UT DA

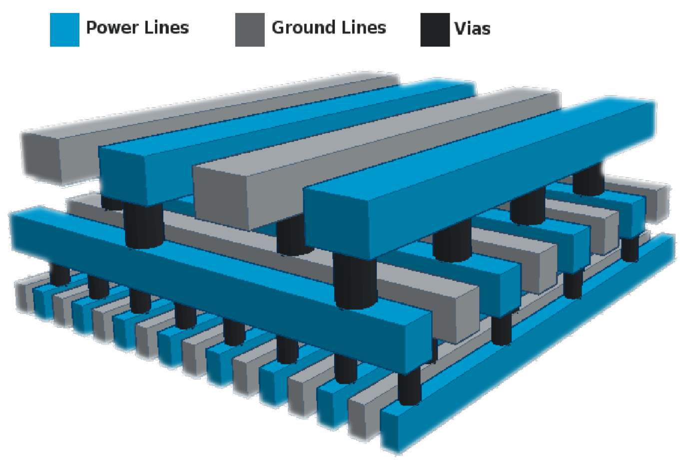
Power Grid Reduction by Sparse Convex Optimization

Wei Ye[1], Meng Li[1], Kai Zhong[2], Bei Yu[3], David Z. Pan[1]

[1]ECE Department, University of Texas at Austin
[2]ICES, University of Texas at Austin
[3]CSE Department, Chinese University of Hong Kong

On-chip Power Delivery Network

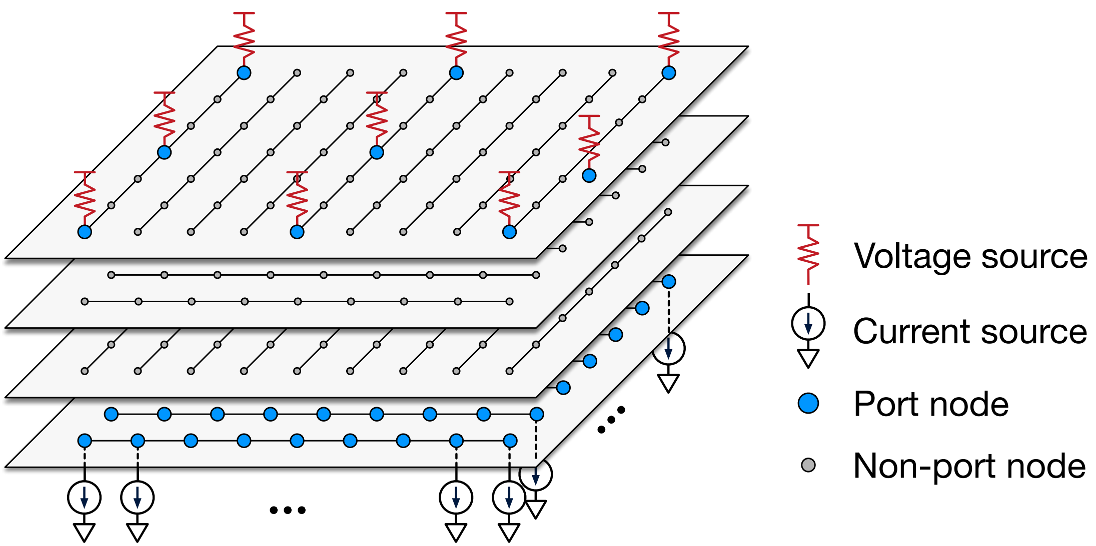
- Power grid
  - Multi-layer mesh structure
  - Supply power for on-chip devices
- Power grid verification
  - Verify current density in metal wires (EM)
  - Verify **voltage drop** on the grids
  - More **expensive** due to increasing sizes of grids
    - e.g., 10M nodes, >3 days



[Yassine+, ICCAD'16]

Modeling Power Grid

- Circuit modeling
  - Resistors to represent metal wires/vias
  - Current sources to represent current drawn by underlying devices
  - Voltage sources to represent external power supply
  - Transient: capacitors are attached from each node to ground
- Port node: node attached current/voltage sources
- Non-port node: only has internal connection



Linear System of Power Grid

- Resistive grid model:

$$Lv = i$$

  - L is $n \times n$ Laplacian matrix (symmetric and diagonally-dominant):

$$L_{i,j} = \begin{cases} \sum_{k,k \neq i} g(i,k), & \text{if } i = j \\ -g(i,j), & \text{if } i \neq j \end{cases}$$

