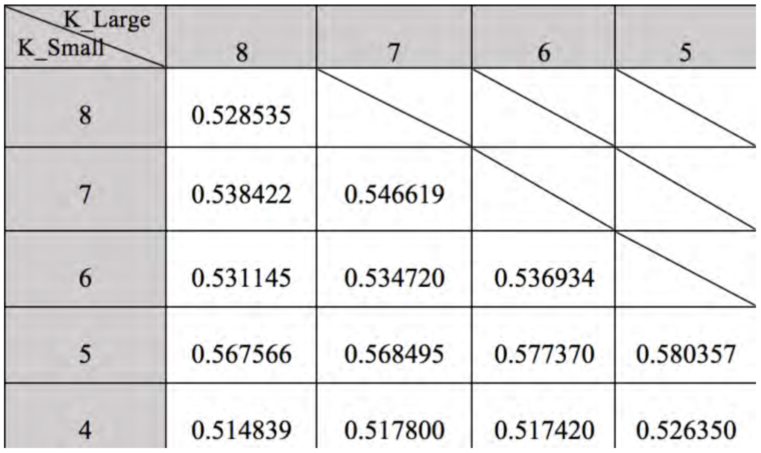
BioNetwork Research: 2020 Winter Report

Henry Ye

During this quarter, I have further tested the performance of the synthetic network generation stamping method, examined existing competing network local alignment algorithms, and added stricter constraints to windowRep seed generation procedure.

1. Synthetic Network Generation

From the last quarter, we have seen that the graphlet distributions of generated synthetic networks were not even close to the ones of original networks based on K-S distance. Since the previous K-S test was using the same K number as the one specified for stamping size, this time we tried using a larger K number to stamp graphlet and a smaller K number to obtain the K-S statistics. The idea behind using different K number is that stamping a bigger graphlet might account for overlapping issue. To test our hypothesis, we experimented with the new procedure on syeast.el using 1M sample, and was able to obtain the following K-S distances.



K_Large / K_Small	8	7	6	5
8	0.528535			
7	0.538422	0.546619		
6	0.531145	0.534720	0.536934	
5	0.567566	0.568495	0.577370	0.580357
4	0.514839	0.517800	0.517420	0.526350

Table 1 (syeast.el, 1M samples): K-S distances between the original and synthetic networks

From Table 1, we can notice that the K-S distances are all quite similar to each other among different choices of K_Large and K_Small, and the results have not been improved much since last time. To find the expected K-S distances, we reran the previous algorithm but compared the graphlet distributions twice on the same original network.

K_Large / K_Small	8	7	6	5
8	0.007426			
7	0.004889	0.005852		
6	0.001766	0.001913	0.001869	
5	0.001503	0.001371	0.001265	0.001785
4	0.001678	0.001610	0.001592	0.001793

Table 2 (syeast.el, 1M samples): K-S distances between the original network and itself

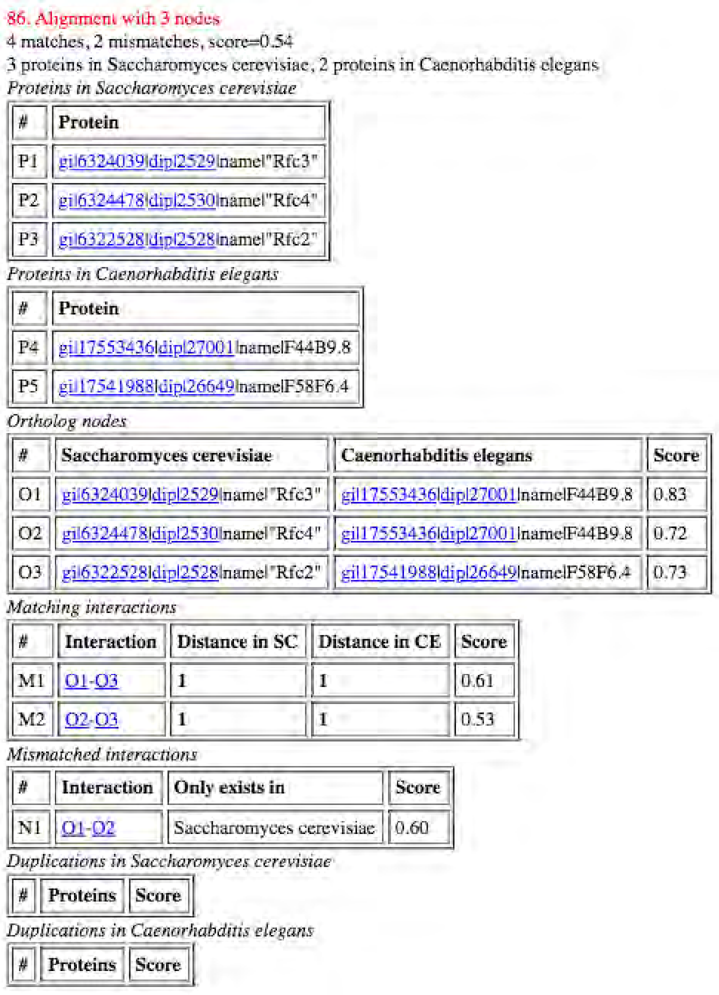
As we can observe from Table 2, the expected K-S distances are still significantly smaller than the ones we have seen from Table 1. Since there is a long way to optimize the current stamping algorithm having K-S distance close to the expected ones, we decided to focus on the Dijkstra and windowRep seed generation at this point.

