Comparing Different Cycle Bases for a Laplacian Solver

Erik G. Boman* Kevin Deweese† John R. Gilbert†

1 Kelner et al.'s Randomized Kaczmarz Solver

Solving linear systems on the graph Laplacian of large unstructured networks has emerged as an important computational task in network analysis [7]. Most work on these solvers has been on preconditioned conjugate gradient (PCG) solvers or specialized multigrid methods [6]. Spielman and Teng, showed how to solve these problems in nearly-linear time [8], later improved by Koutis et al. [5] but these algorithms do not have practical implementations. A promising new approach for solving these systems proposed by Kelner et al. [4] involves solving a problem that is dual to the original system.

The inspiration for the algorithm is to treat graphs as electrical networks with resistors on the edges. The graph Laplacian is defined as $L = D - A$ where D is a diagonal matrix containing the sum of incident edge weights and A is the adjacency matrix. For each edge, the weight is the inverse of the resistance. We can think of vertices as having an electrical potential and net current at every vertex, and define vectors of these potentials and currents as $\vec{v}$ and $\vec{\chi}$ respectively. These vectors are related by the linear system $L\vec{v} = \vec{\chi}$. Solving this system is equivalent to finding the set of voltages that satisfy the currents. Kelner et al.'s SimpleSolver algorithm solves this problem with an optimization algorithm in the dual space which finds the optimal currents on all of the edges subject to the constraint of zero net voltage around all cycles. They use Kaczmarz projections [3] [9] to adjust currents on one cycle at a time, iterating until convergence. They prove that randomly selecting fundamental cycles from a particular type of spanning tree called a "low-stretch" tree yields convergence with nearly-linear total work.

*Sandia National Laboratories, Sandia is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energys National Nuclear Security Administration under contract DE-AC04-94AL85000. eboman@sandia.gov

†UC Santa Barbara Dept. of Computer Science, Supported by Contract #618442525-57661 from Intel Corp. and Contract #8-48252526701 from the DOE Office of Science. kdeweese@cs.ucsb.edu, gilbert@cs.ucsb.edu

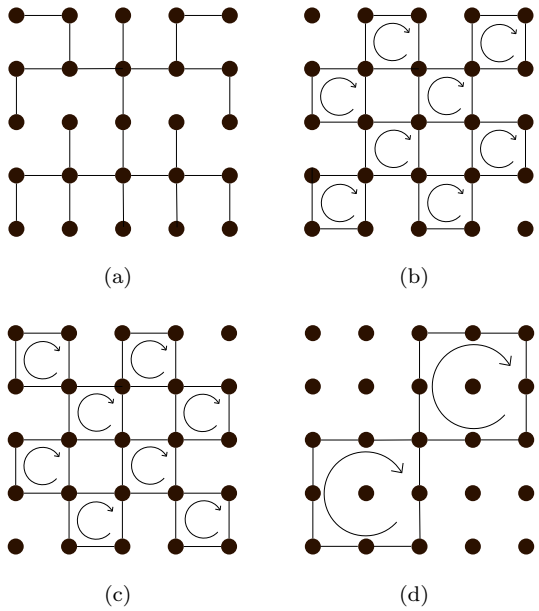

Figure 1: Grid Cycles

2 Choosing the Cycle Basis

We examine different ways to choose the set of cycles and their sequence of updates with the goal of providing more flexibility and potential parallelism. Our ideas include the following.

- Provide parallelism by projecting against multiple edge-disjoint cycles concurrently.
- Provide flexibility by using a non-fundamental cycle basis.
- Provide flexibility by using more (perhaps many more) cycles than just a basis.
- Accelerate convergence by varying the mixture of short and long cycles in the updating schedule.

Sampling fundamental cycles from a tree will require updating several potentially long cycles which will not be edge-disjoint. It would be preferable to update edge-disjoint cycles as these updates could be done in parallel. Instead of selecting a cycle basis from a

spanning tree, we will use several small, edge-disjoint cycles. We expect updating long cycles will be needed for convergence, but we consider mixing in the update of several short cycles as they are cheap to update and have more exploitable parallelism. These cycles can then be added together to form larger cycles to project against in a multigrid like approach.