  - $g_{i,j}$ denotes a physical conductance between two nodes i and j
- A power grid is safe, if $\forall i$:

$$v_i \leq V_{th}$$

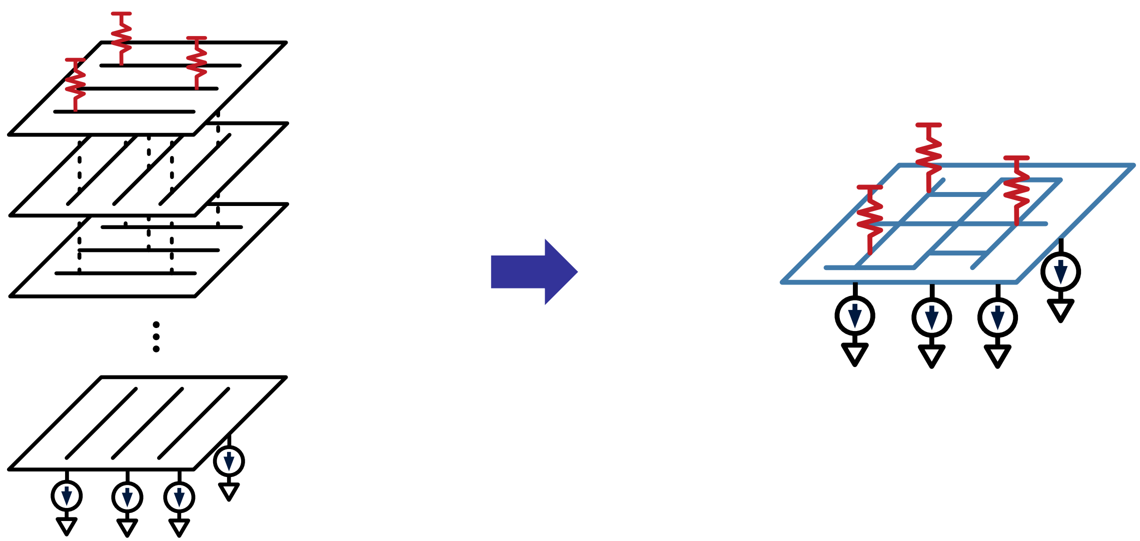
- Long runtime to solve $Lv = i$ for large linear systems

Previous Work

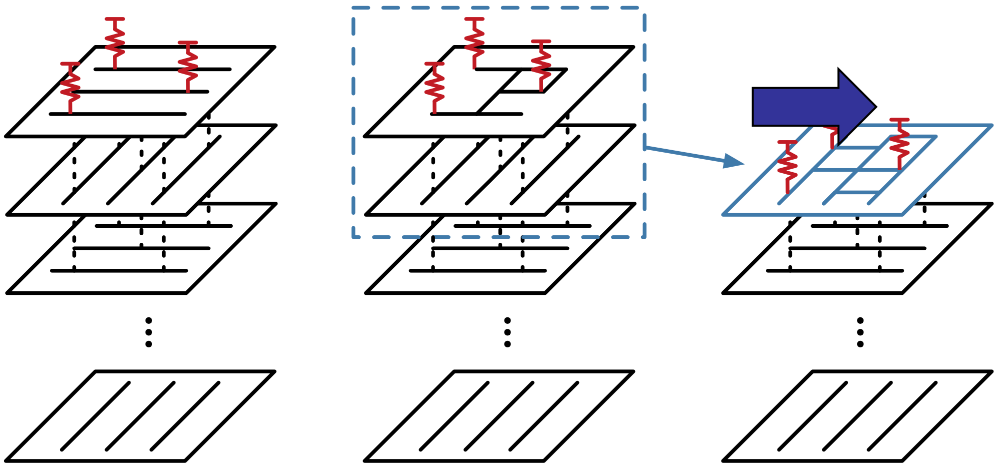
- Power grid reduction
  - Reduce the size of power grid while preserving input-output behavior
  - Trade-off between accuracy and reduction size
- Topological methods
  - TICER [Sheehan+, ICCAD'99]
  - Multigrid [Su+, DAC'03]
  - Effective resistance [Yassine+, ICCAD'16]
- Numerical methods
  - PRIMA [Odabasioglu+, ICCAD'97]
  - Random sampling [Zhao+, ICCAD'14]
  - Convex optimization [Wang+, DAC'15]

