John Adler
Professor Welsh
CS 342 Bioinformatics
April 15, 2020

How I Tried to Design an Assembler

I tried a couple different methods for designing an assembler while working on my program. In my first approach, I created a new dictionary data structure that would keep track of which edges correspond to each read. I had then planned on concatenating a single string sequence using the new data structure. Then I was going to use the string and the python random functions to randomly select start and end points to generate contigs from the string. I was going to generate these contigs until the sum of the lengths of all the contigs generated would be 50 percent. However, this initial approach fell through fairly quickly. Shortly after starting, I got lost on how to actually use the new data structure that I made to concatenate the string sequence, so I moved on from this approach.

In my next approach I thought I could somehow use the list of vertices I had to concatenate the string sequence from which I would assemble the contigs. I had an idea to use python random to pick a random vertex from the list and then find a corresponding edge from the edge dictionary so I could link the vertices together. However, this idea also fell through fairly quickly as I found that I couldn't figure out how to link the vertices with the edges without some other kind of data structure. I realized that I wanted to use a data structure that would allow me to loop through all my kmers so that I could string them together, so I changed my plan again.

The approach I settled on used a couple new lists and would create a sequence from the contigs so I can find 50 percent of the contigs. The first new list I created was ListContigs[] which is a list where I store each contig at an index. The second new list I created is called VertexEdge[] and it contains each unique kmer at an index. I add each kmer to the list while I am looping through the input file to create the De Bruijn graph. To construct the contigs, I started by using a while loop so I could loop through until my EdgeDictionary was empty. My idea is that whenever a kmer is removed from VertexEdge I then subtract one from the dictionary value of that kmer, so: EdgeDictionary[kmer] = EdgeDictionary[kmer] - 1. Then, if the value in the EdgeDictionary is equal to 0, then you remove (pop) the kmer from the dictionary. With this, I could start looping through to create my contigs.

At the start of the while loop, I generate a random number to serve as the random index in VertexEdge[] that will serve as the starting kmer for the contigs. The kmer that is selected as the start of the contig is removed from VertexEdge[] and EdgeDictionary. Then, I use a for loop with the max range of length of VertexEdge[] to construct each contig. I made a sequence of if/elif statements to choose the next kmer that would construct the contig. The if/elif statements would choose the next letter to be added to the contig sequence by choosing A, T, C, or G in that order. If the contig with the newly added letter is not in VertexEdge[], then the new letter is incremented to the next letter and the for loop is continued. This will repeat until the new ending of the contig is found in VertexEdge. Once the new ending for the contig is found, I get the index of that kmer from VertexEdge so I can then remove it from both VertexEdge[] and EdgeDictionary (if needed). After this, I concatenate the new letter to the end of the contig and then the for loop repeats. After the for loop finishes, the contig is appended to ListContigs[]. When the EdgeDictionary runs out of edges, the loop ends.

After I have the complete ListContigs[], I move on to calculate the N50 and L50 values. First, I loop over each contig to add each one of them to a single sequence. Then, once I have that sequence, I can divide the length of the sequence by two so that I can find 50 percent of all the contigs. I then create a new list called ListLargeContigs[] that I will use to basically sort my list of contigs by largest length contigs first. To sort ListContigs[], I use a for loop to loop through each contig and compare them. If the current contig is larger than the current largest contig, the current contig replaces it. Then, the length of the largest contig is subtracted from the length of 50 percent of the contigs since the loop will continue until the length of 50 percent of the contigs is 0. Finally, the largest contig is removed from the original ListContigs[] and added to the ListLargeContigs[] which is the list used to find the N50 and L50 scores. To find the N50 score, I loop through the length of ListLargeContigs[] to find the smallest contig in the list. Then the L50 is simply the length of the ListLargeContigs[]. And that's about it for how I tried to implement the assembler.

I do not think that my assembler entirely works and I think there is still much I could do to improve it. I made a couple data tables below of two different cases. The first is when my program takes in the reads and directly translates them into contigs. The second data table is when I use my assembler.

Contigs as reads:

Size of kmer	reads.txt	sample_c12_r_50_e0.00.txt	sample_c12_r_50_e0.01.txt
5	N50: 22| L50: 1197	N50: 50| L50: 1200	N50: 50| L50: 1200
10	N50: 22| L50: 1197	N50: 50| L50: 1200	N50: 50| L50: 1200
13	N50: 22| L50: 1197	N50: 50| L50: 1200	N50: 50| L50: 1200

My assembler:

Size of kmer	reads.txt	sample_c12_r_50_e0.00.txt	sample_c12_r_50_e0.01.txt
5	N50: 5| L50: 378	N50: 5| L50: 381	N50: 5| L50: 380
10	N50: 10| L50: 8256	N50: 10| L50: 4242	N50: 10| L50: 8385
13	N50: 13| L50: 9190	N50: 13| L50: 4263	N50: 13| L50: 9252

In the graph where my contigs are reads, the size of the kmer clearly doesn't impact either the N50 or the L50 values. Each N50 value comes from the length of the reads in each line of the input file. Since in reads.txt each line has a length of 22 and in the two sample files each line has a length of 50, those end up becoming the N50 values. The L50 value is therefore the number of lines or reads in the input file.

In my assembler, each of the N50 values are equal to the size of the kmer that was used to produce the De Bruijn graph. Then, the L50 values are the number of contigs it took my assembler to reach fifty percent of all the contigs. While I definitely think my assembler is an improvement over the most naive approach, I think there is plenty wrong with it.

With the way I designed my assembler, the N50 value shouldn't always be the size of the kmer. What I tried to do was sort the contigs I created by their size, then choose the largest size contigs first up until the fifty percent of all the contigs was reached. When I was printing my contigs while testing, I realized that my method of constructing the contigs had a number of flaws. The first was that contigs could only be constructed from the reads in the order that I was testing the presence of the next letter in each of my data structures. For example, each of my contigs could only be constructed from the initial kmer + A, T, C, G in that order. So my longest contigs consisted of startkmer+ATCGA. If G is reached, the next letter will loop back to A. I don't know exactly why my code can only assemble the contigs in this way. In addition to this, the longest contig I've been able to create has only had a length of five more than the initial kmer that was used to construct it. I am also not sure why my contigs are being constrained in this way. Along with this, many of my kmers that start each contig do not have a single letter added to them.

There is also likely some error in the way I am calculating my N50 value. Currently, I am looping through the length of fifty percent of the sequences and in each loop, subtracting the length of the contig selected from the length of fifty percent of the sequences. When that fifty percent of the sequences becomes 0, the loop should stop. However, I came across several different scope errors while doing this and the way my loop works at the moment is janky at best.

Overall throughout this assignment, I learned that constructing concise contigs is not only difficult, but also time consuming. For my larger kmer quantities, my program would have to run for at least a minute or two to get an output. This could be due to my less than stellar design, but I was also having difficulty avoiding the nested loops I wrote for this program. The most challenging part for me was figuring out how to use the data structures I had and the new data structures I was creating to actually construct the contigs. Once I had a list of the contigs, it wasn't nearly as hard for me to think through how to get the N50 and L50 values.