Comparing Different Cycle Bases for a Laplacian Solver

Kevin Deweese

June 11, 2014

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 [5]. Most work on these solvers has been on preconditioned conjugate gradient (PCG) solvers or specialized multigrid methods [4]. Spielman and Teng, showed how to solve these problems in nearly-linear time [6] but the algorithm does not have a practical application implementation. A promising new approach for solving these systems proposed by Kelner et al. [2] 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 the diagonal matrix containing the sum of incedent 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 satisfies 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 [1] 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.

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.

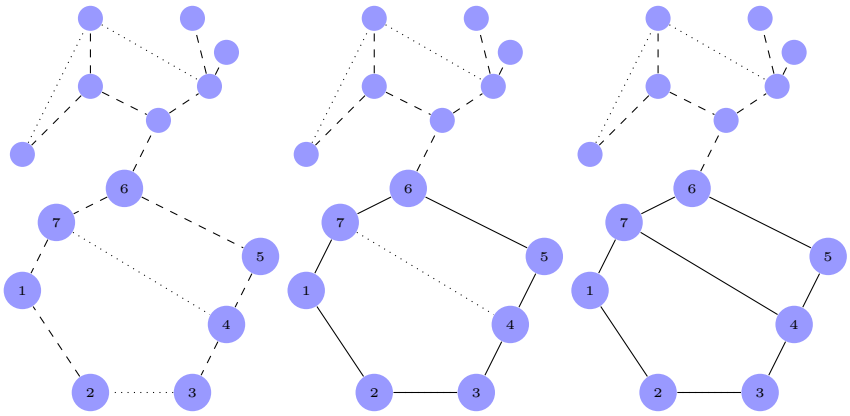
Sampling fundamental cycles from a tree will require updating several potentially long cycles which will not be edge-disjoint. It would be preferable to update edge-disjoint cycles as these updates could be done in parallel. Instead of selecting a cycle basis from a spanning tree, we will use several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

As of this report we have looked at ideas 1, 2, and 4 above and haven't had time to consider many cycles beyond a basis. Much time on this project was spent trying to grow small sets of cycles around nodes and edges but these ideas didn't pan out because we still need a set of cycles that is guaranteed to span the entire cycle space. The technique that worked instead is to start with the fundamental cycle basis and modify it while preserving the rank of the cycle space. We call this algorithm the tree-shortcut algorithm.

---

function TREE-SHORTCUT
  for $e_{i,j} \in E \setminus T$ **do**
    $maxdepth = \min(depth(i), depth(j))$
    $path_{i,j} = \text{Filtered\_Dijkstra}(E \setminus (e_{i,j}), i, j, maxdepth)$
    Replace $fundcycle(e_{i,j})$ with $path_{i,j} + e_{i,j}$
  end for
end function

The algorithm selects an off-tree edge and tries to find a shortest path from the endpoints of the edge on a restricted portion of the graph. The restriction is that it only searches nodes that are closer to the root of the tree than the edge in question. If a cycle is found shorter than the original tree cycle, it must contain at least one non-tree edge which is closer to the root of the tree than the non-tree edge in question. The shortcut cycle replaces the original cycle in the cycle basis. The rest of the original cycle is still represented however as the off-tree edge found in the shortcut path is also represented in the cycle basis. Thus the original cycle is a sum of this internal cycle and the new cycle. A spanning tree is shown in Figure 1(a). When off tree edge $e_{2,3}$ is sampled from the tree a fundamental cycle of length 7 is selected, shown in Figure 1(b). The



(a) Spanning Tree (b) Fundamental Cycle (c) Shortcut Cycle

Figure 1: Tree Shortcut

tree-shortcut algorithm finds the cycle of length 5 going through edge $e_{4,7}$ and replaces the original length 7 cycle with the new cycle shown in Figure 1(c). However the length 4 fundamental cycle of edge $e_{4,7}$ remains and the original cycle is still contained in the new cycle basis by summing these two cycles.

3 Cycle Sampling

In the original Kelner algorithm, cycles are chosen one at a time with probability proportional to tree stretch. One of the purposes of using a different cycle basis is to update multiple, edge-disjoint cycles in parallel. Thus we need a way to select groups of cycles to be updated concurrently. The current approach we are using is to create groups of edge-disjoint cycles and loop through these groups updating every cycle in them, simulating the parallel update of all cycles in them. We use a greedy cycle coloring algorithm to put the new cycles in edge-disjoint groups by coloring cycles the same color if they are edge-disjoint. The only way we currently account for the original sampling probability is that the greedy cycle coloring looks at uncolored cycles in order of decreasing cycle resistance. So the first cycle color should contain more important cycles. We also apply this parallelization to the origonal set of fundamental cycles.

4 Experimental and Results

We performed experiments on a variety of graphs shown in Figure 2. We implemented the tree-shortcut and simple solver algorithms in Matlab, except that we used a random spanning tree for sampling instead of a low-stretch tree. On the small graphs used here theoretically any random spanning tree gives low-stretch. We also haven't implemented a clever data structure Kelner et al. use

Graph	Nodes	Edgesx2
USpowerGrid	4,941	13,188
60 by 60 mesh	3,600	14,160
bcspwr06	1,454	5,300
email	1,133	10,902
data	2,851	30,186

Figure 2: Graphs Used

Graph	Tree	Parallel Tree	Parallel Tree-Shortcut
USpowerGrid	5M	38M	20M
60 by 60 mesh	226M	302M	225M
bcspwr06	1.1M	1.6M	1.4M
email	2.4M	1.8B	171M
data	75M	N/A	1.1B

Figure 3: Total Edges Updated

to quickly update edges. As such the metric we are most interested in is total edges updated, not solve time, shown in Figure 3. In addition we estimate parallelism by looking at the span, or critical path length, the maximum number of edges that would have to be updated by a single processor, which is shown in Figure 4.

5 Discussion and Conclusion

So far we haven't achieved a reduction in total edges by using a different set of cycles (except slightly on the grid). However, we have created a means to update cycles in parallel, improving the span of computation for the tree-shortcut cycles. Applying the greedy cycle coloring to the original set of fundamental cycles makes the span much worse in some cases, and stagnated on the data graph. It seems that the tree-shortcut cycles can be useful as an alternate set of cycles, but more work needs to be done on how to update them correctly. The next thing to try is to not update every cycle in an edge-disjoint group at once, but

Graph	Parallel Tree	Parallel Tree-Shortcut
USpowerGrid	9.3M	2.8M
60 by 60 mesh	10M	34M
bcspwr06	2867k	230k
email	686M	25M
data	N/A	180M

Figure 4: Span of Edges Updated

rather roll the dice for each one of them. We suspect that too much work is being undone by updating less important cycles too frequently. Another thing to consider is adding edges together in a multigrid fashion by adding a cycle to the cycle it split off from during the tree shortcut. By adding enough of these we would get our fundamental basis back and we could go back and forth between these extremes.

References

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

[2] 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.

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

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

[5] 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.

[6] 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.