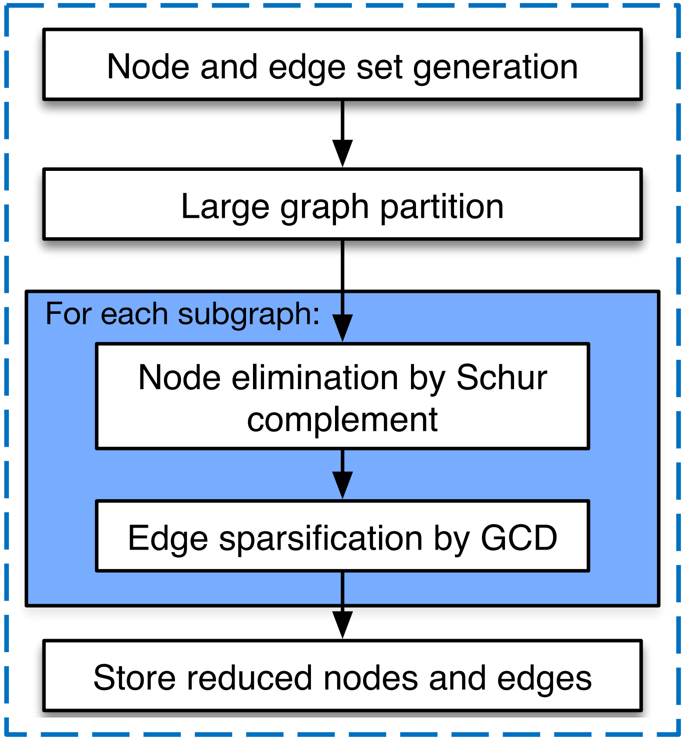
Problem Definition

- Input:
  - Large power grid
  - Current source values
- Output: reduced power grid
  - Small
  - Sparse (as input grid)
  - Keep all the port nodes
  - Preserve the accuracy in terms of voltage drop error



Overall Flow





Node Elimination

- Linear system: $Lv = i$
- L can be represented as a 2×2 block-matrix:

$$L = \begin{bmatrix} L_{11} & L_{12} \\ L_{12}^T & L_{22} \end{bmatrix}$$

- v and i can be represented as follows:

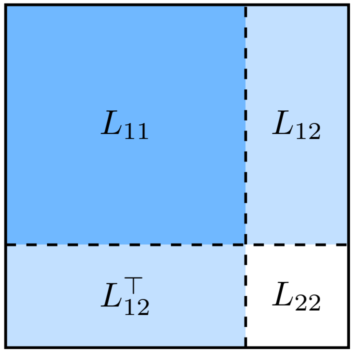
$$v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \text{ and } i = \begin{bmatrix} i_1 \\ 0 \end{bmatrix}$$

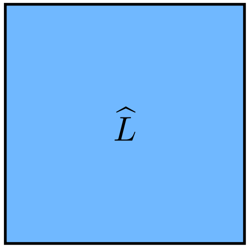
- Applying Schur complement on the DC system:

$$\hat{L} = L_{11} - L_{12} L_{22}^{-1} L_{12}^T$$

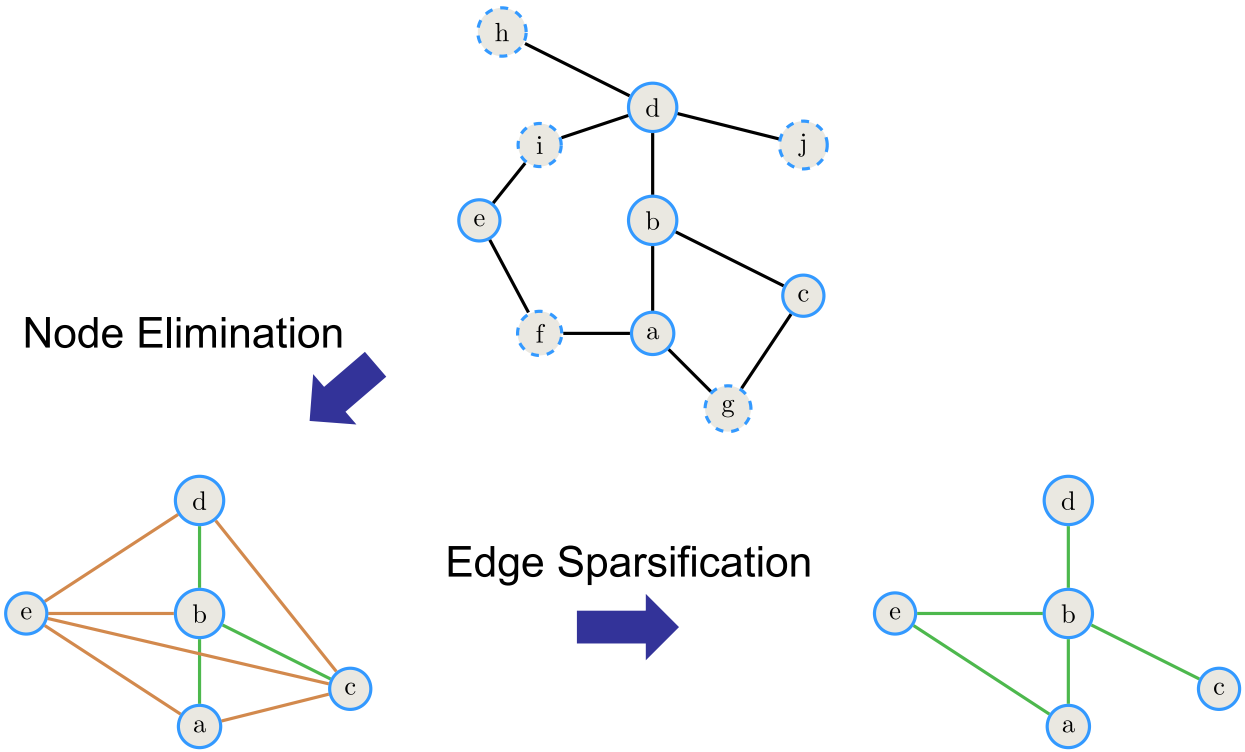
which satisfies:

