Feature Constrained Graph Generation with a Modified Multi-Kernel Kronecker Model

Federico Reyes Gomez [1] **Vamsi Saladi** [1] **Markie Wagner** [1]

Abstract

Graph generation algorithms are a cornerstone of research on graphs, giving us as useful points of comparison, anonymized versions of real-world graphs, or hypothetical synthetic datasets that can be used to test novel algorithms. A key factor of these graph generation algorithms is that they output realistic graphs that exhibit some of the same features as a real target graph, such as small diameters in small-world graphs, certain motif distributions, or clustering coefficients. Thus the problem of creating a random graph that is constrained to certain features is very important. We attempt various methods based on the Kronecker Graph generation model, in addition to using multiple initiator kernels. We first tried to match the spectra of the target graph, leverage motif-based kernels with a Random Kernel Kronecker Method, use an MDP Kronecker Method, and augment the KronFit algorithm to increase the parameter space by using multiple kernels. In the end, we found that features like maximum and minimum degree, along with average degree, were easy to march. But the clustering coefficient, which involves matching local structure, was too complex for our methods. While this problem turned out to be more difficult than we expected, we explore lots of interesting avenues related to multi-kernel Kronecker graph generation.

1. Context

Our goal for this project is to design a graph generation algorithm that outputs a graph that fits the specified input features. We envision a general algorithmic framework that, given graph features such as the clustering coefficient, degree distribution, motif counts, and/or the spectrum of its adjacency matrix and laplacian matrix, can output a random graph of size n that has these features. Possible applications include generating random graphs based on real-world graphs for privacy and anonymization reasons, or for generating graphs to fit hypothetical situations that researchers may want to study.

We based our generation algorithms on the Kronecker Graph Generation algorithm [1] proposed by Leskovec, et al. This algorithm takes a single initiator kernel and repeatedly applies the Kronecker matrix product to generate larger graphs. We propose first determining a set of base kernels to choose from and, instead of multiplying by the same one each time, multiply by different kernels at different levels. We propose this as an alternative (or addition to) stochastic Kronecker graph generation in order to add randomness and variability to the output graphs.

2. Related Work

2.1. Graph Generation

Kronecker graphs were proposed by Leskovec et al. as a way to detail a mathematical approach to generating graphs given certain features (Leskovec et al., 2009). Specifically, given some structural property and a subgraph to emulate, the paper details an algorithm for generating larger graphs that emulate the substructure and maintain self-similarity. The paper does this by detailing a mathematical concept (Kronecker product) that helps maintain a self-similar structure while expanding the size of the adjacency matrix. By generating various different graphs with this algorithm, the paper shows that graphs generated by this method are effective and have similar properties to graphs found in the real world (like heavy tails for the in-degree and out-degree distribution, heavy tails for the eigenvalues and eigenvectors, small diameters, and densification and shrinking diameters over time. Additionally, the paper also provides an algorithm (KronFit) that can be used for fitting the Kronecker graph generation model to large real networks and does so in linear time rather than the naive exponential time.

While the paper presents a very efficient way of finding parameters (an initiator matrix) to fit a real world graph, it still requires a reference. Say you're a scientist studying global economic flow and would like to study a proposed trade deal under a hypothetical condition with higher interconnectivity, you would need to generate that graph somehow in order to study it. Additionally, while not a limitation per se, it would be interesting to study graphs that are generated as a result of Kronecker multiplications between different

kernels, different initiator matrices, instead of recursively multiplying the same one over and over again. The stochastic Kronecker generation method was proposed to vary the graphs generated, but we still hypothesize that we could add even more variance and get more expressive output graphs by using multiple kernels. We originally intended to use a modified version of the Kronecker model as our generation method. The Kronecker model depends on using some initiator structure that can be replicated and duplicated to generate larger graphs. However, there is usually more than one initiator substructure that might fit the necessary constraints and features. In that case, we wanted to be able to alternate and iterate through various possible kernels, and see if using different kernels at each step can help us get us large graphs that can also satisfy our constraints, and see if they are closer to the ground-truth graphs. Additionally, the paper also discusses the possibility of going from a ground-truth graph back to an initiator kernel and then to a random graph; this is a process that we could modify in the future (since we want to go more strictly from features to the random graph), and thus helps provide some insight into the MLE approach to graph generation and feature extraction.

