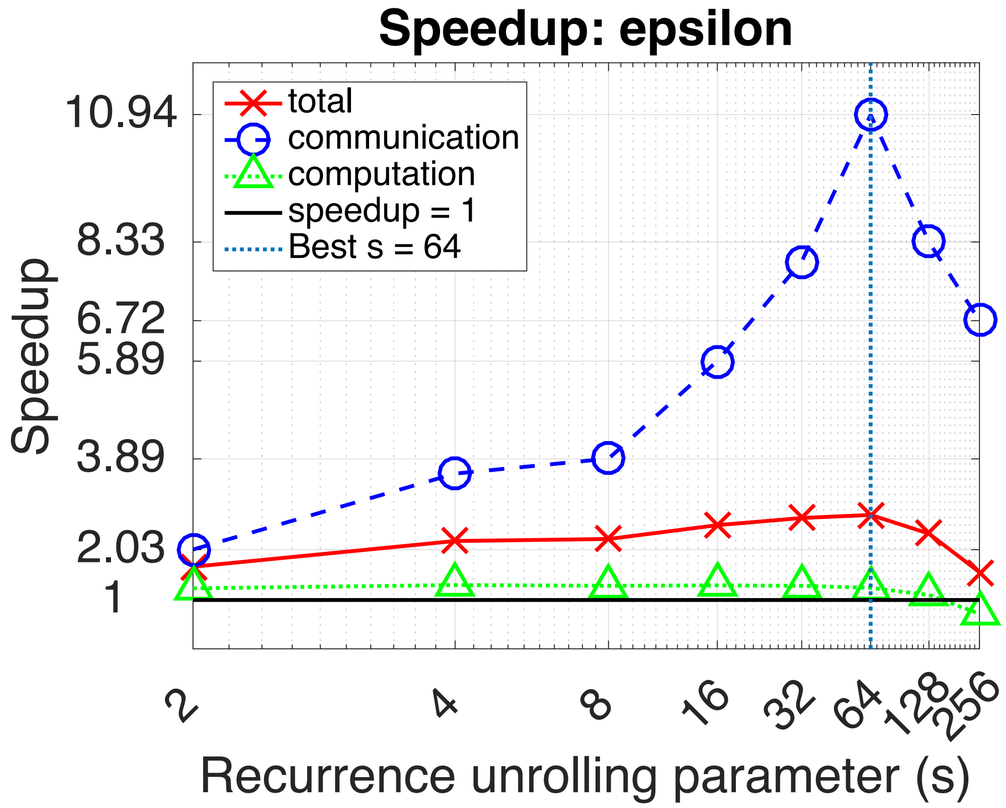




Speed up breakdown

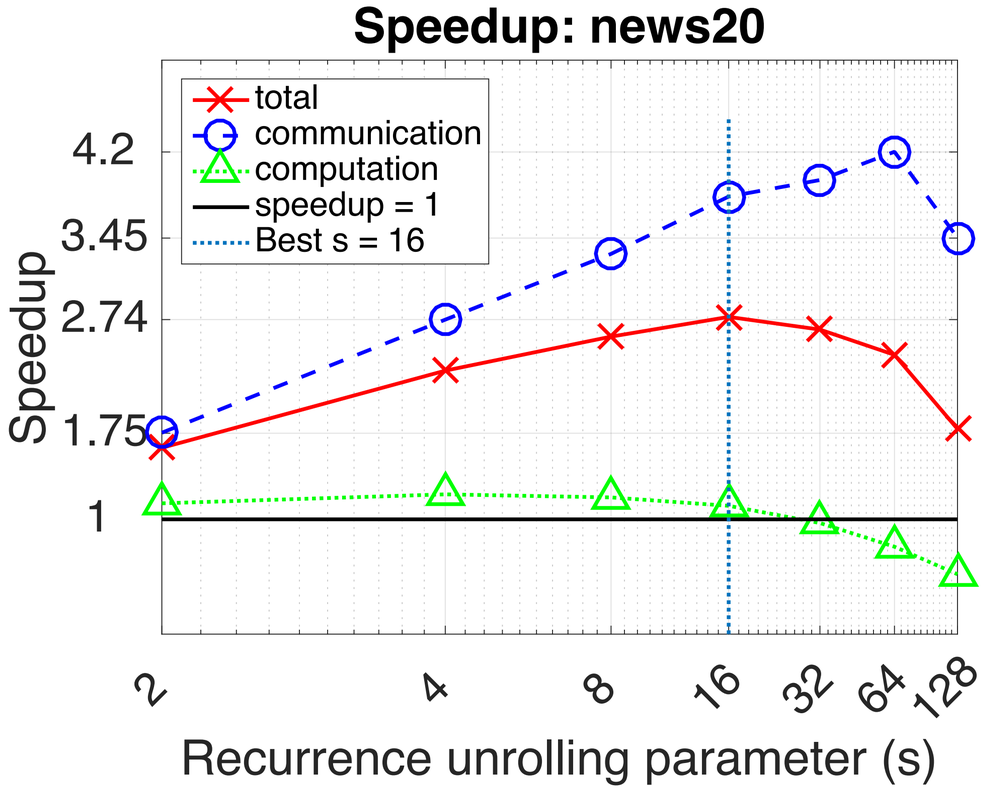
Large communication speedup until bandwidth takes a hit
Computation is maintained due to local cache-efficient (BLAS-3) computations

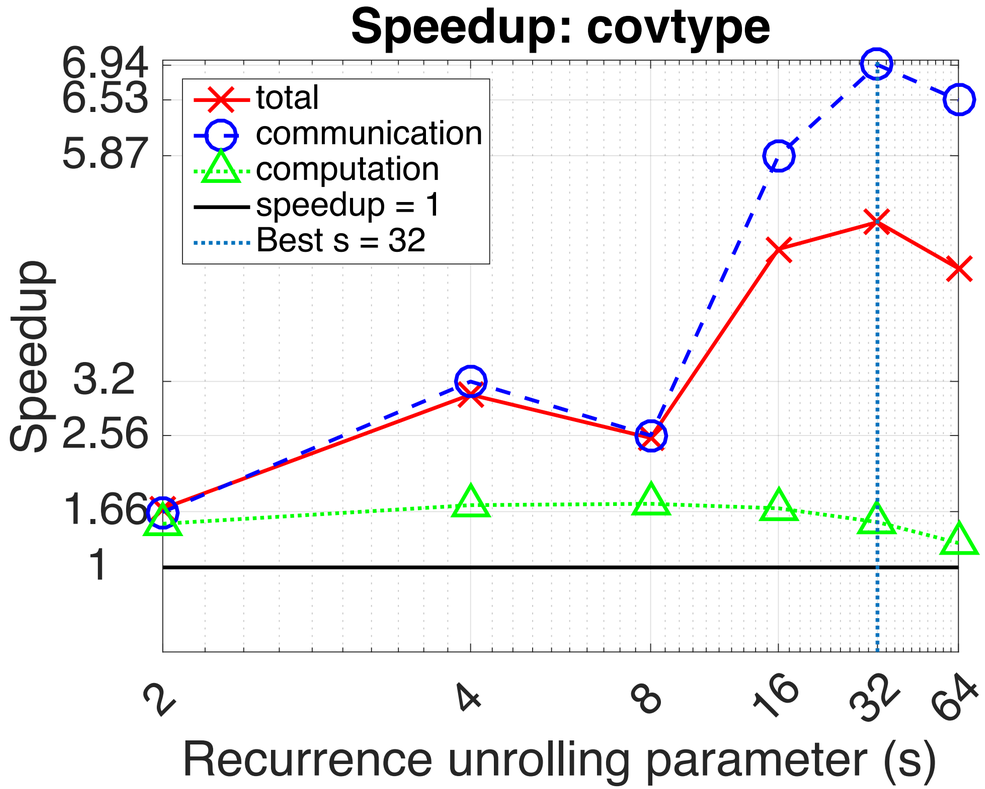




Speed up breakdown

Large communication speedup until bandwidth takes a hit
Computation is maintained due to local cache-efficient (BLAS-3) computations





Variational perspective on local graph clustering

“An optimization approach to locally-biased graph algorithms”, K. Fountoulakis, D. Gleich, M. Mahoney Proceedings IEEE, 2016

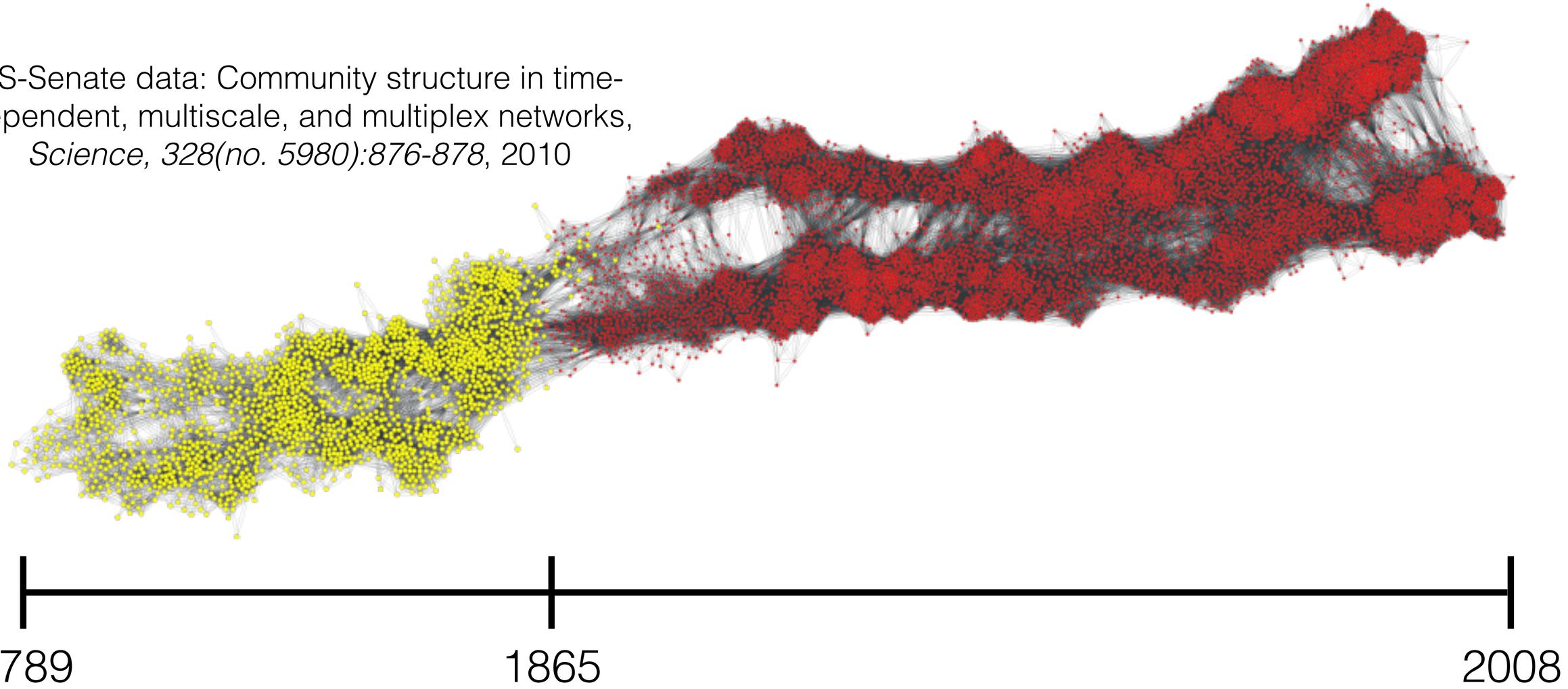
“Variational perspective on local graph clustering”, K.Fountoulakis, F. Khorasani, J. Shun, X. Cheng, M. Mahoney, Math. Prog. B., 2017

“Capacity Releasing Diffusion for Speed and Locality”, D. Wang, K. Fountoulakis, M. Mahoney, S. Rao, ICML 2017

“Parallel Local Graph Clustering”, J. Shun, K. Fountoulakis, F. Khorasani, M. Mahoney, VLDB, 2016

