Eric Buras Master's Thesis

March 22, 2016

1 Abstract

NEED TO WRITE THIS

2 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? Thus, 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 P, a stochastic matrix describing a web graph, v, a distribution vector corresponding to the problem data, and $0 < \alpha < 1$, a damping parameter; solve the linear system:

$$(I - \alpha P)x = (1 - \alpha)v \text{ [?]}$$

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 teleportation edges. I solve the overall linear system by optimally solving the local subgraph using the Algebraic Multigrid method, and combining linear algebra and a much smaller system solve for the teleportation subgraph. This is a new way of solving a complete graph linear system by breaking it down into smaller problems yet still retaining all available information from the graph.

3 Background

3.1 Graph Laplacian Problem

Information about a weighted, undirected graph G with vertex list V and edge list E can be stored in a matrix fThis is called the adjacency matrix which is defined as blah. Then do this for the diagonal matrix. then laplacian is L = D-A. this is done because it is similar to the finite difference discretization of the laplacian on a grid This is called the Laplacian which contains edge connectivity and vertex degree information for the graph. For any vertex $u \in V$, the degree of u is:

$$d(u) = \sum_{v \in V} w_{u,v} \text{ [?]}$$

Let D be the diagonal matrix of degree information for V. The adjacency matrix, A of the graph G is defined as:

$$A(u,v) = w_{u,v} \text{ if } (u,v) \in E \text{ and 0 otherwise. [?]}$$

The Laplacian of G is:

$$L = D - A \text{ [?]}$$

why is this called the laplacian. -1 for all connecting edges. looks like the discretization of the

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. [?]

3.2 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 $O(n^{2.376})$ [?], yet this can be improved upon still (mention nested dissection and memory blowup). 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 (cite get from wikipedia). 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. (cite yousef saad iterative methods for sparse linear systems). Yet

none of these basic solvers take into account attributes of the graph Laplacian that can drastically improve method performance.

3.2.1 Spielman-Teng, Koutis et. al.

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 (define this). The Laplacian for this approximation is similar to the Laplacian for the original graph because of similar spectrum, 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 then able to further decrease the time complexity of S-T for general symmetric diagonally dominant systems [**?**]. (this is not our main problem)

3.2.2 Multigrid Approaches

Algebraic Multigrid utilizes 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 has optimal time complexity and demonstrates good parallel scaling thus it is useful for solving incredibly large problems [**?**]. Three current multigrid approaches 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, 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?

3.2.3 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 [**?**].

3.2.4 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 combine iterate solutions for more accuracy. 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 vertex aggregation step of multigrid. How can vertices be evenly aggregated over graphs with uneven degree distribution?

3.2.5 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 maching to quickly coarsen a graph [**?**]. It remains to be seen whether this approach will be successful for solving an entire Laplacian linear system accurately.

3.3 Novel Graph Splitting

I will now mention a different graph decomposition technique that is crucial in my work. In studying 'small world' networks proposed by Watts and Strogatz [**?**] Chung and 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 $k = 3, l = 3$, and $k = 5, l = 9$ and is a good example of how connected these types of graphs are. 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 edge deletion [**?**]. Thus we are able to split a Laplacian matrix into two matrices with information about a locally connected portion of the graph and information about global edges of the graph for further use in solving the Laplacian linear system.

4 Methodology

4.1 Graph Partitioning

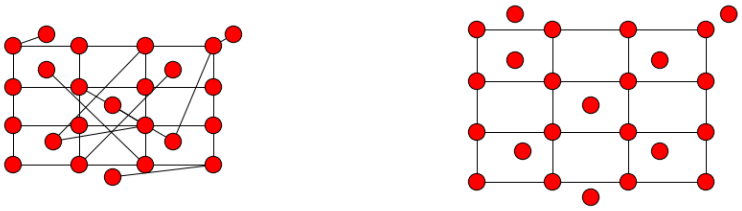
In studying 'small world' networks proposed by Watts and Strogatz [**?**] Fan Chung and Linyuan Lu propose an algorithm that separates a graph into a locally connected component and a teleportation component. This essentially breaks a graph into a group of edges between highly connected nodes with many paths between them, and the remainder of the edges between far sparsely connected nodes. 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. I set these values for k and l for the remainder of the paper. 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 [**?**].

