ABSTRACT

A Multigrid Solver for Graph Laplacian Linear Systems on Power Law Graphs

by

Eric Buras

The Laplacian matrix, L, of a graph, G, contains degree and edge information of a given network. Solving a Laplacian linear system $Lx = b$ provides information about flow through the network, and in specific cases, how that information orders the nodes in the network. I propose a novel way to solve this linear system by first partitioning G into its maximum locally-connected subgraph and a small subgraph of the remaining so-called "teleportation" edges. I then apply optimal multigrid solves to the locally-connected subgraph, and linear algebra and a solve on the teleportation subgraph to solve the original linear system. I show results for this method on real-world graphs from the biological systems of the *C. Elegans* worm, Facebook friend networks, and the power grid of the Western United States.

Chapter 1

Introduction

In 1999, Stanford graduate students Larry Page and Sergey Brin proposed a novel algorithm for ordering the information on the world-wide-web. How do you know what links an internet user is likely to follow from any given web-page? Given a network (graph) of connections between web pages, Page and Brin proposed solving a simple linear system resulting in a vector of importances of the web pages for the network. With the shift of a minor parameter, the PageRank linear system was born. Given L, a stochastic matrix describing a web graph, b, a distribution vector corresponding to the problem data, and $0 < \alpha < 1$, a damping parameter; solve the linear system [**?**]:

$$(I - \alpha L)x = (1 - \alpha)b$$

for x, the PageRank vector. This solution contains information about the importance of a set of web-pages on the internet. The α parameter is used to introduce likelihood that a user clicks on a new random page [**?**].

While Page and Brin went on to make billions of dollars revolutionizing web-search, their algorithm can be thought of in more generic terms for any network of information. David Gleich highlights other applications of the PageRank linear system in his paper reviewing it's simple mathematics and vast reach into other, completely different topics [**?**]. These include chemical bonding networks, macro and micro biological system networks, roads and infrastructural networks, computer hardware and software networks, author and literature networks, and finally social organization networks [?]. Scientists care deeply about the information inherent in the connections of these networks, and finding a way to order that information in a suitable format. Thus PageRank linear systems become GeneRank [?], AuthorRank [?], or MonitorRank [?] linear systems for information stored in genes, co-authorships, and distributed system logs, respectively [?]. The simplicity of the problem formulation combined with the vast amounts of data in practically any subject area shows how influential PageRank has become.

The purpose of this paper is to propose a new method for solving a simple linear system similar to that of PageRank. By utilizing the structure of a graph of a certain class, I split the graph into a large, highly locally-connected subgraph and a much smaller subgraph of the remaining edges, which we will call *teleportation* edges. While the locally-connected subgraph has large diameter, the *teleportation* subgraph edges can jump randomly from one end of the graph to the other. I solve the overall linear system by optimally solving the local subgraph using the algebraic multigrid (AMG) method, combined with a direct solve for the much smaller system of the teleportation subgraph. The traditional graph coarsening algorithms used in AMG are applicable for the locally-connected subgraph because this looks like a geometric grid. The small size of the teleportation subgraph is important as the method complexity depends on more costly dense matrix operations.

Chapter 2

Background

2.1 Graph Laplacian Problem

Information about a weighted, undirected graph G with vertex list V and edge list E can be stored in a matrix. This is called the adjacency matrix which is defined as:

$$A(u,v) = 1 \text{ if } (u,v) \in E \text{ and } 0 \text{ otherwise.}$$

The diagonal matrix, D is a matrix of the degree sequence along the diagonal:

$$D(u,u) = d_u$$

The Laplacian matrix is $L = D - A$. It is called this because it is similar to the finite difference discretization of the Laplacian on a grid.

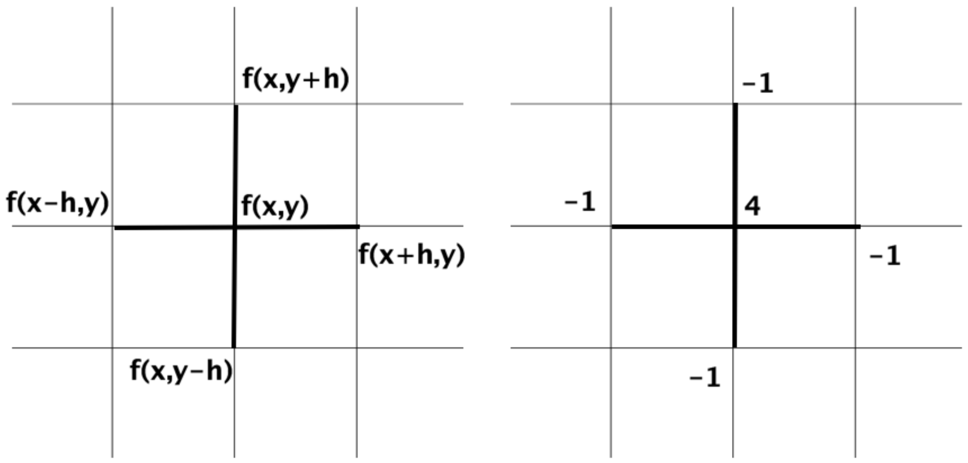


Figure 2.1 : Finite difference discretization on a grid

The 3D finite difference discretization is similar because each edge contributes two terms: one to the adjacency matrix and one to the diagonal matrix. This is how we build multidimensional graph Laplacians. To see how information flows through paths in a network, the graph Laplacian matrix can be used as an operator on an input vector. To find a weighted average of paths in a network one can repeatedly apply the Laplacian operator on an input. This results in a Neumann Series with inverse defined if the series converges in the operator norm [?, ?]:

$$b + Lb + L^2b + L^3b + L^4b + ... = \sum_{i=0}^{\infty} L^i b = (I - L)^{-1}b$$

Thus in problems related to graph regression, spectral graph theory, maximum and minimum cost flow, resistor networks, and partial differential equations it is common to use the inverse operator of the graph Laplacian [?].

2.2 Current Solution Approaches

Solution algorithms for linear systems can be divided into direct methods and iterative methods. Standard direct methods such as LU or Cholesky factorization are accurate and suitable for graphs with small numbers of edges/vertices, however become very costly in terms of memory and time as the size and edge density of the graph increases. Fast matrix inversion can be applied with order $O(N^{2.376})$ [?]. Alternatively one can use nested dissection, however, computational memory requirements blow up as size increases; 2D meshes require $O(NlogN)$ memory, 3D requires $O(N^{\frac{4}{3}})$ memory and the requirements are greater for highly connected power-law graphs [?]. In contrast to direct solvers, iterative methods compute better and better approximate solutions to the linear system. A standard iterative method is the Conjugate Gradient method [?]. To speed up these iterative methods, it is possible to introduce a preconditioner that creates an equivalent linear system that is much easier to solve than the original [?]. Yet none of these basic solvers take into account attributes of the graph Laplacian that can drastically improve method performance.

2.2.1 Spielman-Teng, Koutis et. al.

As an analogy with preconditioning, we want to find an approximation to a graph G with similar spectrum for easier computing. Spielman and Teng introduce the idea of a spectral sparsifier. The Laplacian for this approximation is similar to the Laplacian for the original graph because of their similar spectrums, thus resulting in a good preconditioner for the original linear system. Multiple cycles of sparsification and factorization combine to solve the original problem. They have a multilevel version of this. Spielman and Teng (S-T) were thus able to solve symmetric diagonally dominant systems in nearly linear time [?]. This line of work combined with Vaidya's [?] work on subgraph preconditioners (capacitance matrix methods) resulted in Koutis and Miller's work solving linear systems based on planar Laplacians [?]. Koutis, et. al. were able to extend the results to general symmetric diagonally dominant systems [?].

2.3 Multigrid Approaches

Algebraic multigrid (AMG) creates a coarse basis through agglomeration and then uses a Galerkin operator in multiple graph coarsening cycles to solve a linear system. It is mostly used to solve discretized partial differential equations, but has also become more popular in solving graph Laplacian systems. AMG can have optimal time complexity and demonstrates good parallel scaling, thus it is useful for solving incredibly large problems [?]. Straightforward AMG on these graphs is not optimal because it cannot coarsen the graphs fast enough. So three multigrid approaches to the graph Laplacian problem with heuristics for coarsening are combinatorial multigrid (CMG) from Koutis et. al. [?], Cascadic multigrid from Urschel et. al. [?], and Lean Algebraic Multigrid (LAMG) from Livne and Brandt [?]. All three propose coarsening over the entire graph with heuristics for aggregating edge information. But an important question to ask is: how do you coarsen a graph with edges of varying degrees? How do you know which edges or vertices can be aggregated and still preserve information?

2.3.1 Combinatorial Multigrid (CMG)

Koutis, Miller, and Tolliver propose a combinatorial multigrid solver to solve computer vision problems. This method creates a two-level iterative approach combining the previously mentioned subgraph preconditioning work of Vaidya with algebraic multigrid. For a set of increasingly larger three-dimensional images, CMG required less iterations to converge than a standard multigrid solver in Matlab [?].