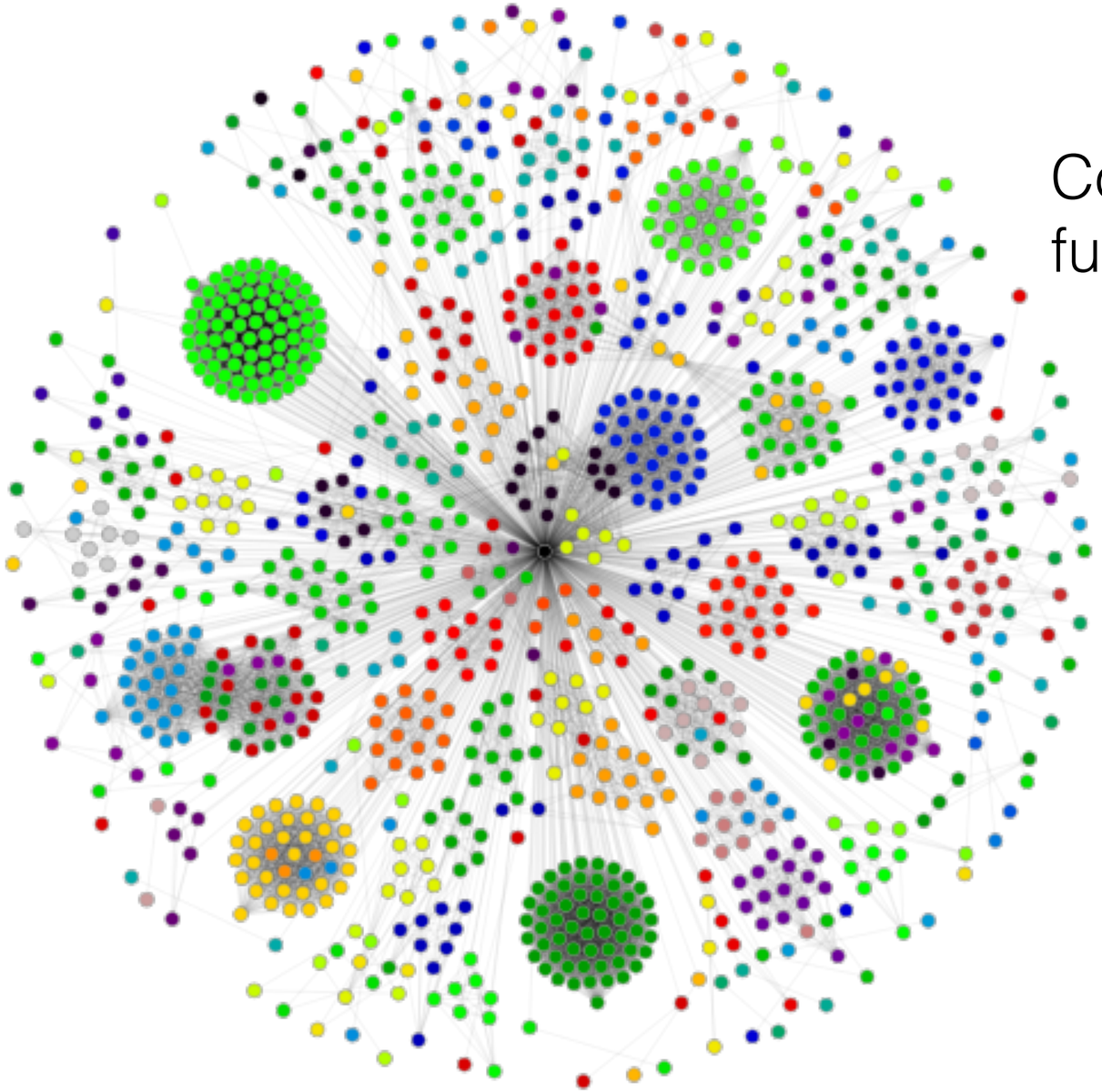
Past and present studies focus on **global** trends of the data

US-Senate data: Community structure in time-dependent, multiscale, and multiplex networks, *Science, 328(no. 5980):876-878*, 2010



The American civil war ended in 1865

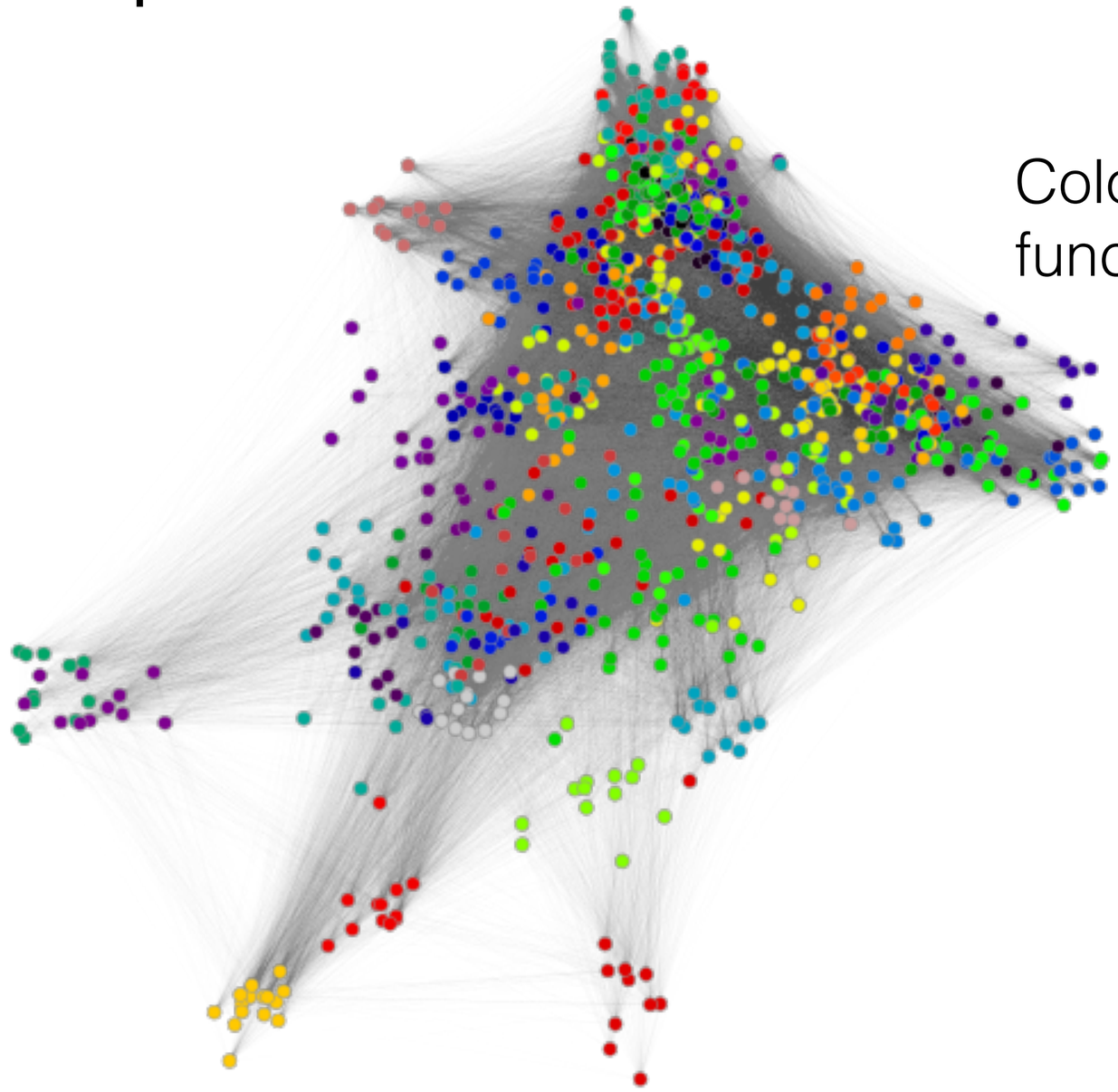
But, most real data have rich **local** structure



Color denotes similar function

Data: The MIPS mammalian protein-protein interaction database. *Bioinformatics, 21(6):832-834*, 2005

And can be very complex



Color denotes similar function

Data: The MIPS mammalian protein-protein interaction database. *Bioinformatics, 21(6):832-834*, 2005

Outline

1. Local graph clustering, definition, examples
2. Example of a state-of-the-art method
3. Variational model
4. Proximal gradient descent

What is local graph clustering and why is it useful?

-Definition: find set of nodes A given a seed node in set B

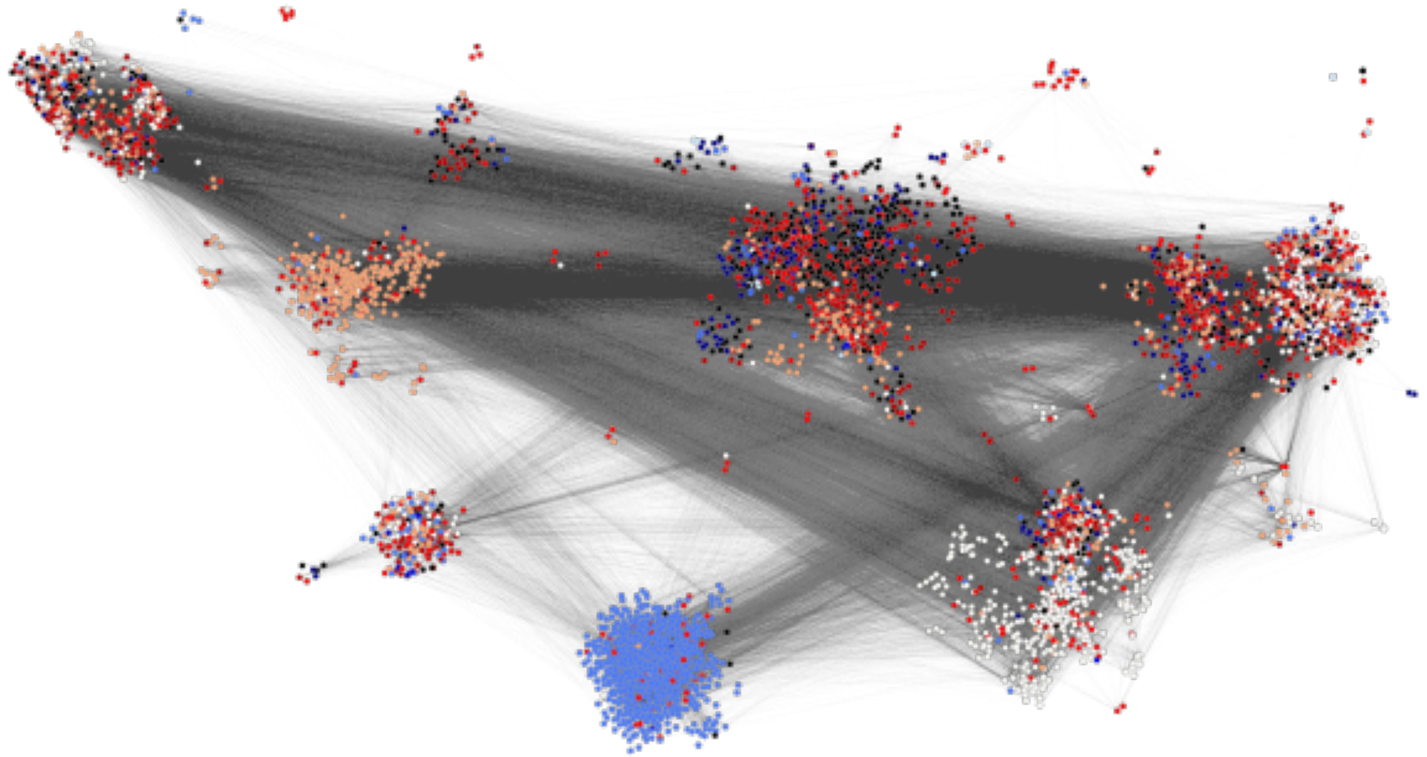
  -Set A has good precision/recall w.r.t set B

  -The running time depends on A instead of the whole graph

-Scalable to graphs with billions of edges

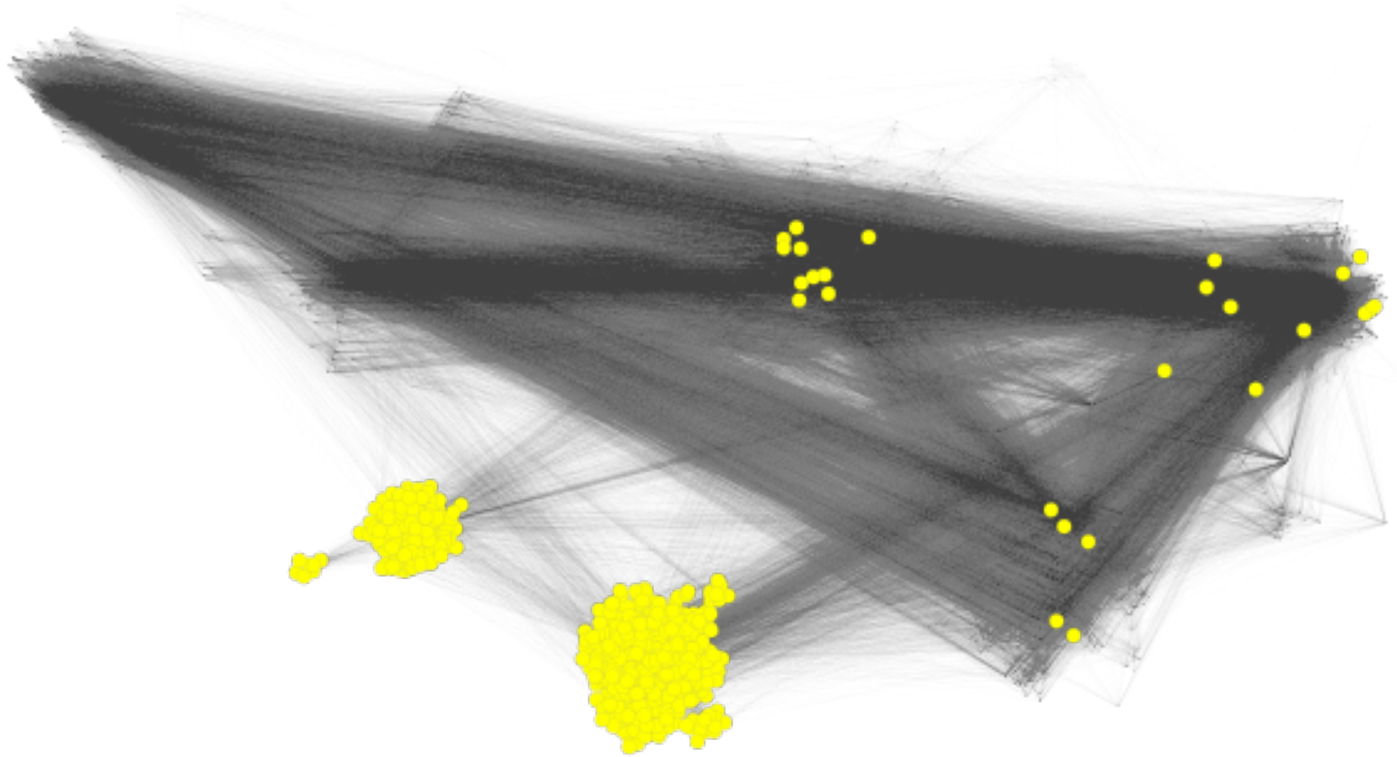
-Ideal for finding small clusters and small neighborhoods

