Graph Reduction Using Near Symmetry

Christopher Healy, Anchit Narain

1 Abstract

In this paper, we propose the mathematical framework for a graph reduction algorithm that focuses on minimizing information lost during compression by aggregating nodes that play nearly if not completely identical structural roles in the graph, i.e. by clustering near and perfect symmetries in the network. We then present the findings of our coded implementation of the proposed symmetry reduction algorithm. We see that perfectly symmetric undirected networks of any size and fully connected networks (or the largest weakly connected component of any undirected network) can be fully reduced by our algorithm. Smaller networks can be reduced by clustering via DBSCAN or OPTICS, but graphs with thousands of nodes can only be clustered by OPTICS. We can then reproduce in imperfect, highly symmetric representation of the original graph from the compressed graph. The reproduced graph, in certain respects is very similar to the original graph.

2 Introduction

Simplifying massive datasets is an increasingly crucial task as the potential to gather data on a previously unheard of scale continues to expand. Concise representations of what are nominally unapproachable complex sets of relationships is crucial for optimizing one's response to the data gained. In the realm of networks specifically, many algorithms scale dramatically with the vertex and edge count of any given network, and many reductions that maintain meaningful representations of the original network can enable feasible analysis that was previously impossible. Graph clustering is an extensive field of research, and at present the literature focuses mostly on clustering by community, which is to say trying to find densely connected groups of nodes in order to create supernodes that effectively surmise the emergent large scale structure of the graph. Significantly less effort has been put towards reduction of the graph by eliminating structural redundancy.

3 Prior Work

3.1 Exploiting symmetry in network analysis (4)

This paper outlines a comprehensive method for reducing the structure of a graph using exact symmetry, defined as invariance under permutations of nodes. Specifically, it focuses on basic symmetric motifs (BSM's). Basic symmetric motifs are defined as a set of orbits of constant size, where an orbit of nodes is invariant under all permutations.

The algorithm condenses each orbit of a BSM into a single new node that is encoded with an associated vector containing the internal degree of each orbit and the number of nodes (from the original graph) in the orbit. This allows a perfect reconstruction of the original graph from the reduced graph, called the "quotient graph." It also defines an upper and lower bound on the total compression of edges, just based on node and edge count.

When applying conventional algorithms to the quotient graph, the time to compute was generally reduced by greater than 50 percent, with maximum reduction at around 80 percent and minimum reduction around 20 percent.

However, the focus only on basic symmetric motifs is limiting, as there are significant patterns potentially overlooked because the symmetries contained within are not basic. If

we were to take a large graph, copy it, and join all old nodes together with their copies by an edge, the resulting graph would be easily reducible with no information loss compared to the original graph. This would speed up high-scaling algorithms by orders of magnitude. However, in most cases, such a symmetry would not be basic, so the algorithms contained here would fail to detect it entirely, and would leave the graph much larger than necessary.

The author points out that BSM's are significantly more common than any sort of complex symmetric motif (CSM). This makes CSM's rare enough to generally discount without much expected opportunity cost. This is certainly true of perfect symmetries, but is largely due to the fragility of larger scale, more convoluted symmetries. If we return to our previous example, if we were to take our graph composed of two connected copies of a smaller graph, and then connect one extra degree-1 node to a random node, the entire symmetry could easily fall apart and we could have a potentially irreducible graph.

However, in larger datasets, one node of low degree has very little causal impact on the structure of the graph, and so for a sufficiently large example, our added node could be simply ignored, and the re-emergent symmetry used to reduce the graph with trivial information loss. While this is a toy example, it does suggest that a looser approach to symmetry grouping has the potential to summarize graphs with relatively small information loss by utilizing more complicated symmetrical structures.

In our compression algorithm, we utilize a generalization of the physically symmetric property of reflectional symmetry. Our algorithm will, over the course of multiple simultaneous breadth-first searches, create sets of particles that can be collapsed, or folded together, while minimizing structural loss. Our reconstruction algorithm utilizes a generalization of the notion of rotational symmetry to reconstruct a symmetric graph of specified node count, degree, and number of strongly connected components.