2.3.2 Lean Algebraic Multigrid (LAMG)

Livne and Brandt ran a 'lean' multigrid algorithm on graph Laplacian systems for almost 4000 real world graphs of varying size and in vastly different fields from the natural sciences to social networks. Their method has three key parts: first, vertices with low degree are eliminated before the graph is coarsened. Second: they aggregate vertices for the coarsening by a proximity heuristic. And third: they apply an energy correction to the Galerkin operator and take linear combinations of solutions to accelerate the iteration. They test their algorithm against CMG, and find that it requires slightly more work, however is more robust overall [?]. One potential downside of LAMG is the heuristics used in the vertex aggregation step. How can vertices be evenly aggregated over graphs with uneven degree distribution?

2.3.3 Cascadic Multigrid

A final alternative form of multigrid was proposed by Urshel et. al. to solve a related problem to the graph Laplacian linear system; they wanted to calculate the Fiedler vector (eigenvector corresponding to the second smallest eigenvalue) of a graph Laplacian. This cascadic multigrid utilizes heavy edge matching to quickly coarsen a graph [?]. It remains to be seen whether this approach will be successful for solving an entire Laplacian linear system accurately.

2.4 Graph Partitioning and Multigrid

I have studied the methods of graph partitioning and sparsification for solving linear systems, and using multigrid to solve the graph Laplacian linear system specifically. I combine these two approaches using Chung and Lu's work [**?**] and optimal multigrid to solve similar linear systems.

Chapter 3

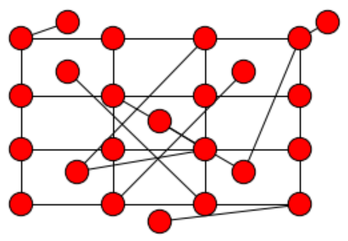
Methodology

3.1 Graph Partitioning

Chung and Lu study ?small world? networks from Watts and Strogatz, specifically graphs which exhibit power law decay in the degree sequence [**?**], and propose an algorithm that separates a graph into a locally-connected component and a teleportation component. This breaks a graph into a group of edges between highly connected nodes with many paths between them, and a remainder graph which is much more sparsely connected and yet of small diameter. Given integers $k \geq 2$ and $l \geq 2$, a (k, l) locally-connected graph will have at least l paths connecting any given pair of nodes where the edges are distinct 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. I set these values for k and l for the remainder of my work. Given graph, G, its maximum k, l-locally connected subgraph is the union of all k, l-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 [**?**]. 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 Partitioning Algorithm

```
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
$P$ is the maximum $k, l$-locally connected subgraph of G.
```


(a) G, grid with random edges

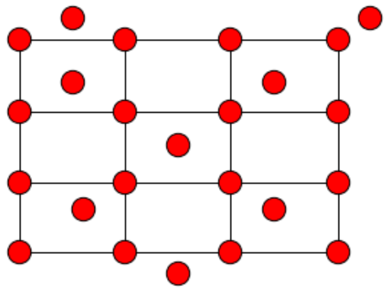

(b) P, max (3,3)-l. c. subgraph of G

Figure 3.1 : Partitioning algorithm results in planar locally connected subgraph

3.1.2 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 edgelists, computing node degrees, converting graphs to Laplacian matrices, etc. which are incredibly useful for simple graph work [?]. Included in this package is a function for finding the locally connected subgraph of a graph based upon the work of Chung and Lu above [?]. Throughout much of the course of my work I used this function to partition my graphs. However for graphs with many edges, this function is inefficient because it utilizes a shortest path algorithm [?] to test each edge's connectivity. This algorithm has similar complexity for any given k, l connected subgraphs, however I only care about $k, l = 3$. Thus I wrote an optimized code that simply searches through all potential paths of length 2 and 3 in the graph. This requires orders of magnitude less work than running a shortest path algorithm for each edge. 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 refine and harden my code, and ultimately contribute it to the NetworkX package. The existing function is still important for computations with arbitrary k and l, however, and I hope to repair the errors discovered in the function.

3.2 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. Because P_L is singular, I add the diagonal of T_L to P_L, and thus have $L = P_L + T_L$ where L is the Laplacian for the entire graph and P_L and T_L are slightly altered versions of the Laplacians. To solve a Laplacian linear system $Lx = b$, I now solve the two subgraph Laplacians and use linear algebra.

3.2.1 Linear Algebra: Woodbury Matrix Identity

I use 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 I will assume that P_L is very large and sparse, and T_L is very sparse and low rank. I will examine the conjecture that T_L is low rank in Section 4.2.3. Here is a breakdown of how I solve the Laplacian linear system:

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

3.2.2 Algebraic Multigrid of Locally Connected Subgraph

The locally connected subgraph has a large diameter and resembles the grid graph in some dimension if drawn a certain way in space. In 2 dimensions, the maximum locally-connected subgraph for $k, l = 3$ looks planar. This would mean that it can be drawn on a piece of paper without any edges crossing. Using work beginning with Gary Miller [?], we know that an algebraic multigrid (AMG) solver is optimal for a planar graph Laplacian matrix as the solution space is split into multiple cycles of coarsened solving [?]. Thus AMG would be a natural choice for P_L. 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 locally-connected portion, preserving edge information in the coarse levels. 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 [**?**, **?**]

3.2.3 Low Rank SVD of Teleportation Subgraph

The Woodbury matrix identity requires use of the singular value decomposition (SVD) for the teleportation Laplacian T_L. 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. Currently I am taking the full-rank SVD of T_L and removing the columns of U and rows of V that correspond to negligible singular values. 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 $r \times n$. The full rank SVD has $O(n^3)$ complexity and dominates the complexity of the entire method. However, in future work I can optimize the SVD to account for the low-rank structure of T_L. This change will be very valuable in solving many-node networks.

3.3 Model Complexity

3.3.1 Graph Splitting

The small-world networks that Watts-Strogatz and Chung-Lu worked with roughly follow a power law degree distribution where the proportion of nodes with d degree scales with $d^{-\gamma}$ with $2 < \gamma < 3$ [?, ?]. In lay terms, there are a few nodes with high degree and many nodes with low degree (≤ 3 e.g.). This is similar to an exponential distribution, however a power law distribution has a much longer and fatter tail. Given probability density function of the degree distribution: $P(d)$, we can compute the expected degree in a sequence $E[d]$ with $\zeta(x)$, the Riemann zeta function:

$$P(d) = \frac{d^{-\gamma}}{\zeta(\gamma)}, \zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$
$$E[d] = \sum_{d=1}^{\infty} d \frac{d^{-\gamma}}{\zeta(\gamma)}$$
$$E[d] = \frac{\zeta(\gamma - 1)}{\zeta(\gamma)}$$

As γ increases from two to three, the expected value of the distribution decreases. For most realistic values of γ, the expected value is small, and this is why they are sometimes described as graphs with "preferential attachment"; edges are added between nodes that already have many edges [?]. It is important to note that graphs with finite nodes can be described with $\gamma \leq 2$. The degree decay of these distributions is stretched. These stretched power law distributions do not have an expected value because they are not well-defined for node numberings approaching infinity, however an average value can be calculated over a set of finite nodes. This might be a more appropriate fit for datasets described in the Results section.

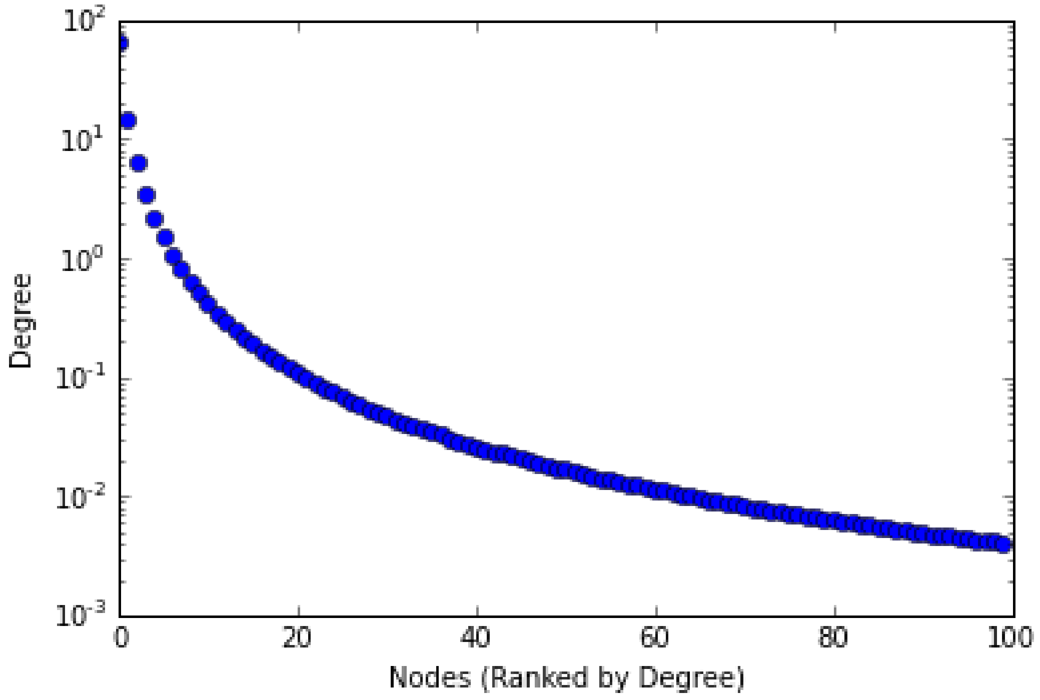


Figure 3.2 : Power-law distribution (pdf) of degree sequence multiplied by the number of nodes with $\gamma = 2.1$. Note the rapid degree decay after the first 5 nodes. Integer rounding of the degrees could simulate a degree distribution.