Facebook social network: colour denotes class year



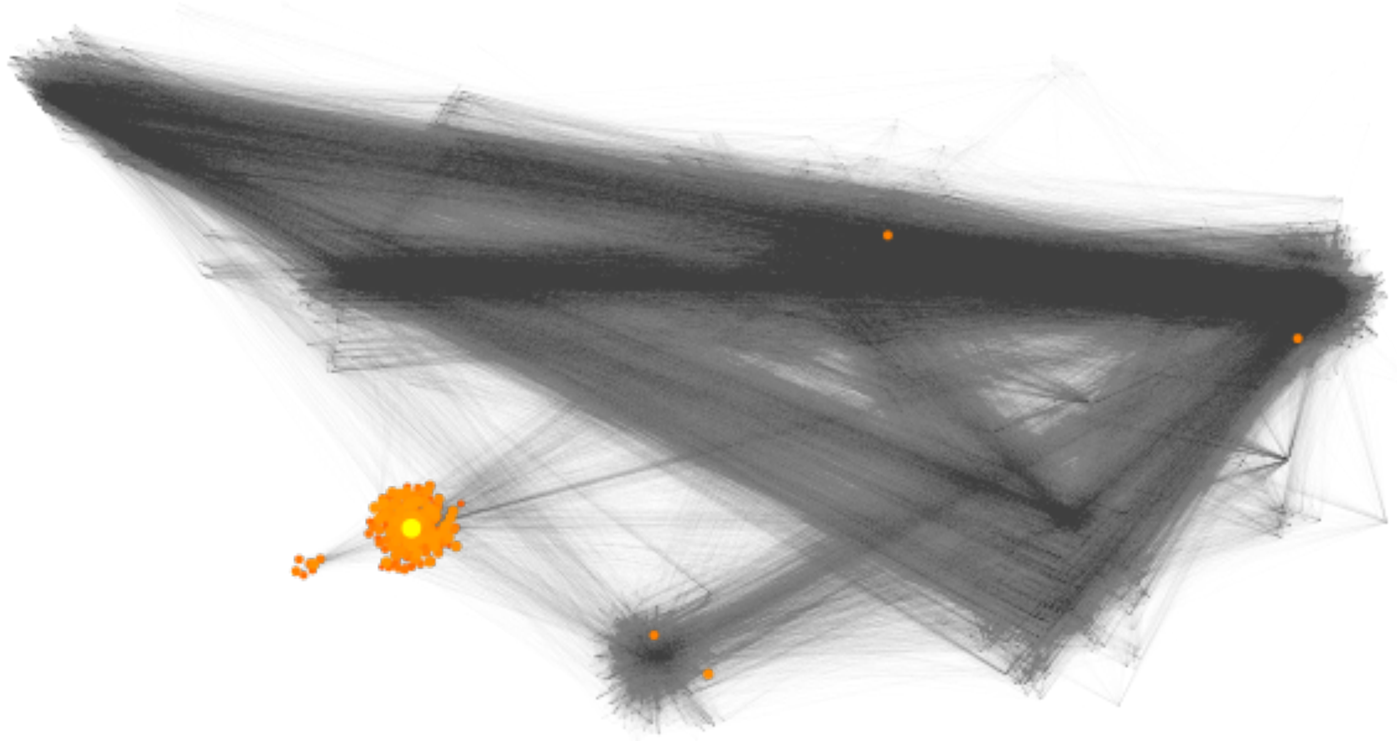
Data: Facebook John Hopkins, A. L. Traud, P. J. Mucha and M. A. Porter, Physica A, 391(16), 2012

Global spectral: finds 20% of the graph



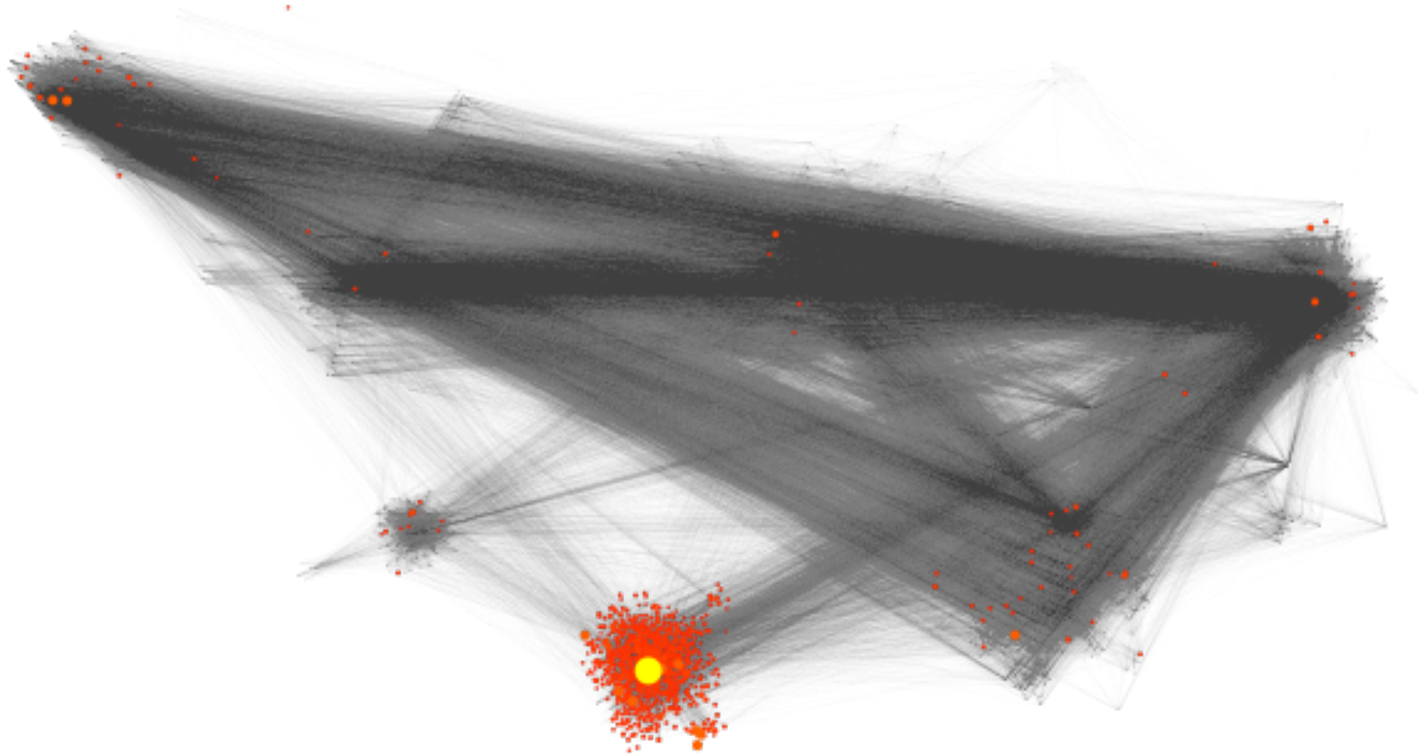
Data: Facebook John Hopkins, A. L. Traud, P. J. Mucha and M. A. Porter, Physica A, 391(16), 2012

Local graph clustering: finds 3% of the graph



Data: Facebook John Hopkins, A. L. Traud, P. J. Mucha and M. A. Porter, Physica A, 391(16), 2012

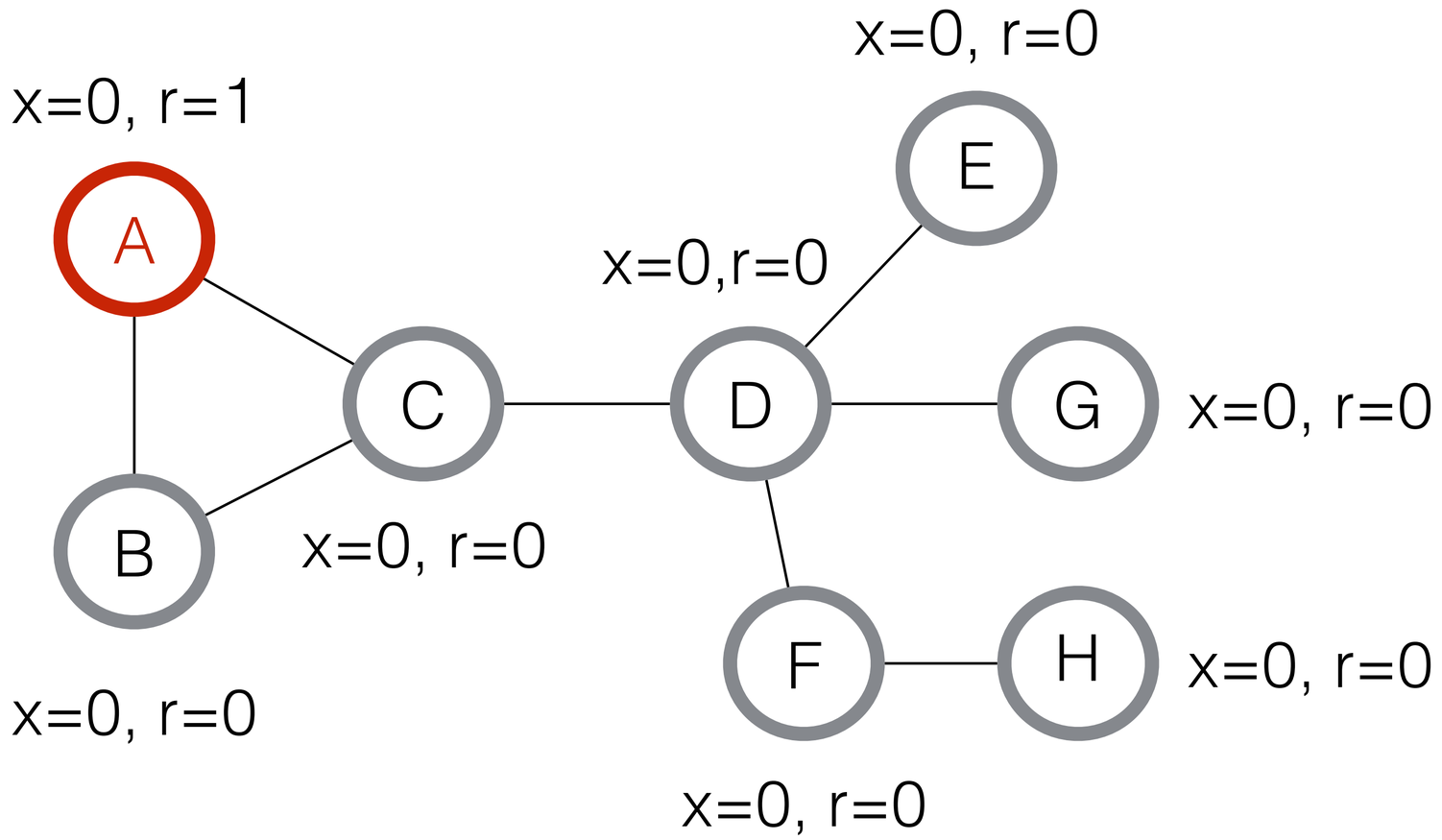
Local graph clustering: finds 17% of the graph



Data: Facebook John Hopkins, A. L. Traud, P. J. Mucha and M. A. Porter, Physica A, 391(16), 2012

Approximate Personalized PageRank

Algorithm idea: iteratively spread probability mass around the graph.