3.2 Synthesis with ReFex and RolX

Using RolX and ReFex (1) to classify node roles, and using the quotient network to efficiently reduce graph structures to a smaller form can be thought of as two different extremes of the same parameterized process.

Both algorithms essentially sort nodes from a network into groups, trying to optimize for grouping structurally similar nodes. RolX sacrifices massive quantities of information about the technical structure of the graph in order to sort nodes into a few, easily digestible buckets, while a quotient network only sorts into buckets when it can do so with absolutely no loss of information.

If we consider the ReFex embedding of two nodes grouped into one by a quotient network, we can easily note that they must have the exact same structural role in the graph and thus the same embedding. Assuming that no two nodes that play different structural roles are embedded by ReFex as the same vector (an event which would be trivially unlikely given enough recursions), we can roughly model the symmetry reduction as a clustering algorithm in the ReFex embedding space that only clusters identical vectors (The quotient network may not even group these vectors, if the symmetric motif they form is not a BSM). On the other hand, we can model RolX as a clustering algorithm in the same space that clusters many heterogeneous vectors into several large classifications.

Because of the fragility of complex symmetry motifs and the corresponding rarity of perfect CSM's, the primary investigation of this project is to investigate the tradeoff between information loss and complexity in the reduction of naturally occurring networks.

We will present an algorithm parameterized by any given attitude towards this tradeoff – in the limit where we want to minimize information loss, our algorithm should emerge, roughly, to the generation of a quotient network. In the limit where we do not care about

information loss, our algorithm should emerge, roughly, to structural role extraction. We thus will parameterize a clustering algorithm in a space of ReFex embeddings that optimizes for some tradeoff between these two values.

By taking the symmetry reduction model, generalizing it to work on CSM's instead of BSM's, and loosening the requirements for perfect symmetry, i.e. loosening the requirements for the clustered vectors to be identical in the Refex embedding space, one can ideally can exploit near symmetries to significantly improve the reduction of a large network without incurring disastrous information loss.

4 Symmetry

4.1 Symmetry as a continuous variable

While the definition of symmetry using automorphisms is undoubtedly useful, and has proved invaluable in past analyses (including PAPER1), for the purposes of this algorithm, we will instead define a symmetry metric, evaluating the "degree" of symmetry between two nodes instead of simply defining orbits of nodes, wherein only pairs of nodes sharing an orbit are symmetric. This will allow us to recognize and account for near symmetries.

4.2 Idealized Metrics:

It is useful to start by thinking of an idealized metric for symmetry. Though computationally infeasible, these ideas will motivate our approach towards softening the problem. We will assume, for now, an unweighted, undirected, uncolored, connected network N with m edges and n nodes labelled $n_1, ...n_n$.

4.2.1 Idealized Metric 1

For each node n_i, create the graph N_i, by taking N, coloring all nodes $n_j, j \neq i$ white and n_i black, and then removing all labels from the graph. We can then define the symmetry between two nodes $S(n_i, n_j)$ in terms of how different the graphs N_i and N_j are. Since all graphs N_i have the same number of nodes and edges, we can evaluate this difference as the minimum number of edges that must be moved to turn N_i into N_j. If the two graphs are identical, then $S(n_i, n_j) = 0$, which is the maximum value, and, if the graphs take x edge shifts to become the same, then $S(n_i, n_j) = -x$. While this metric is computationally infeasible to calculate at a large scale, it has some desirable properties, including a transitive inequality shared by the measure of the distance between points in Euclidean space:

$$|S(n_i, n_k)| \leq |S(n_i, n_j)| + |S(n_j, n_k)|$$

4.2.2 Idealized Metric 2

The previous metric does not take into account the distance of the edge shifts that occur from the black nodes on the graph. Assuming we will be using graphs to model dynamic phenomena where proximity of nodes in the graph amplifies causal effects, the effects of differences in graph structure of farther out nodes does not matter as much as the differences in graph structure of the two nodes in the same neighborhood when deciding whether they can be reduced. Therefore we would like to adjust our old symmetry metric into a new $S_2(n_i, n_j)$ that scales according to the inverse of the distance of the edge shifts from both black nodes as we transition. Therefore, if the minimizing set of edge shifts to turn N_i into N_j is $(x_{1i}, x_{1j}, y_{1i}, y_{1j}), ..., (x_{zi}, x_{zj}, y_{zi}, y_{zj})$ where z is the total number of shifts, x_{ab}