2. Competing Network Alignment Algorithms

To test the performance of Dijkstra network alignment algorithm, we have to compare it against existing popular network aligners. In the past several weeks, I have examined MAWISH (2006), networkBLAST (2008), AlignNemo (2012), GASOLINE (2014), and GLAlign (2019).

2.1 MAWISH

MAWISH was presented in “Pairwise Alignment of Protein Interaction Networks” by Mehmet. It weighs the edges of the alignment graph by the ortholog similarity scores and defines the alignment scores by combining match, mismatch and duplication scores of ortholog protein pairs. The implementation of this algorithm came with the paper. Based on their given sample datasets, the program can output the exact found alignment and corresponding alignment scores (see figure 1). However, the algorithm will fail on our networks, throwing “Inconsistent number of ortholog pairs” error, even though the input files are formatted in the same way as the given sample data.



86. Alignment with 3 nodes
4 matches, 2 mismatches, score=0.54
3 proteins in Saccharomyces cerevisiae, 2 proteins in Caenorhabditis elegans

Proteins in Saccharomyces cerevisiae

#	Protein
P1	gi|6324039|dip|2529|name|"Rfc3"
P2	gi|6324478|dip|2530|name|"Rfc4"
P3	gi|6322528|dip|2528|name|"Rfc2"

Proteins in Caenorhabditis elegans

#	Protein
P4	gi|17553436|dip|27001|name|F44B9.8
P5	gi|17541988|dip|26649|name|F58F6.4

Ortholog nodes

#	Saccharomyces cerevisiae	Caenorhabditis elegans	Score
O1	gi|6324039|dip|2529|name|"Rfc3"	gi|17553436|dip|27001|name|F44B9.8	0.83
O2	gi|6324478|dip|2530|name|"Rfc4"	gi|17553436|dip|27001|name|F44B9.8	0.72
O3	gi|6322528|dip|2528|name|"Rfc2"	gi|17541988|dip|26649|name|F58F6.4	0.73

Matching interactions

#	Interaction	Distance in SC	Distance in CE	Score
M1	O1-O3	1	1	0.61
M2	O2-O3	1	1	0.53

Mismatched interactions

#	Interaction	Only exists in	Score
N1	O1-O2	Saccharomyces cerevisiae	0.60

Duplications in Saccharomyces cerevisiae

#	Proteins	Score

Duplications in Caenorhabditis elegans

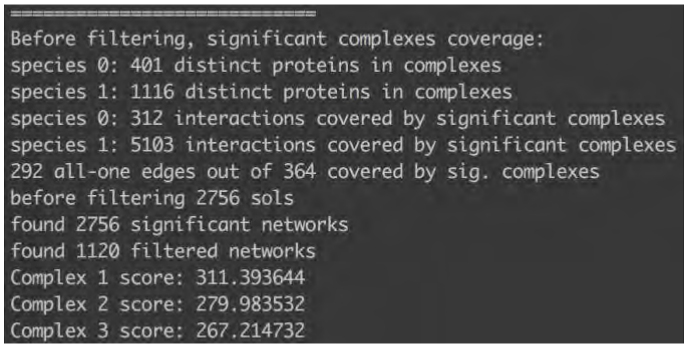
#	Proteins	Score

Figure 1: A sample output from MAWISH on given SCerevisiae, CElegans networks.

(SC: 5157 nodes, 18192 edges; CE: 3345 nodes, 5988 edges; Orthologs: 15740)

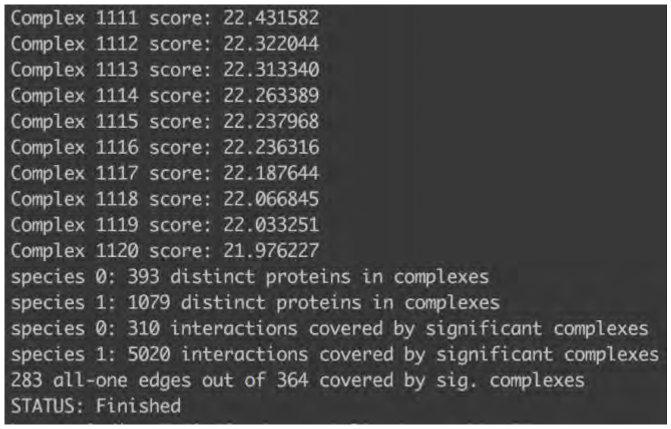
2.2 NetworkBLAST

The NetworkBLAST algorithm was designed to identify protein complexes that are conserved in evolution across species. Each found complex is later assigned a likelihood ratio score to test its fit to a protein complex model (an alignment) versus the null hypothesis that it arises at random. There is an online web server implementation of this algorithm. However, it only produces some simple statistics of found complex (alignment) but without the exact interactions, which turns out to be unhelpful in our comparison. Following Figure 2 is a sample program output:



```
================================
Before filtering, significant complexes coverage:
species 0: 401 distinct proteins in complexes
species 1: 1116 distinct proteins in complexes
species 0: 312 interactions covered by significant complexes
species 1: 5103 interactions covered by significant complexes
292 all-one edges out of 364 covered by sig. complexes
before filtering 2756 sols
found 2756 significant networks
found 1120 filtered networks
Complex 1 score: 311.393644
Complex 2 score: 279.983532
Complex 3 score: 267.214732
```