$$\hat{L} v_1 = i_1$$





Node Elimination (cont'd)



- Output graph keeps all the nodes of interest
- Output graph is **dense**
- Edge sparsification: sparsify the reduced Laplacian without losing accuracy

Edge Sparsification

- Goal of edge sparsification
  - **Accuracy**
  - **Sparsity** reduce the nonzero elements off-the-diagonal in L
- Formulation (1):

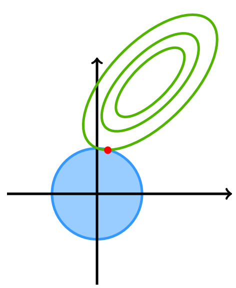
$$\min_{X\in\mathbb{R}^{n\times n}} \boxed{\frac{1}{2m}\sum_{k=1}^{m}\|(X-L)v_k\|_2^2} + \lambda\boxed{\|X\|_0}, \quad \text{s.t. } X \text{ is a Laplacian matrix}$$

- Formulation (2): [Wang+, DAC2014]

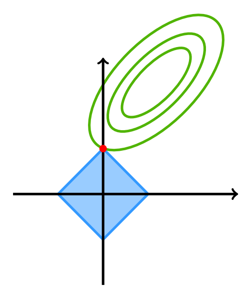
$$\min_{X\in\mathbb{R}^{n\times n}} \frac{1}{2m}\sum_{k=1}^{m}\|(X-L)v_k\|_2^2 + \lambda\boxed{\|X\|_1}, \quad \text{s.t. } X \text{ is a Laplacian matrix}$$

$$\min_{X\in\mathbb{R}^{n\times n}} \frac{1}{2m}\sum_{k=1}^{m}\|(X-L)v_k\|_2^2 + \lambda\sum_{i=1}^{n}X_{i,i}, \quad \text{s.t. } X \text{ is a Laplacian matrix}$$

L2 norm



L1 norm



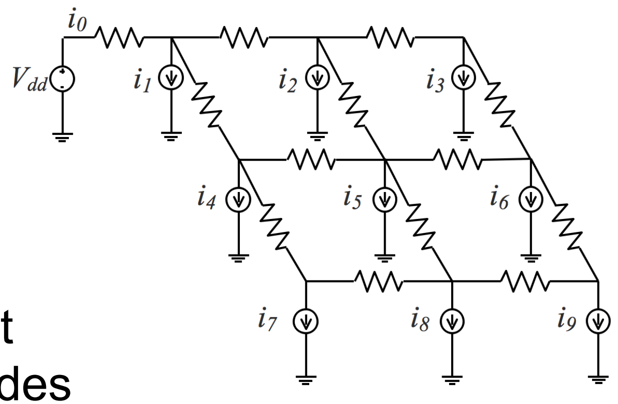
Edge Sparsification

- Formulation (2): [DAC2014 Wang+]

$$\min_{X\in\mathbb{R}^{n\times n}} \frac{1}{2m}\sum_{k=1}^{m}\boxed{\|(X-L)v_k\|_2^2} + \lambda\sum_{i=1}^{n} X_{i,i}, \quad \text{s.t. } X \text{ is a Laplacian matrix}$$

$$\boxed{|\Delta i_0|^2 + |\Delta i_1|^2 + \cdots |\Delta i_9|^2}$$

$$i_0 \gg i_k, \forall i = 1, \cdots, 9$$



Problem: accuracy on the Vdd node does not guarantee accuracy on the current source nodes