denotes the path length from n_b to the edge removed from the graph in the ath edge shift, and y_{ab} denotes the path length from n_b to the edge added to the graph in the ath edge shift, then:

$$S_2(n_i, n_j) = -\sum_{k=1}^{z}(\frac{1}{x_{ki}} + \frac{1}{x_{kj}} + \frac{1}{y_{ki}} + \frac{1}{y_{kj}})$$

Unfortunately, in this case we lose the previous transitive property.

4.3 ReFex as a Practical Substitute

The distance between ReFex embeddings, which depends only on the structure of the graph, can serve as as substitute for the idealized metrics above. Given that the embedding v_i of a node can be written purely as a product of N_i and is generally well behaved, we can expect that all other things being equal, increasingly adding variations to N_i will increasingly change the embedding of v_i from its original location, and thus the distance between these vectors serves as a decent proxy for the idealized symmetry metrics laid out above.

Take a ReFex embedding of an arbitrary network with arbitrary (standard) starting parameters, and recursion performed by aggregating data only from a node's immediate neighborhood. Take the symmetry metric equal to the negative distance between node embeddings $S_3(n_i, n_j) = -D(v_i, v_j)$. Note that if the two nodes have $S(n_i, n_j) = 0$ and $S_2(n_i, n_j) = 0$, they must have identical embeddings, and so $S_3(n_i, n_j) = -D(v_i, v_j) = 0$.

Note that being a euclidean representation of distance, this metric satisfies the geometric property above:

$$|S(n_i, n_k)| \leq |S(n_i, n_j)| + |S(n_j, n_k)|$$

Additionally, note that all other factors being equal, nodes with further away from each other (where distance is measured as the shortest path between any 2 nodes) will have less impact on each other's embeddings. This is because data from original embeddings travels at a "speed" of only one edge per recursion, and on recursion r with p initial starting parameters and a recursive expansion factor of f, the first pf^r elements of each embedding have been fixed, determined only by the r-hop neighborhood. We can further tactically normalize dimensions of the embeddings to increase this effect, as we will detail later.

4.4 Theorem on Symmetric Nodes

Theorem: Take two perfectly symmetric nodes n_i and n_j. We can establish a one to one correspondence between the neighbors of n_i and n_j such that each pair of neighbors is also perfectly symmetric.

We can establish this clearly with either of our interpretations of perfect symmetry from 3.2. We said two nodes were perfectly symmetric iff the graph created by coloring one of those nodes black, every other node, white, and removing all labels was identical to the graph created by coloring the other node black, every other node white, and removing all labels.

Call the original network N, and call the induced colored graph, for each of two symmetric nodes i and j, N_i and N_j We have that $N_i = N_j$. Since these graphs are identical, we now relabel the nodes to create a graph B, keeping them constant across N_i, N_j. Jump to a specific neighbor of the black node in B. Now, create a new colored graph A from

N_i/N_j by coloring this new node black and coloring the original black node white. Note that the black node in A corresponds to a specific node v in N that was transformed from N to N_i to B to A, and a specific node u in N that was transformed from N to N_j to B to A. Note also that $N_v = A = N_u$, by the definition of these colored graphs. In that case, then $N_v = N_u$ and so v and u are symmetric. Apply this method for each neighbor of A to establish a symmetric relation between each neighbor of i to a neighbor of j.

Note that very frequently these symmetric nodes will just be the same node. This is ok. Nodes are trivially perfectly symmetric with themselves. Also note that this property transitively applies to the n-hop neighbors of any two perfectly symmetric nodes.