4.1.1 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.

4.1.2 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-Lu above (cite?). Throughout much of the course of my work I used this function



(a) G, grid with random edges (b) P, max (3,3)-l. c. subgraph of G

Figure 1: Partitioning algorithm results in planar locally connected subgraph

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?) 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 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.

4.2 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.

4.2.1 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. Here is a breakdown of how I solve the Laplacian linear system:

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

4.2.2 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 [**?**], 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 [**?**]. 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 [**?**, **?**]

4.2.3 Low Rank SVD of Teleportation Subgraph

The Woodbury matrix identity requires use of the singular value decomposition 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 take the low rank singular value decomposition (cite?) to solve this small portion which has $O(r^2 n)$ complexity. This change will be very valuable in solving many-node networks.

4.3 Model Complexity

4.3.1 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 $d^{-\gamma}$ usually with $2 < \gamma < 3$ [**?**, **?**]. In lay terms, there are less and less nodes with increasing degree. This is similar to an exponential distribution, however there is a much longer and fatter tail (cite something). 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:

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

For now, let's assume our degree sequence follows this 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. This gives us expected number of paths: $(E[d])^2 + (E[d])^3$. Then it is simple to count the number of edge-disjoint paths. If there are two or less 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. 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 two iterations at most (see results).

4.3.2 Linear Algebra and Solves

After splitting the entire 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 matrix, the method is $O(n^3)$, however with work to implement the low rank SVD, it is $O(r^3 + r^2n)$. It is clear that the complexity of the solve depend 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.

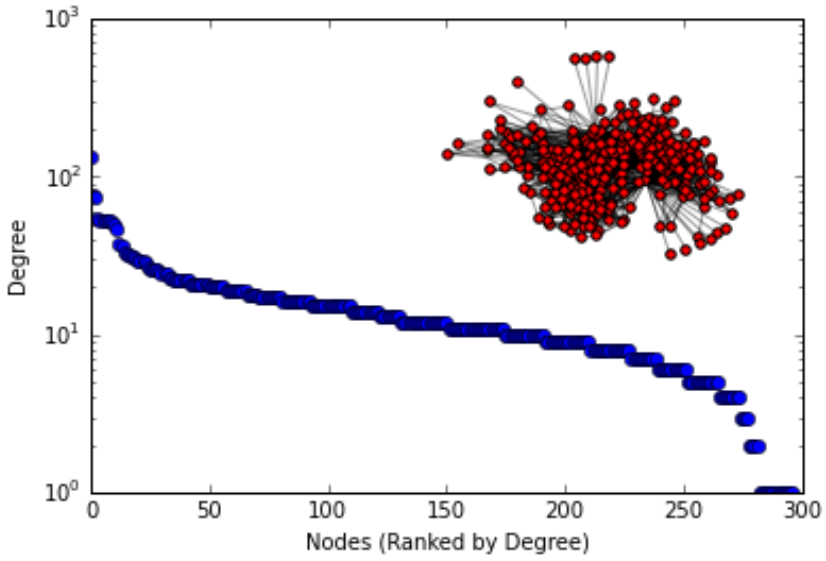


Figure 2: Neural Network of C. Elegans [**?**, **?**]

Eric Buras Thesis: Results

5 Results

5.1 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 such as social networks, biological processes, and an electric grid. 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).