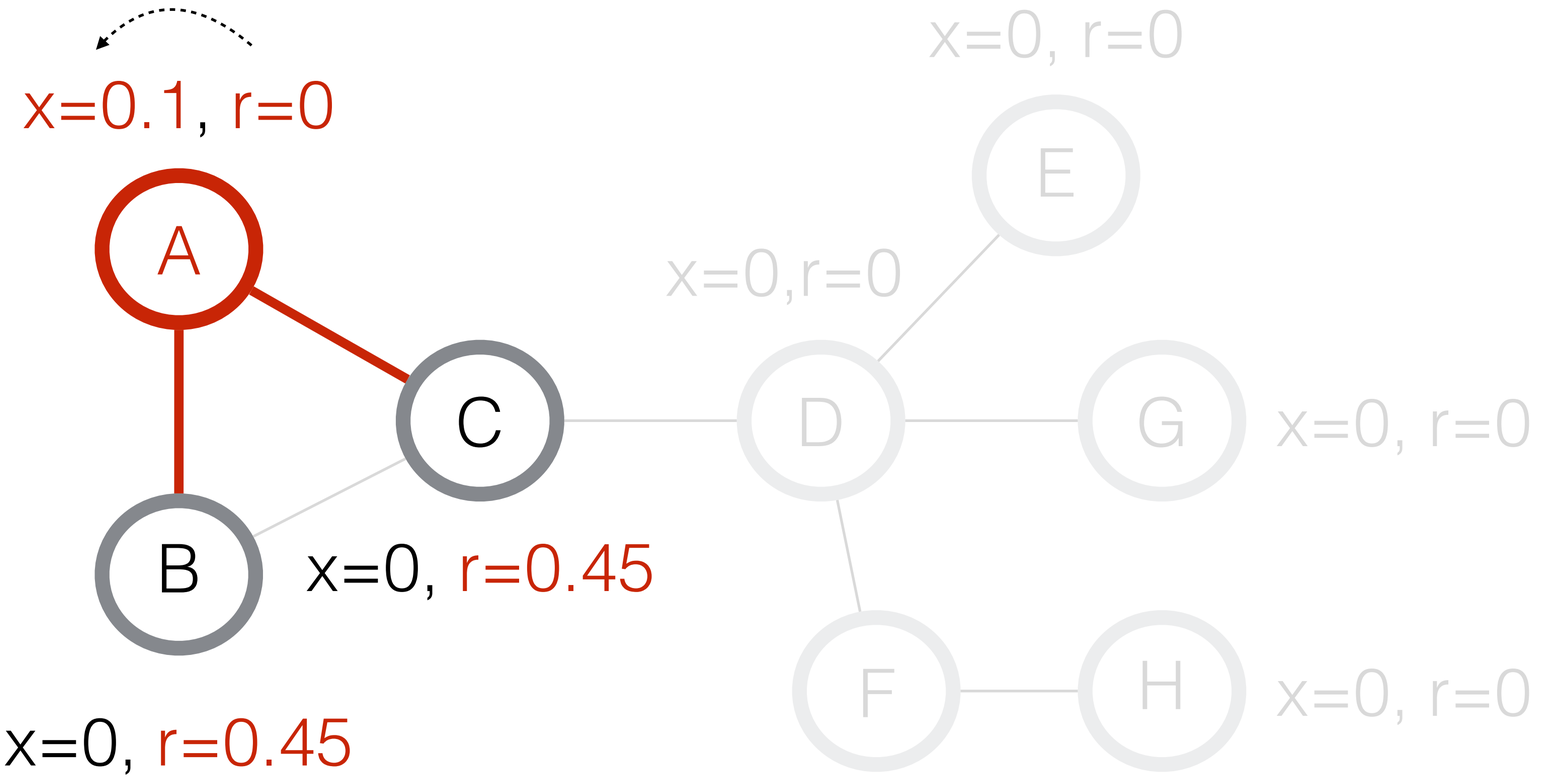


R. Andersen, F. Chung and K. Lang. Local graph partitioning using Page-Rank, FOCS, 2006

Approximate Personalized PageRank

Algorithm idea: iteratively spread probability mass around the graph.

Transfer **alpha** (10%) mass from r to x



The other 90% of the mass is distributed evenly to neighbors of A

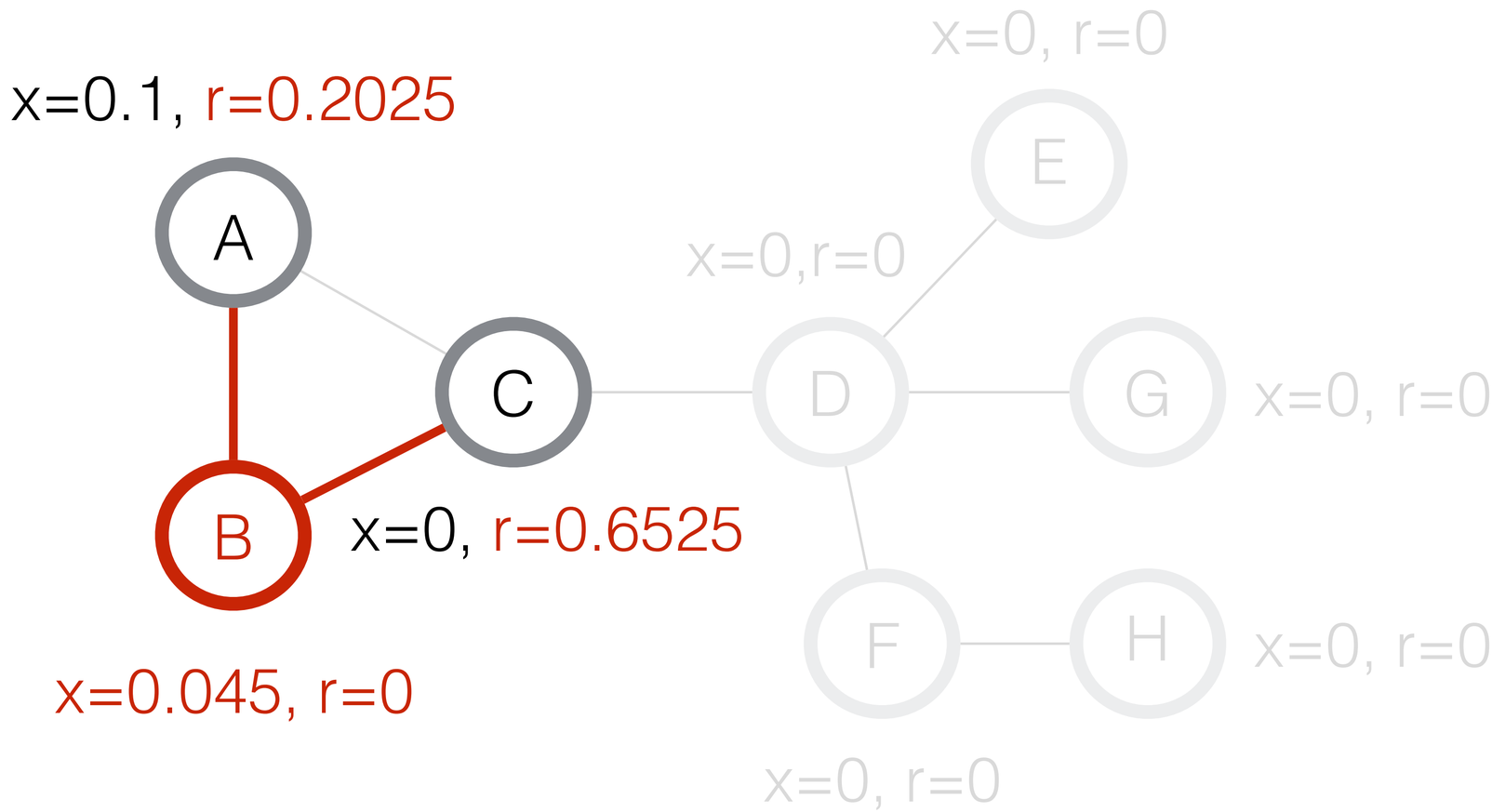
$$\alpha = 0.1$$

R. Andersen, F. Chung and K. Lang. Local graph partitioning using Page-Rank, FOCS, 2006

Approximate Personalized PageRank

Algorithm idea: iteratively spread probability mass around the graph.

$$\alpha = 0.1$$

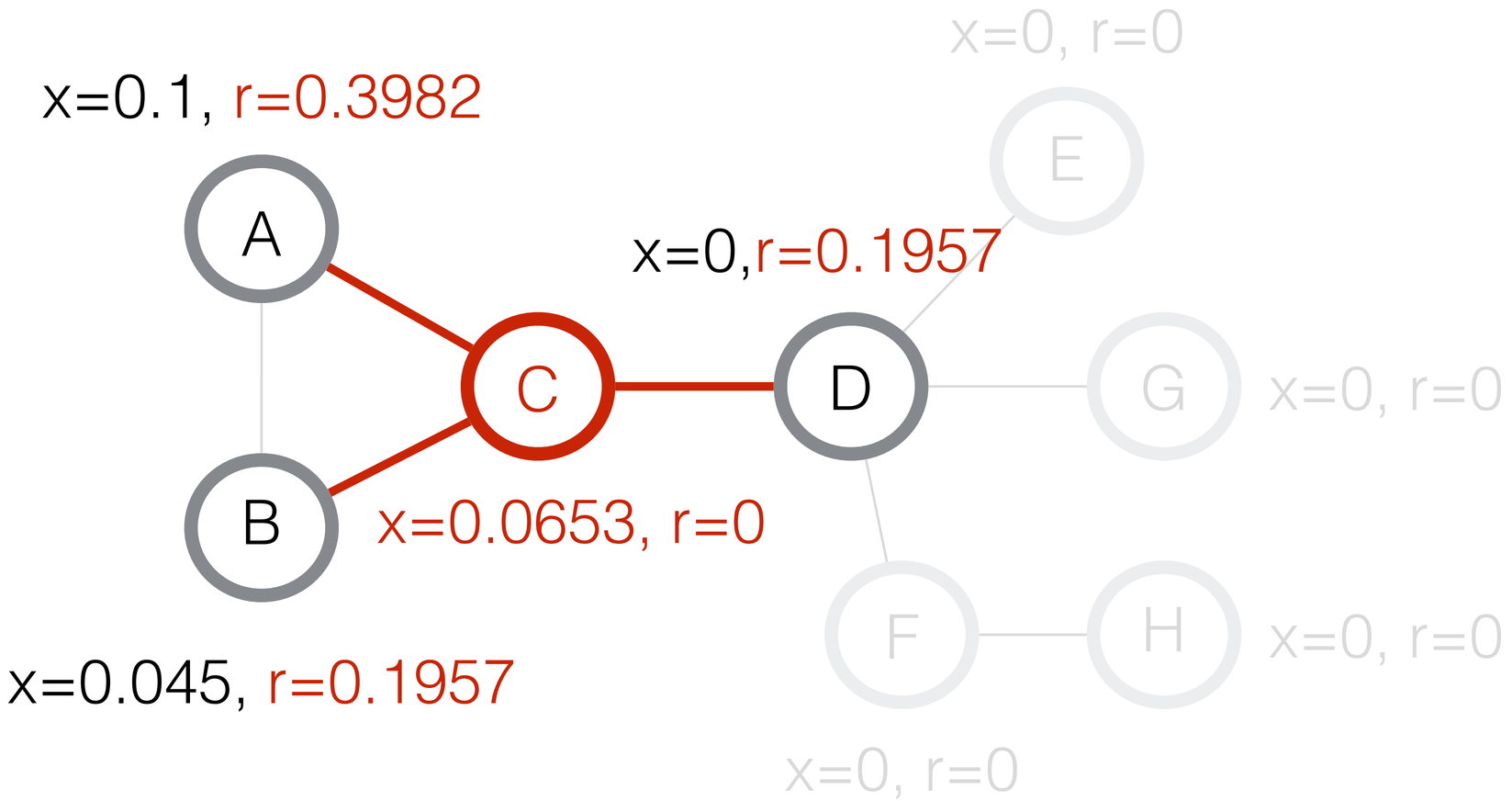


R. Andersen, F. Chung and K. Lang. Local graph partitioning using Page-Rank, FOCS, 2006

Approximate Personalized PageRank

Algorithm idea: iteratively spread probability mass around the graph.

$$\alpha = 0.1$$



R. Andersen, F. Chung and K. Lang. Local graph partitioning using Page-Rank, FOCS, 2006

Approximate Personalized PageRank

Algorithm idea: iteratively spread probability mass around the graph until

$$\max_i \frac{r_i}{d_i} \leq \rho\alpha$$

- ρ: termination parameter
- d_i: number of edges of node i

Variational model of APPR

Observation: The optimality conditions of an l1-regularized convex problem imply the termination condition of APPR.

$$\text{minimize } \frac{1-\alpha}{2}\|Bx\|_2^2 + \alpha\|H(\mathbf{1}-x)\|_2^2 + \alpha\|Zx\|_2^2 + \rho\alpha\|Dx\|_1$$

where

- B: is the incidence matrix
- D: Degree matrix
- H = diag(initial prob. dist. over nodes)
- Z = D - H
- α: teleportation parameter
- ρ: l1-reg. hyper-parameter

Termination conditions vs optimality conditions

Termination criteria of Approximate Personalized PageRank

$$\max_i \frac{r_i}{d_i} \leq \rho\alpha$$

Optimality conditions of the variational model

$$\frac{r_i}{d_i} = \rho\alpha, \;\; x_i \neq 0$$

$$\frac{r_i}{d_i} \leq \rho\alpha, \;\; x_i = 0$$

Properties of the variational problem

- **Theorem:** The volume of the optimal solution is bounded by 1/ρ

- **Theorem:** Same combinatorial theoretical guarantees for local graph clustering

- **Crucial:** The model decouples the output from the algorithm.