5 The Algorithm

5.1 Overview

Our algorithm, GRUNS (Graph Reduction Using Near Symmetry), works in several steps:

1. **Recurse** Recurse on Weakly Connected Components of the Graph
2. **Embed**: Construct Refex embeddings of each node in the dataset, and normalize these embeddings appropriately.
3. **Cluster**: Run unsupervised clustering algorithms on these embeddings and group them into new nodes by cluster. The better the clustering, the better the graph reduction in the next step.
4. **Reduce**: Appropriately arrange the resulting clusters into a reduced graph.

5.2 Recurse

We apply the algorithm separately to each Weakly Connected Component of the graph.

5.3 Embed

5.3.1 Parameters

1. Initial Features: list of neighborhood features to be initially embedded within each node. In our implementation, the initial features used are degree, number of edges in Egonet, and the number of edges leaving the Egonet.
2. Recursions (R): Scalar integer value, determined by how many recursions we think we can calculate. In our implementation, we used $R = 2$.
3. Maximum hop distance (k): number of hops summed over during recursion. In our implementation, we used $k = 3$.

5.3.2 Starting Features

For each node, calculate the Initial Features, and embed them in a vector.

5.3.3 Recursion

For the number of recursions, and for each node, we will multiply the size of the node's embedding by $1 + k$. Partition this vector into $\frac{1}{1+k}$ subvectors. On the ith subvector, sum over the previous embeddings of the i-hop neighborhood of the node. (For i = 0, this is just the nodes original embedding).

5.3.4 Normalization

Since we will be clustering over distance, we do not wish to standardize the norms of vectors in the feature space: two vectors $a = \lambda b$ are not structurally identical and the distance value should reflect that. What we will do instead is to normalize the values across each dimension. We will subtract from every element of each vector the average value of that element across vectors, and then divide each element of each vector by the standard deviation of that element across vectors.

5.4 Clustering

Once we successfully have embedded each node from the original graph, we cluster them using Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and Ordering Points to Identify the Clustering Structure (OPTICS) unsupervised clustering algorithms. Since we do not know the optimal number of clusters a priori, we implement methods that learn the optimal number of clusters for a variety of distance metrics that calculate a the distance between paired embedding vectors in the embedding space.

The various metrics we tested were: ['cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan','braycurtis', 'canberra','chebyshev', 'correlation', 'hamming', and 'minkowski'].

Then, we test the accuracy of of each distance metric's clustering on known perfectly symmetric networks, networks made up completely of separte perfect symmetries, and networks with only some symmetries, and some with only near symmetries, all for varying network sizes. After all this test benching, we select the handful of distance metrics and other associated hyperparamenters specific to both DBSCAN and OPTICS that provide the most accurate clustering of the network across a variety of network sizes.

We considered using agglomerative hierarchical clustering, but both Scipy and Scikit learn implementations of this algorithm failed at clustering perfectly symmetric graphs, like the Petersen Graph, into a single supernode. We found that the underlying code from these libraries assumes a default minimum of 2 clusters, which fails for perfectly symmetric networks.

DBSCAN is a density-based clustering non-parametric algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking points that lie alone in low-density regions (whose nearest neighbors are too far away) as outliers. It starts with an arbitrary seed point which has at least MinPts points nearby within a distance (or "radius") of ε. We do a breadth-first search (BFS) along each of these nearby points. For a given nearby point, we check how many points it has within its radius. If it has fewer than MinPts neighbors, this point becomes a leaf–we don't continue to grow the cluster from it. If it does have at least MinPts, however, then it's a branch, and we add all of its neighbors to the FIFO queue of our BFS. Once the BFS is complete, we're done with that cluster, and we never revisit any of the points in it. We pick a new arbitrary seed point that is not already part of another cluster, and grow the next cluster. This continues until all of the points have been assigned. If a point has fewer than MinPts neighbors, AND it's not a leaf node of another cluster, then it's labeled as a "Noise" point that does not belong to any cluster. As such, we see that the accuracy of clustering via DBSCAN is highly dependent on the input parameters ε, MinPts, and the distance metric in which ε is defined.

OPTICS is another algorithm for finding density based clusters in spatial data, and is similarly fine tuned by the input MinPts and choice of distance metric. However, the sklearn OPTICS is more forgiving than DBSCAN in that it can take a parameter max ε, which is the the maximum distance between two samples for one to be considered as in the

neighborhood of the other. As such, OPTICS accepts a range of ε values, making it less sensitive to this parameter than DBSCAN.

