BioNetwork Research: 2019 Fall Report

Henry Ye

During this quarter, I have provided network alignment seeds to Utsav & Tina, implemented the neighboring window distribution table, and experimented with synthetic network generation using a stamping-like algorithm and MCMC random walk.

1. Network Alignment seeds

From the last few quarters, we have tried various windowReps sampling methods, graphlet and window sizes on PPI networks, hoping to use sampled windowReps as network database index. However, the obtained results were not promising. Considering we only tested on PPI networks, at the beginning of this quarter, I obtained several new networks in different fields such as AutoSys, Facebook social network, and FoodWeb from Sridevi. Unfortunately, poor performance persists. After all experiments, we realized that it's improbable for windowReps to be used as database index, and then decided to move on to network alignment. Collaborating with Utsav & Tina on their Dijkstra alignment algorithm project, I am providing them with windowReps (maximizer and maximizer with distance) as initial alignment seeds.

2. Neighboring Window Distribution Table

In the new synthetic network generation project, an MCMC random walk algorithm was proposed. Based on the neighboring graphlet distributions and removed/added nodes, we can determine the canonical form for the next stamped graphlet. However, as a first step, we need to make the neighboring graphlet distribution table. Through the BLANT MCMC sampling methods, we can easily keep track of the canonical integers and node differences between current and next sampled

graphlets. In the end, I successfully implemented this functionality in the BLANT repo. The following Figure 1 shows a sample command with its corresponding results:

```
$ ./blant -k4 -md -sMCMC -n100000 -t4 networks/SCerevisiae.el
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 68342 0 0 4130 3740 164 395 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0
0 0 0 4153 0 0 2298 1328 287 273 0
0 0 0 3739 0 0 1309 3740 69 1165 504
0 0 0 181 0 0 287 64 41 24 0
0 0 0 382 0 0 307 1146 24 533 286
0 0 0 0 0 0 0 509 0 234 346
```

Figure 1: The row index is the ordinal canonical of the current graphlet, whereas the column index represents the ordinal canonical of the neighboring graphlet. The table entries note such graphlet counts.

3. Synthetic Network Generation

3a. Stamping-like Algorithm

Before we start implementing MCMC random walk, we devised a more naïve way to generate a synthetic network through stamping the sampled k-node graphlet (g1) from the graphlet concentration onto the randomly chosen k-node graphlet (g2) from the synthetic network. Regardless of current edges of g2, we remove all its edges and add them back based on the canonical form of g1. Since there are no neighboring graphlets involved, we can use current BLANT MCMC and NBE sampling methods to obtain the graphlet concentration map. After repeating the above steps, we might encounter the problem where the number of synthetic network edges far exceeds the one of the original network. As a result, we will remove a specified number

of edges in the synthetic network once it starts having more edges than the original one. If we make the step of adding and removing edges as one iteration, we can define the stopping criteria of this algorithm: either the synthetic network passes the K-S test or the number of iterations exceeds the specified maximum number. By following the above procedure, we got the following results for 5-node graphlet on syeast.el after 1,000,000 samples:

GintOrdinal	OriginalPDF	SyntheticPDF	OriginalCDF	SyntheticCDF
4	0.026854	0.054506	0.026854	0.054506
10	0.149888	0.464738	0.176742	0.519244
11	0.089671	0.014887	0.266413	0.534131
14	0.098516	0.407260	0.364929	0.941391
15	0.108122	0.025451	0.473051	0.966842
16	0.010064	0.003675	0.483115	0.970517
17	0.080566	0.001655	0.563681	0.972172
18	0.000218	0.000000	0.563899	0.972172
19	0.008116	0.000000	0.572015	0.972172
22	0.123448	0.023589	0.695463	0.995761
23	0.069229	0.002573	0.764692	0.998334
24	0.088124	0.000250	0.852816	0.998584
25	0.026380	0.000342	0.879196	0.998926
26	0.000607	0.000375	0.879803	0.999301
27	0.005750	0.000647	0.885553	0.999948
28	0.029329	0.000052	0.914882	1.000000
29	0.002011	0.000000	0.916893	1.000000
30	0.041921	0.000000	0.958814	1.000000
31	0.002630	0.000000	0.961444	1.000000
32	0.022679	0.000000	0.984123	1.000000
33	0.015876	0.000000	1.000000	1.000000

Distance: 0.576462
Obtained K_statistics: 12.962093
Obtained p_value: 0.000000

Figure 2: A graphlet distribution comparison between the original network and synthetic network

From Figure 2, we can notice that the maximum CDF distance found was around 0.58, which means that the two networks were not even close to each other. Considering there might be some bias introduced by BLANT sampling methods, we compared the graphlet distributions on the same network twice to examine the expected K-S statistics.

```
$ ./blant -k5 -gNBE -c0.05 -n1000000 networks/syeast.el
Steps made to remove Edges:  0
GintOrdinal     OriginalPDF     SyntheticPDF    OriginalCDF     SyntheticCDF
4       0.026849        0.034532        0.026849        0.034532
10      0.148221        0.104399        0.175070        0.138931
11      0.090835        0.100965        0.265905        0.239896
14      0.096368        0.087387        0.362273        0.327283
15      0.107794        0.112366        0.470067        0.439649
16      0.010318        0.006520        0.480385        0.446169
17      0.081108        0.099195        0.561493        0.545364
18      0.000210        0.001040        0.561703        0.546404
19      0.008180        0.009872        0.569883        0.556276
22      0.122995        0.136292        0.692878        0.692568
23      0.069234        0.072529        0.762112        0.765097
24      0.089380        0.076110        0.851492        0.841207
25      0.026212        0.034460        0.877704        0.875667
26      0.000572        0.000000        0.878276        0.875667
27      0.005883        0.002928        0.884159        0.878595
28      0.029267        0.030461        0.913426        0.909056
29      0.002012        0.002955        0.915438        0.912011
30      0.042689        0.049250        0.958127        0.961261
31      0.002652        0.001845        0.960779        0.963106
32      0.023155        0.020263        0.983934        0.983369
33      0.016066        0.016630        1.000000        1.000000
Distance:  0.036139
Obtained K_statistics:  0.812607
Obtained p_value:   0.523774
```

Figure 3: A graphlet distribution comparison between the original network and itself.

From Figure 3, we can observe that the two graphlet distributions from the same network were close enough to each other so that they passed the K-S test. Compared to the previous results, we decided to move on to other methods since it would be challenging for this method to have a massive improvement.

3b. MCMC Random Walk

Unlike the previous stamping-like method, the MCMC random walk algorithm will utilize the neighboring distribution. Starting from one sampled k-node graphlet (g_curr), we decide the canonical form for the next k-node graphlet (g_next) after removing and adding one node randomly from g_curr. However, the major challenge of this method is to construct g_next so that it can have a sampled canonical form. Since g_next and g_curr share k-1 common nodes, which means that those two graphlets have the same k-1 rows of their

corresponding adjacency matrix, we have to permute the last row of the adjacency matrix of g_next so that it can match with the sampled canonical form. However, sometimes, after permuting all possibilities for the last row, we still cannot construct expected g_next. Due to the factorial runtime and the uncertainties, the generating process for this method becomes too slow to complete. Thus, for future work, we can look for a better way to construct g_next.

4. Comparisons of current network indexing methods

During the last two weeks of this quarter, I started looking for current network indexing methods and trying to compare them with BLANT. There are a couple of papers, such as REFBSS, SAGA, QNET, and RINQ, which show competing results. However, there is no source code published along with the paper. As a result, in the current stage, I am reaching out to the authors for their implementations so that I can begin testing them against BLANT.