BioNetwork Research: WindowReps 2018 Fall Report

Henry Ye

During the past summer and fall quarter, I have been working on windowReps project, trying different methods of finding windowReps and comparing their performances.

Background

K-node windowReps are designed to reduce graph storage and help graph alignment by using a k-node graphlet to represent a larger window. A successful windowReps finding method must fulfill two requirements: **Uniqueness and Stability**. Uniqueness means that the windowReps in two totally different windows should be relatively distinct with each other (unique globally). Also, the ones found in one particular window should have few duplicates (unique locally) as well. On the other hand, Stability requires that windowReps in neighboring windows (when the window oozes around) should keep the same (Stable locally). In order to find such windowRep finding methods, 8 methods have been tested.

1. Minimizer & Maximizer

As the first idea of windowReps, minimizer was initially inspired by a paper about sequence alignment and storage reduction, which suggested using a subsequence with minimal value to represent a longer sequence. Since each graphlet can be represented by an integer from its lower triangular adjacency matrix, we sample out every k-node graphlets inside a larger window to find the one with the **smallest** canonical integer representation. However, it turns out that this method has a terrible performance on the Uniqueness requirement because most graphlets in a window have small canonical integer values. Whereas few graphlets have large canonical integer values, we changed our strategy of using Maximizer (the graphlet with the largest canonical integer) as our windowRep. In this case, Maximizer works very

well on the global Uniqueness. However, it changes too frequently to fulfill the local Stability requirement.

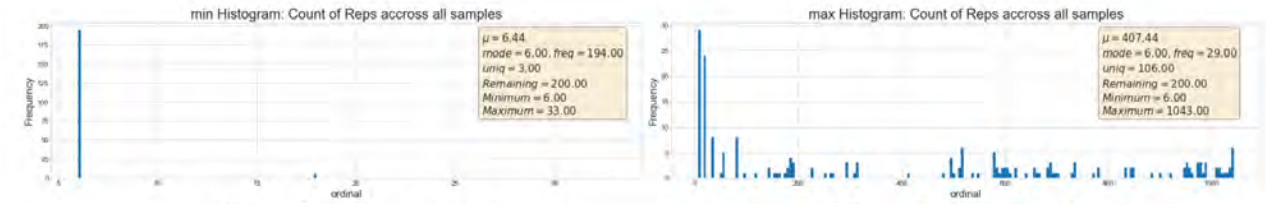

Figure 1: Minimizer & Maximizer Global Uniqueness Checking (Scerevisiae, W18, K7): From these two graphs, we can see that Minimizers only have 3 unique canonical values out of 500 samples. However, maximizers tend to spread much more uniformly than minimizers, even though it still some spikes in the low canonical value areas.

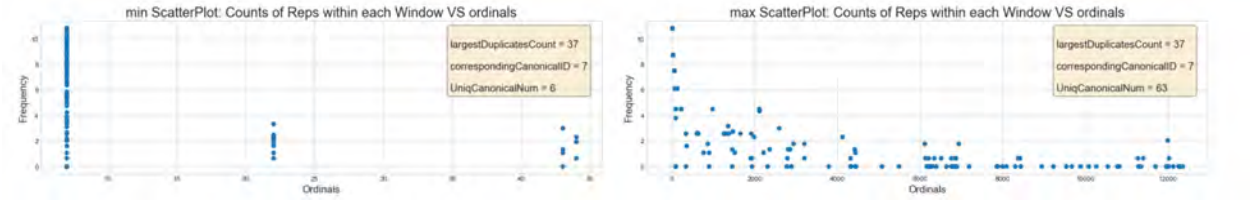

Figure 2: Minimizer & Maximizer Local Uniqueness Checking (Scerevisiae, W18, K7): Within the same window, Minimizers tend to have much more duplicates and much fewer unique canonicals integers than Maximizers.

```
min :    [100, 82, 70, 64, 60, 57]
max :    [100, 37, 19, 11, 5, 3]
```

Figure 3: Minimizer & Maximizer Stability Checking (Scerevisiae, W18, K7): From the statistics above, after 5 steps, Minimizer has the same windowRep 93% percent of the time. However, after 2 steps, only 41% Maximizer has the same windowRep.

2. Maximizer & Minimizer with Distance to the median number edges

Considering most graphlets have small value canonical integer values, we put an edge restriction on our previous Minimizer and Maximizer definitions that the chosen ones should have closest number of edges to the median, $\binom{k}{2}/2$, hoping the more balanced windowReps can give a better performance.