For now, let's assume our degree sequence follows the original power law distribution. To partition the graph using Chung-Lu for $k = 3$ and $l = 3$, for each node we must search through all paths of lengths 1, 2, and 3 away from the node. We only continue if path of length 1 exists. If so, we take the expected value of the degree sequence and raise it to the second power to give number of paths of length 2. We do it similarly for the third power. Thus, the expected work is $O(E[d]^3)$. Then it is simple to count the number of edge-disjoint paths. If there are two or fewer edge-disjoint paths, the edge (path length 1) is removed. For a graph with m edges, this requires $m * ((E[d])^3)$ searches for one iteration of the cycle. For computational complexity of real-world graphs, simply replace the $E[d]$ value with the calculated average degree. We continue iterating this cycle until no edges can be removed. I do not have a bound on the number of iterations required to remove all teleportation edges, however in experiments with real-world graphs, it seems they require three iterations at most (including the final iteration where no edges are deleted).

3.3.2 Linear Algebra and Solves

After splitting the entire n-vertex graph into locally-connected subgraph and teleportation subgraph, we must count the number of floating point operations (FLOPs) for each operation in the method. Listed are the operations with flop count for $n \times n$ matrices P_L and rank-r T_L, $n \times r$ matrix U, $r \times r$ matrix S, $r \times n$ matrix V, and $n \times 1$ vector, b:

Operation	**O(FLOPs)**
$USV = T_L$	$O(n^3)$
S^{-1}	$O(r)$
$y = P_L^{-1}b$ (MG)	O(n)
$y_1 = Vy$	$O(rn)$
$Q = P_L^{-1}U$ (r MG solves)	$O(rn)$
$Q_1 = VQ$	$O(r^2n)$
$Q_2 = S^{-1} + Q_1$	$O(r^2)$
$y_2 = Q_2^{-1}y_1$	$O(r^3)$
$y_3 = Uy_2$	$O(rn)$
$y_4 = P_L^{-1}y_3$ (MG)	$O(n)$
$x = y - y_4$	$O(n)$

In summary: given $n \times n$ graph Laplacian matrix and rank-r teleportation Laplacian matrix, the method is $O(n^3)$, however with work to implement a rank-revealing SVD, it can be decreased to depend on r. Usually, the complexity of the method would depend on the order of the linear system solves. However, I can obtain optimal work complexity with multigrid solves on the locally-connected subgraph Laplacian, P_L, because I know how to get sufficient coarsening ratios for the coarse solves.

It is clear that outside the SVD, the complexity of the solve depends on r, the rank of the teleportation matrix T_L. We will show in the results section how the theoretical complexity of the algorithm compares to the calculated timings for the graph partitioning and each operation in the linear system solve.

Chapter 4

Results

4.1 Graphs

We have a set of graphs that are similar to the power-law graphs from Watts-Strogatz and Chung-Lu [?, ?]. These encompass a wide range of subjects from biological processes, social networks, and urban infrastructure. To standardize the datasets, I coerced them into undirected, unweighted networks, however I can solve a similar linear system if they are in their original form. In addition to a short description of the datasets and what it means to solve their Laplacian linear system, there is an appendix with figures for each graph: a spy plot of the Laplacian matrix, a degree histogram, a simple network graphic to illustrate connections in the data, and a plot of the singular values of T_L to show its low-rank structure.

4.2 *C. Elegans* Worm Data

The *C. Elegans* worm has been studied extensively due to the low complexity of it's biological systems. The smallest graph in this results section is the completely mapped nervous system of 297 neurons in the worm with 2148 connections from initial experimental data given by White et. al. in 1986 [?, ?]. Solving the Laplacian linear system of a neural network is analogous to finding the PageRank vector of influences for each neuron. The neural edge information can be combined with anatomical information about brain regions to study which parts of the brain are more important for different functions. Another possible study identifies how an organism's brain of an organism develops as it ages [?].

The second *C. Elegans* dataset describes the 453 metabolic processes and their 2025 connections that comprise the metabolism of the worm [?]. Again, this is a simple network of a simple organism. Solving the Laplacian linear system can offer insights into the key metabolic processes.

The final *C. Elegans* dataset is an order of magnitude larger than the previous two. The worm's genetics encode proteins that ultimately display phenotypes. Networks of this type are described as interactomes. The graph I study has 912 proteins with 22,738 edges for protein-protein interactions with corresponding, scientifically-observed phenotypes [?]. Solving the Laplacian linear system for a protein network can highlight unobserved, tangential protein-phenotype relationships [?].

While the *C. Elegans* worm is an incredibly, biologically simple organism, it is useful as a test case for solving much larger linear systems for more complex organisms. The ultimate goal, of course, to map the biological systems of humans. Already, work is being done to map the human interactome [?, ?] and parts of the human neural network [?]. All three of the worm networks are comprised of a highly locally-connected subgraph and small teleportation subgraph making them amenable to my method.

4.2.1 Facebook Friend Circles

The largest graph I work with contains the 88,234 Facebook friend connections between 4039 users [?]. Social scientists are interested in identifying sources of influence in a social network. By solving the Laplacian linear system associated with the Facebook graph I can solve a so-called 'reverse' PageRank system. This finds the origins of influence; for a circle of Facebook friends, it might identify how those people became connected through a few key people. This can be applied to any network of human interactions including other social networks, or organizational systems [?]. This graph is also highly locally-connected, and is a good example of how my method scales to larger networks.

4.2.2 Western US Power Grid

The final dataset illustrates how my method fails. The Western US power grid has 4941 nodes and 6594 connections [?]; this is a much lower edge/node ratio than the previous datasets and explains why this graph is not very locally-connected. Whereas the previous 4 examples resulted in low-rank teleportation Laplacian matrix, T_L for the power grid is almost full-rank. As seen further, the resulting linear system solve requires way too many floating point operations. Solving this system can describe the flow of electricity through the grid [?], however my method clearly is not appropriate. Despite this particular power-grid graph failing, those who supplied the data may have removed much of the locally-connected graph prior to publication. The edgelist may only contain major power lines, whereas a full dataset would have many many more minor nodes and edges.

4.2.3 Graph Overview

The key part of my algorithm is partitioning a graph into a large locally-connected subgraph and a small teleportation subgraph. Four of these examples fully fit into this class of graphs, whereas the power grid most certainly does not as evidenced below. The important attribute to look for is low rank of the teleportation Laplacian matrix relative to the number of the nodes (which is the size of the entire square Laplacian matrix). Here is a table summarizing the datasets:

Graph (nodes, edges)	Rank T_L	Rank/nodes
Neural (297, 2148)	22	.0741
Metabolic (453, 2025)	49	.1082
Protein (912, 22738)	26	.0285
Facebook (4039, 88234)	180	.0446
Power (4941, 6594)	4284	.8670

4.3 Graph Partitioning

Partitioning the complete graph into the maximum locally-connected subgraph and the teleportation graph is the unique part of this algorithm. This requires an initial computational step as an overhead cost. Any graph must only be partitioned once, as the unique edgelists can be written into files for future use. In the methodology section I mentioned that each graph follows a power-law degree distribution with $2 < \gamma < 3$ which affects its expected degree. However, for finite node graphs, γ can be less than 2 and the degree sequence decay is stretched. Let's take a look at the degree histrogram for the *C. Elegans* neural network compared to a typical power-law degree distribution and a stretched power-law distribution:

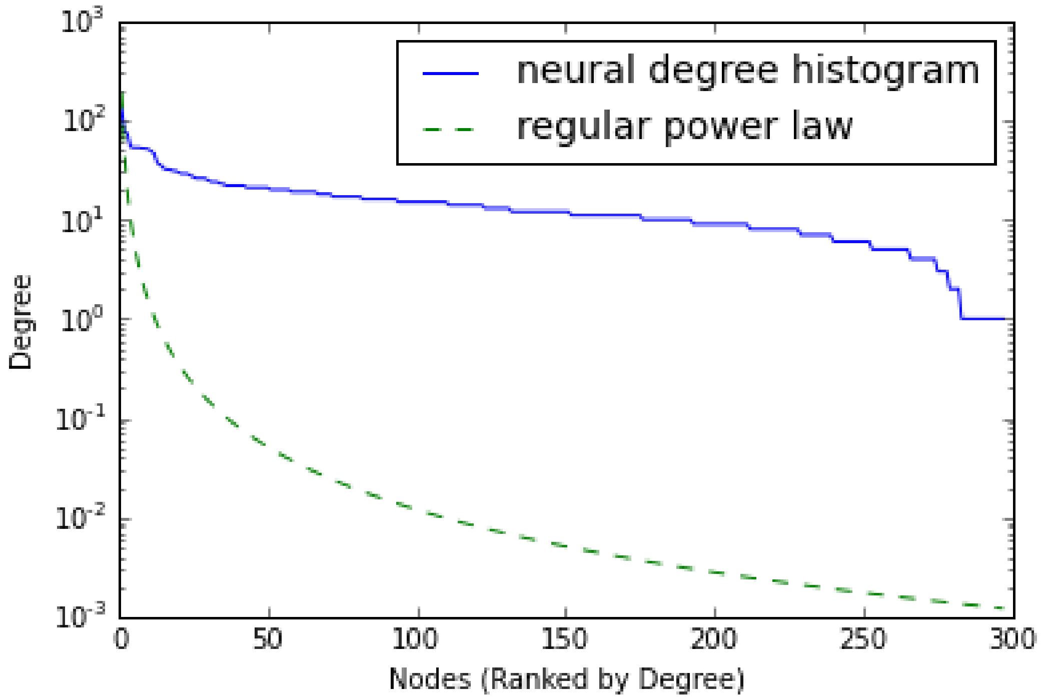


Figure 4.1 : Degree sequences of Neural Network and standard power law distribution ($\gamma = 2.1$)

