An Empirical Study of Cycle Toggling Based Laplacian Solvers

Kevin Deweese[1] John Gilbert[1] Gary Miller[2] Richard Peng[3]
Hao Ran Xu[4] Shen Chen Xu[2]

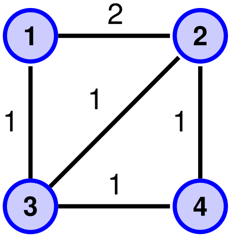
[1]UCSB

[2]Carnegie Mellon [3]Georgia Tech [4]MIT

SIAM Workshop on Combinatorial Scientific Computing, 2016

Problem Description

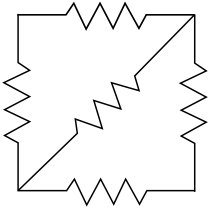
- Our focus: Solve the the system of equations $Lx = b$ where L is a graph Laplacian matrix



$$\begin{pmatrix} 3 & -2 & -1 & 0 \\ -2 & 4 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}$$

Electrical Networks

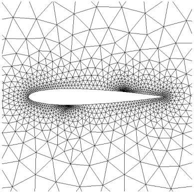
- Edge weights of Laplacians are actually conductances, or inverse resistances $w_e = 1/r_e$
- Right hand side b contains current demands at every vertex
- Left hand side x contains potential, or voltage at every vertex
- Can also consider a flow, or current on every edge



Applications

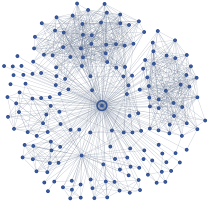
Graphs with regular degree structure, 2D/3D meshes

- Finite element analysis
  - Electrical and thermal conductivity
  - Fluid flow modeling
- Image processing
  - Image segmentation, inpainting, regression, classification



Graphs with irregular degree, problems in network analysis

- Maximum flow problems
- Graph sparsification
- Spectral clustering



Solvers With Good Asymptotic Complexity

Primal (solves for vertex potentials)

- Linear times polylog. Spielman and Teng, 2006
- Nearly $m \log n$. Koutis, Miller, and Peng, 2011
- Nearly $m \log^{1/2} n$. Cohen et al., 2016

Solvers With Good Asymptotic Complexity

Primal (solves for vertex potentials)

- Linear times polylog. Spielman and Teng, 2006
- Nearly $m \log n$. Koutis, Miller, and Peng, 2011
- Nearly $m \log^{1/2} n$. Cohen et al., 2016

Dual (solves for edge flows)

- A simple, nearly $m \log^2 n$, combinatorial algorithm. Kelner, Orecchia, Sidford, and Zhu, 2013

Solvers With Good Asymptotic Complexity

Primal (solves for vertex potentials)

- Linear times polylog. Spielman and Teng, 2006
- Nearly $m \log n$. Koutis, Miller, and Peng, 2011
- Nearly $m \log^{1/2} n$. Cohen et al., 2016

Dual (solves for edge flows)

- A simple, nearly $m \log^2 n$, combinatorial algorithm. Kelner, Orecchia, Sidford, and Zhu, 2013 Simplest

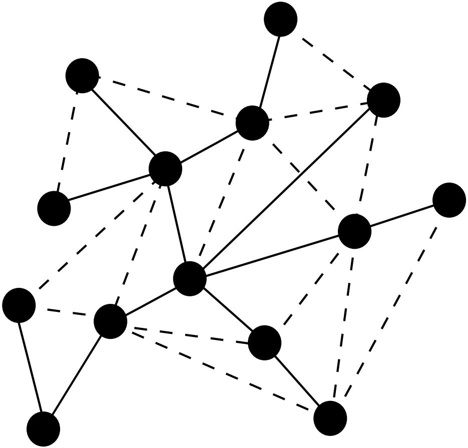
UCSB

Our Goal

- Provide some understanding of Kelner et al. (cycle toggling) implementations
- Introduce a useful class of test problems, *heavy path* graphs, for exploring these methods
- Examine performance behavior of different cycle toggling methods

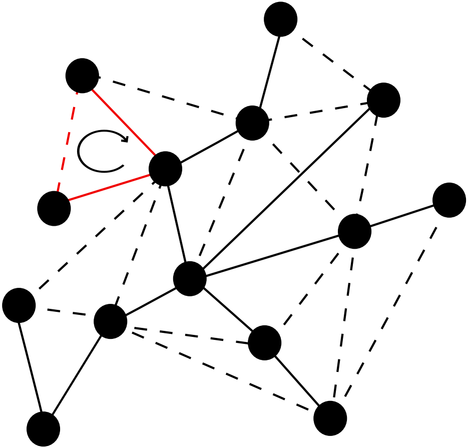
Kelner et al.'s Method (Cycle Toggling)