…

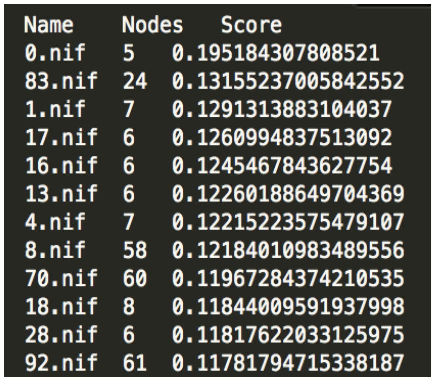


```
Complex 1111 score: 22.431582
Complex 1112 score: 22.322044
Complex 1113 score: 22.313340
Complex 1114 score: 22.263389
Complex 1115 score: 22.237968
Complex 1116 score: 22.236316
Complex 1117 score: 22.187644
Complex 1118 score: 22.066845
Complex 1119 score: 22.033251
Complex 1120 score: 21.976227
species 0: 393 distinct proteins in complexes
species 1: 1079 distinct proteins in complexes
species 0: 310 interactions covered by significant complexes
species 1: 5020 interactions covered by significant complexes
283 all-one edges out of 364 covered by sig. complexes
STATUS: Finished
```

Figure 2: Found Protein Complex between MM (Species 0) and HS (Species 1).

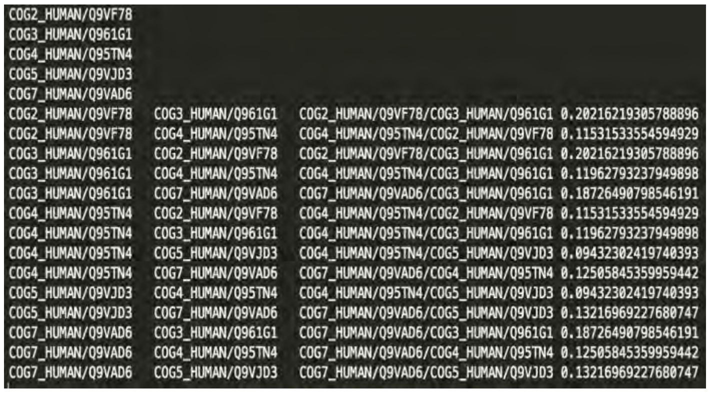
2.3 AlignNemo

AlignNemo utilizes ortholog pairs to merge input networks through constructing a union graph, whose nodes are consisted of ortholog pairs (composite) and unique nodes in each of input networks (simple). The edges of the union graph are weighted by the number of paths (of length at least 2) connecting the two nodes. With the constructed union graph, AlignNemo will extract and rank each k-node subgraphs, and then extend top-ranking subgraphs to form an alignment. The implementation of this algorithm also came with the paper. Given the sample dataset Human (12113 nodes, 83321 Edges), Fly (8043 nodes, 25999 Edges) and their ortholog file (7063 pairs), the program was able to complete and produce the following results (Figure 3).



```
Name    Nodes   Score
0.nif   5    0.195184307808521
83.nif  24   0.13155237005842552
1.nif   7    0.1291313883104037
17.nif  6    0.1260994837513092
16.nif  6    0.1245467843627754
13.nif  6    0.12260188649704369
4.nif   7    0.12215223575479107
8.nif   58   0.12184010983489556
70.nif  60   0.11967284374210535
18.nif  8    0.11844009591937998
28.nif  6    0.11817622033125975
92.nif  61   0.11781794715338187
```

Figure 3a: Alignment Summary Report



```
COG2_HUMAN/Q9VF78
COG3_HUMAN/Q961G1
COG4_HUMAN/Q95TN4
COG5_HUMAN/Q9VJD3
COG7_HUMAN/Q9VAD6
COG2_HUMAN/Q9VF78   COG3_HUMAN/Q961G1   COG2_HUMAN/Q9VF78/COG3_HUMAN/Q961G1 0.20216219305788896
COG2_HUMAN/Q9VF78   COG4_HUMAN/Q95TN4   COG4_HUMAN/Q95TN4/COG2_HUMAN/Q9VF78 0.11531533554594929
COG3_HUMAN/Q961G1   COG2_HUMAN/Q9VF78   COG2_HUMAN/Q9VF78/COG3_HUMAN/Q961G1 0.20216219305788896
COG3_HUMAN/Q961G1   COG4_HUMAN/Q95TN4   COG4_HUMAN/Q95TN4/COG3_HUMAN/Q961G1 0.11962793237949898
COG3_HUMAN/Q961G1   COG7_HUMAN/Q9VAD6   COG7_HUMAN/Q9VAD6/COG3_HUMAN/Q961G1 0.18726490798546191
COG4_HUMAN/Q95TN4   COG2_HUMAN/Q9VF78   COG4_HUMAN/Q95TN4/COG2_HUMAN/Q9VF78 0.11531533554594929
COG4_HUMAN/Q95TN4   COG3_HUMAN/Q961G1   COG4_HUMAN/Q95TN4/COG3_HUMAN/Q961G1 0.11962793237949898
COG4_HUMAN/Q95TN4   COG5_HUMAN/Q9VJD3   COG4_HUMAN/Q95TN4/COG5_HUMAN/Q9VJD3 0.09432302419740393
COG4_HUMAN/Q95TN4   COG7_HUMAN/Q9VAD6   COG7_HUMAN/Q9VAD6/COG4_HUMAN/Q95TN4 0.12505845359959442
COG5_HUMAN/Q9VJD3   COG4_HUMAN/Q95TN4   COG4_HUMAN/Q95TN4/COG5_HUMAN/Q9VJD3 0.09432302419740393
COG5_HUMAN/Q9VJD3   COG7_HUMAN/Q9VAD6   COG7_HUMAN/Q9VAD6/COG5_HUMAN/Q9VJD3 0.13216969227680747
COG7_HUMAN/Q9VAD6   COG3_HUMAN/Q961G1   COG7_HUMAN/Q9VAD6/COG3_HUMAN/Q961G1 0.18726490798546191
COG7_HUMAN/Q9VAD6   COG4_HUMAN/Q95TN4   COG7_HUMAN/Q9VAD6/COG4_HUMAN/Q95TN4 0.12505845359959442
COG7_HUMAN/Q9VAD6   COG5_HUMAN/Q9VJD3   COG7_HUMAN/Q9VAD6/COG5_HUMAN/Q9VJD3 0.13216969227680747
```