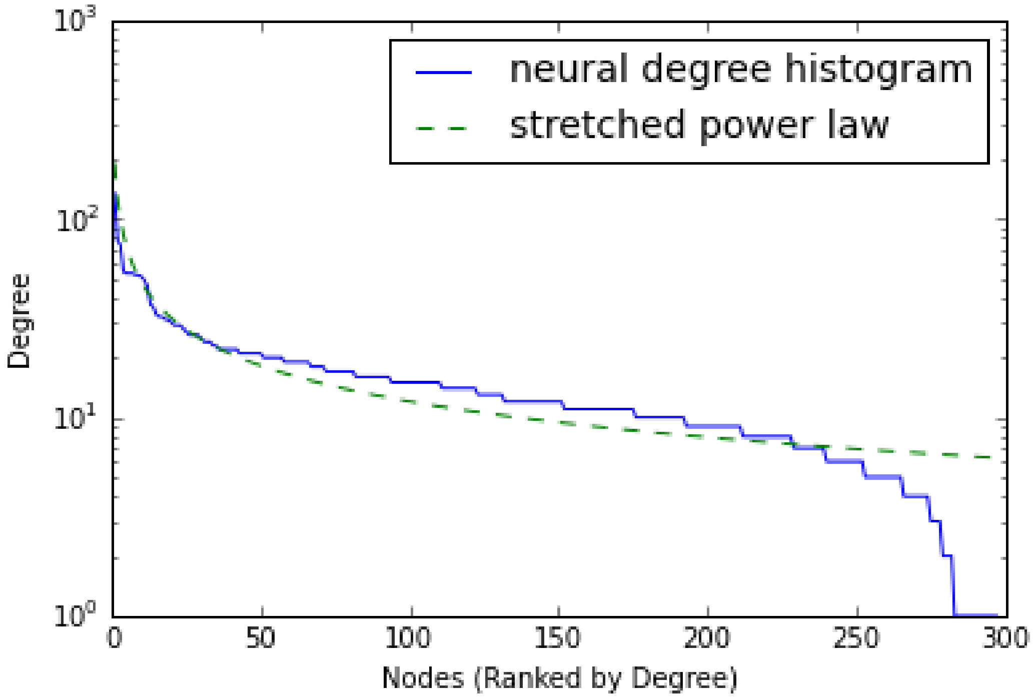


Figure 4.2 : Degree sequences of Neural Network and stretched power law distribution ($\gamma = .6$ in numerator)

Clearly the stretched power-law distribution fits the data better. The finite node power-law distribution has $E[d] \approx 1$, the stretched power-law distribution has $E[d] = 14.39$ and the average degree of the neural network is 14.46. This matters because it affects the complexity of the partitioning algorithm which has $O(N_I \times m \times (E[d]^3))$. I want to more accurately predict the computational cost of the partitioning algorithm.

4.3.1 Table of Partitioning results

Graph (V, E)	Avg. Deg.	Nx Part. (s)	Part. (s)	N_I
Neural (297, 2148)	14.46	11	1.6	3
Metabolic (453, 2025)	8.94	12	2.3	3
Protein (912, 22738)	49.86	1915	53	2
Facebook (4039, 88234)	43.69	11593	480	3
Power (4941, 6594)	2.669	2.1	.33	3

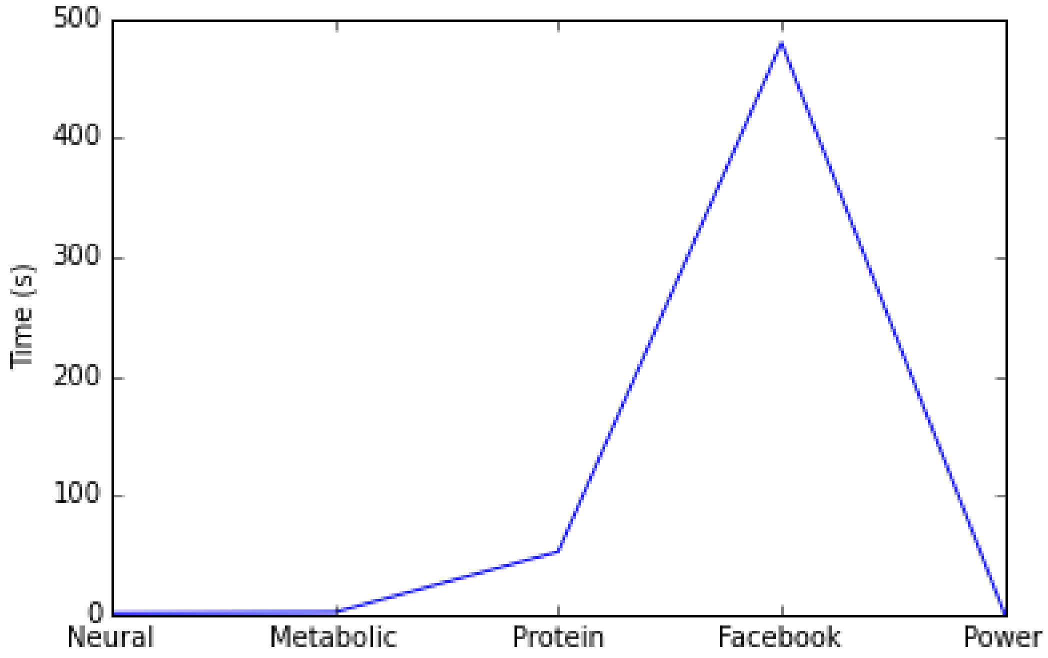


Figure 4.3 : Partitioning graph times. Note how the time increases by the third power for the first four. The Partitioning drops for the power grid with low average degree.

The partitioning times are in line with the theoretical complexity except for the metabolic network which has a similar number of edges but lower average degree than the neural network, and takes longer to partition. I hypothesize that because the graph is double the size, there is an additional overhead to using the NetworkX functions to create pathways and loop through all possible edges. I add the NetworkX partitioning times (Nx Part. (s)) to show how much more computation the NetworkX partitioning requires with the shortest path algorithm. Granted, searching through increasing path lengths requires exponentially more work, so this is probably infeasible for increasing k beyond 5. In addition I show the number of iterations of edge deletion, N_I, until the maximum locally-connected subgraph is recovered.

4.4 Linear System Solve

I want to dig deeper into the solve portion to determine if the operations correspond with their given theoretical complexities. Here is a table of the timings for the individual operations:

Operation	$O(r,n)$	**Neur.**	**Meta.**	**Prot.**	**FB**	**Pow.**
$USV = T_L$	$O(n^3)$	.0334	.0737	.4183	38.47	72.86
S^{-1}	$O(r)$	.0005	.0007	.0001	.0023	5.104
$y = P_L^{-1}b$ (MG)	$O(n)$	.0857	.0962	.3552	1.347	.1152
$y_1 = Vy$	$O(rn)$	.0013	.0015	.0002	.0023	.0702
$Q = P_L^{-1}U$ ($r\times$MG)	$O(rn)$	.3292	.7360	.5190	23.11	106.9
$Q_1 = VQ$	$O(r^2n)$	.0006	.0036	.0013	.4124	285.1
$Q_2 = S^{-1} + Q_1$	$O(r^2)$	.0012	.0018	.0003	.0011	.4157
$y_2 = Q_2^{-1}y_1$	$O(r^3)$	.0023	.0025	.0013	.0099	86.49
$y_3 = Uy_2$	$O(rn)$	.0001	.0002	.0003	.0025	.1867
$y_4 = P_L^{-1}y_3$ (MG)	$O(n)$	.0128	.0070	.1183	.8249	.0059
$x = y - y_4$	$O(n)$	.0003	.0003	.0003	.0004	.0003
Total	$O(n^3)$	**.51**	**.966**	**1.44**	**64.46**	**560**