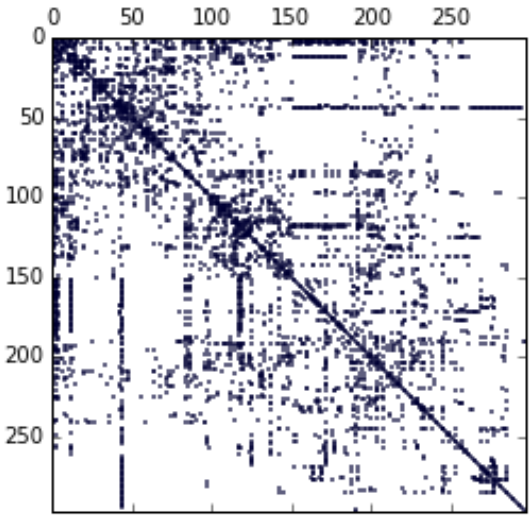


Figure 3: Spy Plot of Neural Network Laplacian Matrix

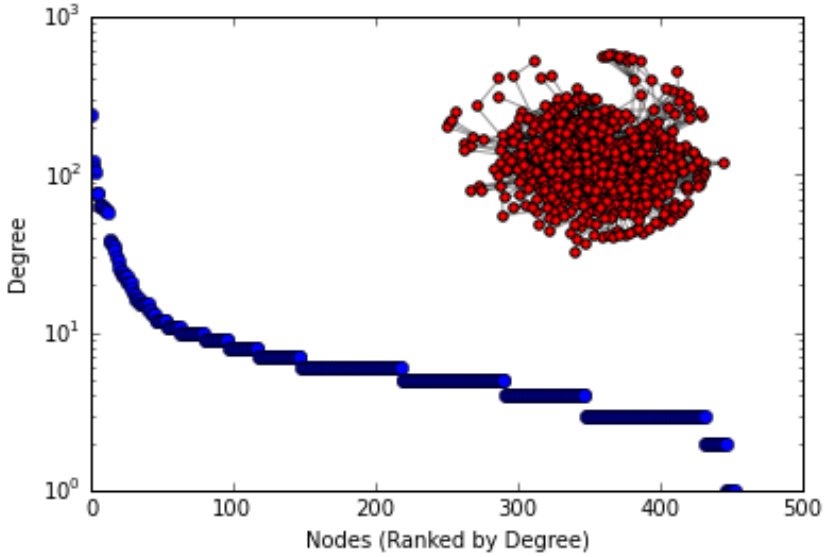


Figure 4: Metabolic Network of C. Elegans [**?**]