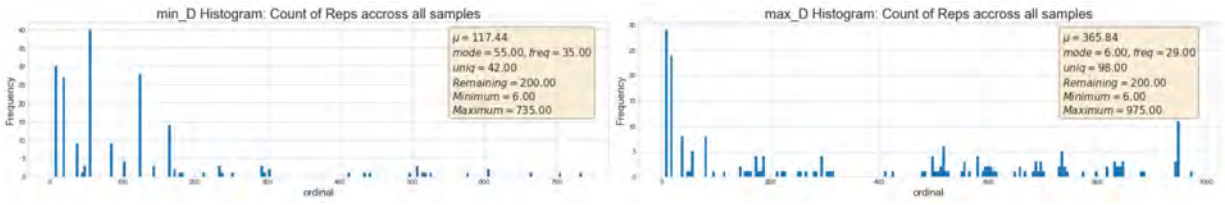

Figure 4: Minimizer & Maximizer with Distance Global Uniqueness Checking (Scerevisiae, W18, K7): Compared to Figure 1, It's noticeable that the Minimizer with Distance performs much better than Minimizer in terms of having much more unique windowReps and less spikes. To the histogram of Maximizer with Distance, it also looks more uniform than Figure 1.

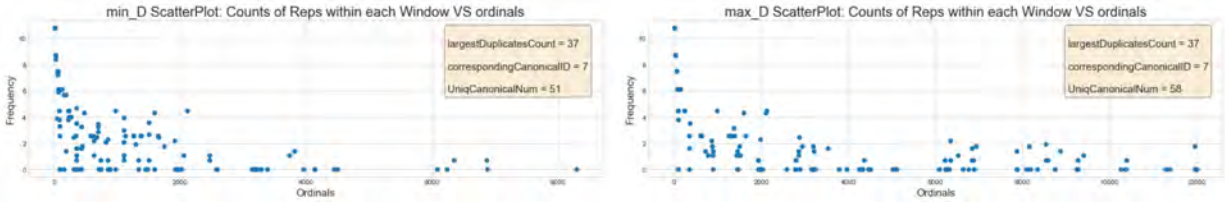

Figure 5: Minimizer & Maximizer with Distance Local Uniqueness Checking (Scerevisiae, W18, K7): Similar to the results obtained from Figure 4, Minimizer with Distance has much more unique and fewer duplicates for each windowReps than Minimizer. For the Maximizer with Distance, the duplicates checking results are not different a lot from previous Maximizer.

```
min_D :    [100, 74, 53, 43, 33, 27]
max_D :    [100, 64, 43, 32, 20, 17]
```

Figure 6: Minimizer & Maximizer with Distance Stability checking (Scerevisiae W18 K7): Compared to Figure 3, we can find out that the Minimizer with Distance is not as stable as Minimizer. However, Maximizer with Distance becomes more stable than Maximizer due to its balanced number of edges.

3. Least Frequent Min & Max

After using Min and Max definitions and obtain their performance results, we were thinking about trying something else to define the windowRep. Considering windowReps should somehow uniquely define the larger window, the k-node graphlet appears least frequently in the window should represent the window well. Since there might be a tie, we define two methods: one is to choose the minimal canonicals whereas the other one chooses the maximal canonicals.

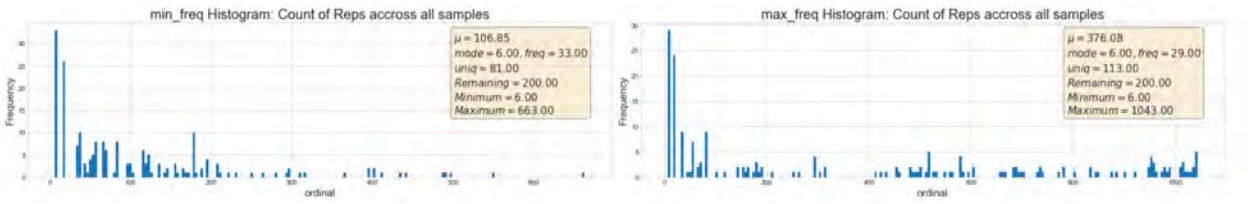

Figure 7: Least Frequent Min and Max Global Uniqueness Checking (Scerevisiae, W18 K7): As expected, Least Frequent method gives us the most number of unique windowReps compared to the previous methods.