2.2. Prescribed Feature Constraints

In a paper by Ying and Wu, four main features are used to constrain generated graphs: the spectrum of the adjacency matrix, the second eigenvalue of the Laplacian matrix, the harmonic mean of the shortest distance, and the transitivity measure of a graph (Ying & Wu, 2009). They chose these features because of their expressiveness and their relation to more traditional graph features such as the maximum degree, chromatic number, clique number, and extent of branching in a connected graph. They propose a generation algorithm based on edge switching: Given an input graph G to model, randomly switch edges if the resulting graph still satisfies the specified features. If not, revert the switch and choose a new random edge. They empirically show that this method is effective in preserving various key features that they analyze.

Ying's paper presents a novel way to fit a graph to certain features. KronFit implicitly fits a graph to certain features by using a Maximum Likelihood Estimation method, while the proposed switching algorithm fits the graph to specific user-specified features. This is closer to what we want, but this switching algorithm still requires a reference graph that serves as the input to the algorithm. Additionally, this paper is much less rigorous than the Kronecker paper.

This paper presents a feature-constraining paradigm and four key features that we would ideally try to use in our algorithm, although we ended up choosing simpler features for our limited timeframe. The paper also presents this backtracking optimization algorithm that, though less precise than a gradient descent optimization algorithm, may have better practical results with our problem.

2.3. Measuring Graph Similarity

An important factor in graph generation algorithms is in evaluating it and determining similarity of different graphs. A paper by Koutra et al. focuses on the idea of graph similarity and subgraph matching, and how we can develop algorithms for both to return metrics (Koutra et al., 2011). For example, the problem of graph similarity can be summarized as a problem of finding a one-to-one matching between the nodes of one graph and the nodes of the other (if they node sets are the same size), or at least matching nodes from one graph to another (if the node sets are different sizes). More specifically, the paper explores various approaches and key attributes in determining graph similarity such as edit distance, and graph isomorphisms and key feature extraction. The paper also goes through the idea of using belief propagation and other iterative measures like SimRank, which is a successful algorithm for evaluating self-similarity in graphs. Additionally, the paper explores the idea of subgraph matching, and how it can use Substructure Index-based Approximate Graph Alignment (SAGA), a measure developed so that there are fewer constraints. For example, SAGA can ignore node gaps, node mismatches and graph structural differences and does not require any constraints to be designed in advance. The paper also explores the idea of using tensor decomposition and PCA analysis to determine further subgraph matching.

We see that although the paper does in fact explore various features of graphs and their mathematical properties to evaluate a similarity metric, we see that it does not in fact take advantage of many other attributes. For example, there is no indication that motif counts factored into their metric at all. We also don't see any indication that the authors used any sort of community metrics or role metrics in matching nodes, but relied more on their mathematical spectra and eigenvalue properties. Finally, though there is a lot of exploration into subgraph matching, there is not a lot of exploration into the idea of subgraph permutations, or the idea that we can find some sort of mapping from one subgraph in one graph to a subgraph in the other. This is an idea that has been explored as an analogue to node mappings, and is something to consider if we use their similarity measures for our application.

3. Datasets

To start testing out various ideas, we first chose a dataset. We focused on biological datasets because of the relatively small size of the graphs in this domain. We used the Biological Network Repository source to find small biological networks.