- Select cycle (with probability proportional to stretch) from a fundamental cycle basis
- Update flows around cycle



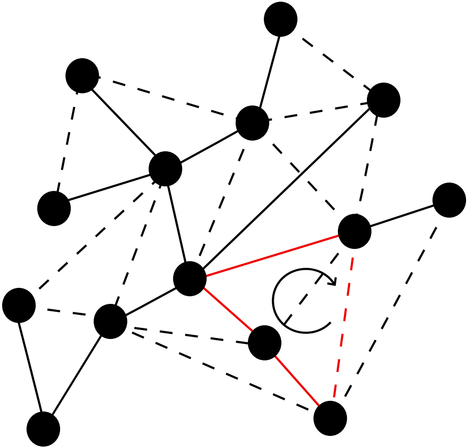
Kelner et al.'s Method (Cycle Toggling)

- Select cycle (with probability proportional to stretch) from a fundamental cycle basis
- Update flows around cycle



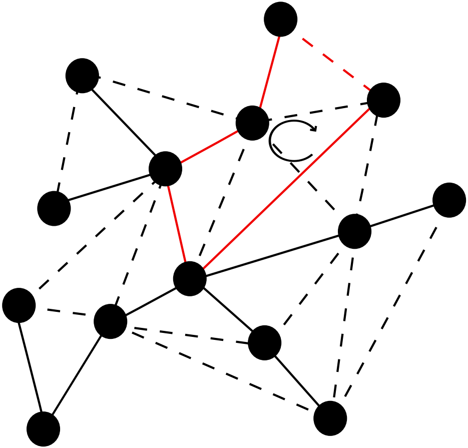
Kelner et al.'s Method (Cycle Toggling)

- Select cycle (with probability proportional to stretch) from a fundamental cycle basis
- Update flows around cycle



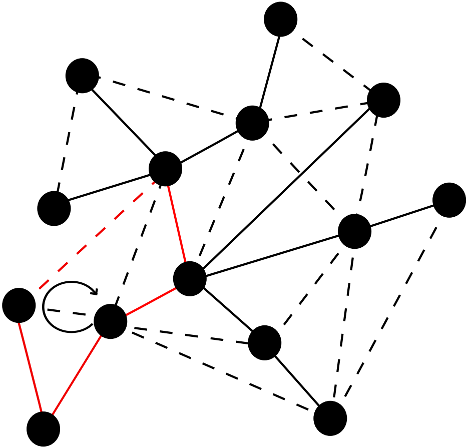
Kelner et al.'s Method (Cycle Toggling)

- Select cycle (with probability proportional to stretch) from a fundamental cycle basis
- Update flows around cycle



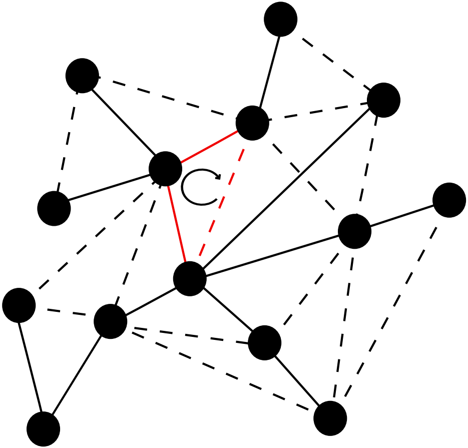
UCSB

- Select cycle (with probability proportional to stretch) from a fundamental cycle basis
- Update flows around cycle



Kelner et al.'s Method (Cycle Toggling)

- Select cycle (with probability proportional to stretch) from a fundamental cycle basis
- Update flows around cycle



Cycle Toggle Methods

total cost $=$ number of cycle toggles $\times$ cost per cycle toggle

Cycle Toggle Methods

total cost $=$ number of cycle toggles $\times$ cost per cycle toggle

proportional to tree stretch

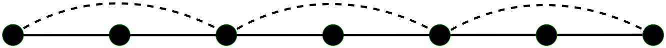
Cycle Toggle Methods

total cost = number of cycle toggles $\times$ cost per cycle toggle

clever implementations

Heavy Path Graphs

- Path graph + edges weighted such that the path is low-stretch tree
- Used to explore fundamental questions of cycle toggling approaches
- Can be tuned to have various stretch and spectral properties