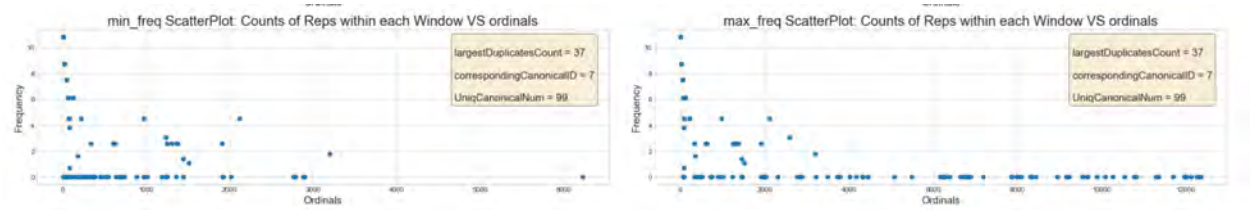

Figure 8: Least Frequent Min and Max Local Uniqueness Checking (Scerevisiae, W18 K7): Compared to Figure 2 and 5, these two methods also find the windowReps that have fewer duplicates than precious 4 methods.

```
min_freq :    [100, 48, 31, 23, 18, 13]
max_freq :    [100, 53, 36, 26, 19, 14]
```

Figure 9: Least Frequent Min and Max Stability Checking (Scerevisiae, W18 K7): As a tradeoff of being unique, the changing rate for Least Frequent methods becomes large. Both methods lose a half windowReps within only a step.

4. Median Frequent Min & Max

After a second thought about Uniqueness and Stability requirements and the results from previous methods, I found out that Uniqueness and Stability are actually tradeoffs with each other. When the windowReps are relatively unique and have fewer duplicates, it must have a very high variance. On the other hand, when the windowReps are stable and don't change a lot when the window moves, it must have a low variance. That's why Min methods have more duplicates and less unique windowReps are more stable compared to Max methods. Considering this tradeoff, I came up with using the graphlets with the median frequent appearance as the windowReps. If there is a tie, the Min and Max choosing criterion will be used.

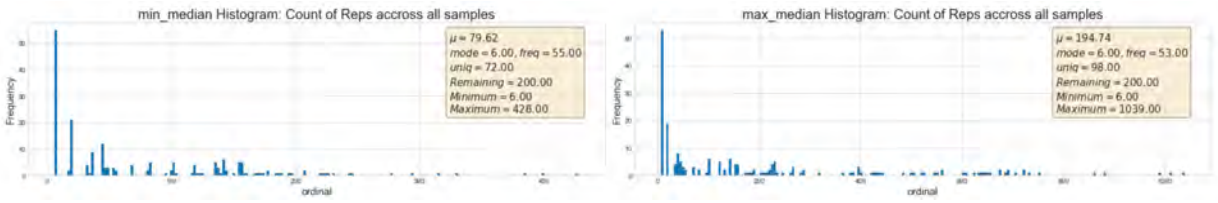

Figure 10: Median Min and Max Global Uniqueness Checking (Scerevisiae, W18 K7): Compared to Figure 4 and 7, Median choosing methods have less unique windowReps and spread less uniformly.

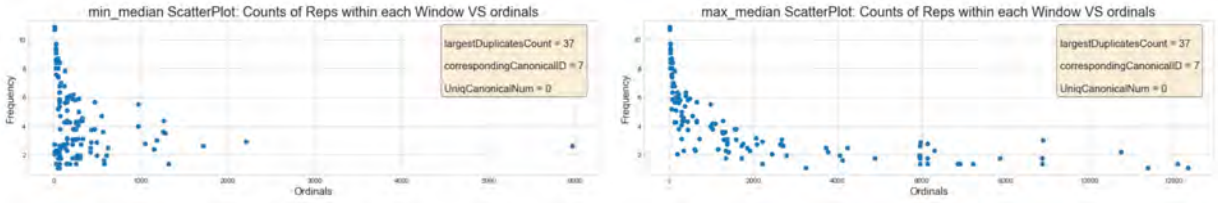

Figure 11: Median Min and Max Local Uniqueness Checking (Scerevisiae, W18 K7): Compared to Figure 5 and 8, Median choosing methods have more duplicates in each sample.

```
min_median :    [100, 40, 30, 25, 22, 21]
max_median :    [100, 42, 32, 24, 22, 21]
```

Figure 12: Median Min and Max Stability Checking (Scerevisiae, W18 K7): Based on the stats above, we can see that Median choosing methods performs worst among all previous methods, around 60% of the windowReps changed merely after one step.

At last, I also tried using the graphlet with median ordinal value instead of median frequency. However, similar to the previous Frequency Median methods, the result from Ordinal Median method is still not satisfying.