Figure 3b: Alignment 0.nif with interaction and alignment scores

However, when testing this program on our Mouse (6653 nodes, 17576 edges) and Rat (2266 nodes, 3312 edges) networks, it failed to finish (or need to take a really long time). One possible reason is that there are 1,840,342 orthologs between those two species, which will make the union graph too large to be built.

2.4 GASOLINE

GASOLINE utilizes a seed-and-extend approach, where the seeds are found by MCMC with the Gibbs Sampling algorithm. Then it will extend or shrink the alignments by another Gibbs sampling while taking into account both sequence and topological similarities. In the final stage, all found alignments are ranked by ISC (index of structural conservation) and their sizes. Since their source code is not published, we cannot obtain its results on our networks at this point.

2.5 GLAlign

The general idea of GLAlign is first to use global alignment algorithms to extract the topological information and then perform local alignment methods. Because there can be different combinations of global aligners (e.g. SANA, MAGNA++, NETAL, etc) and local aligners (e.g. AlignNemo, AlignMCL, LocalAli, etc), the performance varies across various architectures. The authors of GLAlign published their source code in R, which can be investigated in the next quarter.

3. WindowRep Seed Generation

During the last few weeks of this quarter, I added new constraints to the seed generation process and have been testing the Uniqueness/Stability tradeoffs of the new methods. The first constraint is an ambiguity check. Considering there are multiple node permutations of a triangular canonical graphlet, the found triangular seeds might not have node-to-node correspondence. Thus, we need to find one consistent node representation, which led to the unambiguous graphlet - having unique orbit for each of its nodes. The second constraint is to only record the top K windowReps based on their total degree or the number of edges 1-step away.

	Time_7	Time_8	AvgLS_7	AvgLS_8	AvgLSInt_7	AvgLSInt_8
AThaliana	826m30.456s	665m6.531s	4.21349	3.13131	9.94036	5.24476
CElegans	173m46.736s	213m22.216s	2.78506	2.45109	3.66435	2.99401
DMelanogaster	70m26.121s	79m9.440s	2.76805	2.36022	4.97347	3.32484
HSapiens	141m34.918s	127m5.935s	3.13647	2.57371	6.1627	3.01568
MMusculus	1607m14.972s	1291m23.327s	4.3073	3.17103	14.3266	10.5263
RNorvegicus	1555m35.278s	1346m48.660s	3.92608	3.02707	12.3813	11.8017
SCerevisiae	326m4.590s	351m1.854s	3.28616	2.64253	6.68151	3.59712
SPombe	271m35.996s	322m13.798s	2.87851	2.48219	4.9456	3.51288