Proximal gradient descent for local graph clustering

$$f(x) := \frac{1-\alpha}{2}\|Bx\|_2^2 + \alpha\|H(\mathbf{1}-x)\|_2^2 + \alpha\|Zx\|_2^2 \qquad g(x) := \rho\alpha\|Dx\|_1$$

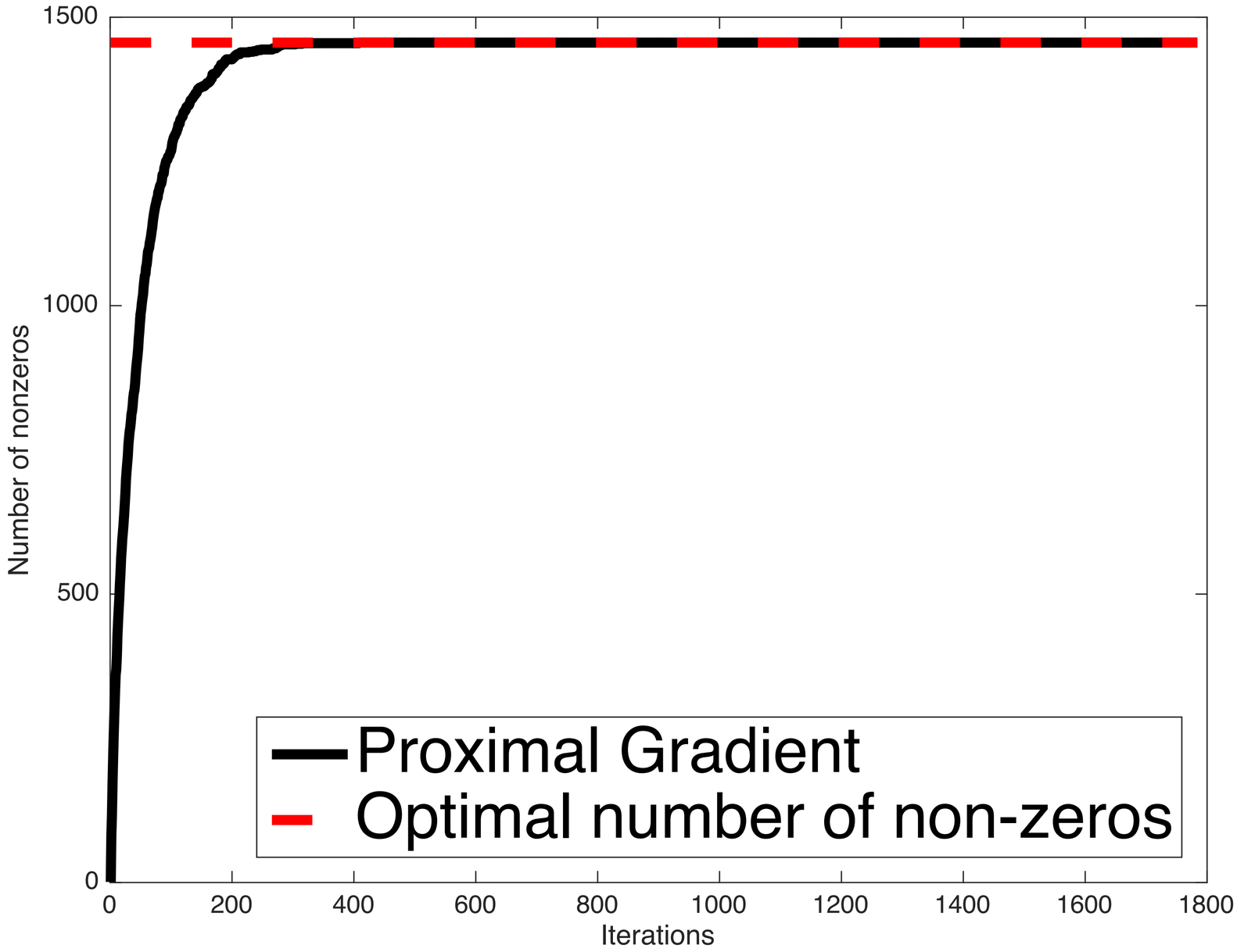
Proximal gradient descent

$$x_{k+1} := \operatorname{argmin}\ g(x) + \underbrace{f(x_k) + \langle \nabla f(x_k), x - x_k \rangle}_{\text{first-order Taylor approximation}} + \underbrace{\frac{1}{2}\|x - x_k\|_2^2}_{\substack{\text{upper bound on the} \\ \text{approximation error}}}$$

Requires careful implementation to avoid excessive running time

- Need to maintain a set of non-zero nodes
- Update x and gradient only for non-zero nodes and their neighbors at each iteration

Theorem: non-decreasing non-zero nodes



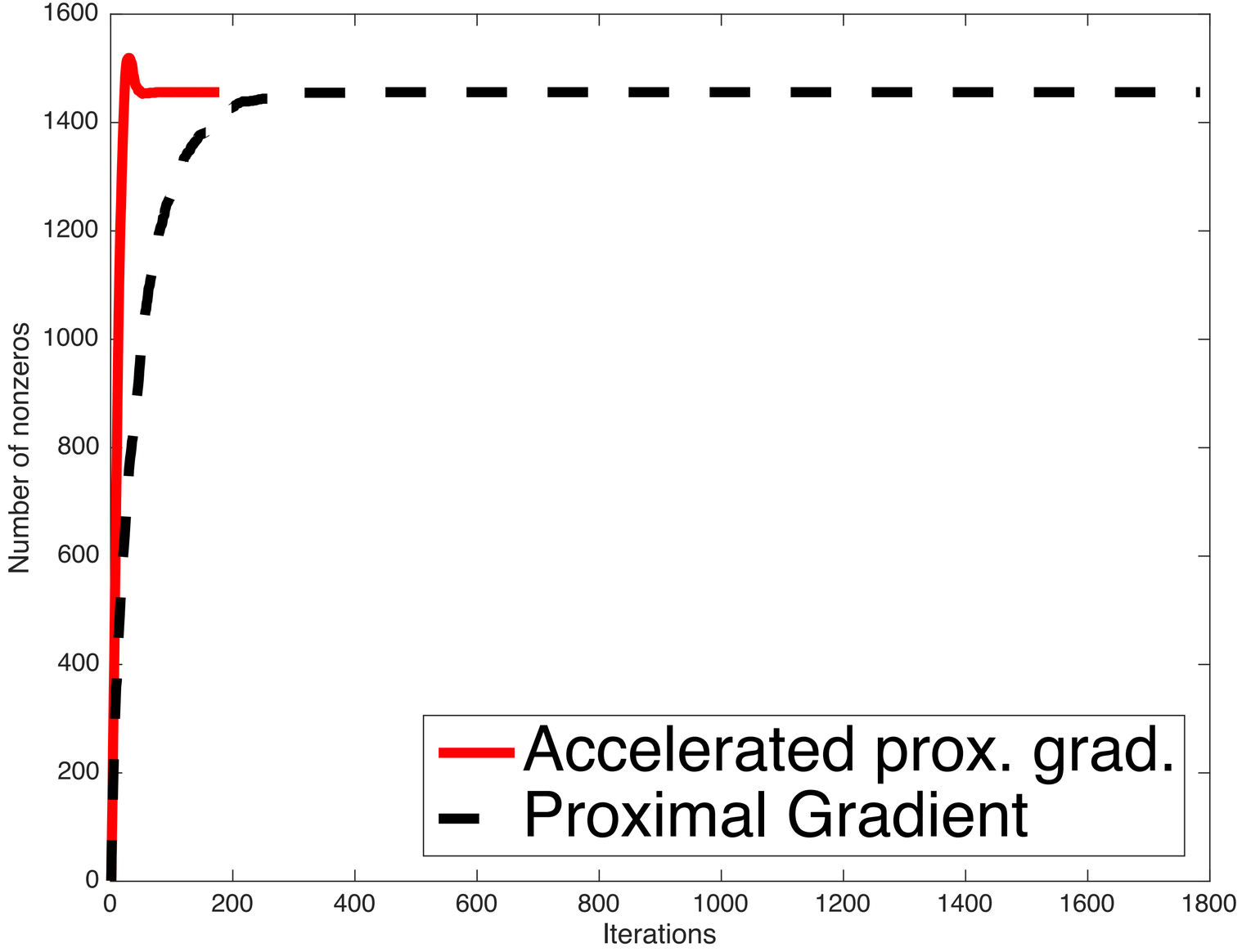
Worst-case running times

	Weighted graphs	Unweighted graphs
Prox. grad.	$\mathcal{O}\left(\frac{(\lvert\mathcal{S}_*\rvert + \widehat{\mathrm{vol}}(\mathcal{S}_*))}{\mu}\log\left(\frac{2}{\epsilon^2\rho^2\alpha^2\min_j d_j}\right)\right)$	$\mathcal{O}\left(\frac{2}{\rho\mu}\log\left(\frac{2}{\epsilon^2\rho^2\alpha^2\min_j d_j}\right)\right).$
APPR	✗	$\frac{1}{\alpha\rho}$

$$\mu := \alpha + \frac{1-\alpha}{4}\lambda_{min}(\mathcal{L}_{\mathcal{S}_*})$$

$\mathcal{L}_{\mathcal{S}_*}$: sub-matrix of normalized Laplacian

Open problem: is accelerated prox. grad. a local algorithm?



Gradient descent running time

$$\mathcal{O}\left(\frac{(|\mathcal{S}_*| + \widehat{\mathrm{vol}}(\mathcal{S}_*))}{\mu} \log\left(\frac{2}{\epsilon^2 \rho^2 \alpha^2 \min_j d_j}\right)\right)$$

Accel. gradient descent

$$\mathcal{O}\left(\frac{\mathrm{vol}(\mathcal{G})}{\sqrt{\mu}} \log\left(\frac{2}{\epsilon^2 \rho^2 \alpha^2 \min_j d_j}\right)\right)$$

$$\mathcal{O}\left(\frac{|\mathcal{S}_*| + \mathrm{vol}(\mathcal{S}_*)}{\sqrt{\mu}} \log\left(\frac{2}{\epsilon^2 \rho^2 \alpha^2 \min_j d_j}\right)\right)$$

Software

LocalGraphClustering on GitHub

- Written in Python with C++ routines when required

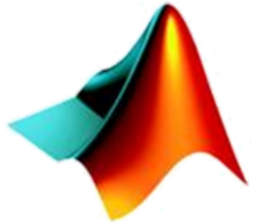
- Graph analytics on 100 million edges graph on a 16GB RAM laptop

- Demonstrations on social and bioinformatics networks

- 8 Python notebooks with numerous examples and graph visualizations

- Video presentations

- 12 methods and pipelines



MATLAB

- **pdNCG**: primal-dual Newton Conjugate Gradients

- **MFIPMCS**: Matrix-free Interior Point Method for Compressed Sensing

- **Trillion**: Scalable instance generator for l1-regularized least-square problems

- **FCD**: Flexible Coordinate Descent

* Can be found at Edinburgh Research Group in Optimization repo and my personal website

Thank you!

Other work during my PhD

“A flexible coordinate descent method”, K. Fountoulakis, R. Tappenden, Computational Optimization and Applications, 2018

“Performance of first- and second-order methods for l1-regularized least squares problems”, K. Fountoulakis, J. Gondzio, Computational Optimization and Applications, 2016

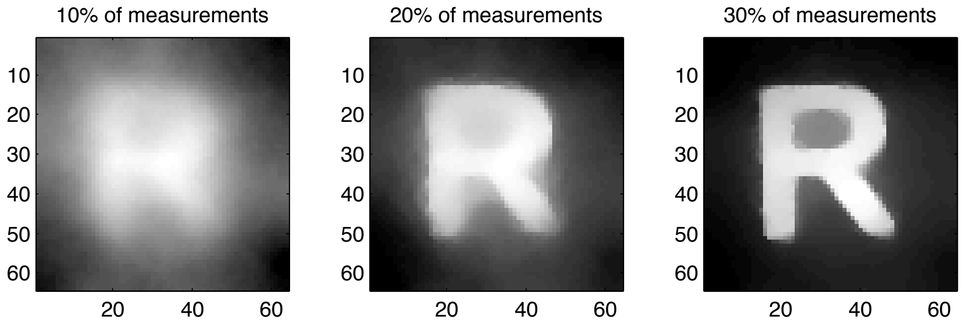
“A second-order method for strongly-convex l1-regularization”, K. Fountoulakis, J. Gondzio, Mathematical Programming, 2016

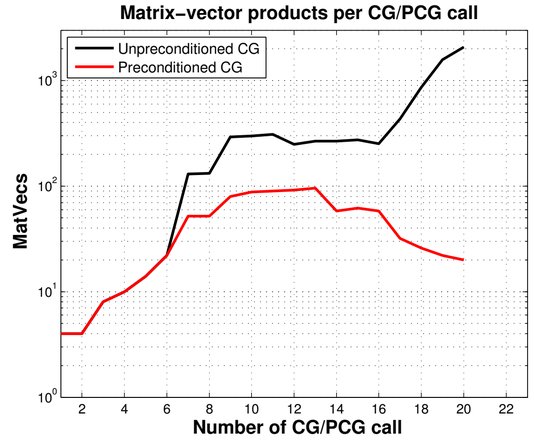
“A preconditioner for a primal-dual Newton conjugate gradients method for compressed sensing problems”, I. Dassios, K. Fountoulakis, J. Gondzio, Mathematical Programming, 2015

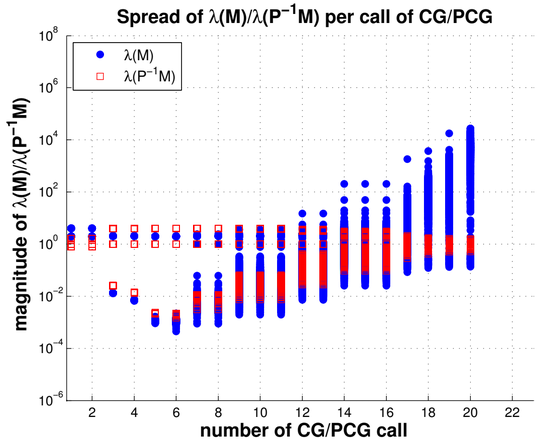
“Matrix-free interior point method for compressed sensing problems”, K. Fountoulakis, J. Gondzio, P. Zhlobich, Mathematical Programming Computation, 2014

Newton-type methods for image processing

$$\text{minimize } \lambda\|\nabla x\|_1 + \frac{1}{2}\|Ax - b\|_2^2$$







Parallel local graph clustering methods in shared memory

- Why shared memory? Currently the largest publicly available graphs can be stored in computers with shared memory

- We parallelize 4 local spectral methods + rounding
  1. Approximate PageRank (as demonstrated in previous slides)
  2. Nibble
  3. Deterministic HeatKernel Approximate PageRank
  4. Randomized HeatKernel Approximate PageRank
  5. Sweep cut rounding algorithm

Based on

Parallel Local Graph Clustering, J. Shun, K. Fountoulakis, F. Khorasani, M. Mahoney, VLDB, 2016

Overview of results

-3-16x faster than serial version

-Parallelization allowed us to solve problems of billions of nodes and edges.