Conclusion:

Among all previous windowRep choosing methods, **Maximizer with Distance has the best performance**. It provides a large number of unique windowReps but still keep relatively stable when the window oozes around.

Future

I will use the previous methods on the intersection of all windowReps found in three adjacent window, hopefully it can give some extend of stability (within three steps or more).

Important code implementation changes

DFS -> Use combination of nodes in the window to find all windowReps

Mega Window -> MCMC Ooze method

Both changes made in algorithms help reduce the running time significantly.

Other output plots:

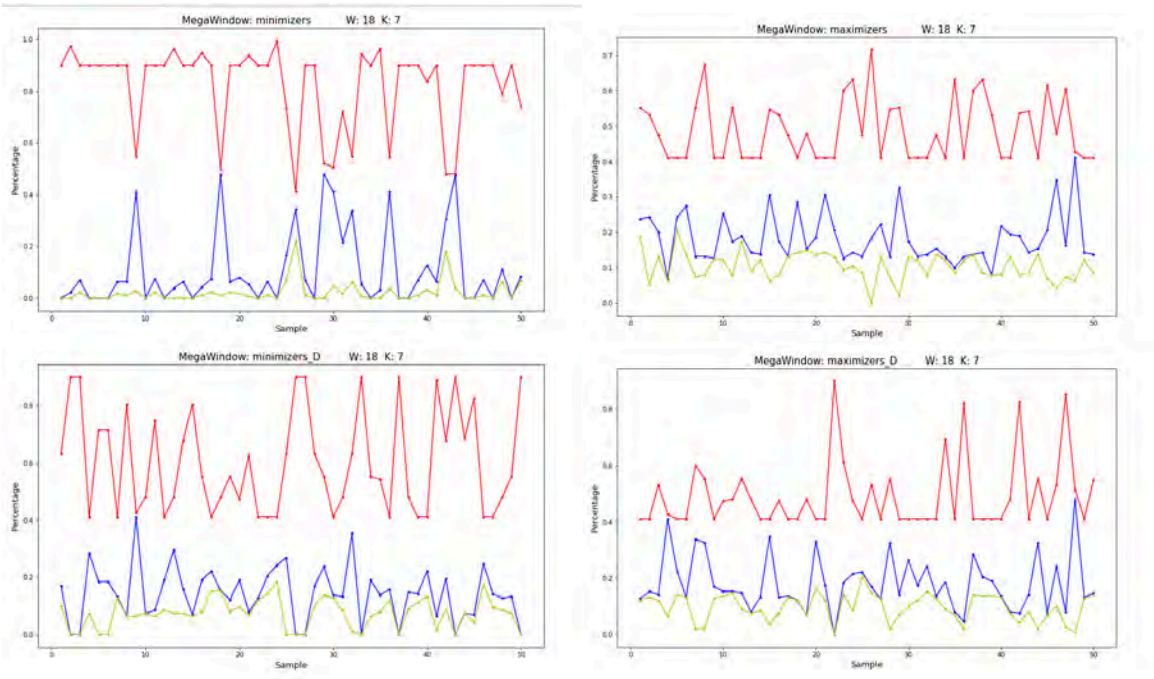

Figure 12: The percentage of most frequent windowReps compared to the second and third frequent windowReps using Mega window methods (2 steps away, across 50 samples)

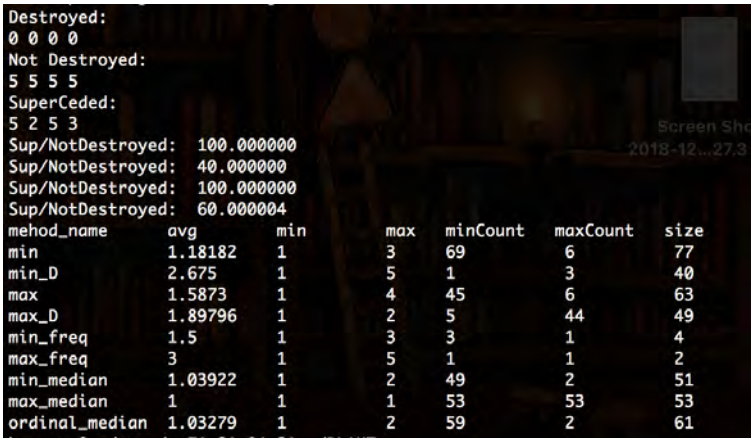

Figure 13: Superseded and Not Destroyed rate with a summary of the how windowRep changes within 5 steps among all choosing methods