Table 1. Description of the Datasets with High Level Features

Name	Nodes	Edges
Diseasome	516	1188
SC-TS	636	3959
SC-HT	2084	63027
Enzymes-G123	90	127

We ended deciding on the following four graphical models:

3.1. Bio-Diseasome Network

This is a network that details out the human disease network the and set of common genes that are known to be associated with each of these networks. As we can see from Table 1. this is a network that is relatively small and only has 516 nodes and 1188 edges. We can visualize the Bio-Diseasome Network below:

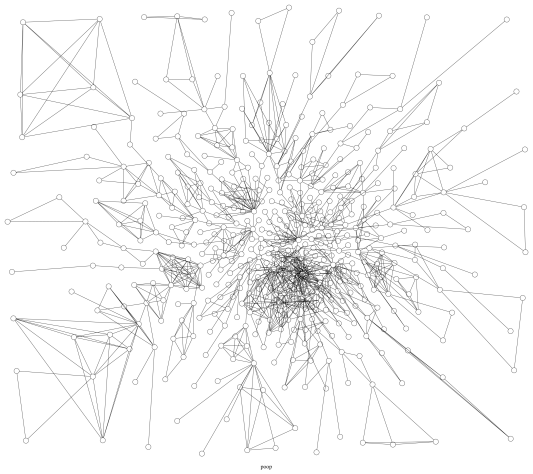

Figure 1. Bio-Diseasome Network visualized

We have the following list of additional qualities that we know about this graph:

- Minimum Degree: 1
- Maximum Degree: 50
- Average Degree: 4
- Average Clustering Coefficient: 0.63583
- Density: 0.00894

3.2. Bio-SC-TS

The Bio-SC-TS network details out the interaction of nucleic acid and can be visualized here:

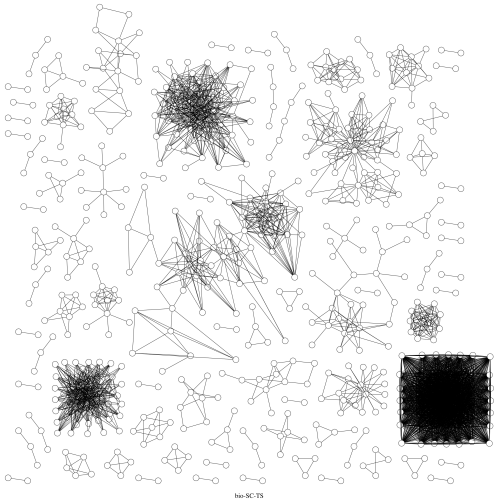

Figure 2. Bio-SC-TS Network visualized

This is a graph with 636 nodes and 3959 edges, and We can see that there are a lot of fragmented pieces to this network that are very connected. We have the following list of additional qualities that we know about this graph:

- Minimum Degree: 1
- Maximum Degree: 66
- Average Degree: 12
- Average Clustering Coefficient: 0.47124
- Density: 0.01961

3.3. Bio-SC-HT

This is a graph that deals with nucleic acid interactions with the predictions of RNAi phenotypes. The graph has 2084 nodes and 63,027 edges. This network is unfortunately too large to picture here, so we have just detailed out the other aspects of this network:

- Minimum Degree: 1
- Maximum Degree: 472
- Average Degree: 60
- Average Clustering Coefficient: 0.3491
- Density: 0.02904

3.4. Enzymes-G123

The Enzymes-G123 network details out the chemical and molecular structure of the mutant MAN47 enzymes, g-123. We see it visualized below:

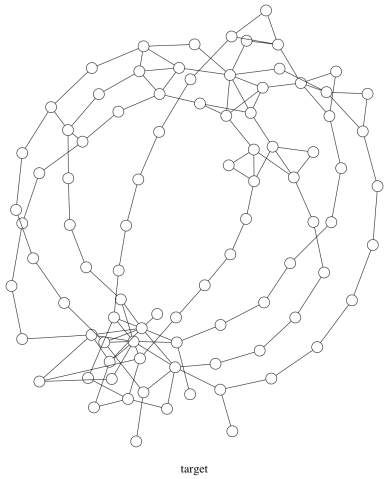

Figure 3. Enzymes-G123 Network visualized

This is a graph with 90 nodes and 127 edges, and we can see that there are two major halves to this molecule that are connected vertically and horizontally. We have the following list of additional qualities that we know about this graph:

- Minimum Degree: 1
- Maximum Degree: 9
- Average Degree: 2.82
- Average Clustering Coefficient: 0.1051
- Density: 0.0159

4. Initial attempts

In order to make this project tractable, we first had to make some decisions to limit the scope of the project. First of all, we decided that coming up with a unique loss function would be too ambitious for a single-quarter project. Thus we initially decided to focus on the backtracking approach similar to [2] instead of a loss function, gradient-descent type of approach.