- Formulation (3):

$$\min_{X\in\mathbb{R}^{n\times n}} \frac{1}{2m}\sum_{k=1}^{m}\|((X-L)v_k)\circ \boxed{w}\|_2^2 + \lambda\sum_{i=1}^{n} X_{i,i}, \quad \text{s.t. } X \text{ is a Laplacian matrix}$$

  - Weight vector: $w_0 = 1/n, w_i = 1, \forall i = 1, \cdots, n$
  - Strongly convex and coordinate-wise Lipschitz smooth

Coordinate Descent (CD) Method

- Update one coordinate at each iteration
- Coordinate descent:

  Set $t = 1$ and $X^1 = 0$

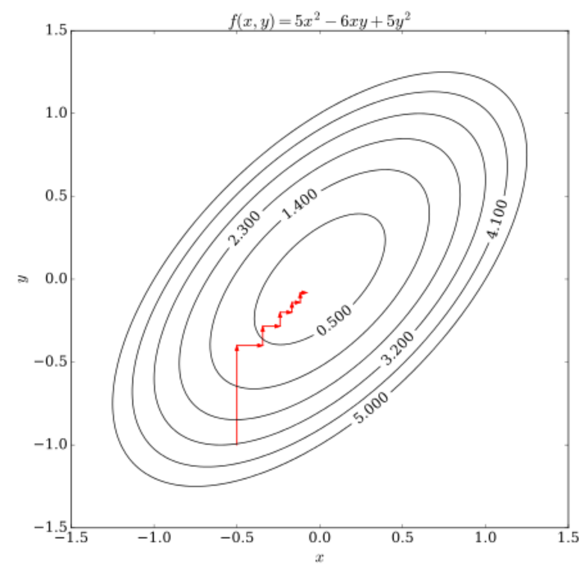
  For a fixed number of iterations (or convergence is reached):

  Choose a coordinate (i, j)

  Compute the step size δ^* by minimizing

  $$\operatorname*{argmin}_{\delta} f(X + \delta e_{i,j})$$

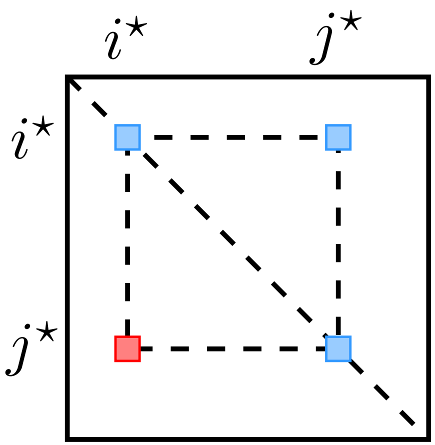
  Update $X_{i,j}^{t+1} \leftarrow X_{i,j}^{t} + \delta^*$

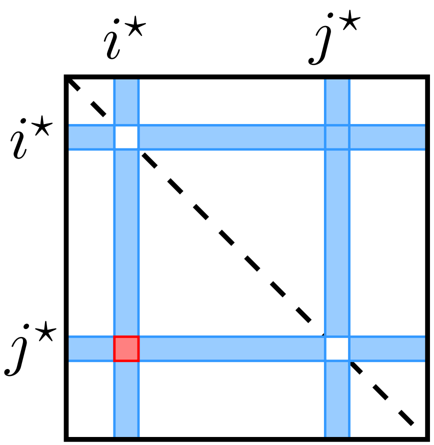


- How to decide the coordinate?
  - Cyclic (CCD)
  - Random sampling (RCD)
  - **Greedy coordinate descent** (GCD)

CD vs Gradient Descent

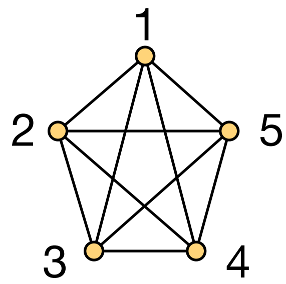
- Gradient descent (GD) algorithm:
$$X^{t+1} \leftarrow X^t - \alpha \nabla f(X)$$
- GD/SGD update $O(n^2)$ elements in X and gradient matrix G at each iteration
- CD updates $O(1)$ elements in X (Laplacian property)
- CD proves to update $O(n)$ elements in G for Formulation (2) and (3).

