Eric Buras Thesis: Complexity

March 21, 2016

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 k degree scales with $k^{-\gamma}$ 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). We can compute the expected degree in a sequence following a power law:

$$\text{P(d)} = \frac{d^\gamma}{\zeta(\gamma)} \text{ where } \zeta(x) \text{ is the Riemann-Zeta function.}$$
$$\text{E[d]} = \sum_{d=1}^{\infty} d * \frac{d^\gamma}{\zeta(\gamma)}$$
$$\text{E[d]} = \frac{\zeta(\gamma-1)}{\zeta(\gamma)}$$

so expected degree is between 1.5 and 2 for our range of γ. However this vastly understates the tail in most of these real data degree distributions.
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 $m * ((E[d])^2 + E[d]^3)$ searches for one iteration of the cycle. We continue iterating this cycle until no edges can be removed. For a random graph or a graph without a large locally-connected subgraph, this process would require many cycles. This is shown in the power grid graph from Watts-Strogatz [**?**] Fortunately I have found real-world graphs with a large local subgraph, thus the number of iterations is small. I cannot give a bound on this number, but my datasets require less than four cycles to partition all teleportation edges. However it is important to remember that my datasets do not follow the above power law distribution. The expected degree of node in each of my datasets is much higher, requiring more work for this algorithm than at first glance.

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), find the order of the low-rank singular value decomposition, take into account a low rank direct solve, and count the multigrid solves. For convenience, here is the algorithm to solve the graph laplacian using the Woodbury matrix identity:

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

Given nxn graph Laplacian matrix, rank-k teleportation matrix with $k < \frac{n}{20}$ we find $2k^2n + k^2 + 4kn + k + n$ flops, $k + 2$ multigrid solves, and one dense kxk direct solve. It is clear that the complexity of the solves depend on k, the rank of the teleportation matrix T_L.