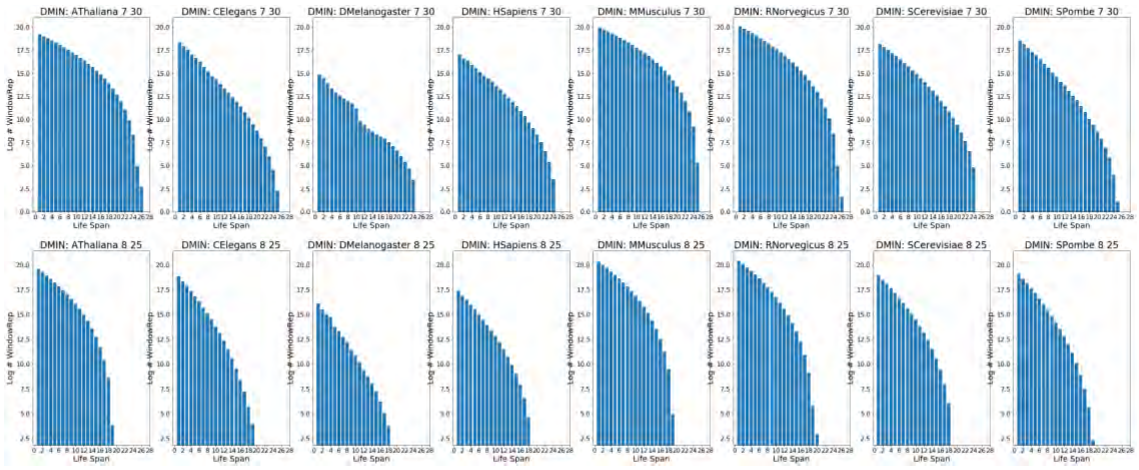


Figure 4a: WindowRep Lifespan from DMIN

	Time_7	Time_8	AvgLS_7	AvgLS_8	AvgLSInt_7	AvgLSInt_8
AThaliana	173m54.684s	142m54.520s	3.45508	2.49258	5.6159	3.29735
CElegans	68m50.352s	70m52.514s	3.01448	2.53616	4.07152	3.26319
DMelanogaster	65m32.021s	76m0.100s	3.85478	2.68652	6.69431	3.61011
HSapiens	108m47.653s	89m46.263s	4.88799	2.69051	6.78742	3.23886
MMusculus	214m33.105s	181m58.959s	3.74122	2.69129	5.95871	3.64354
RNorvegicus	241m25.412s	201m46.018s	3.86228	3.16781	5.9633	4.51572
SCerevisiae	182m41.242s	160m34.737s	4.48474	2.87138	7.0441	3.57678
SPombe	128m22.984s	120m19.464s	3.64324	2.62436	5.06572	3.4863

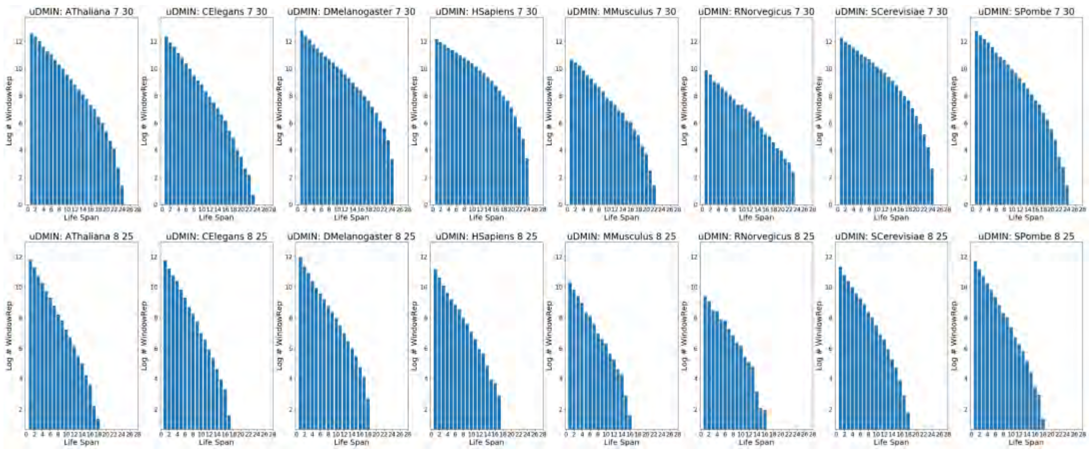


Figure 4b: WindowRep Lifespan from DMIN + unambiguous check (uDMIN)

	Time_7	Time_8	AvgLS_7	AvgLS_8	AvgLSInt_7	AvgLSInt_8
AThaliana	202m40.998s	148m18.201s	3.04501	2.40443	4.55895	2.94988
CElegans	76m34.029s	69m41.669s	2.86832	2.40192	3.7234	3.06967
DMelanogaster	65m1.850s	71m51.774s	3.14461	2.51993	4.57114	3.33777
HSapiens	133m2.568s	88m6.906s	2.60239	2.06687	4.25453	2.68913
MMusculus	259m16.361s	184m45.265s	3.30973	2.48412	4.78064	3.18942
RNorvegicus	271m0.523s	200m28.244s	3.38792	2.87743	5.01732	4.18634
SCerevisiae	207m2.568s	154m11.791s	2.98904	2.29957	4.21126	2.71963
SPombe	133m2.739s	122m54.720s	2.82629	2.31315	3.91756	3.08453