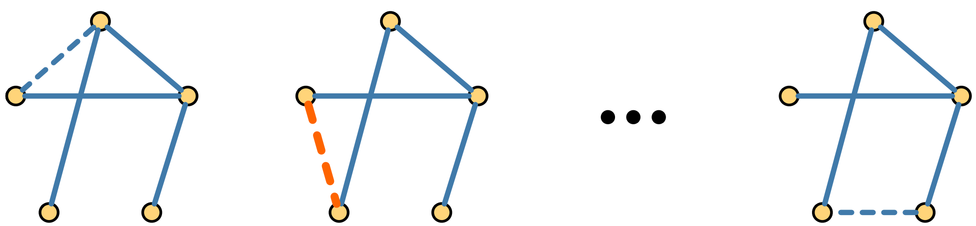


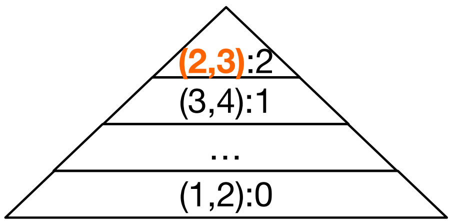


Greedy Coordinate Descent (GCD)

Input L

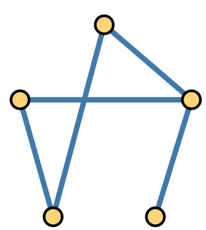




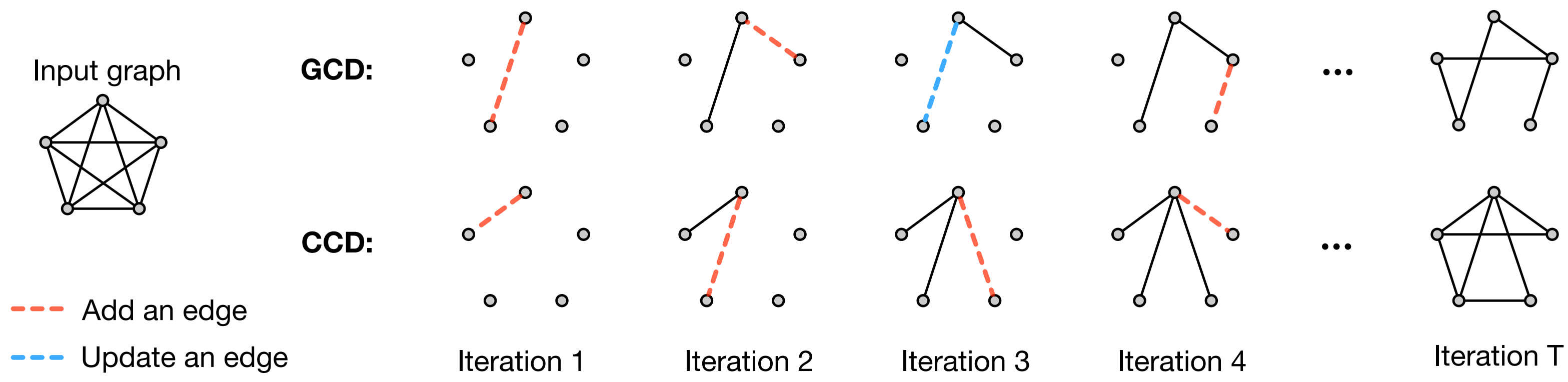


Max-heap

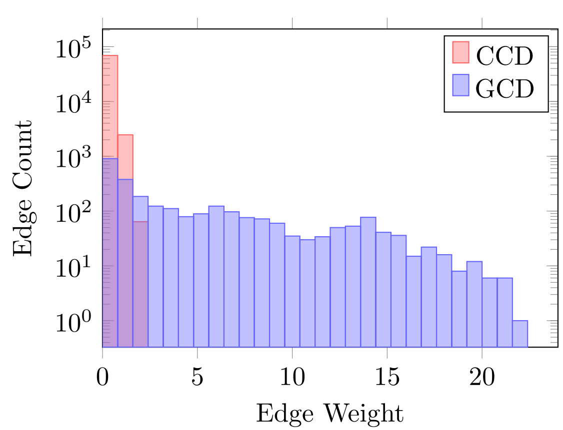
Output X



GCD vs CCD



- GCD produces sparser results
  - CCD (RCD) goes through all coordinates repeatedly
  - GCD selects the most significant coordinates to update



GCD Coordinate Selection

- General Gauss-Southwell Rule:

$$(i^*, j^*) = \arg\max_{(i,j)\in[n]\times[n]} |G_{i,j}|$$

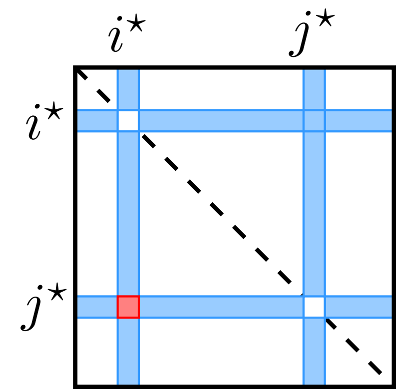
- Observation: the objective function is quadratic w.r.t. the chosen coordinate
- GCD is stuck for some corner cases:

- A new coordinate selection rule:

$$(i^*, j^*) = \arg\max_{(i,j)\in[n]\times[n]} |G_{i,j}| \ \text{ s.t. } G_{i,j} > 0 \text{ or } y_{i,j} \neq 0$$

GCD Speedup

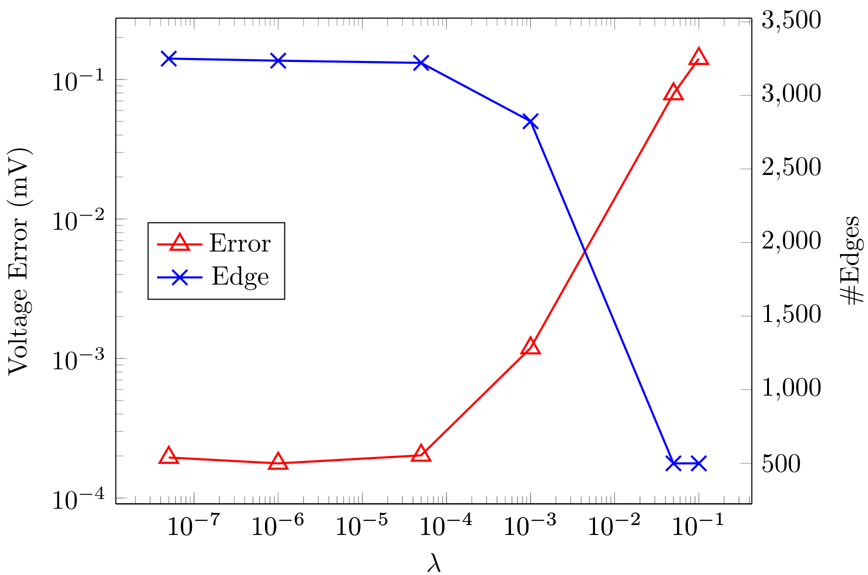
- Time complexity is $O(n^2)$ per iteration
  - traverse $O(n^2)$ elements to get the best index
  - As expensive as gradient descent
- Observation: each node has at most n neighbors → heap

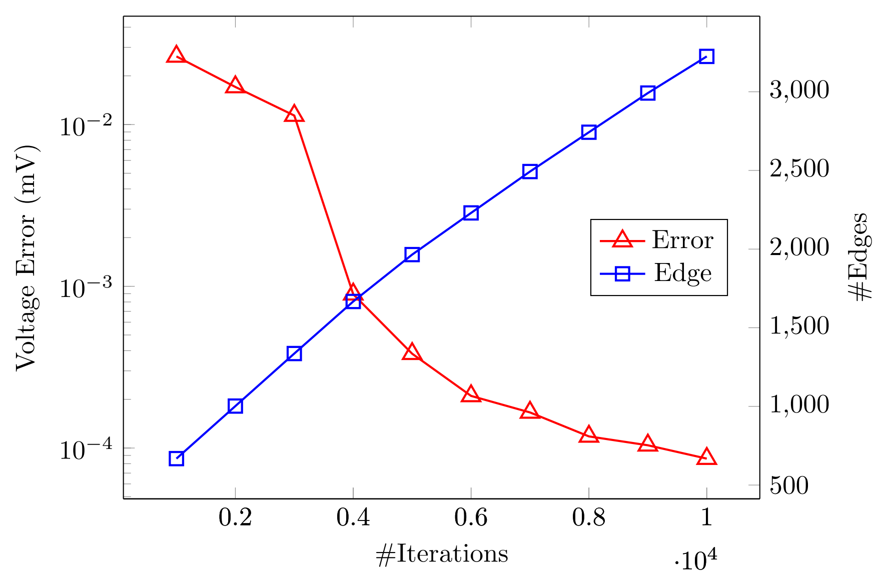


- Heap to store $O(n^2)$ elements in G:
  - Pick the largest gradient, $O(1)$
  - Update $O(n)$ elements, $O(n \log n)$
- Lookup table
  - $O(n^2)$ space; $O(1)$ for each update
- **Improved** time complexity $O(n \log n)$