Next we decided to first determine what are the best features to test such an algorithm.

One large part of our project depends on being able to generate initiator kernels that fit the desired features. In theory, we could then use these to generate a larger graph by taking repeated Kronecker products.

We initially tried to focus on using the spectrum of the Adjacency matrix as the feature to model. We took our single test graph, the bio-diseasome graph, and found the spectrum and related eigenvectors of it. Since we want to generate smaller kernels (of size 3 for our first attempt) then we took the 3 largest eigenvalues of the target graph, generated a random basis for $\mathbb{R}^3$. We then generated a new adjacency matrix $A' = \lambda_1 v_1 v_1^T + \lambda_2 v_2 v_2^T + \lambda_3 v_3 v_3^T$. We then divided by the max to normalize and generated a new graph by taking all values greater than 0.5. Finally, we constrained the generated graph to have the same number of edges as the target graph. After this was all done, we plotted the graph and, to no surprise, we didn't get very good results.

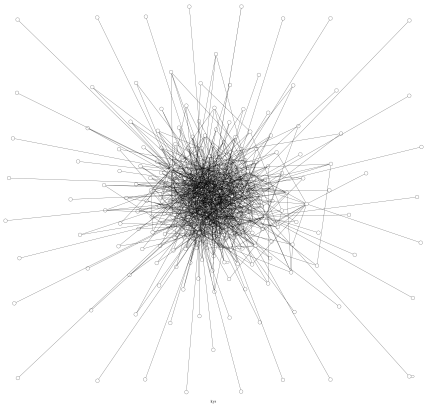

Figure 4. First Attempt at a Generated Graph

This looks very much like a random graph and visually not at all like our target graph. We also did some quantitative checks, first with the clustering coefficient. The clustering coefficient of the target graph was 0.63583 and the clustering coefficient of our generated graph was around 0.09.

From this experiment, we realized that the spectra of the Adjacency matrix alone is not expressive enough to maintain structure of a graph. Note that in this case we generated a graph of the same size of the target node to test our initial assumption that the spectra would retain features of the graph, but in our actual algorithm the intent of this would be to generate the small 3 or 4×4 initiator kernels. Nevertheless, we see that this approach doesn't work.

5. Random Kernel Kronecker Method

5.1. Random Selection

This is the first of the novel approaches proposed in this paper. We hypothesized that we could try the approach of randomly selecting a new kernel at each iteration from a base set of kernels. These kernels were chosen as some of the more common 3 and 4 node motifs in the graph. Note that this choice was done visually by us, not automatically.

We measured four main metrics: 1) Average Degree (AD), 2) Maximum Degree (MAX), 3) Minimum degree (MIN), and 4) the Clustering Coefficient of the graph (CLUST COEF)

After implementing this approach, we get the following results on each of the graphs:

For the bio-Diseasome graph,

GRAPH	A.D.	MAX	MIN	CLUST. COEF.
TARGET	4	50	1	0.63583
GENERATED	6.3	62	1	0.3421

Table 2. Random Selection of Kernels at Each iteration (bio-Diseasome)

We see from these results, that the additional choice of kernels vastly improves our ability to get the clustering coefficient closer to where it needs to be, while none of the other properties are wildly affected. Thus, this results seems to confirm the intuition that there is potential in offering these choices for the kernel at each iteration.

For the bio-SC-TS graph, we have the following results:

GRAPH	A.D.	MAX	MIN	CLUST. COEF.
TARGET	12	66	1	0.47124
GENERATED	9.3	65	1	0.3732

Table 3. Random Selection of Kernels at Each iteration (bio-SC-TS)

Once again, we see that these results are significantly better than the results of the baselines currently used, and thus we have a reason to continue to pursue this idea.