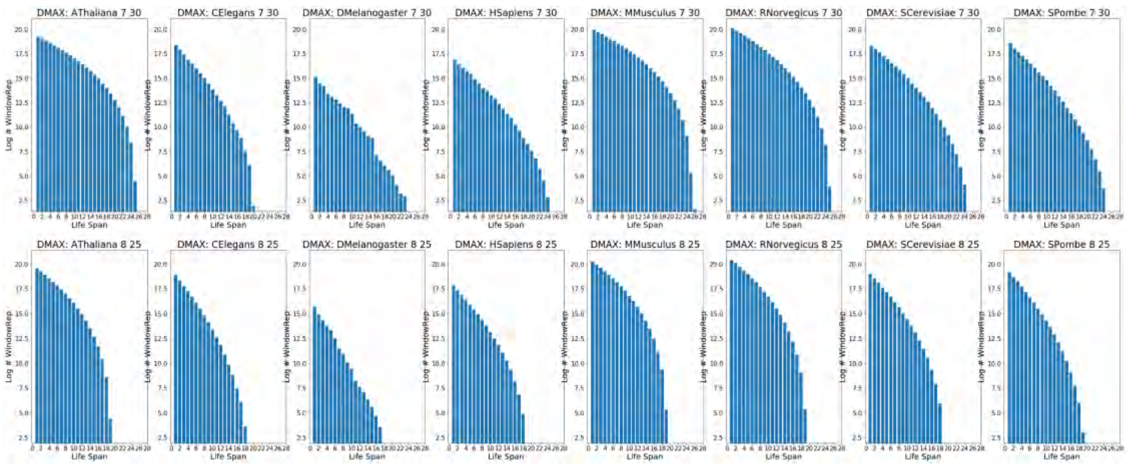


Figure 4c: WindowRep Lifespan from DMAX

	Time_7	Time_8	AvgLS_7	AvgLS_8	AvgLSInt_7	AvgLSInt_8
AThaliana	838m0.213s	651m17.051s	4.20809	3.09733	5.45951	4.04476
CElegans	165m31.459s	209m8.541s	2.58542	2.30562	3.16456	2.61301
DMelanogaster	79m25.562s	76m53.218s	2.73008	2.08815	4.52352	3.25662
HSapiens	130m33.586s	153m35.859s	3.01204	2.536	3.81534	2.62055
MMusculus	1574m26.697s	1221m24.013s	4.24531	3.14723	9.69932	8.10154
RNorvegicus	1495m17.965s	1317m44.420s	3.81897	3.00298	9.5816	9.72763
SCerevisiae	331m49.859s	359m45.964s	3.20449	2.60762	3.91236	2.997
SPombe	265m1.085s	337m53.554s	2.90289	2.46822	3.43525	2.84765

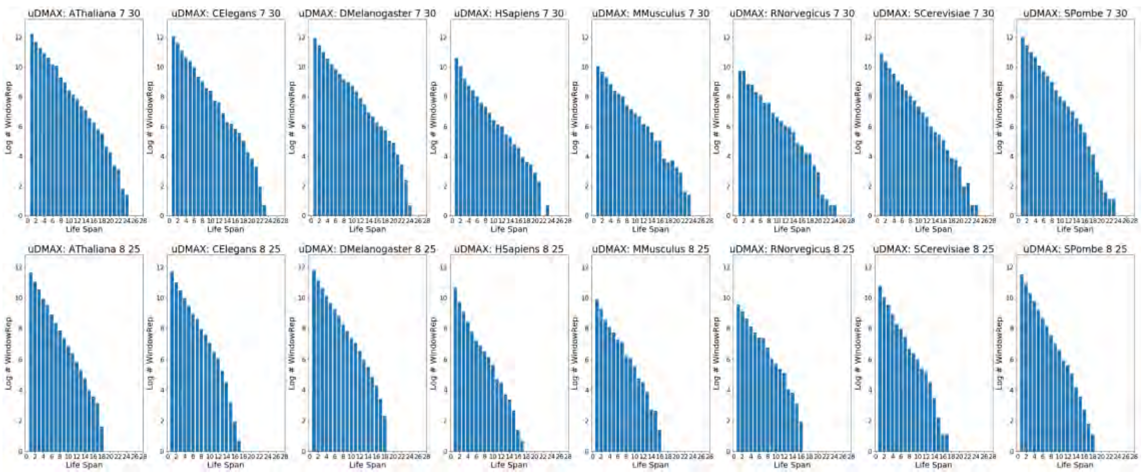


Figure 4d: WindowRep Lifespan from DMAX + unambiguous check (uDMAX)

According to Figure 4, we can notice that the lifespan of windowReps decays linearly on a log scale (exponentially on the normal scale) for all methods and the average lifespan is around 4 steps. Such results indicate that the unambiguous check is unable to improve the stability of windowRep sampling methods. Next, I utilized the frequency mode straight from BLANT to examine the uniqueness of windowReps obtained from the new methods.