We define MinPts for each algorithm as 1, since we would like to place isolated nodes in their own cluster as opposed to clustering them with other isolated nodes. We also use the above listed metrics as they allow give us a good mix of euclidean and non euclidean definitions of distance, and give us a variety of options with which to optimize the clustering. Then, we wrote multiple helper functions to find/learn the optimal value for ε for both of these clustering methods as there are no existing libraries that do so. We used the pseudocode from (2) as a skeleton for our implementation of an algorithm for finding optimal ε values. In essence, we find a suitable value for epsilon by calculating the distance (defined differently for each tested metric) to the nearest n points for each point, sorting these distances from lowest to highest, and plotting the results. Then we look to see where the curvature is at its maximum along this plot.

However, finding regions of maximum curvature (also known as knee or elbow points) for discrete datasets is still considered an open problem, so we implement a modified variant of the algorithm presented in (3) to find regions of maximum curvature along the above defined plot for each separate metric given the concavity of the plot, a list of x and y values for each entry in the plot, and the fact that the plot is increasing over the entire region. Given these parameters, we are able to solve for the optimal value of ε for DBSCAN and set the upper bound on the range of distances for OPTICS.

We also found that DBSCAN is extremely sensitive to this optimal value of ε and as such, will only cluster perfect symmetries, leaving near-symmetries independently clustered. Increasing the value of ε beyond this optimal value even slightly leads to imperfect clustering. However, with OPTICS we were able to introduce a secondary parameter β that we add to the input max ε to further the upper bound on the clustering radius of any given embedding in the embedding space. This parameter allows us to vary the degree of symmetries that we're willing to cluster together. With $\beta = 0$, we primarily cluster perfectly symmetric embeddings, but as we increase β we can cluster more near symmetries as well. These clusters are reduced into new single nodes in the Reduce Step.

5.5 Reduce

This step in the algorithm takes as an input the graph and the clusters of nodes provided by the second step in the algorithm. It is parameterized by nothing.

The algorithm consists of a series of reductions until every cluster is completely contained in its own node. Each time we make a reduction, embed in each node a new four dimensional vector. Since we can combine nodes multiple times, we also transfer the previous embedded vectors into those nodes. So, at the end, each clustering node will have some amount of four dimensional vectors embedded inside them, where the number of vectors is equal to the number of reductions it took the algorithm to collapse all of the nodes.

5.5.1 Reduction Vector

Every time we collapse some nodes into one, they must be of the same cluster. They also must have the same prior vectors (In the perfectly symmetric case, this should always be true if they are of the same cluster). The elements of our new vectors are as follows.

1. The number of nodes collapsed in this cluster, in this reduction
2. The internal degree of the nodes collapsed in this reduction, if all edges had weight one

3. The number of disjoint subgraphs collapsed together in this cluster, in this reduction

4. The internal edge weights between nodes in this reduction

5.5.2 Reduction

1. Arbitrarily choose a cluster that has not been collapsed completely. Take each node in that cluster (call these nodes "cluster nodes") and simultaneously conduct a breadth first search of this graph, through each nth iteration labelling the n-hop unexplored neighborhood of cluster node i with the vector (i,n).

   Along the way, keep track of the clusters of each neighbor and store the resulting near-one-to-one correspondences between n-hop neighbors of each cluster node. We maintain an array indexed by node and origin tree, greedily assigning as many correspondences as possible between the elements of this array at every step, where one can only assign a correspondence between two nodes if they are in a different origin trees and the same cluster.

2. In a round where two of our breadth first searches collide with each other, we combine all of the one to one correspondences we have established between those two searches, including of course the source nodes. We do the standard graph aggregation, but we don't need to worry about self edges, that is encoded in the vectors. The edges between the component nodes' new combined nodes are just summed up for the edge between the new combined nodes.