Finally, for the bio-SC-HT graph, we have the following results:

GRAPH	A.D.	MAX	MIN	CLUST. COEF.
TARGET	60	472	1	0.3491
GENERATED	34	298	1	0.2942

Table 4. Random Selection of Kernels at Each iteration (bio-SC-HT)

We see that among the three target graphs, this methods performs the worst on this graph. This is partly just due to the size of the graph, and how incredibly difficult it can be to generate a graph of that size, and maintain such specific structural properties.

5.2. Extension

We were limited by time, but a more theoretically sound way to approach this method would be to enumerate the subgraphs and then draw kernels randomly proportional to their counts in the graph. The steps would be as follows:

1. Read in a target graph
2. Enumerate all 3-5 node motifs
3. Generate a set of kernels corresponding to the enumerated motifs and a vector of motif counts
4. At each step of the Kronecker graph generation, randomly sample a kernel proportionally to its count in the original graph
5. Continue until full graph is generated

We could not test this approach since enumerating the graphs is NP-Hard problem and we didn't have the bandwidth to tackle this problem in addition to the other work we did.

We also note that using motifs larger than 5 nodes would probably be more expressive, but would probably be too computationally expensive.

5.3. Markov Decision Process

We also decided to take another, more unorthodox, approach to the problem. The problem of generating a random graph with a pre-subscribed set of features can also be formulated as an optimal policy problem in a Markov Decision Process. In other words, at each step of the process, what is the ideal kernel to select from a base set of kernels to yield a graph that most closely represents the target graph. Choosing a kernel at each iteration of the Kronecker process is the action step in our Markov Decision Process (MDP).

Our state space is simple as well. The state space is the set of all possible kernel combinations. Normally, this would be a rather large state space, but since we only chose about 5 relevant kernels that are valid initiators for these graphs, and we only require about 4 or 5 iterations, this is only a state space size of 625 or 3125, which is quite manageable.

5.3.1. SIMILARITY COMPARISON METRICS

Part of what makes this such an interesting problem is the lack of existing, open-source graph comparison metrics. Though industry has suggested approaches, like Substructure Index-based Approximate Graph Alignment (SAGA) and Approximate Constrained Subgraph Matching (ACSM) as evaluation metrics, there is no accepted standard.

Thus, we decided that we needed a reasonable way to compare how good our generated graphs were that took inspiration from these metrics. There are some clear and obvious important characteristics of similar graphs. First, we have the maximum and minimum degree. Next, we have structural properties like clustering coefficient, density, and average degree. Thus, we have a proposed similarity metric to use for our MDP models.

Consider two graphs G_1 and G_2. Now, let each have a feature vector v_1 and v_2. Our comparison metric will be:

$$C(G_1, G_2) = ||v_1 - v_2||_2^2$$

Now, what is the feature vector? It is a normalized vector of the four features listed above: maximum degree, minimum degree, clustering co-efficient, and average degree.

Given these metrics, and normalizing among them to account for magnitude differences, we are able to simulate a comparison metric among two graphs.

5.3.2. Policy Iteration

Now, this problem falls under policy iteration, since our goal is to find the optimal choice of kernel choices such that the final graph resembles our target graph in some way. Thus, we want to follow the general algorithm described below:

Algorithm 4.2 Policy iteration

```
function PolicyIteration(π_0)
    k ← 0
    repeat
        Compute U^{π_k}
        π_{k+1}(s) = arg max_a (R(s,a) + γ Σ_{s'} T(s' | s,a) U^{π_k}(s')) for all states s
        k ← k + 1
    until π_k = π_{k-1}
    return π_k
```

Figure 5. Policy Iteration Psuedocode

Following this policy iteration idea, we attempted trails on the Enzymes-g123 dataset and the bio-Diseasome dataset, but were unable to get reasonable policies. Most of the issue was with the fact that every iteration, to evaluate the cost at that point in the Kronecker generation, we would take a subgraph of the target graph and evaluate a cost on that subgraph and our current generated graph. However, this does not necessarily represent the target graph particularly well, especially at early stages in the process. For example, after making 2 choices in the MDP, we would have a generated graph of 9-16 nodes, but to evaluate a cost, we would take a 9-16 node subgraph of the target graph and evaluate the cost. Clearly, there are scale issues with this idea, and they greatly affected our ability to get proper results.