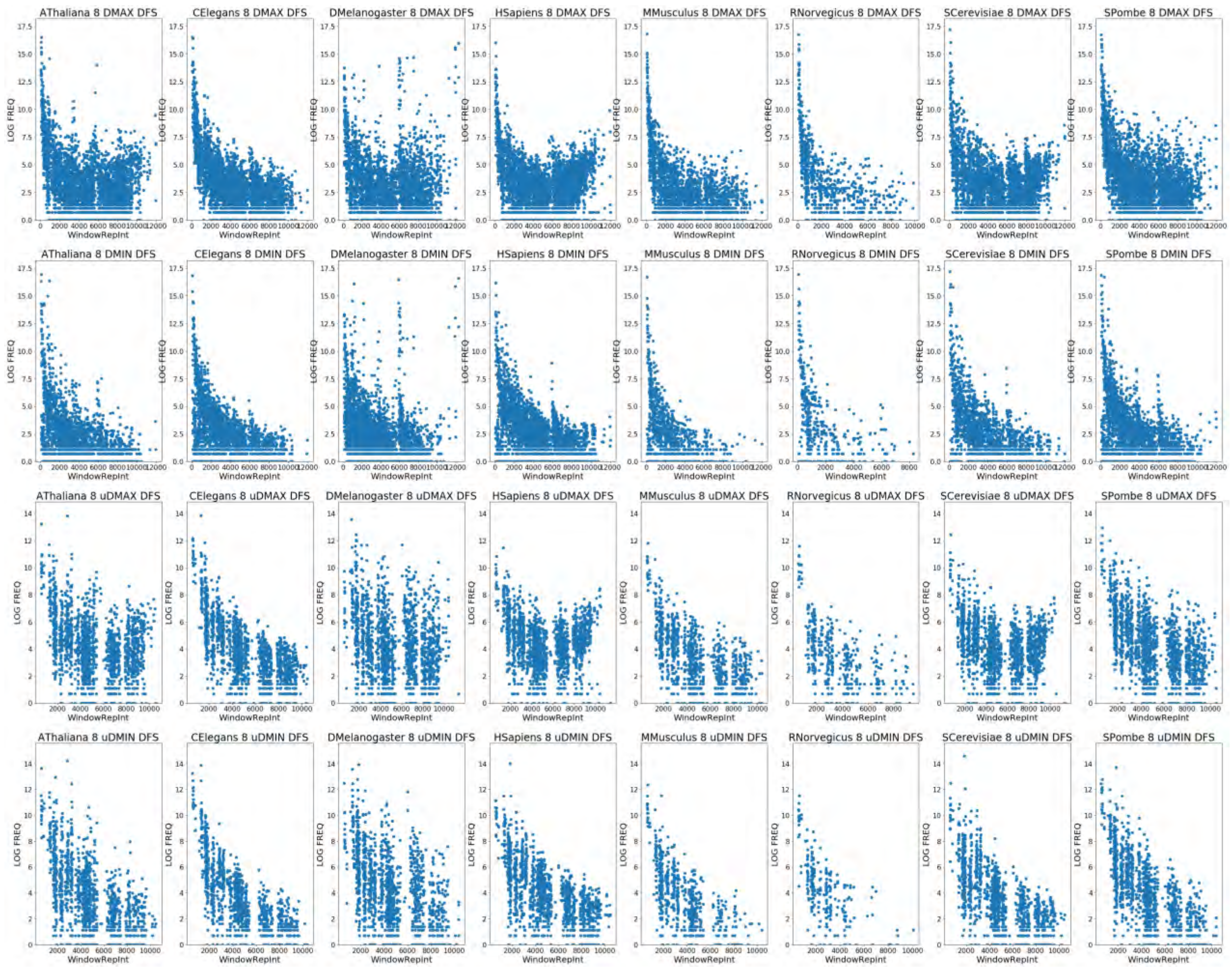
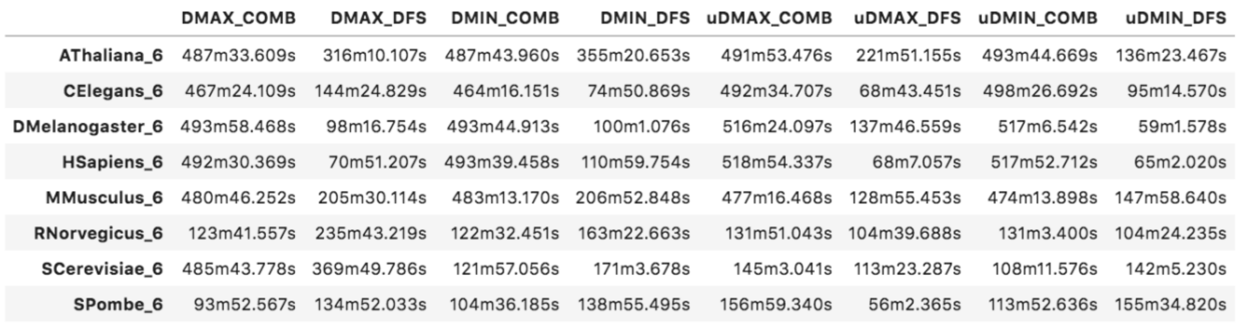


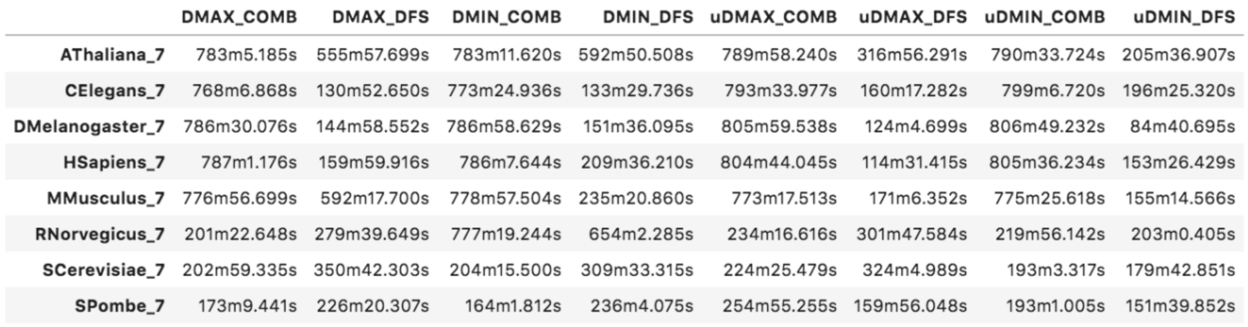
Figure 5: Frequencies of each found windowRep canonicals using different sampling methods.

From Figure 5, we can observe that there are far less found unique canonicals after applying the ambiguity check, which is expected since there is only a limited number of graphlets that have unambiguous property.

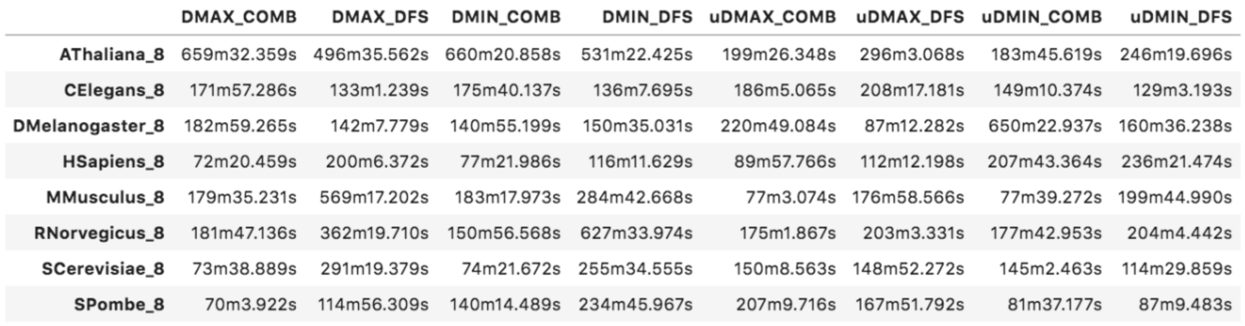
While obtaining the above frequency statistics from BLANT, I was able to compare the run time between greedy combination and DFS-like iteration methods. The tables below are clear to show that in most cases, DFS has better runtime performance than COMB.