Cycle Toggle Methods

Support two operations

- Query: (find voltage drop, $\sum_e r_e f_e$ along cycles)
- Update: (alter flow of the cycle by Δ)

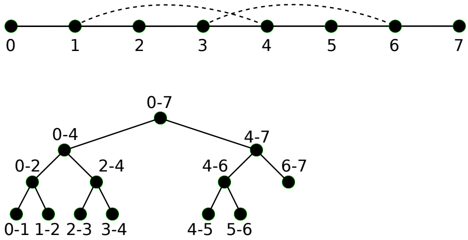
We consider two strategies

- Single level with fancy data structures
- Multilevel divide-and-conquer

Toggle Method 1: Single Level with Data Structures

Balanced binary search trees can be used to provide $O(\log n)$ query and update operations on intervals of a path graph.

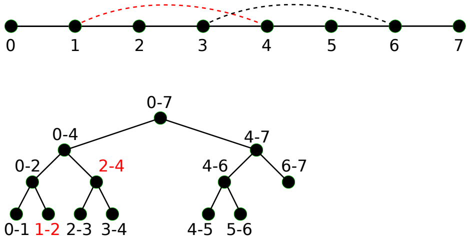
- Create BST representing path intervals
- Store voltage drop, $\sum rf$ at every sub-interval
- Initialize a lazy tag at every interval to 0



Toggle Method 1: Single Level with Data Structures

Balanced binary search trees can be used to provide $O(\log n)$ query and update operations on intervals of a path graph.

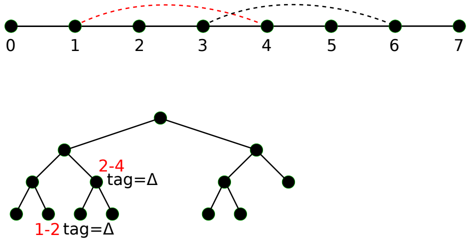
- Query interval 1-4
- Add sums of intervals 1-2 and 2-4



Toggle Method 1: Single Level with Data Structures

Balanced binary search trees can be used to provide $O(\log n)$ query and update operations on intervals of a path graph.

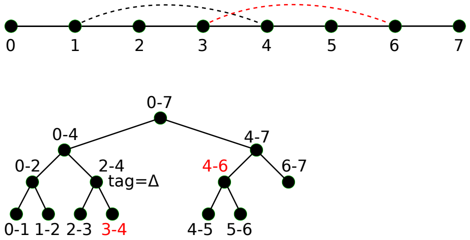
- Apply update Δ to interval 1-4 by updating intervals 1-2, 2-4, and all ancestor intervals
- Set tag of intervals 1-4 and 1-2 to Δ



Toggle Method 1: Single Level with Data Structures

Balanced binary search trees can be used to provide $O(\log n)$ query and update operations on intervals of a path graph.

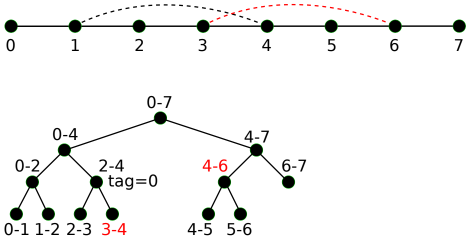
- Query interval 3-6
- Encounter non-zero tag



Toggle Method 1: Single Level with Data Structures

Balanced binary search trees can be used to provide $O(\log n)$ query and update operations on intervals of a path graph.

- Push tag information to children
- Set tag to 0



Toggle Method 1: Single level with Data Structures

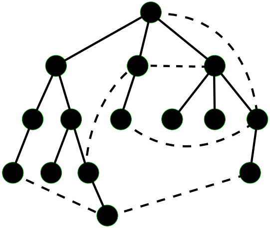
- Can be extended to general graphs
- BSTs can be combined with *heavy light* decomposition to yield $O(\log^2 n)$ per update
- Can be improved to $O(\log n)$ with virtual trees

Toggle Method 2: Multilevel Divide-and-Conquer

- Preselect a batch of K cycles to update
- Use this knowledge to reduce problem size (contraction, path compression)
- Operate on the batch of cycles recursively
- Updating a batch is $O(n \log n)$ if K is $O(n)$