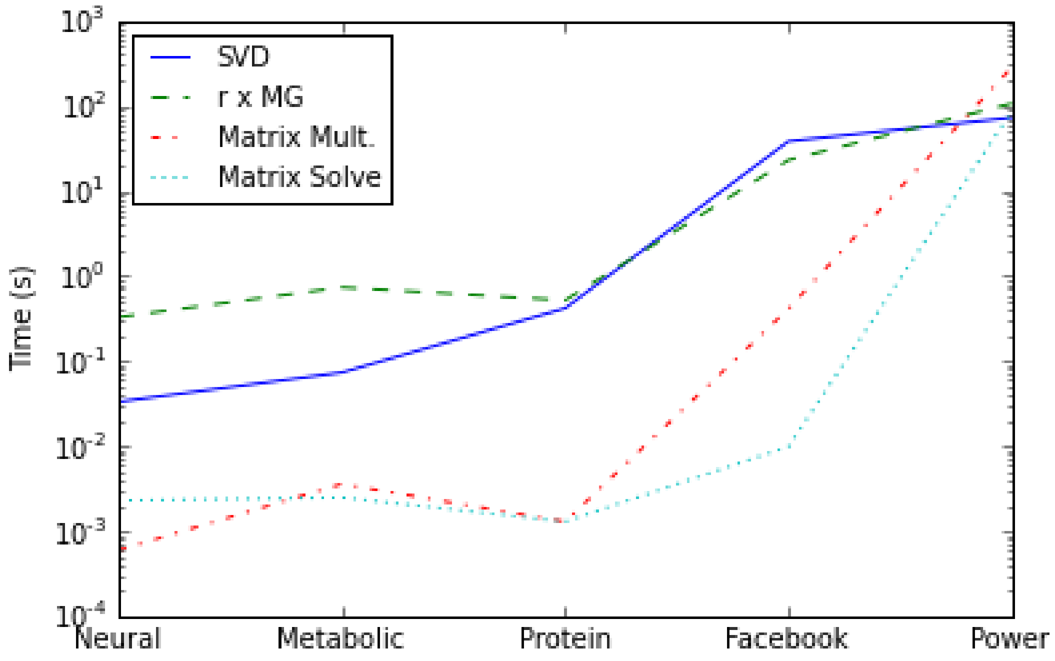


Figure 4.4 : Solve times for the limiting operations. Note: does not scale for full-rank power grid graph

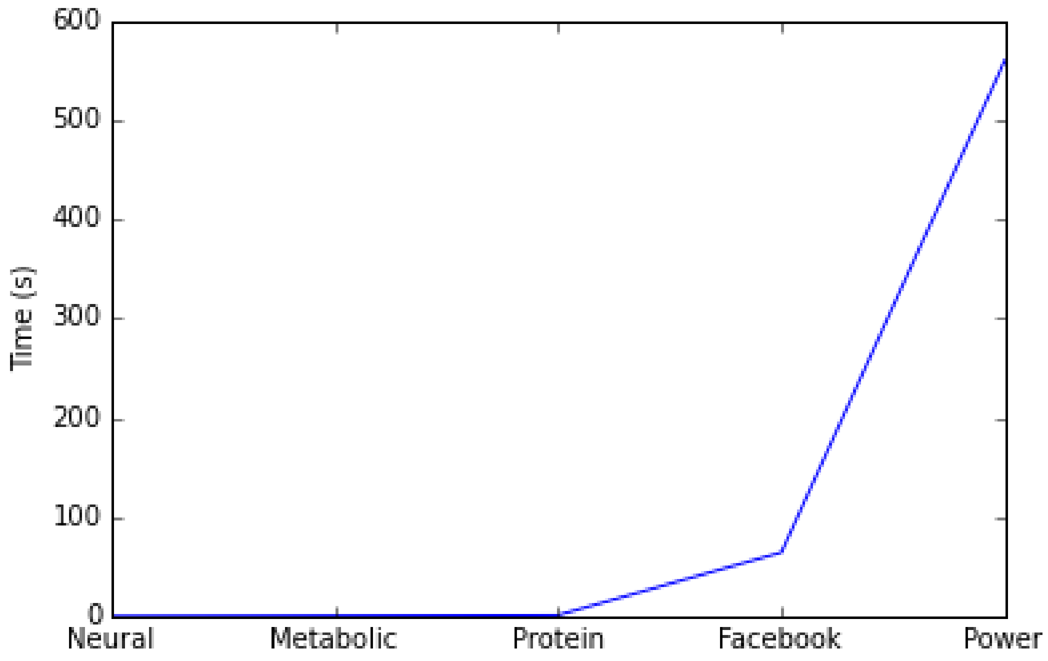


Figure 4.5 : Linear system solve times.

For the first four examples (not including the power grid solve), the limiting operations are the singular value decomposition and multiple right hand side multigrid solves, $Q = P_L^{-1}U$.

4.4.1 SVD and Multiple Multigrid Solves

The SVD is an efficient operation that allows for peak performance (work done per data input) [?]. Thus it is beneficial to have this as a limiting operation. There are alternative algorithms for computing the SVD such as an analogous method to randomized QR [?] that are faster. In addition, it is possible to utilize the low-rank structure of T_L to further decrease computation. I tried removing parts of the SVD corresponding to non-negligible singular values but found that even the removal of one singular value causes large error in the final solution. It is also ideal to be constrained by a multiple right-hand-side multigrid solve step. By vectorizing the multiple right-hand-sides I can achieve more computational work per multigrid solve. This combined with low-rank aspect will greatly speed up the overall solve routine.

4.4.2 Power Grid

However for the full rank power-grid solve, the limiting operations are the SVD, the matrix-matrix multiplication, $Q_1 = VQ$, and the matrix solve, $y_2 = Q_2^{-1} y_1$. These will not scale well for larger graphs without the low-rank structure of the teleportation subgraph Laplacian matrix. Thus graphs with low average degree, large V, and lacking a large locally-connected portion should not be used for solving Laplacian linear systems with my method.

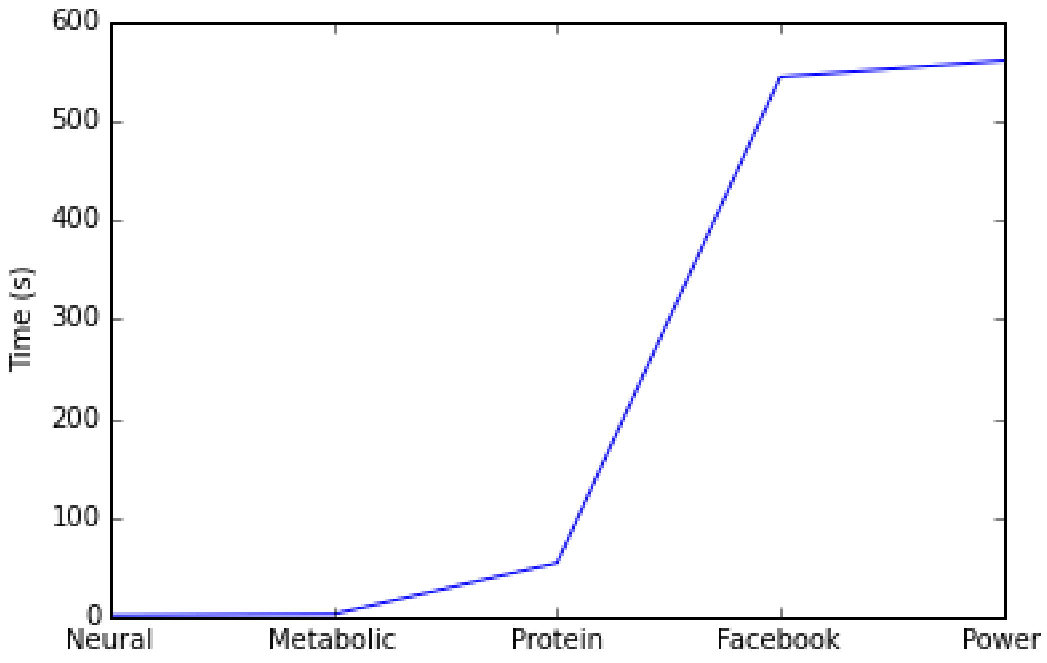


Figure 4.6 : Total solve time for each graph Laplacian linear system.