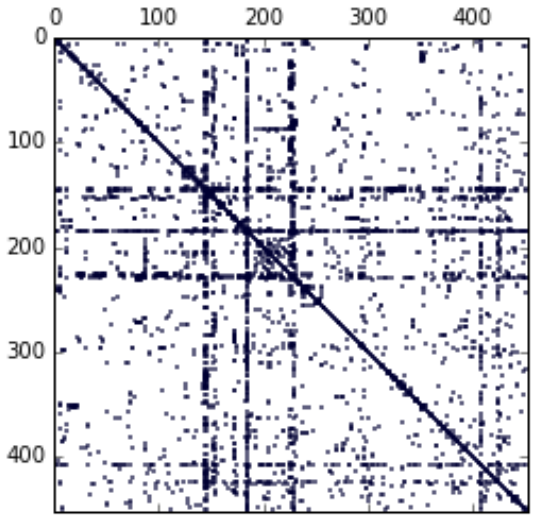


Figure 5: Spy Plot of Metabolic Network Laplacian Matrix

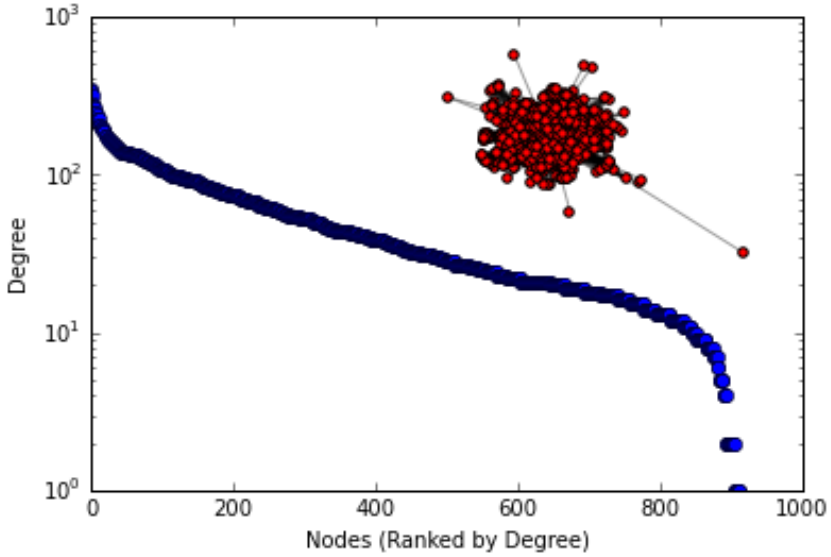


Figure 6: Gene Network Encoding Proteins of C. Elegans [**?**]

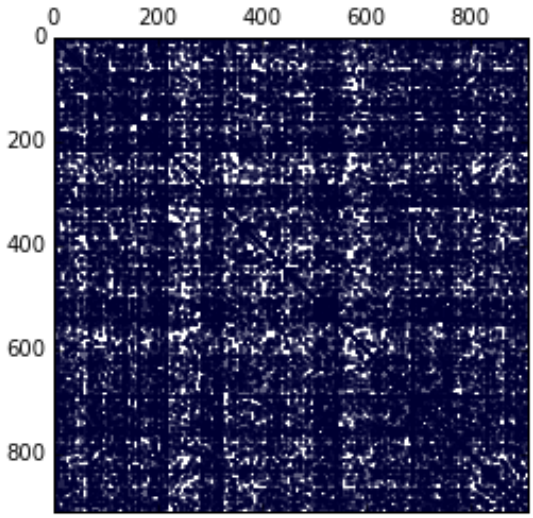


Figure 7: Spy Plot of Gene Network Laplacian Matrix

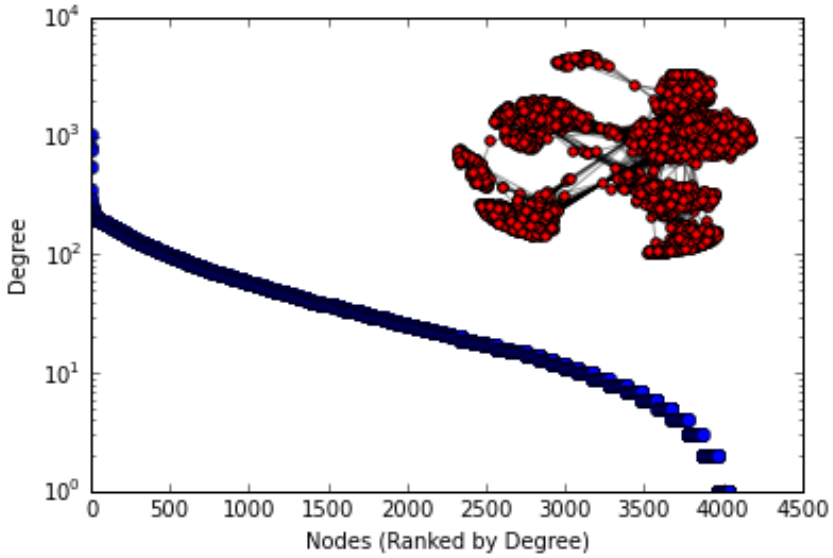


Figure 8: Facebook Friend Network [**?**]