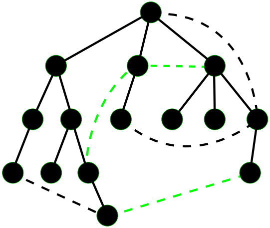
UCSB

Toggle Method 2: Multilevel Divide-and-Conquer



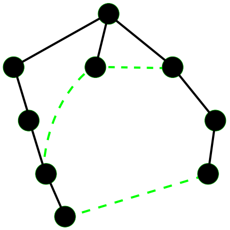
Toggle Method 2: Multilevel Divide-and-Conquer

- Sample cycles



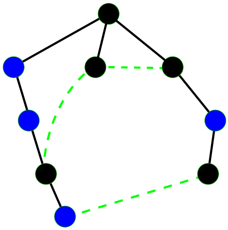
Toggle Method 2: Multilevel Divide-and-Conquer

- Sample cycles
- Contract graph on selected cycles



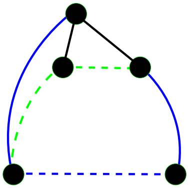
Toggle Method 2: Multilevel Divide-and-Conquer

- Sample cycles
- Contract graph on selected cycles
- Remove degree-2 vertices



Toggle Method 2: Multilevel Divide-and-Conquer

- Sample cycles
- Contract graph on selected cycles
- Remove degree-2 vertices



Experimental Setup: Methods to Compare

- Single level, general graphs
- Single level, heavy path optimized
- Multilevel, general graphs
- Multilevel, heavy path optimized

Experimental Setup: Methods to Compare

- Single level, general graphs
- Single level, heavy path optimized
- Multilevel, general graphs
- Multilevel, heavy path optimized

Want to compare average cycle update time

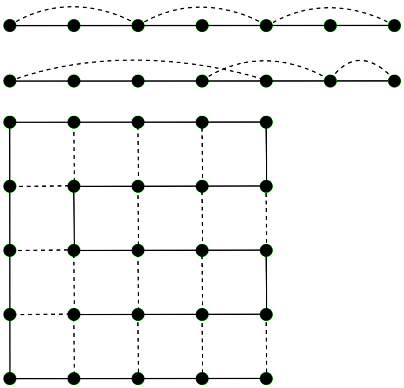
UCSB

Experimental Setup: Methods to Compare

- Single level, general graphs
- Single level, heavy path optimized
- Multilevel, general graphs
- Multilevel, heavy path optimized
- Jacobi preconditioned conjugate gradient

Experimental Setup: Heavy Path Graphs

- Fixed Cycle Length (2 and 1000)
- Random Cycle Length
- 2D Mesh
- 3D Mesh



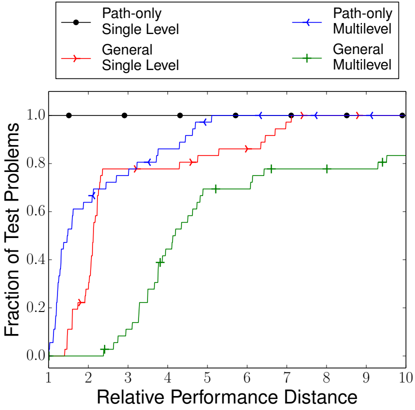
Experimental Setup: Heavy Path Graphs

- Edge weights chosen for different stretch behavior
  - Uniform stretch: Stretch is 1 for every cycle
  - Exponential stretch: Stretch of every cycle is sampled from an exponential distribution
- Graph size in vertices 5×10^4, 10^5, 5×10^5, 10^6

Experimental Setup

- Right hand sides
  - Random: Select x and form $b = Lx$
  - (-1,1): Route one unit of flow from one endpoint of path to the other
- Residual tolerance 10^{-5}

Performance Profile: Cycle Toggle Time



Cycle Toggle Time

	Path-only Single Level	Path-only Multilevel	General Single Level	General Multilevel
% of problems solver is best	100	0	0	0

Cycle Toggle Time

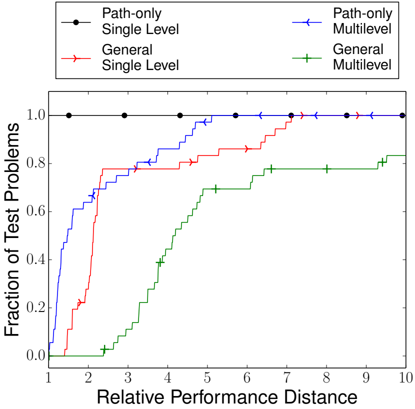
	Path-only Single Level	Path-only Multilevel	General Single Level	General Multilevel
% of problems solver is best	100	0	0	0
% within factor 2 of best	100	60	20	0