	DMAX_COMB	DMAX_DFS	DMIN_COMB	DMIN_DFS	uDMAX_COMB	uDMAX_DFS	uDMIN_COMB	uDMIN_DFS
AThaliana_6	487m33.609s	316m10.107s	487m43.960s	355m20.653s	491m53.476s	221m51.155s	493m44.669s	136m23.467s
CElegans_6	467m24.109s	144m24.829s	464m16.151s	74m50.869s	492m34.707s	68m43.451s	498m26.692s	95m14.570s
DMelanogaster_6	493m58.468s	98m16.754s	493m44.913s	100m1.076s	516m24.097s	137m46.559s	517m6.542s	59m1.578s
HSapiens_6	492m30.369s	70m51.207s	493m39.458s	110m59.754s	518m54.337s	68m7.057s	517m52.712s	65m2.020s
MMusculus_6	480m46.252s	205m30.114s	483m13.170s	206m52.848s	477m16.468s	128m55.453s	474m13.898s	147m58.640s
RNorvegicus_6	123m41.557s	235m43.219s	122m32.451s	163m22.663s	131m51.043s	104m39.688s	131m3.400s	104m24.235s
SCerevisiae_6	485m43.778s	369m49.786s	121m57.056s	171m3.678s	145m3.041s	113m23.287s	108m11.576s	142m5.230s
SPombe_6	93m52.567s	134m52.033s	104m36.185s	138m55.495s	156m59.340s	56m2.365s	113m52.636s	155m34.820s



	DMAX_COMB	DMAX_DFS	DMIN_COMB	DMIN_DFS	uDMAX_COMB	uDMAX_DFS	uDMIN_COMB	uDMIN_DFS
AThaliana_7	783m5.185s	555m57.699s	783m11.620s	592m50.508s	789m58.240s	316m56.291s	790m33.724s	205m36.907s
CElegans_7	768m6.868s	130m52.650s	773m24.936s	133m29.736s	793m33.977s	160m17.282s	799m6.720s	196m25.320s
DMelanogaster_7	786m30.076s	144m58.552s	786m58.629s	151m36.095s	805m59.538s	124m4.699s	806m49.232s	84m40.695s
HSapiens_7	787m1.176s	159m59.916s	786m7.644s	209m36.210s	804m44.045s	114m31.415s	805m36.234s	153m26.429s
MMusculus_7	776m56.699s	592m17.700s	778m57.504s	235m20.860s	773m17.513s	171m6.352s	775m25.618s	155m14.566s
RNorvegicus_7	201m22.648s	279m39.649s	777m19.244s	654m2.285s	234m16.616s	301m47.584s	219m56.142s	203m0.405s
SCerevisiae_7	202m59.335s	350m42.303s	204m15.500s	309m33.315s	224m25.479s	324m4.989s	193m3.317s	179m42.851s
SPombe_7	173m9.441s	226m20.307s	164m1.812s	236m4.075s	254m55.255s	159m56.048s	193m1.005s	151m39.852s



	DMAX_COMB	DMAX_DFS	DMIN_COMB	DMIN_DFS	uDMAX_COMB	uDMAX_DFS	uDMIN_COMB	uDMIN_DFS
AThaliana_8	659m32.359s	496m35.562s	660m20.858s	531m22.425s	199m26.348s	296m3.068s	183m45.619s	246m19.696s
CElegans_8	171m57.286s	133m1.239s	175m40.137s	136m7.695s	186m5.065s	208m17.181s	149m10.374s	129m3.193s
DMelanogaster_8	182m59.265s	142m7.779s	140m55.199s	150m35.031s	220m49.084s	87m12.282s	650m22.937s	160m36.238s
HSapiens_8	72m20.459s	200m6.372s	77m21.986s	116m11.629s	89m57.766s	112m12.198s	207m43.364s	236m21.474s
MMusculus_8	179m35.231s	569m17.202s	183m17.973s	284m42.668s	77m3.074s	176m58.566s	77m39.272s	199m44.990s
RNorvegicus_8	181m47.136s	362m19.710s	150m56.568s	627m33.974s	175m1.867s	203m3.331s	177m42.953s	204m4.442s
SCerevisiae_8	73m38.889s	291m19.379s	74m21.672s	255m34.555s	150m8.563s	148m52.272s	145m2.463s	114m29.859s
SPombe_8	70m3.922s	114m56.309s	140m14.489s	234m45.967s	207m9.716s	167m51.792s	81m37.177s	87m9.483s

4. Future Work

1. Contact GHOST authors for their source code and experiment with the GLAlign R code. Meanwhile, look for other local alignment algorithms with Burwin.

2. Keep testing the seed performance after applying Top K constraints, and think about other constraints that can produce reliable seeds.