4.5 Dataset Figures

4.5.1 *C. Elegans* Neural Network

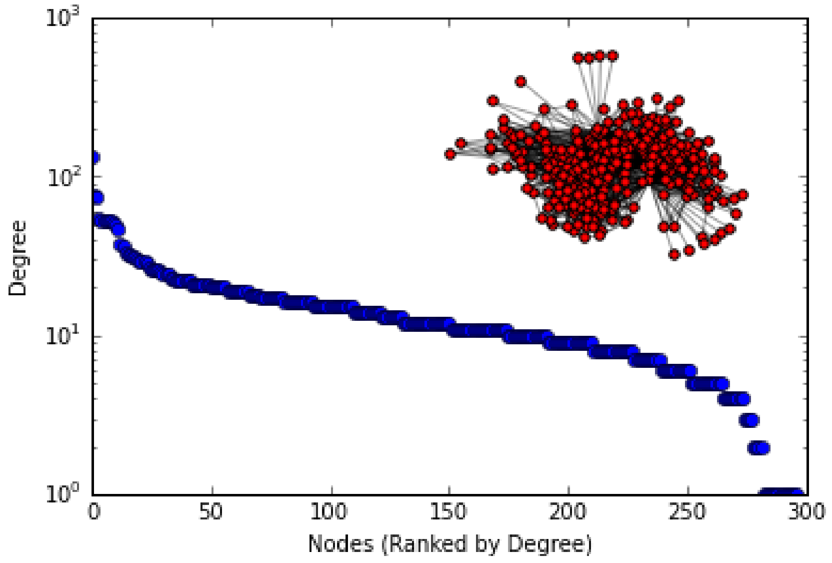


Figure 4.7 : Neural Network of *C. Elegans*

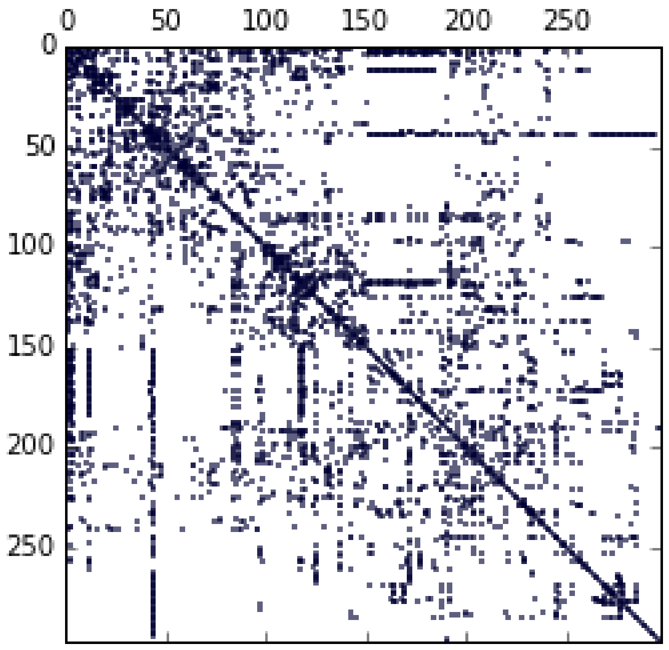
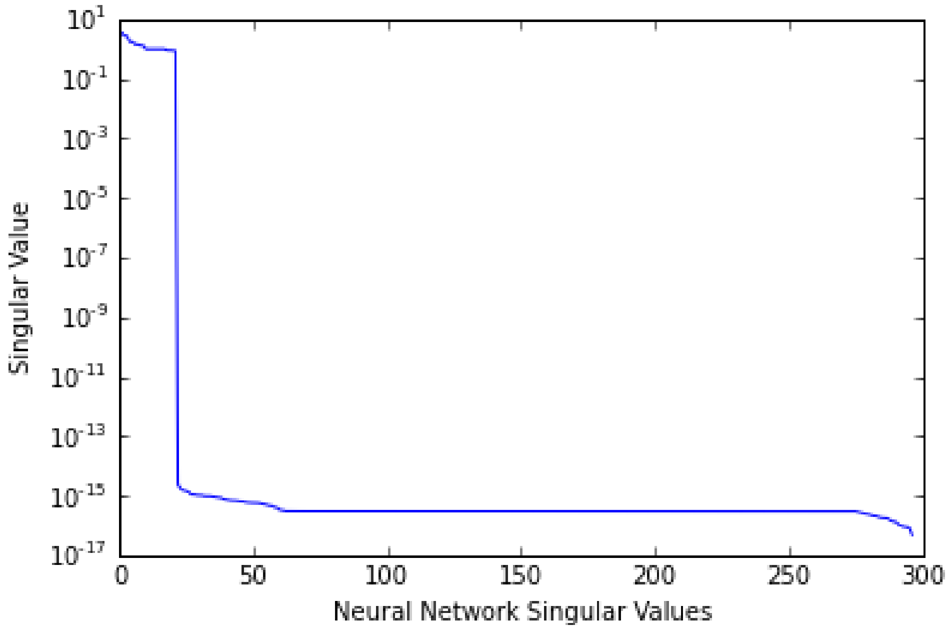


Figure 4.8 : Spy Plot of Neural Network Laplacian Matrix



4.5.2 *C. Elegans* Metabolic Network

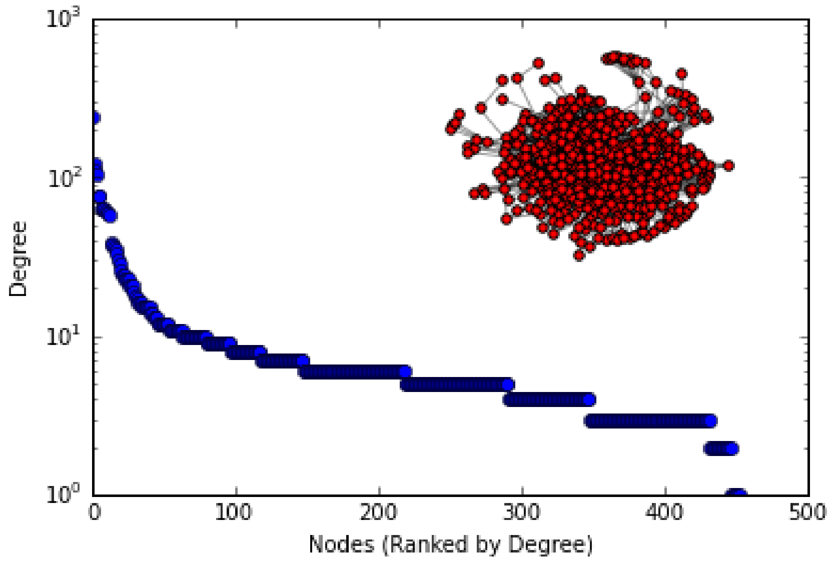


Figure 4.9 : Metabolic Network of *C. Elegans*

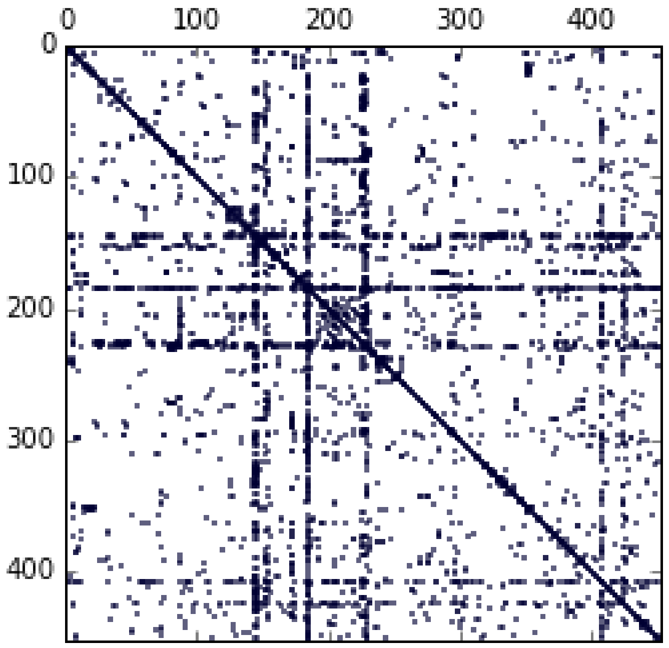
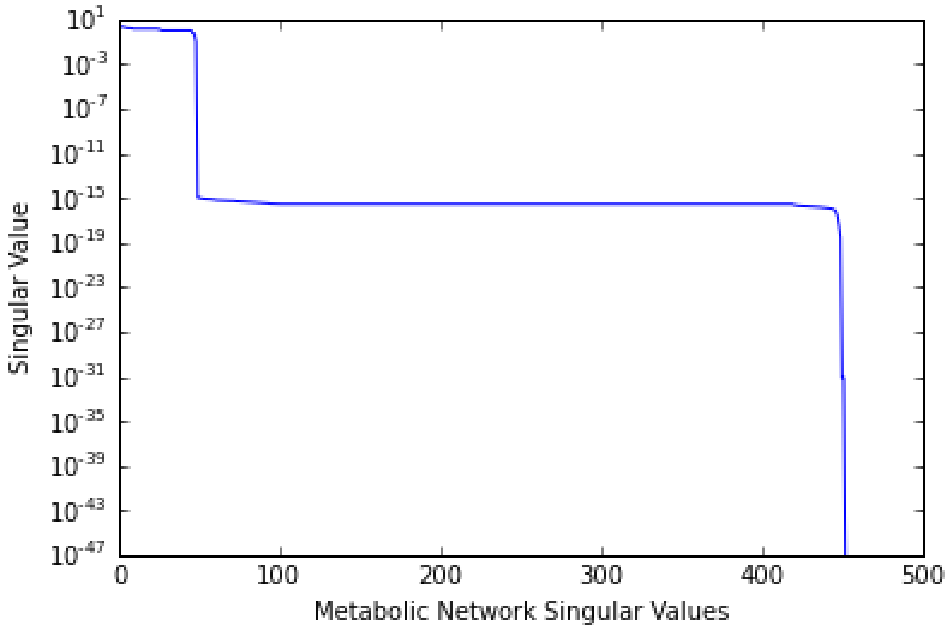