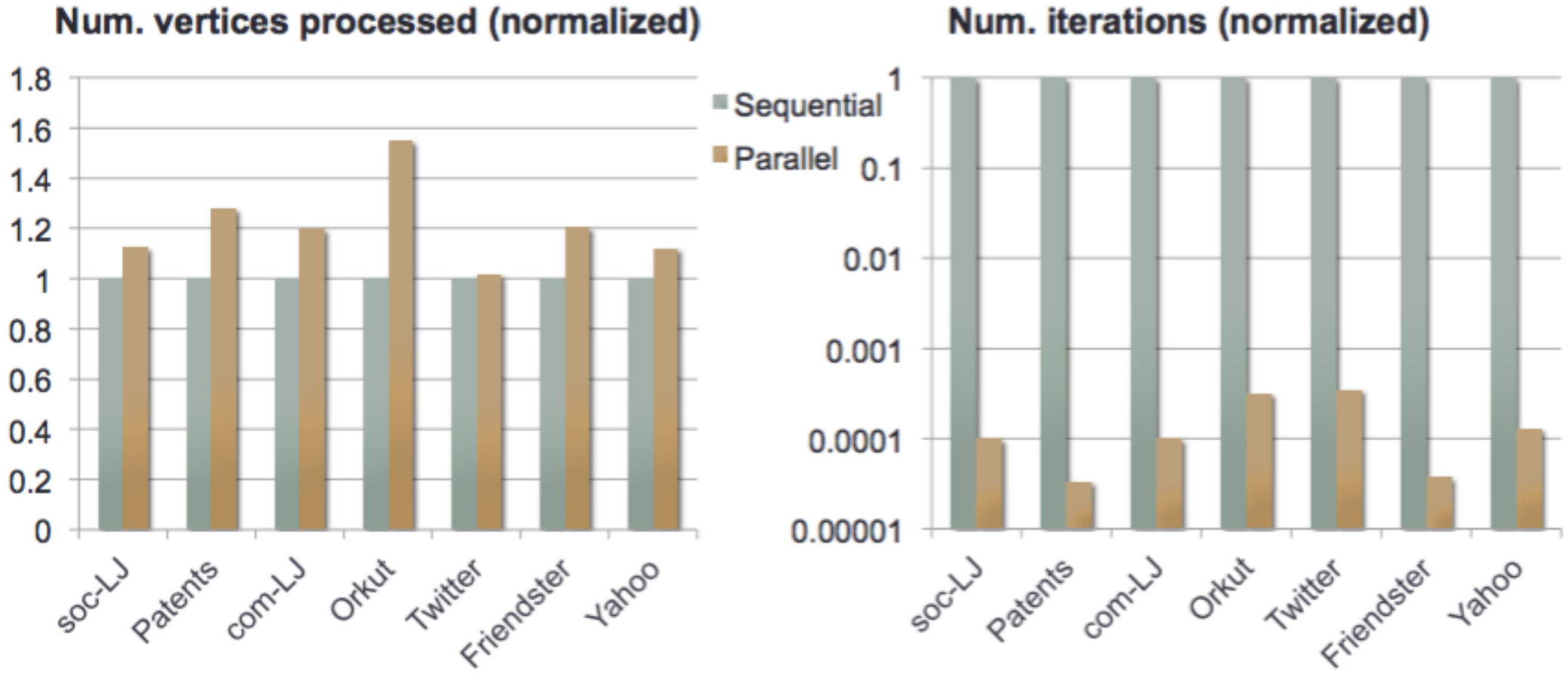
An example: Parallel Approximate Personalized PageRank

- Serial version: picks a node from candidates to distribute mass to its neighbors
- Parallel: pick **all** nodes from candidates and distribute mass simultaneously
- Asymptotic work (FLOPS) remains the same
  - But work is parallelized
  - We pay a small communication cost among cores

Data

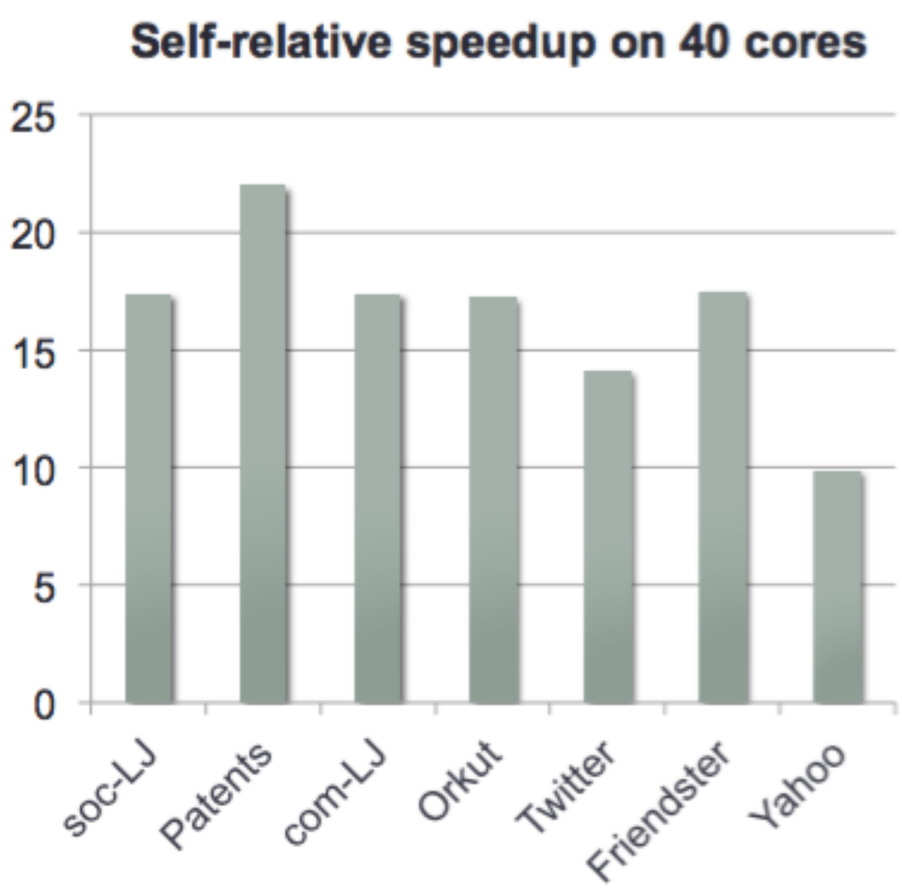
Input graph	Num. vertices	Num. edges
soc-JL	4,847,571	42,851,237
cit-Patents	6,009,555	16,518,947
com-LJ	4,036,538	34,681,189
com-Orkut	3,072,627	117,185,083
Twitter	41,652,231	1,202,513,046
Friendster	124,836,180	1,806,607,135
Yahoo	1,413,511,391	6,434,561,035

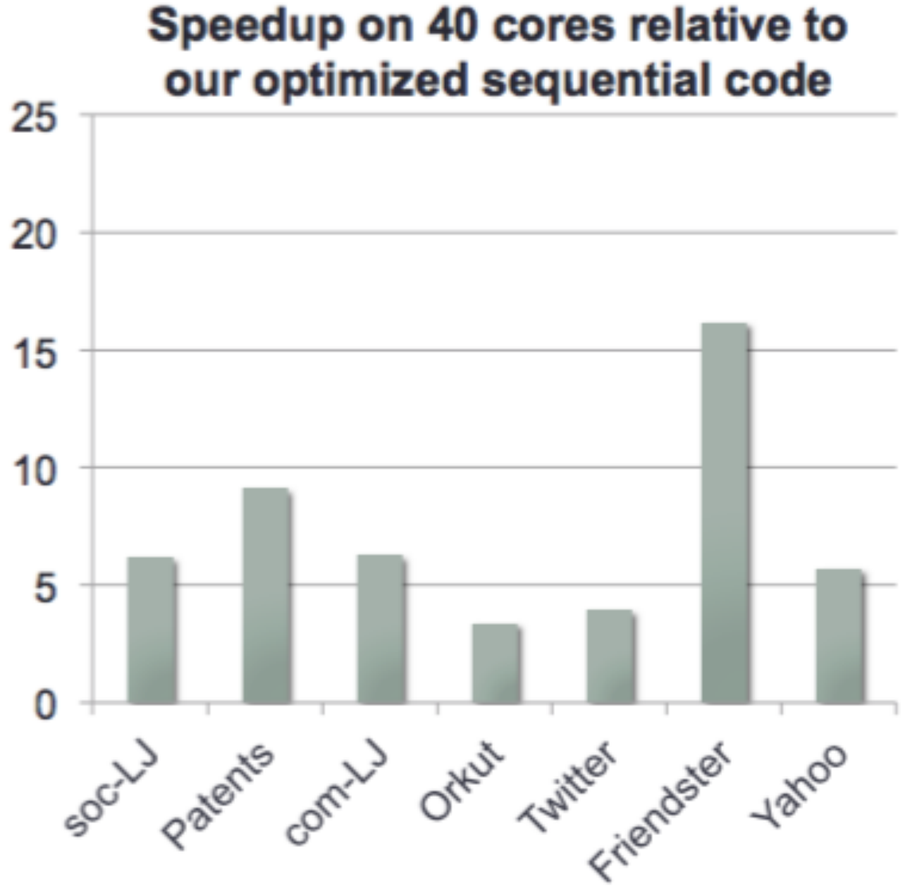
Performance



- Slightly more work for the parallel version
- Number of iterations is significantly less

Performance





- 3-16x speed up
- Speedup is limited by small active set in some iterations and memory effects

Future directions: short term

- DARPA D3M (and HIVE) funds the project on graph analytics.

- Expand variational perspective to flow methods

- Prove that accelerated gradient descent has local running time

- Speed up existing pipelines using local graph clustering

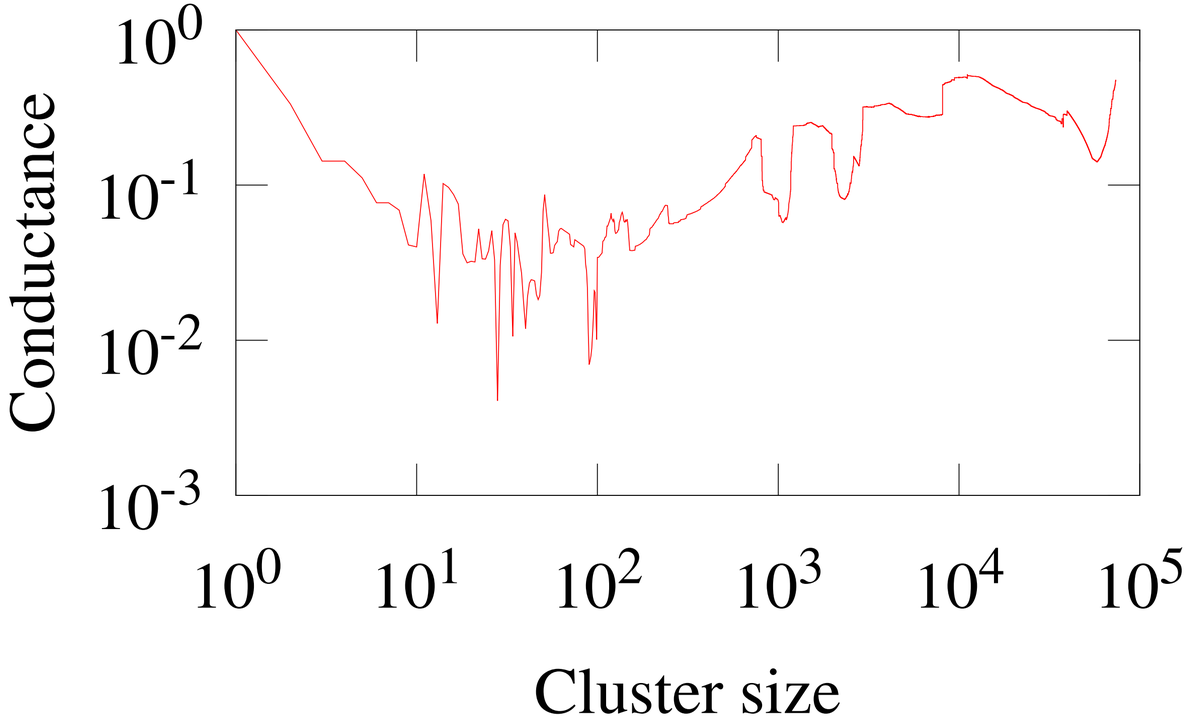
- A tutorial on global and local graph methods

