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
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 and edge set generation

Large graph partition

For each subgraph:

Node elimination by Schur complement

Edge sparsification by GCD

Store reduced nodes and edges
Node Elimination

- Linear system: $Lv = i$
- $L$ can be represented as a $2 \times 2$ block-matrix:
  \[
  L = \begin{bmatrix}
  L_{11} & L_{12} \\
  L_{12}^T & L_{22}
  \end{bmatrix}
  \]
- $v$ and $i$ can be represented as follows:
  \[
  \begin{align*}
  v &= \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \quad \text{and} \quad i = \begin{bmatrix} i_1 \\ 0 \end{bmatrix}
  \end{align*}
  \]
- Applying **Schur complement** on the DC system:
  \[
  \hat{L} = L_{11} - L_{12} L_{22}^{-1} L_{12}^T
  \]
  which satisfies:
  \[
  \hat{L}v_1 = i_1
  \]
Output graph keeps all the nodes of interest
Output graph is **dense**
Edge sparsification: sparsify the reduced Laplacian without losing accuracy
Goal of edge sparsification

- **Accuracy**
- **Sparsity** reduce the nonzero elements off-the-diagonal in \( L \)

Formulation (1):

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

Formulation (2): [Wang+, DAC2014]

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

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

♦ Formulation (2): [DAC2014 Wang+]

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

\[|\Delta i_0|^2 + |\Delta i_1|^2 + \cdots + |\Delta i_9|^2\]

\[i_0 \gg i_k, \forall i = 1, \cdots, 9\]

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

♦ Formulation (3):

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

› 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 $\delta^*$ by minimizing
  $$\arg\min_\delta f(X + \delta e_{i,j})$$
  
  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

![Input graph](image)

- Iteration 1
- Iteration 2
- Iteration 3
- Iteration 4
- Iteration T

![Chart](image)
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:

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

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