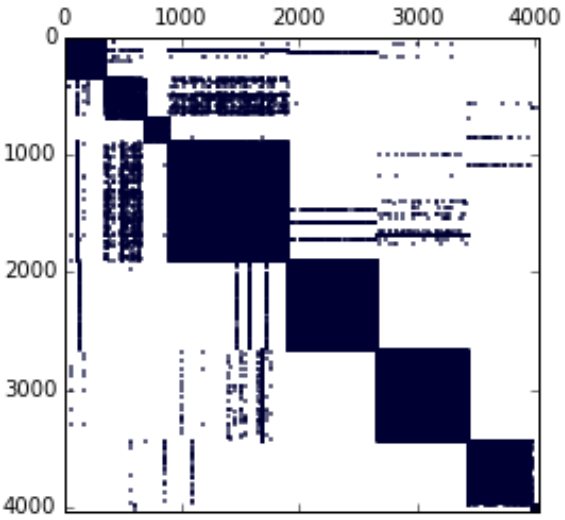


Figure 9: Spy Plot of Facebook Network Laplacian Matrix

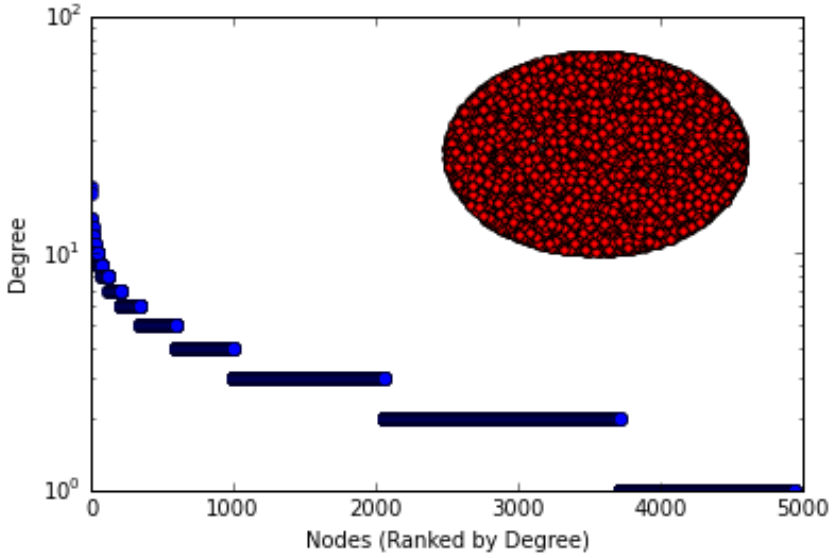


Figure 10: Network of Western Power Grid [**?**]

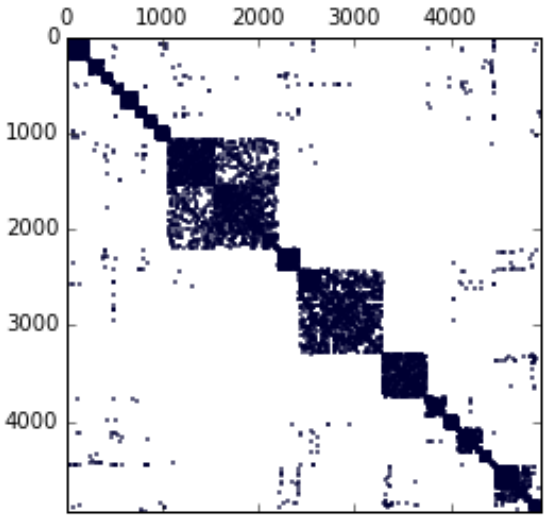


Figure 11: Spy Plot of Power Grid Laplacian Matrix

Graph (nodes, edges)	Avg. Deg.	Rank T_L	Nx part. (s)	Part. (s)	Iter.
Neural (297, 2148)	14.46	22	11	1.6	2
Metabolic (453, 2025)	8.94	49	12	2.3	2
Gene (912, 22738)	49.86	26	1915	53	1
Facebook (4039, 88234)	43.69	180	11593	480	2
Power (4941, 6594)	2.669	4284	2.1	.33	2

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	**Order**	**Neural**	**Meta**	**Gene**	**FB**	**Power**
$USV = T_L$	$O(n^3)$	.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

As observed, the operation timings correspond to their theoretical floating point operation orders. 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, $Q = P_L^{-1}U$. These operations can be optimized using the low-rank SVD and by vectorizing the multiple right hand solves so that only one multigrid solve must be done. 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, $y_2 = Q_2^{-1}y_1$. Graph Laplacian linear systems without low-rank T_L should not be solved using this method.

6 Conclusions and Future Work

NEED TO WRITE THIS