Future directions: long term

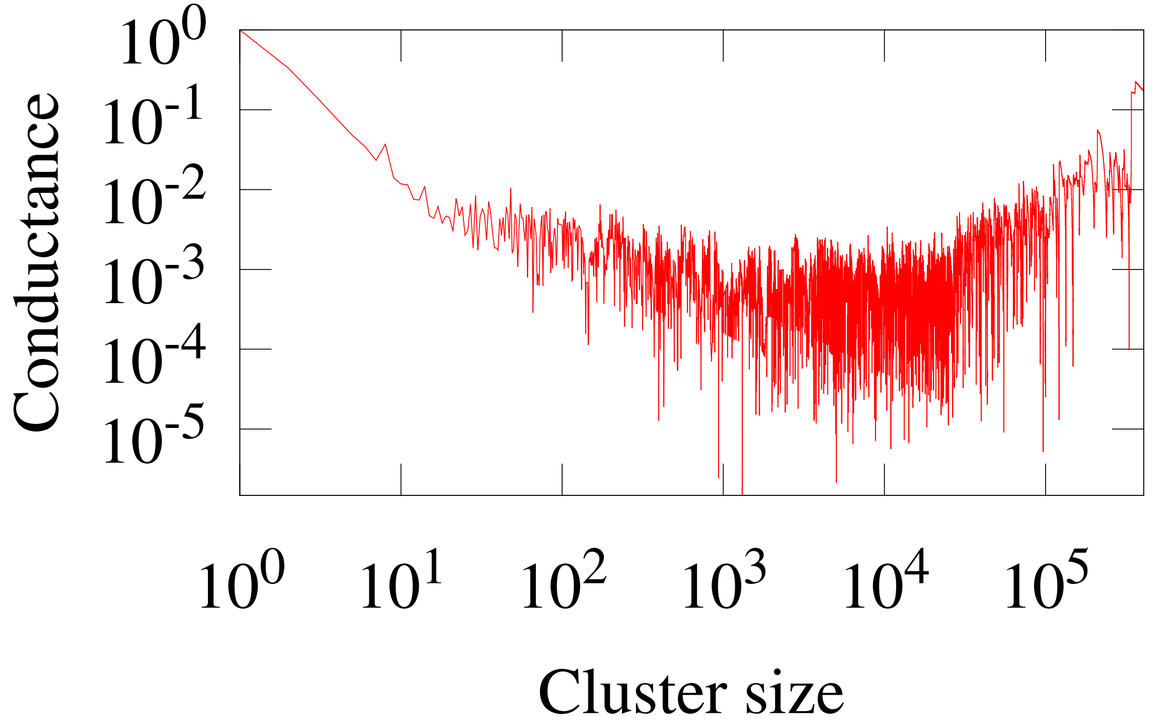
- **My objective:** make local analysis of data useful for downstream applications scientists
- Target major applications, social network analysis, bioinformatics and medical sciences
- Target data-science and machine learning venues, i.e., KDD, ICML, NIPS
- Make the software as easy as possible for the 99% case

Graph analytics on graphs with more than a billion nodes

Friendster, 124M nodes, 1.8B edges

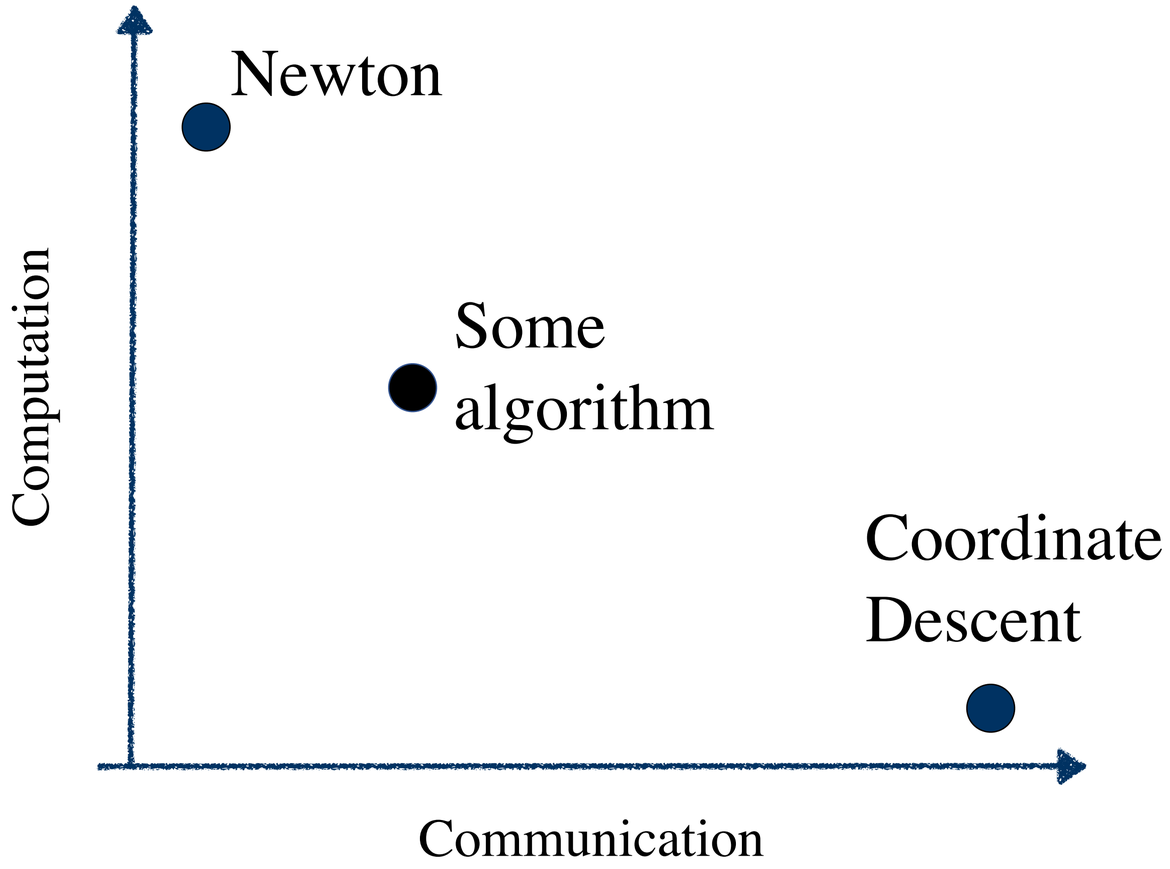


Yahoo, 1.4B nodes, 6.4B edges



- $O(10^5)$ approximate PPR problems were solved in parallel for each plot,
- Agrees with conclusions of [Leskovec et al. 2008], i.e., good clusters tend to be small.

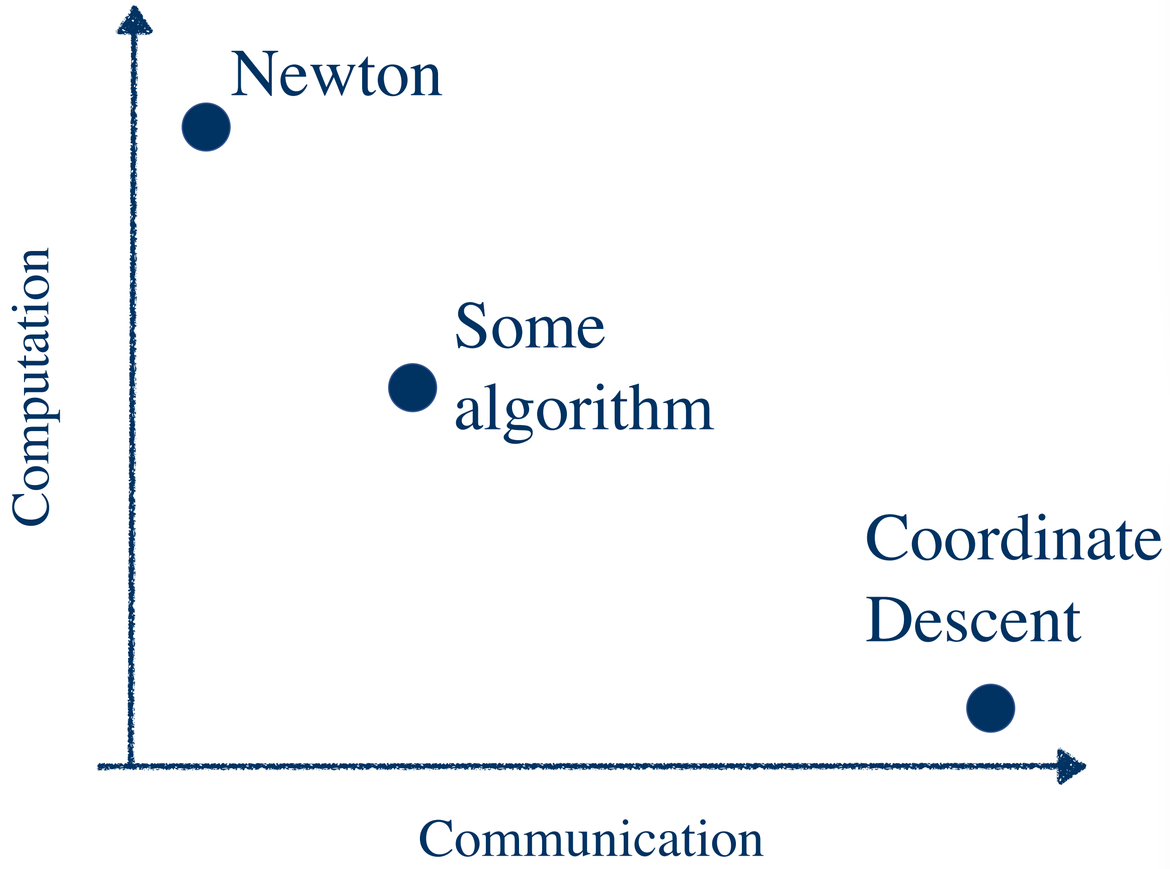
Trade-offs and existing approaches



Current approach:

choose an algorithm based on computation and communication trade-off

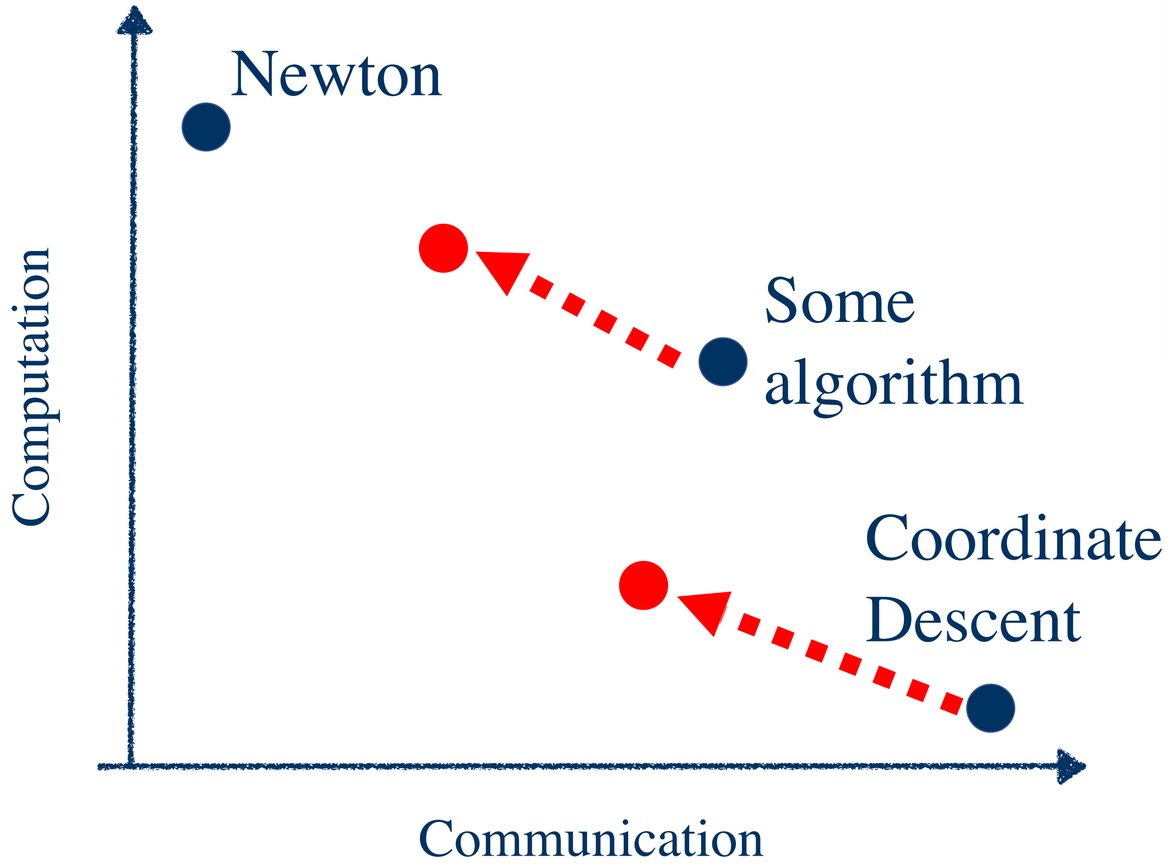
Trade-offs and existing approaches



What happens if there is no algorithm with the required trade-off?