   Important Note: In any step of the breadth first propagation, we have two types of potential collisions. First, where both breadth first searches search the same node. Second, where one breadth first search searches a node that has been previously searched by another breadth first search. Note here that if both of these things occur in one round, we only combine the one to one correspondences of the groups whose associated collision occurred by the second method. Collisions occurring by the first method are said to have occurred "after" collisions by the second method in any given round and thus are only checked for if collisions occurring by the second method have not occurred.

3. Check to see if any clustered nodes are presently separated. If so, go back to step 1. If not, return the reduced graph.

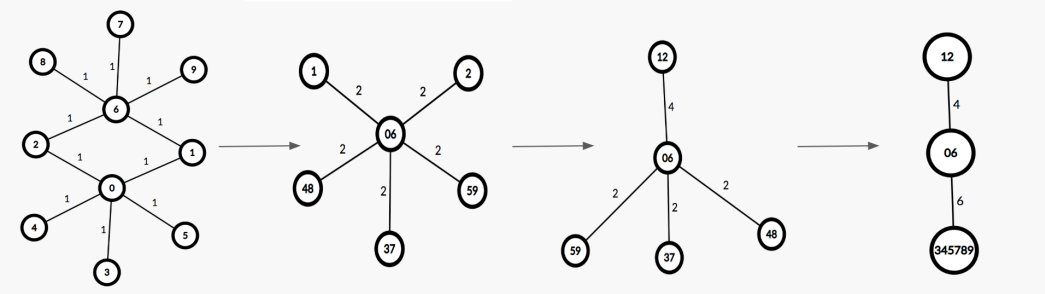

Figure 1: Here, we show the actual progression of reduction that the algorithm completed on a 10-node undirected network containing 3 distinct perfect symmetries. The reconstruction algorithm, in this case, will reproduce the graph exactly.

6 Reconstruction

Our reconstruction algorithm is simply built: For each node in our compressed graph, we iterate through the each layer of input vectors and resulting nodes, extending that node to a simple construction of a vertex-transitive induced graph, and distributing the associated edges over the new nodes.

6.1 Creating A Vertex-Transitive Graph

6.1.1 Inputs

:

1. Node Count n
2. Degree d
3. Number of Weakly Connected Components c
4. Edge weight of Internal Components w

(If the requested degree and node count are both odd, we add a node such that $n+ = 1$. In practice this rarely occurs.)

6.1.2 Creation

We create n nodes, indexed from 0 to $n-1$. If the degree is odd, we create edges from each node i to another node $i + \frac{n}{2}$. Then, the remaining edges that must be added per node are even. For each node, we then create edges to the nodes that have the nearest index (modular in the number of nodes), until we have a vertex transitive graph. It's vertex transitivity is clear: under cyclic permutations of indices, the structure of the graph is trivially maintained.

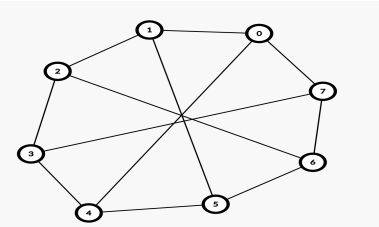

Figure 2: Example of constructed transitive graph with 8 nodes and degree 3

6.1.3 Integration

We then replace our target node in the graph with this newly created symmetric graph. Using arbitrary approximations, we can then distribute the edges from the original node to other nodes near evenly amongst the newly created nodes. We approximate an even distribution (in the case of imperfect symmetry) by assigning an an original edge with weight g to a selection of g nodes in the newly created subgraph, prioritizing nodes that have smaller degree and letting each newly created edge have weight 1. If an original edge has weight higher than the number of nodes, we assign each new node an edge to that neighbor, with each edge having the appropriate integer approximation of the fraction between the original weight and total number of newly added nodes.