Because of this scale issue, at early iterations, the generated graphs learned from very small subgraphs, which tended to be highly connected. Thus, the generated graphs tended to be extremely highly connected, to the point where visualizing them was like looking at a large black box because of how many edges were present.

5.3.3. Improvements to Consider

Due to time constraints, we were not able to implement certain natural improvements to this method. First, we would consider applying an edge constraint so that the final generated graph does not have an incredibly large number of edges because the initial kernel choices promoted high connectivity. Next, we would have to devise a better, more mathematically sound, loss function that took into account certain structural properties like motifs or subgraph properties. Though finding motifs and enumerating counts would be an incredibly difficult implementation, we could look at other structural properties like diameter as well.

6. Multi-Kernel KRONFIT

As we saw that a lot of our initial attempts were not giving us ideal results, we decided to try to implement a modification to an existing algorithm: KRONFIT.

6.1. KRONFIT

KRONFIT is an algorithm presented by (Leskovec et al., 2009) that can be used to fit the Kronecker graph generation model to large real networks in linear time rather than the naive exponential time. It does so by calculating a likelihood of a Graph given a generated graph and permutation,

$$P(G|P,\sigma) = \prod_{(u,v)\in G} P[\sigma_u, \sigma_v] \prod_{(u,v)\notin G} (1P[\sigma_u, \sigma_v])$$

where P is the stochastic adjacency matrix of the generated graph.

In order to prevent the exponential sum over all possible permutations of the graph, the algorithm randomly samples a permutation based on the likelihood of that permutation given the target Graph and P using a Metropolis sampling method.

This likelihood is differentiable and the kernel values are thus optimized by calculating the gradient of the likelihood and updating the parameters accordingly.

6.2. Motivation

We notice that KRONFIT uses a single initiator matrix and, due to the fact that the Kronecker product of an $n \times n$ matrix with an $m \times m$ matrix results in an $mn \times mn$ matrix, then the outputted final graphs are constrained to be powers of the size of the initiator matrix.

However, the algorithm doesn't ever assume that there is only a single initiator matrix. It merely uses the final generated graph, or more precisely its associated stochastic adjacency matrix, to calculate the likelihood of these parameters. Thus we can add multiple kernels and run the algorithm as designed.

One of our biggest motivations for this project was extend-

ing Kronecker-like methods with multiple kernels, so we decided to re-implement KRONFIT but use multiple kernels of different sizes instead of a single initiator matrix. We did this in order to get more granularity with the output size of KRONFIT, and to be able to fit more graphs.

6.3. Method

Given a graph with n nodes, let p_n be a list containing the prime factorization of n. For example, let $n = 90$. Thus $p_n = [2, 3, 3, 5]$. Instead of our graph generation being the repeated Kronecker product of a single initiator matrix, our graph generation becomes

$$G_{\text{generated}} = p_n^{(0)} \otimes p_n^{(1)} \otimes p_n^{(2)} \otimes p_n^{(3)}$$

Aside from this modification, the bulk of the algorithm stays the same. We decided to implement it using TensorFlow 2.0 so that it would automatically calculate gradients based on our negative likelihood loss function as described in the original Kronecker Graphs paper (Leskovec et al., 2009).

6.4. Testing

We note that our implementation is much less efficient than the optimized SNAP version so we ran our tests on the Google Cloud Platform with a small 90-node graph from (Rossi & Ahmed, 2015), ENZYMES-g123, a cheminformatics graph representing enzyme molecules.