We need to wait until a mathematician comes up with a solution

Our approach



Take existing algorithms and make them communication avoiding

Outline of the approach and results

Choose your favorite algorithm

Re-organize it to make it communication avoiding

=

Load balanced processors

Scalability to 1000+ of processors or more

Algorithm 1 Block Coordinate Descent (BCD) Algorithm

Input: $X \in \mathbb{R}^{d\times n}, y \in \mathbb{R}^n$, $H > 1$, $w_0 \in \mathbb{R}^d$, $b \in \mathbb{Z}_+$ s.t. $b \leq d$
for $h = 1, 2, \cdots, H$ **do**
  choose $\{i_m \in [d] | m = 1, 2, \ldots, b\}$ uniformly at random without replacement
  $\mathbb{I}_h = [e_{i_1}, e_{i_2}, \cdots, e_{i_b}]$
  $\Gamma_h = \frac{1}{n}\mathbb{I}_h^T X X^T \mathbb{I}_h + \lambda \mathbb{I}_h^T \mathbb{I}_h$
  $\Delta w_h = \Gamma_h^{-1}\left(-\lambda \mathbb{I}_h^T w_{h-1} - \frac{1}{n}\mathbb{I}_h^T X z_{h-1} + \frac{1}{n}\mathbb{I}_h^T X y\right)$
  $w_h = w_{h-1} + \mathbb{I}_h \Delta w_h$
  $z_h = z_{h-1} + X^T \mathbb{I}_h \Delta w_h$
Output w_H

Communication for parallel computation of Gram matrix (every iteration)

Communication for parallel computation of all required inner products (every outer iteration)

No communication for inner loop

Algorithm 2 Communication-Avoiding Block Coordinate Descent (CA-BCD) Algorithm

Input: $X \in \mathbb{R}^{d\times n}, y \in \mathbb{R}^n$, $H > 1$, $w_0 \in \mathbb{R}^d$, $b \in \mathbb{Z}_+$ s.t. $b \leq d$
for $k = 0, 1, \cdots, \frac{H}{s}$ **do**
  for $j = 1, 2, \cdots, s$ **do**
    choose $\{i_m \in [d] | m = 1, 2, \ldots, b\}$ uniformly at random without replacement
    $\mathbb{I}_{sk+j} = [e_{i_1}, e_{i_2}, \cdots, e_{i_b}]$
  let $Y = \left[\mathbb{I}_{sk+1}, \mathbb{I}_{sk+2}, \cdots, \mathbb{I}_{sk+s}\right]^T X$.
  compute the Gram matrix, $G = \frac{1}{n} Y Y^T + \lambda I$.
  for $j = 1, 2, \cdots, s$ **do**
    Γ_{sk+j} are the $b \times b$ diagonal blocks of G.
    $$\Delta w_{sk+j} = \Gamma_{sk+j}^{-1}\left(-\lambda \mathbb{I}_{sk+j}^T w_{sk} - \lambda \sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T \mathbb{I}_{sk+t} \Delta w_{sk+t}\right) - \frac{1}{n}\mathbb{I}_{sk+j}^T X z_{sk} - \frac{1}{n}\sum_{t=1}^{j-1}\left(\mathbb{I}_{sk+j}^T X X^T \mathbb{I}_{sk+t} \Delta w_{sk+t}\right) + \frac{1}{n}\mathbb{I}_{sk+j}^T X y\right)$$
    $w_{sk+j} = w_{sk+j-1} + \mathbb{I}_{sk+j} \Delta w_{sk+j}$
    $z_{sk+j} = z_{sk+j-1} + X^T \mathbb{I}_{sk+j} \Delta w_{sk+j}$
Output w_H

Why communication matters

Definition and architectural trends

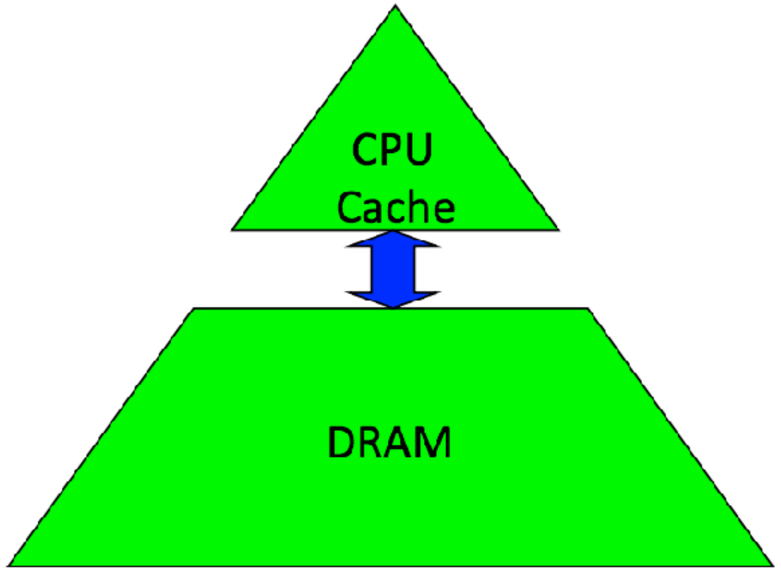
Modelling communication

Useful communication primitives

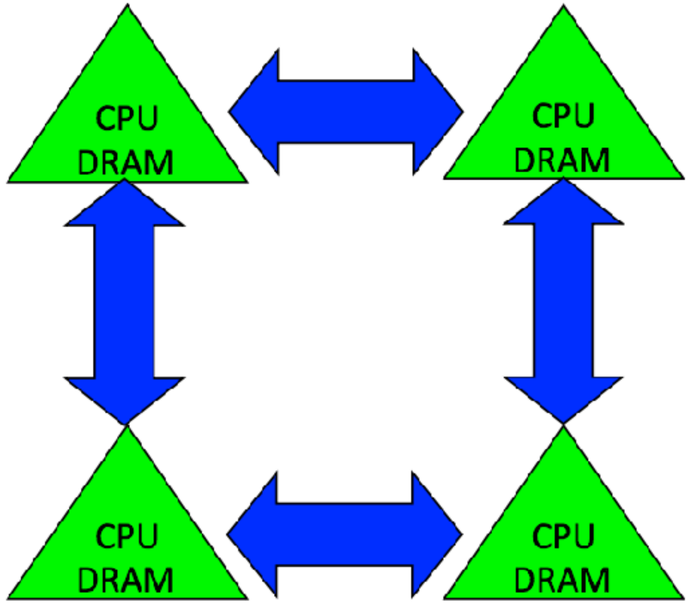
Definition

Communication is data movement.

Sequential



Parallel (Distributed-Memory)



Courtesy:
Schwartz

Architectural trends

Processor speed << Communication speed

Gap is growing

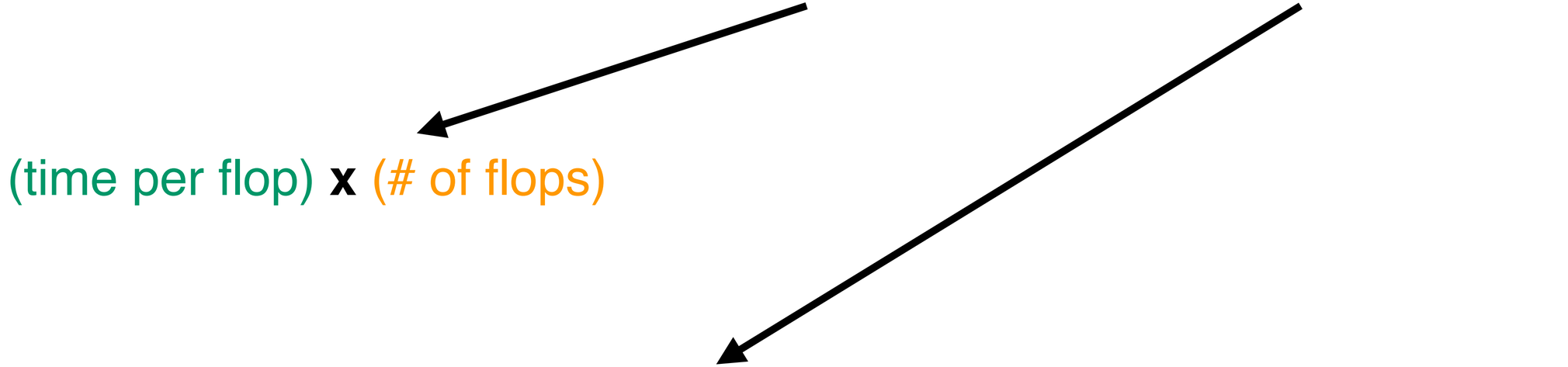
Need for faster optimization algorithms with less communication requirements

Modelling communication

Hardware Parameters

Algorithm Parameters

Running Time = Computation time + Communication time



(time per flop) **x** (# of flops)

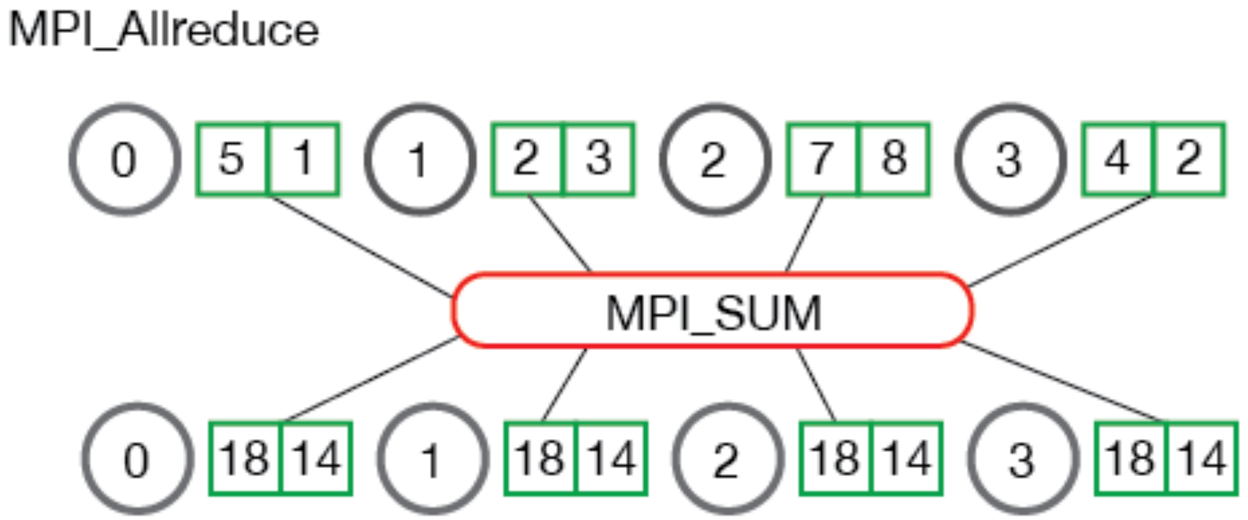
(time per word) **x** (# of words) **+** (time per message) **x** (# of messages)

Bandwidth of algorithm

Latency (or start-up time) of hardware

Communication primitives

Array of size **n** with **P** processors.



Courtesy: Kendall

Cost of one MPI_Allreduce

Number of words	**Number of Messages**
O(log P)	**O(log P)**

Thakur, et. al. 2005

An example: coordinate descent

Pseudo-code

$A_i^T \text{residual}$

- Sample a column of data

- Compute partial derivative

- Update solution

- Repeat

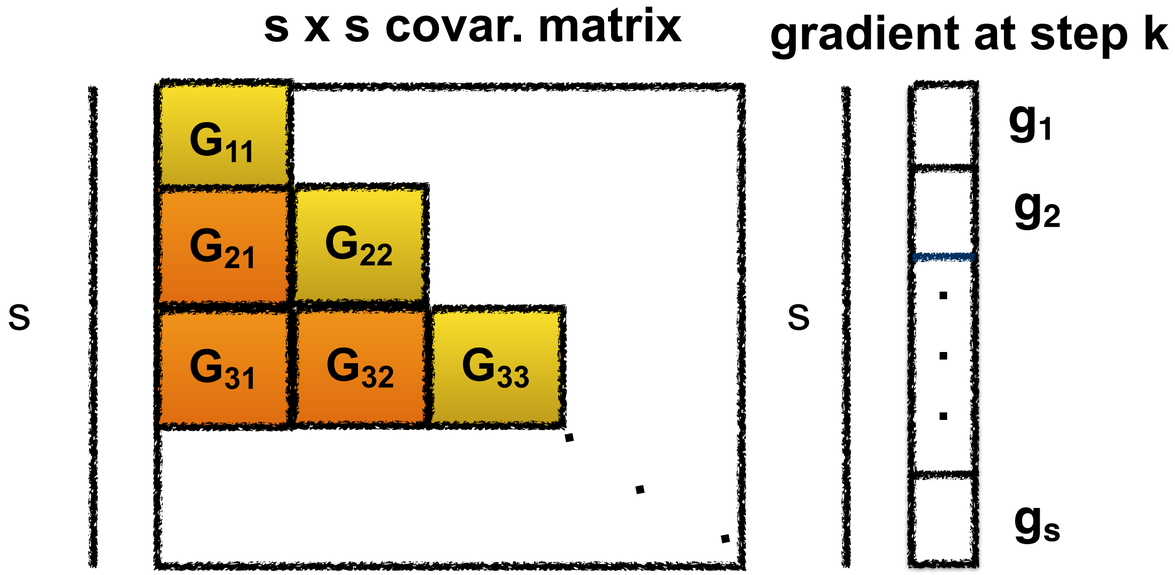
How all these are related to numerical optimization?

- **First-order** methods are often preferred in the serial setting due to fast convergence to low accuracy solutions

- **However,** in the parallel setting first-order methods can be communication bound depending on the number of cores and the network architecture

- Each iteration is usually very inexpensive but the algorithm needs many iteration to converge and each iteration requires a communication round

Avoid start-up costs by viewing the algorithm as a recurrence



Compute in parallel with one communication round all quantities that are needed for the next "s" iterations

Compute "s" directions

Redundantly on all processors

No communication required