UCSB

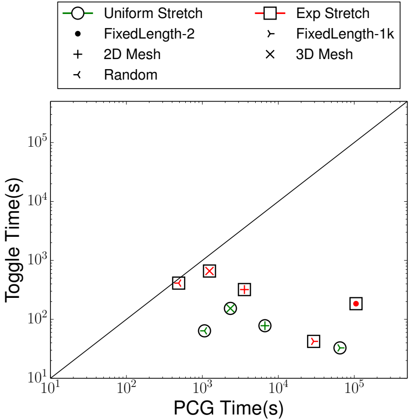
Cycle Toggle Time

	Path-only Single Level	Path-only Multilevel	General Single Level	General Multilevel
% of problems solver is best	100	0	0	0
% within factor 2 of best	100	60	20	0
% within factor 10 of best	100	100	100	80

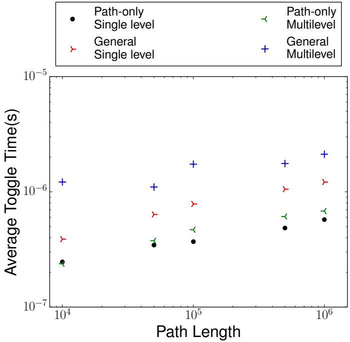
Performance Profile: Cycle Toggle Time



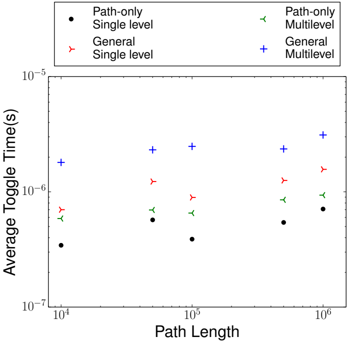
PCG Comparison (to General Single Level)



Weak Scaling

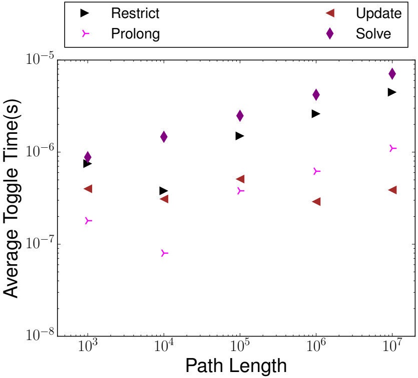


3D Mesh Exponential Stretch



Fixed Length 1k Uniform Stretch

Timing Breakdown of Recursive Toggling



Summary of Results

- Heavy path graphs are a useful model to consider
  - Simplify implementation details
  - Cycle toggling outperforms PCG
- Single level with fancy data structures performs better than multilevel recursive
- Check out our code and data at https://github.com/sxu/cycleToggling

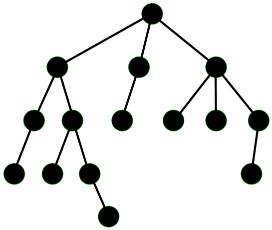
Future Work

- Combine primal and dual solvers
- Examine floating point ops required for recursive cycle updates
- Further explore heavy path graphs

Thank You

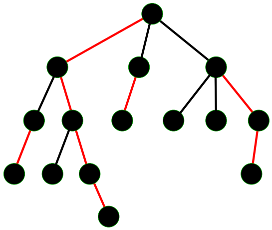
Henning Meyerhenke

Heavy Light Decomposition



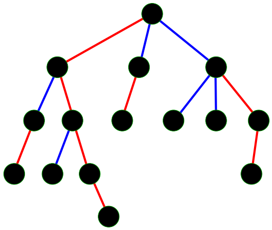
- Arbitrarily root tree

Heavy Light Decomposition



- Arbitrarily root tree
- Mark child edges to largest subtree as *heavy*
- Maximal length paths of *heavy* edges are called *heavy chains*

Heavy Light Decomposition



- Arbitrarily root tree
- Mark child edges to largest subtree as *heavy*
- Maximal length paths of *heavy* edges are called *heavy chains*
- Mark other edges as *light*

BST + HLD

- A path from any vertex to the root intersects $O(\log n)$ *heavy chains* and $O(\log n)$ *light* edges
- $O(1)$-cost operations on *light* edges and $O(\log n)$-cost operations on *heavy chains* via binary search trees
- Theoretical bound of $O(\log^2 n)$ per operation, good running time experimentally

Stretch Dependency