An example of these cycles can be seen on the 5 by 5 grid graph in Figure 1. Figure 1(a) shows a spanning tree in which each cycle is determined by an edge not in the tree. The smallest cycles of a non-fundamental scheme are shown in Figures 1(b)(c). All the cycles in each of these two figures are edge-disjoint and can be updated in parallel. They can also be summed together as in Figure 1(d).

3 Preliminary Experiments and Results

We performed our initial experiments on grid graphs of various sizes. We used a non-fundamental set of cycles with a hierarchical ordering. The smallest set of cycles are updated. Then the cycles are coarsened and the next level of cycles are updated. This is done until reaching the perimeter cycle before resetting back to updating the smallest cycles. We also implemened the SimpleSolver algorithm in Matlab, except that we used a random spanning tree for sampling instead of a low-stretch tree. We also haven't implemented a clever data structure Kelner et al. use to quckly update edges. We also compared our results to PCG with Jacobi.

The metric we choose for comparison is the total number of edges updated, or matrix elements touched in CG. We can see the total work measured in edges updated in Table 1. Also shown in the table is an estimated potential parallelism using the work-span model [10]. The span, or critical path length, is the maximum number of edges that would have to be updated by a single processor if we can split the work over infinitly many processors.

Grid Size (Vertices)	25	289	4,225
Fundamental Cycles Work	8K	1.4M	296M
Alternative Cycles Work	1K	.08M	4M
Alternative Cycles Span	.5K	8.4K	105.8K
PCG Work	1K	.09M	5M

Table 1: Edges Updated

4 Conclusions and Future Work

Our preliminary experiments show that choosing a non-fundamental set of cycles can save significant work compared to a fundamental cycle basis, and can be at least competitive with PCG.

We are exploring ways to find a non-fundamental cycle basis of more general graphs; one challenge is how best to find large sets of short edge-disjoint cycles for parallelism. Our ideas for cycle finding include short-cuts to the spanning tree cycles and growing small cycles locally around vertices and edges. We also plan to make a rigorous comparison with several other preconditioned CG methods, including incomplete Cholesky and support-graph techniques.

We note that any of these graph Laplacian solvers can be extended to general symmetric diagonally dominant systems via standard reduction techniques. [1] [2].

References

[1] E. G. Boman, D. Chen, B. Hendrickson, and S. Toledo. Maximum-weight-basis preconditioners. *Numerical Linear Algebra Appl.*, 11:695–721, 2004.

[2] K. Gremban. *Combinatorial Preconditioners for Sparse, Symmetric, Diagonally Dominant Linear Systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, October 1996.

[3] S. Kaczmarz. Angenäherte auflösung von systemen linearer gleichungen. *Bulletin International de lAcademie Polonaise des Sciences et des Lettres*, 35:355–357, 1937.

[4] J. A. Kelner, L. Orecchia, A. Sidford, and Z. A. Zhu. A simple, combinatorial algorithm for solving SDD systems in nearly-linear time. In *Pro 45th ACM Symp. Theory of Comp.*, (STOC '13), pages 911–920, New York, 2013.

[5] I. Koutis, G. L. Miller, and R. Peng. Approaching optimality for solving sdd systems. *CoRR*, abs/1003.2958, 2010.

[6] O. E. Livne and A. Brandt. Lean algebraic multigrid (LAMG): Fast graph Laplacian linear solver. *SIAM Scientific Comp*, 34(4):B499–B522, 2012.

[7] D. A. Spielman. Algorithms, graph theory, and linear equations in Laplacian matrices. In *Proceedings of the International Congress of Mathematicians*, volume 4, pages 2698–2722, 2010.

[8] D. A. Spielman and S. Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the Thirty-sixth Annual ACM Symp. on Theory of Comp.*, STOC '04, pages 81–90, New York, NY, USA, 2004. ACM.

[9] S. Toledo. An algebraic view of the new randomized Kaczmarz linear solver. Presented at the Simons Institute for the Theory of Computing, 2013.

[10] B. Wilkinson and M. Allen. *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers.* Prentice Hall, 2004.