All of our generated data was run using 5 epochs. We report the feature values for the target graph, three sample runs, and the average over 100 runs.

6.5. Results

The initial training attempt with Multi-Kernel Kronfit was on the Enzymes-G123 dataset, which was the smallest of our four networks. Enzymes-G123 was chosen to test the principle of the idea so that we could determine if it was worth pursuing larger graph. The following table details our results with this graph.

GRAPH	A.D.	MAX	MIN	C.C.	DIAM.
TARGET	2.82	9	1	0.1051	12
GEN. -1	3.02	8	1	0.0371	9
GEN. -2	2.95	7	1	0.0063	9
GEN. -3	2.98	7	1	0.0147	10
AVG. GEN.	**2.98**	**7.33**	**1**	**0.0193**	**9.66**

Table 5. Multiple Attempts of Multi-Kernel Kronfit

From the table, it is clear that the generated graph matched the average degree and maximum/minimum degree relatively well, but did not do particularly well in matching the clustering coefficient.

A typical example of a generated graph is shown to compare to our target graph:

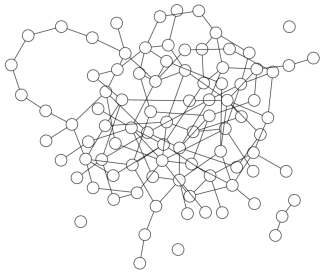

Figure 6. Generated Graph from Multi-Kernel Kronfit

As we can see from this image, the generated image does not necessarily incorporate the large structural scheme of two halves that was prevalent in our original network. However, it does represent the general connectivity of the target graph relatively well, at least to the naked eye.

6.6. Discussion

This algorithm didn't do particularly well at matching the clustering coefficient of the target graph. This is probably because the clustering coefficient holds a lot of data related to local structure, while it's harder for a graph generation algorithm that starts from a small sample space to learn how to recreate that local structure. I'm sure that with more and better training, it's possible for it to have worked, but we didn't find this with our limited trials.

Additionally, we weren't able to run the algorithm that many times due to a lot of issues in fine-tuning the training that presented a major block in the project. We had issues with vanishing and then exploding gradients that initially led to sub-par results.

We suspect this is due to the fact that our parameter space is much larger than the original algorithm. Apart from having multiple sized kernels, our first run had a separate kernel for each prime factor of the number of nodes. One way to mitigate this would be to share kernels across sizes. For example, with $n = 90$, we would have four iterations with the kernels corresponding to the sizes $[2, 3, 3, 4]$, but three kernels, of sizes 2×2, 3×3, and 4×4. In this example, the gains are small but in other examples this might help reduce the parameter space and improve training considerably.

7. Code

Find our code at the following repo:

https://github.com/vsaladi99/CS224W_FinalProject

Contributions

Federico: Problem formulation, coming up with the Multi-Kernel KRONFIT algorithm, writing that code in TF, testing, writing the paper and calculating results.

Vamsi: Coming up with Random Kernel Kronecker Method, running most tests with that and earlier attempts, attempting a Markov Decision Process, calculating graph statistics, writing the paper, and plotting graphs.

Markie: Technical problem formulation, literature review research, writing the paper and poster.

Acknowledgements

We would like to thank Jure Leskovec for providing us the opportunity to do this project and for providing us the knowledge necessary to carry out the research. We would additionally like to thank the Biological Network Repository for providing the datasets that we used for our project.

References

Koutra, D., Parikh, A., Ramdas, A., and Xiang, J. Algorithms for graph similarity and subgraph matching. In *Proc. Ecol. Inference Conf*, volume 17, 2011.

Langley, P. Crafting papers on machine learning. In Langley, P. (ed.), *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pp. 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

Leskovec, J., Lang, K. J., Dasgupta, A., and Mahoney, M. W. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

Rossi, R. A. and Ahmed, N. K. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015. URL http://networkrepository.com.

Ying, X. and Wu, X. Graph generation with prescribed feature constraints. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pp. 966–977. SIAM, 2009.