

# Eric Buras Thesis: Methodology

March 17, 2016

## 1 Graph Partitioning

In studying 'small world' networks proposed by Watts and Strogatz [7] Fan Chung and Linyuan Lu propose an algorithm that separates a graph into a locally connected component and a global component. Finding a local subgraph is akin to finding a locally connected portion of a graph. Given integers  $k \geq 2$  and  $l \geq 2$ , a  $(k, l)$  locally connected graph will have at least  $l$  paths connecting any given two nodes with distinct edges in each path. The length of each path can be at most  $k$  edges for this pair. A grid network can be described locally with  $k = 3$  and  $l = 3$ , and is a good example of how a planar graph is connected. Given graph  $G$ , its maximum locally connected subgraph is the union of all locally connected subgraphs within the entire graph. It is important that this maximum is unique, and can be found through an iterative edge deletion algorithm[3]. Search through all edges in the graph, and remove an edge if it does not have at least  $l$  edge-disjoint paths of length less than or equal to  $k$ . Repeat this cycle until no edges can be removed. Chung and Lu prove that this algorithm succeeds regardless of the order of edges removed [3].

### 1.1 Pseudocode

```
Given Graph G
P = copy(G)
While: Deleted_edges == True:
    Deleted_edges = False
    For edge in P.edges:
        (start_node, end_node) = edge
        Search through paths of length  $i = 2 : k$  from start_node to end_node
        count = number of edge-disjoint paths
        if count  $\leq l$ :
            P.remove(edge)
            Deleted_edges = True
```



(a) Grid with random edges

(b) The planar portion is kept intact

Figure 1: Here is a small example of the graph partitioning in action

## 1.2 NetworkX Library and My Additions

There is an open source Python library for graph theory and computation called NetworkX written by Hagberg, Schult, and Swart from Los Alamos National Lab. I use many of their functions for reading through edgelist, computing node degrees, converting graphs to Laplacian matrices, etc which are incredibly useful for simple graph work [5]. Included in this package is a function for creating finding the locally connected subgraph of a graph based upon the work of Chung-Lu above (cite?). Throughout much of the course of my work I used this function to partition my graphs. However for graphs with many edges, this function adds a huge time complexity overhead because it utilizes a shortest path algorithm (cite Dijkstra?). This algorithm should work comparably for any given  $k, l$  connected subgraphs. However I am focused on  $k, l = 3$  so I wrote I an optimized code that simply searches through all potential paths of length 2 and 3 in the graph. This requires much less work than running a shortest path algorithm several thousand times. In the process, and after much debugging and testing, I think that the established NetworkX function has some errors. I now only use my simplified code which speeds up the overall algorithm greatly. Results for partitioning timing are included in the Results section. I hope to work more on my code and submit it as an open-source contribution to the NetworkX package. The established function for partitioning a graph is still important, however, and I hope to work further to find the errors in the function.

## 2 Laplacian Solver

I have partitioned a graph into its locally connected subgraph,  $P$ , and the graph of the teleportation edges,  $T$ . These subgraphs can be converted to Laplacian form with  $P_L$  matrix of connections and degree information for the locally connected part, and  $T_L$ , similar for the teleportation part. With a minor diagonal operation on  $T_L$ , we have  $L = P_L + T_L$  where  $L$  is the Laplacian for the entire graph. To solve a Laplacian linear system  $Lx = b$  we now solve the

two subgraph Laplacians and use linear algebra.

## 2.1 Linear Algebra: Woodbury Matrix Identity

I utilize the Woodbury matrix Identity:

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1}$$

with  $A$  replaced by  $P_L$ ,  $U$  and  $V$  component matrices of the SVD of  $T_L$ , and  $C$  replaced by a diagonal of the singular values of  $T_L$ . For a class of graphs we will assume that  $P_L$  is very large and sparse, and  $T_L$  is very sparse and low rank.

## 2.2 Summary of Solving the Laplacian Linear System

$$\begin{aligned} Lx &= b \\ x &= L^{-1}b \\ x &= (P_L + T_L)^{-1}b \\ x &= (P_L + USV)^{-1}b \\ &\text{Use Woodbury matrix identity} \\ x &= (P_L^{-1} - P_L^{-1}U(S^{-1} + VP_L^{-1}U)^{-1}VP_L^{-1})b \\ x &= P_L^{-1}b - P_L^{-1}U(S^{-1} + VP_L^{-1}U)^{-1}VP_L^{-1}b \end{aligned}$$

## 2.3 Algebraic Multigrid of Locally Connected Subgraph

The locally connected subgraph is planar, meaning it can be drawn on a piece of paper without any edges crossing (proof it is planar?). Using work beginning with Gary Miller [6], we know that an Algebraic Multigrid solver is optimal for this planar graph laplacian matrix as the solution space is split into multiple cycles of coarsened solving [2]. Previous approaches to using multigrid (LAMG, CMG, Cascadic) to solve Laplacian linear systems do not have a systematic method of graph coarsening. They are based on heuristics for identifying which edges to keep in the multiple levels. Whereas these algorithms are prone to losing edge information in the multiple coarsening cycles, my algorithm runs multigrid only on the planar portion, preserving correct edge information in the coarse levels. (do i need to cite?) There are many multigrid solves in this algorithm, and it is important to optimize performance. Thus I use the Portable, Extensible Toolkit for Scientific Computation (PETSc) and its python library petsc4py to run multigrid solves [1, 4]

## 2.4 Low Rank SVD of Teleportation Subgraph

For graphs I am interested in, the number of edges in the Teleportation subgraph is very small relative to the size of the original graph. This causes the laplacian matrix of teleportation subgraph,  $T_L$ , to have very low rank structure. I can then take the low rank singular value decomposition (cite?) to solve this small portion. Given  $n \times n$  matrix  $T_L$  with rank  $r$ , we compute  $USV = T_L$  where  $T_L$

has  $r$  non-negligible singular values. Thus  $U$  is tall-skinny  $n \times r$ ,  $S$  is an  $r \times r$  diagonal matrix of the non-negligible singular values, and  $V$  is short-fat  $r \times n$ . This low-rank SVD is also important in decreasing the floating point operations in the matrix-matrix, and matrix-vector operations.

## References

- [1] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015.
- [2] Achi Brandt and Oren E Livne. *Multigrid techniques: 1984 guide with applications to fluid dynamics*, volume 67. SIAM, 2011.
- [3] Fan Chung and Linyuan Lu. *Complex Networks*, chapter The Small World Phenomenon in Hybrid Power Law Graphs, pages 89–104. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.
- [4] Lisandro D Dalcin, Rodrigo R Paz, Pablo A Kler, and Alejandro Cosimo. Parallel distributed computing using python. *Advances in Water Resources*, 34(9):1124–1139, 2011.
- [5] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008.
- [6] Gary L Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM Journal on Computing*, 24(5):1002–1017, 1995.
- [7] Duncan J Watts and Steven H Strogatz. Collective dynamics of ‘small-world’ networks. *nature*, 393(6684):440–442, 1998.