Figure 4.10 : Spy Plot of Metabolic Network Laplacian Matrix



4.5.3 *C. Elegans* Protein Network

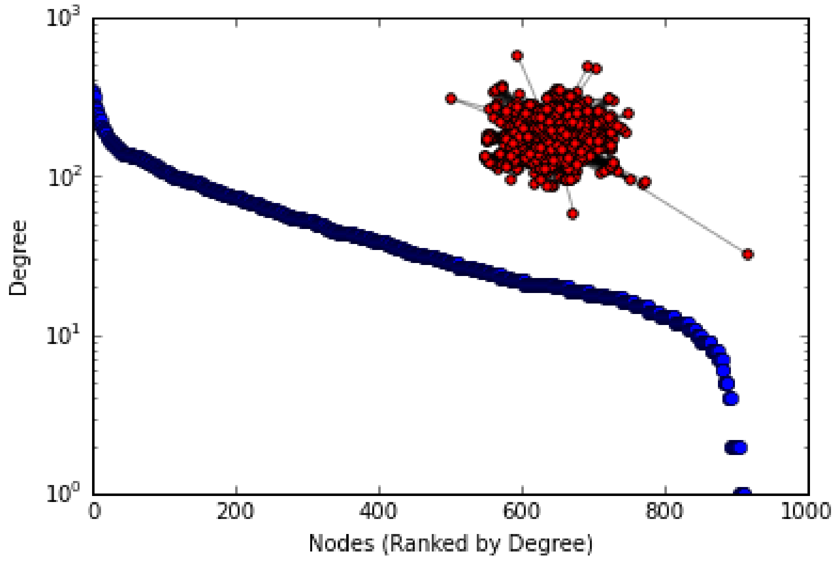


Figure 4.11 : Protein Network with Corresponding Phenotypes of *C. Elegans*

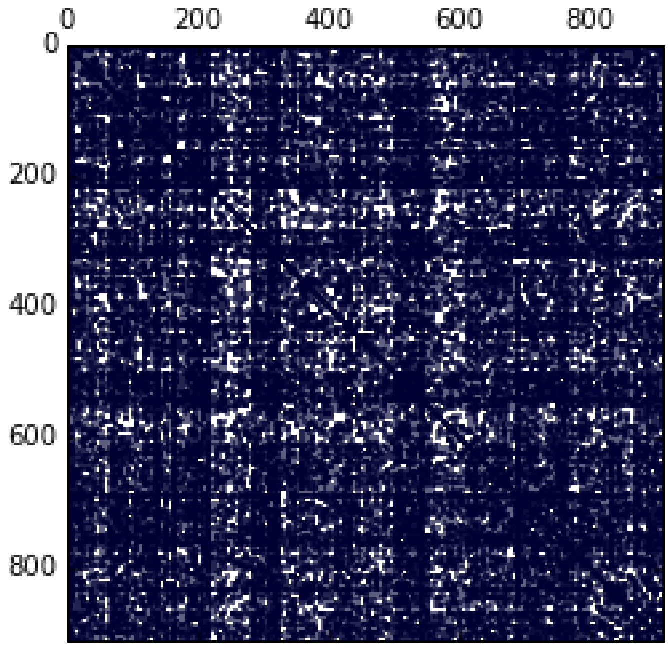
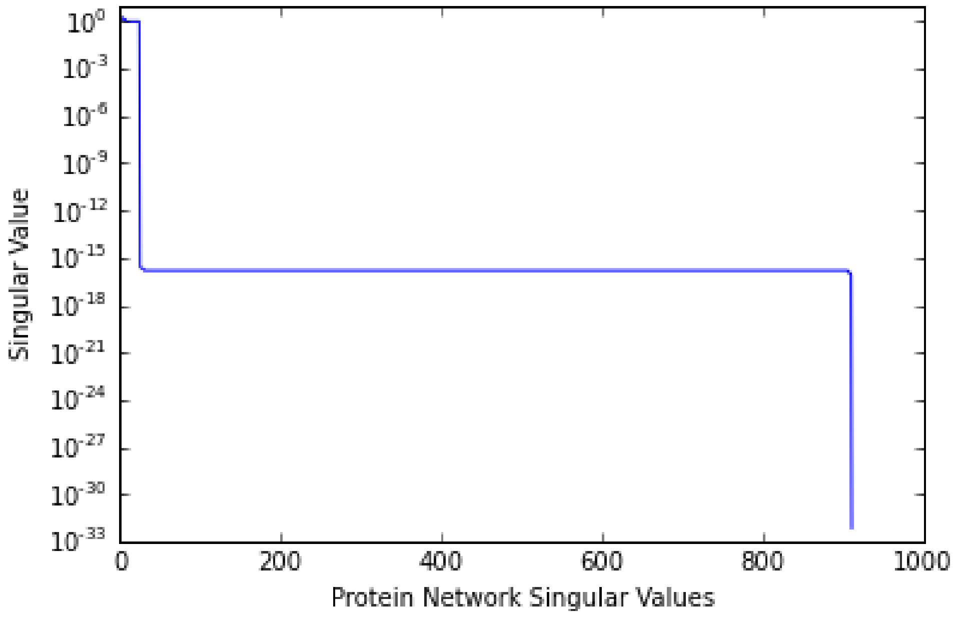


Figure 4.12 : Spy Plot of Protein Network Laplacian Matrix



4.5.4 Facebook Friend Network

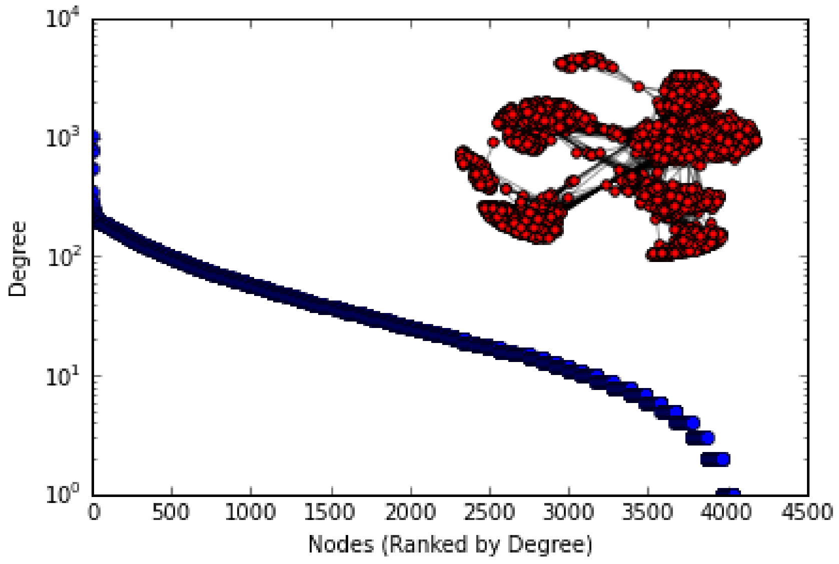


Figure 4.13 : Facebook Friend Network

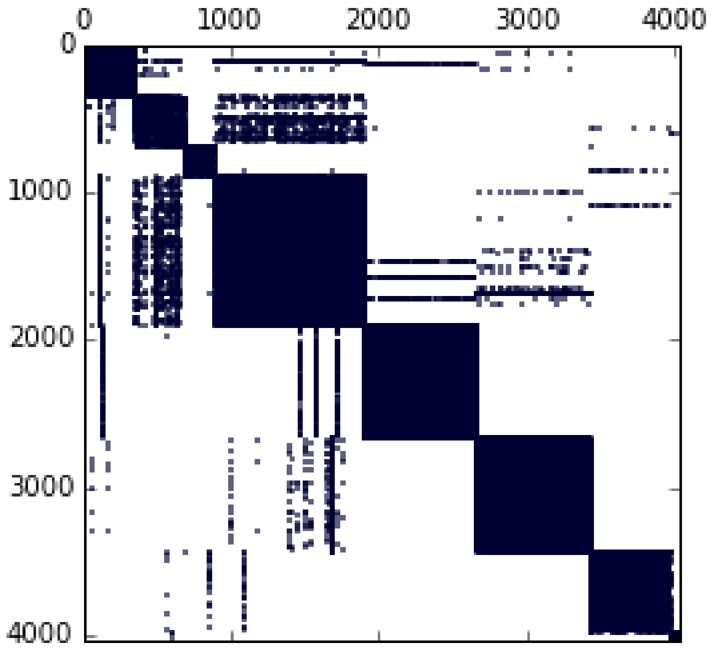
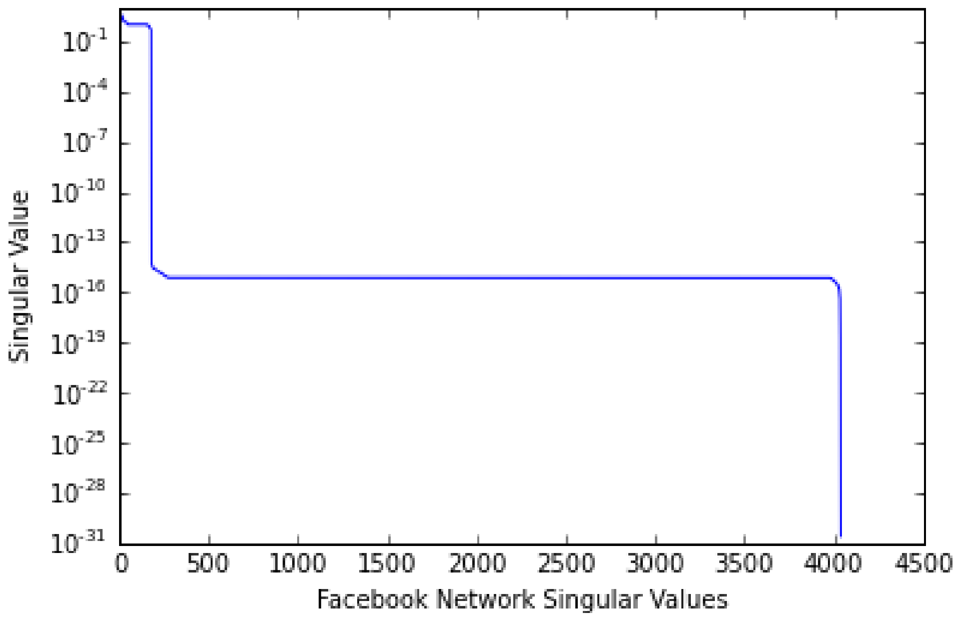