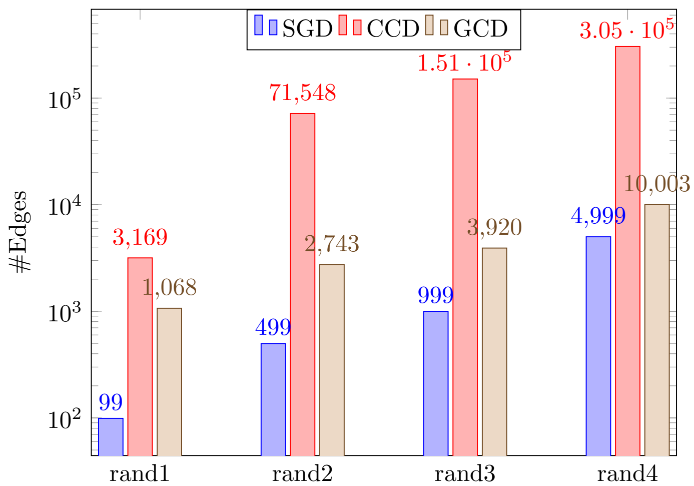
Experimental Results

- Sparsity and accuracy trade-off
- Accuracy and runtime trade-off

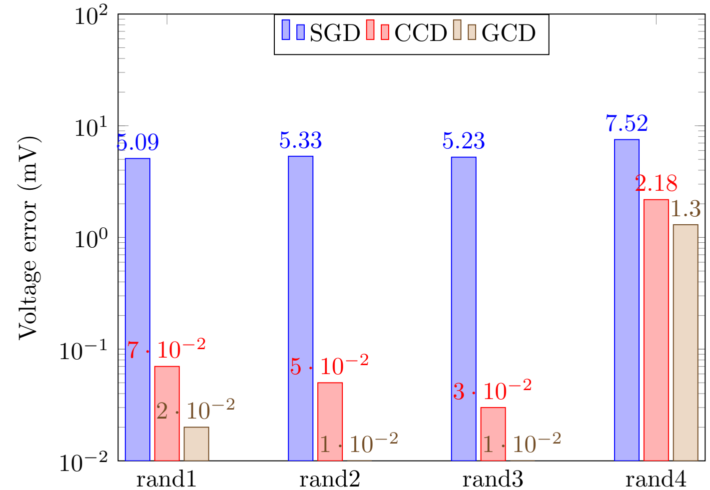




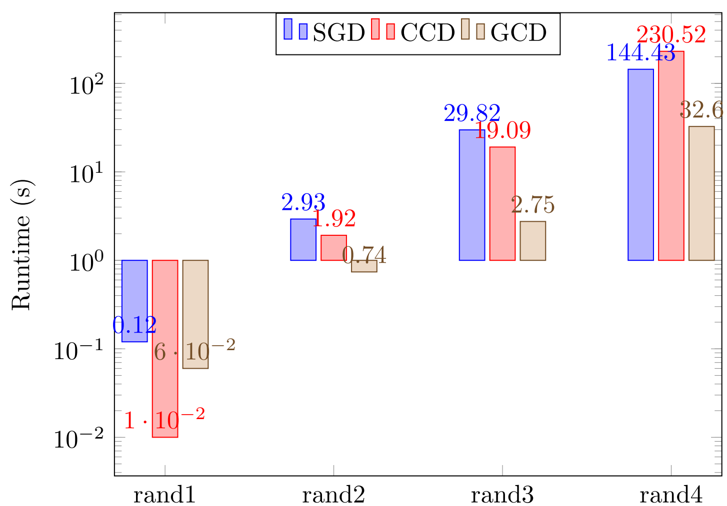
Gradient Descent Comparison



Sparsity



Accuracy



Runtime

Experimental Results

CKT		ibmpg2	ibmpg3	ibmpg4	ibmpg5	ibmpg6
#Port Nodes	Before	19,173	100,988	133,622	270,577	380,991
	After	19,173	100,988	133,622	270,577	380,991
#Non-port Nodes	Before	46,265	340,088	345,122	311,072	481,675
	After	**0**	**0**	**0**	**0**	**0**
#Edges	Before	106,607	724,184	779,946	871,182	1283,371
	After	48,367	243,011	284,187	717,026	935,322
Error		**1.2%**	**0.7%**	**4.8%**	**2.2%**	**2.0%**
Runtime		38s	106s	132s	123s	281s

Conclusion

- Main Contributions:
  - An iterative power grid reduction framework
  - Weighted convex optimization-based formulation
  - A GCD algorithm with optimality guarantee and runtime efficiency for edge sparsification

- Future Work:
  - Extension to RC grid reduction

Thanks