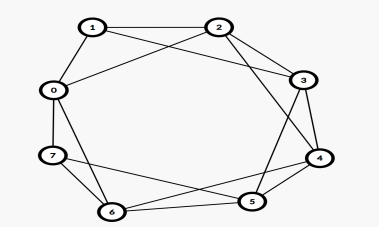

Figure 3: Example of constructed transitive graph with 8 nodes and degree 4

7 Results and Conclusions

We tested our coded implementation of the proposed algorithm on toy models like that shown in Figure 1 to make sure the intended functionality worked on smaller networks. After that, we worked on larger networks, such as the Arxiv GR-QC (General Relativity and Quantum Cosmology) collaboration network from the SNAP website. It covers scientific collaborations between authors' papers submitted in the General Relativity and Quantum Cosmology category on the ArXiv. If an author i co-authored a paper with author j, the graph contains a undirected edge from i to j. If the paper is co-authored by k authors this generates a completely connected (sub)graph on k nodes. It contains 5242 total nodes, and the number of nodes in the largest weakly connected component is 4158.

While both DBSCAN and OPTICS clustering worked equally well on the toy model networks, on networks with $>$ 1000s of nodes, the DBSCAN algorithm begins to cluster almost all nodes in the same cluster, mostly incorrectly. OPTICS on the other hand, is much more robust when dealing with a very large number of unique clusters, each containing a very large number of nodes. Of the metrics tested, we found that the cityblock, manhattan,and minkowski metrics always clustered correctly on both small and large networks (ranging from 10 to 5000 nodes). We decided upon specifically using the minkowski metric for reconstruction efforts for time efficiency as this was recommended in (5). In general, we would recommend using the OPTICS clustering algorithm with the minkowski (or cityblock and manhattan) metrics for most accurately clustering both small and large networks.

We compressed the graph down as much as the clustering would allow, reducing the number of nodes from 4158 to 2199.

Upon reconstruction, there were notable similarities and differences between the reconstructed graph and the original graph. The degree distributions between the two graphs (shown below) are nearly identical, which is promising, but due to some error in our implementation that we could not find, the reconstruction could not complete the last step in the unfolding in the large, imperfect symmetry case, leaving a graph that has still been compressed, with 3398 total nodes. This is obviously not intended, but does give us an opportunity to observe the structural similarities between the partially reduced graph and the original graph.

While the complete coded implementation of the proposed symmetry reduction algorithm is as of yet incomplete and not completely functional, we were completely successful in embedding, clustering, and reducing the graphs. We plan on continuing this project by fine tuning our initial, naive reconstruction algorithm.

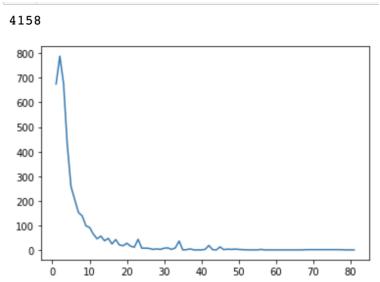

Figure 4: Degree Distribution for largest WCC in GR-QC dataset.

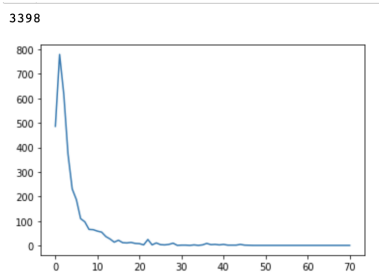

Figure 5: Degree Distribution for compressed and then reconstructed largest WCC from GR-QC dataset.

8 Division of Labor

Chris worked primarily on the implementation of the graph compression and reconstruction algorithms. Anchit worked on literature review and the implementation, testing, and tuning of all the clustering algorithms used. Both students contributed roughly equally to the development of the algorithm and the paper.

References

K. Henderson and B. Gallagher *et al.* Rolx: Structural role extraction mining in large graphs. *Association for Computing Machinery*, Aug. 2012.

N. Ramh and I. Sitanggang *et al.* Determination of optimal epsilon (eps) value on dbscan algorithm to clustering data on peatland hotspots in sumatra. *IOP Conf. Series: Earth and Environmental Science*, Aug. 2016.

B. Raghavan *et al.* Finding a 'kneedle' in a haystack: Detecting knee points in system behavior. *ACM Digital Library*, Jun. 2011.

R. Garcia *et al.* Exploiting symmetry in network analysis. *ArXiv*, Jul. 2019.

S. Dahal *et al.* Effect of different distance measures in result of cluster analysis. *Aalto University*, Apr. 2015.