Figure 4.14 : Spy Plot of Facebook Network Laplacian Matrix



4.5.5 Power Grid

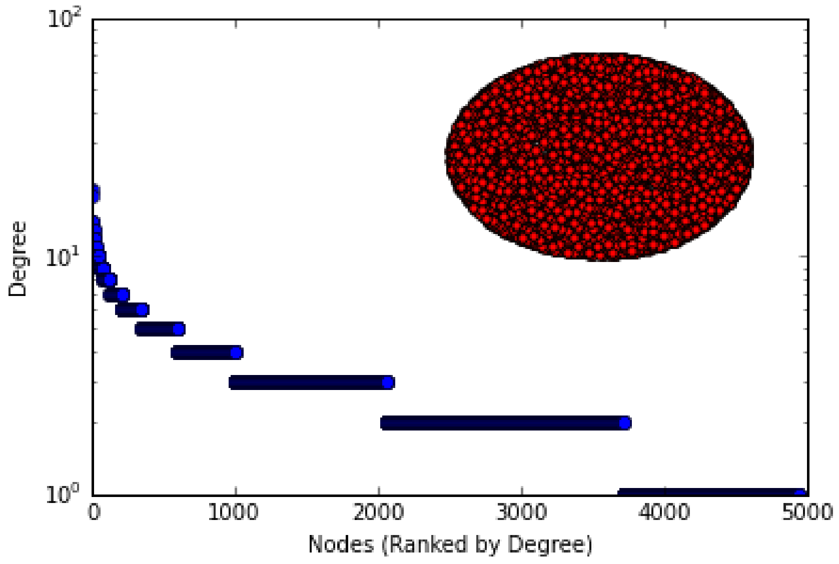


Figure 4.15 : Network of Western Power Grid

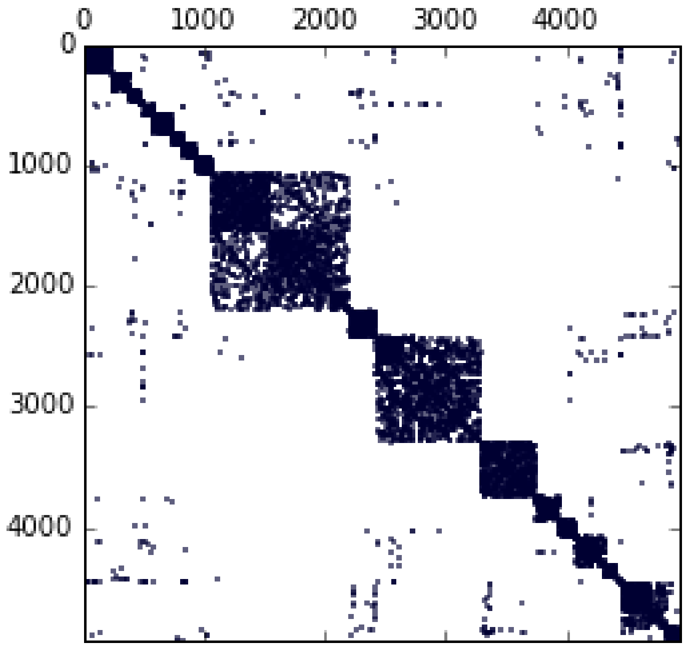
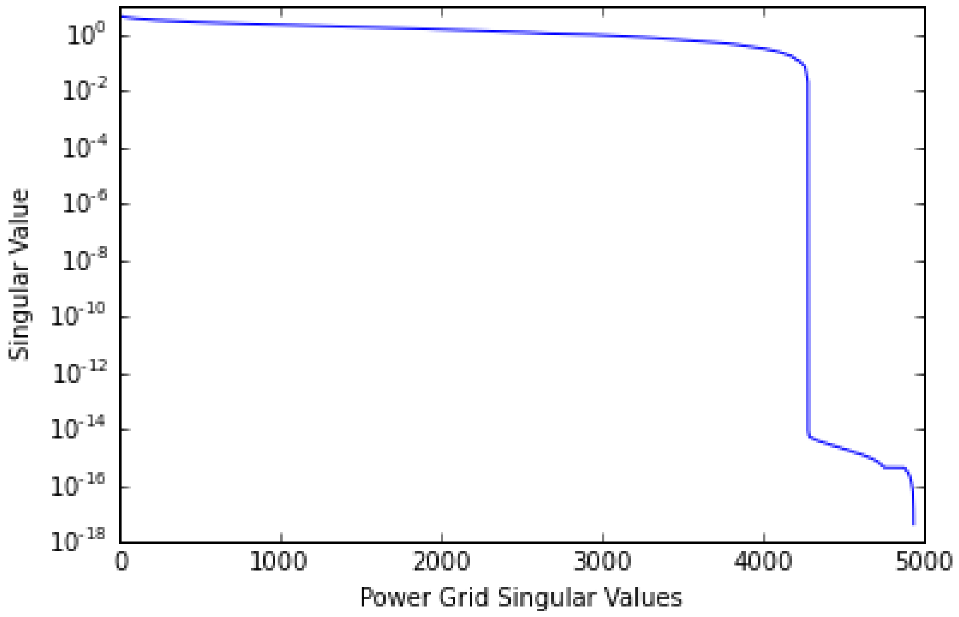


Figure 4.16 : Spy Plot of Power Grid Laplacian Matrix



Chapter 5

Conclusions and Future Work

In conclusion I have utilized maximum locally-connected subgraph partitioning from Chung and Lu to partition a graph into a locally-connected subgraph and a teleportation subgraph. I then combine optimal multigrid solves on the locally-connected subgraph with linear algebra and a direct solve on the teleportation subgraph. I have now solved the original Laplacian linear system in a simple and straightforward manner that can be replicated easily. I give a complexity model for the partitioning algorithm, $O(N_I \times m \times E[d]^3)$, and a suboptimal complexity for the linear system solve, $O(n^3)$. Using the Woodbury matrix identity [?] I solve the Laplacian linear systems for biological networks of *C. Elegans*, for Facebook friend circles, and show how my method does not work for the power grid of the Western US. The individual operations align with their theoretical computational cost. However, there is much to continue working on.

5.1 Graph Partitioning

My graph partitioning algorithm is written in python mainly because of ease-of-use and because of the NetworkX library that provides simple graph computations. However I can rewrite much of the code in C to efficiently create graph structs. I can then test local-connectivity for edges in parallel, drastically decreasing time to partition. I would also like to submit my graph partitioning algorithm to the open source NetworkX library as an alternative to the k,l_connected_subgraph function for small k [?].

5.1.1 Theoretical Bounds

In addition to working on the code for the algorithm, I would like to work with graph theorists to classify graphs with large locally-connected components. There might be bounds on the size of the teleportation subgraph which can lead to bounds on the rank of the teleportation Laplacian matrix. This could help establish a class of graphs for which my method is appropriate.

5.2 Linear System Solve

Most of my code is built on PETSc which is written in C and optimized for sparse matrices [?], however the initial singular value decomposition is computed using numpy [?] and does not utilize the low-rank (r) nature of the teleportation Laplacian matrix. This results in $O(n^3)$ operations instead of order dependent on r operations. Also, PETSc does not have a vectorized multiple right-hand-side solve technique that will drastically decrease computations for the $Q_1 = VQ$ step in the linear system solve that is causing a bottleneck. If I implement this, I can speed up the system solve.

5.3 Larger Graphs and Finding Meaningful Solutions

I only tested my method on graphs of small to medium size due to memory constraints on my laptop. I would like to test these and much larger graphs on a cluster to determine how my method scales. Because I am especially interested in biological systems, I would like to test my method on more complex systems than *C. Elegans.*

Finally, and most importantly, I did not have enough time to truly evaluate solutions to the Laplacian linear systems for my example graphs. I need to do more work to find appropriate right hand sides to solve against. Currently I am solving against a random right hand side. What are the most important regions of the worm's nervous system? Are there any hidden protein-phenotype expressions that could be important in unraveling the worm's genome? Who are the origins of influence in the Facebook friend networks? These are the real questions scientists are asking, I have proposed a method to answer them, and would like to see the final results.