# Project Option 2: Simulated Chromosome Assembly

Nick Karlovich, Cameron Wonchoba, Josh Schlumpberger, Alex Patel

*Abstract*— **Researchers have access to a ton of shotgun data from the human genome. These shotguns are usually around 150 base pairs long. The sequence of the human genome can be found through merging these shotgun reads. One approach to reassembling these shotguns into a human genome is to use De-Bruijn graphs where an Eulerian path can be used for reassembly. De-Bruijn graphs are derived from shotgun data with edges defined as kmers. The Eulerian path through the graph represents the assembled sequence. Originally, a proposed solution for assembly was to generate a De-Bruijn graph where a Hamiltonian path would be used for reassembly. However, determining the Hamiltonian path of a graph is non-polynomial. By representing the assembly problem as a graph whose nodes are k-1mers and edges are kmers, we reduce the problem to finding an Eulerian Path - a path in which all edges of the graph are visited. This can be determined in polynomial time. However, in practice the De-Bruijn graph produced from the shotgun data may not always be Eulerian, thus causing issues in the assembly. In this paper, we show that there are issues with the basic implementation of building a De-Bruijn graph and finding an Eulerian path to reassemble a sequence by attempting to reassemble a human chromosome. We found that repeated segments of the sequence often create unbalanced nodes within our graphs, preventing us from correctly reassembling the sequence. We also found that coverage and kmer size affects the reliability of the De-Bruijn graph. Our results demonstrate the rigor of the assembly problem, and the potential issues one might run into when attempting to build a naïve sequence assembler. We anticipate our paper to be a starting point for learning about sequence assembly and provide some potential problems with a naïve sequence assembler.**

## I. PREVIOUS FINDINGS

One method for reconstructing a sequence is using a Hamiltonian circuit. Reconstruction with a Hamiltonian circuit was first introduced in 1988 by the name of Sequencing by Hybridization. At the time, nobody believed it would work, and today they are still right. In our project, we show and reaffirm the inefficiency of Sequencing by Hybridization (Hamiltonian).

Modern gene reconstruction algorithms such as Abyss [1] and Velvet [2] use Eulerian circuit based algorithms rather than Hamiltonian based approaches. In Velvet's paper, the researchers make some optimizations to the basic Eulerian circuit to make it more viable. One optimization includes simplification, whose job is to simplify "dead-end" paths into one large k-mer to save memory. Error correction is another optimization that Velvet makes. Errors are one of the largest problems that reconstruction algorithms face. These optimizations help Velvet improve their results when reconstructing genomes of real-world data with Eulerian circuits. In our project, we will not be implementing any of the optimizations proposed and used in Velvet or other Eulerian based genome re-assemblers. In the future, adding these optimizations may be a good next step for expanding this project.

## II. RESULTS

We developed software to simulate shotgun reads, convert shotgun reads to kmers, build De-Bruijn graphs, and traverse Eulerian circuits. A variety of tests were run to determine the strengths and weaknesses of our implementation. Two important factors a researcher must decide when sequencing is what shotgun coverage and kmer size to select. To find the optimal shotgun coverage, we varied the coverage and found an average coverage of 8 was sufficient to have satisfactory assembly (Figure 3). With nearly every alignment algorithm using kmers, a logical question to ask is what size kmer works best. In our testing, we found that a kmer size of at least 19 produced the best result (Figure 5).

The two most significant flaws in our implementation happened when we introduced simulated error in our shotgun reads (Figure 4) and assembled large segments (Figure 6). Our implementation doesn't handle any percentage of errors particularly well. Even when using the best rotated score, the assembly accuracy still dropped under a score of 50 with any error. We believe the best potential solution to the error problem is to use an error correction algorithm. Eulerian circuits are not built to handle errors in data, so the data must be cleaned before using it. In our further research, we found MisEd [4] to be a promising start for correcting errors in shotgun reads. In Figure 6, we tested how our algorithm performed with increasingly large segments of the chromosome. The accuracy of our assembly drops considerably as larger segments are used. We attempted to fix this by using the best rotated score. A rotated string is an ordered rearrangement of the letters. An example of all the rotated strings for the string "abcd" can be seen below:

```
abcd — 0 Rotations
dabc — 1 Rotation
cdab — 2 Rotations
bcda — 3 Rotations
abcd — 4 Rotations
```

The same process is done for the assembled sequence. For each rotation, the score is computed, and then the best score is used as the best rotated score. While finding the best rotated score increased our assembly accuracy to near perfect, knowing what the end sequence will look like usually isn't realistic. However, if a researcher knows how the sequence starts, accuracy will be much better. Additionally, the time taken for large sequence lengths goes up exponentially (Figure 7). Throughout our code, we used best practices for threading to reduce this result, but the heavy reliance on memory during sequence comparison ultimately slows the algorithm the most. Instead of using Needleman-Wunsch or Levenshtein distance as we did, a simple base by base string comparison would've been much faster and yielded similar results.

## III. CONCLUSION

The purpose of our project was to reassemble simulated shotgun reads of the human chromosome using De-Bruijn graphs and Eulerian circuits with the goal of achieving near-perfect assembly accuracy and high efficiency. We had success assembling chromosome segments under 1000 bases with no shotgun error. However, larger segments or segments with any percentage of error caused the assembly to lose significant accuracy. There is room for improvement in our selection of starting nodes in the Eulerian circuit algorithm. Finally, there is a significant need for the error correction of shotgun reads.

## IV. METHODS

To test the performance and efficacy of the Hamiltonian and Eulerian circuits, an already assembled Human DNA sequence is needed. Chromosome 22 was downloaded from `http://hgdownload.cse.ucsc.edu/goldenPath/hg38/chromosomes/` (this chromosome is selected because it is the smallest in size). For purposes of testing, a subset of this chromosome is used - we selected different sized subsets depending on the test.

Shotgun reads are simulated from our obtained sequence. The length of our shotguns was computed using the following formula:

$$min(\frac{SequenceLength}{2}, 150) \tag{1}$$

The number of shotguns that are generated depends on the coverage we want to see. This can be generated by the following formula:

$$\frac{Coverage * SequenceLength}{ShotgunLength} \tag{2}$$

These shotguns are then split up into all of the possible unique kmers. These kmers enable us to make two types of De-Bruijn graphs.

The first De-Bruijn graph is used to compute a Hamiltonian cycle. This graph uses kmers as its nodes and weights (the number of connections between each vertex) as its edges (Figure 1). The Hamilitonian cycle is computed by using depth-first search.

```
def hamiltonian_path(kmer_string, visited):
    outgoing = kmer_string.outgoing
    visited.add(kmer_string)
    for kmer in outgoing:
        if kmer not in visited:
            sub_problem = hamiltonian_path(kmer, visited)
            output.add(sub_problem.score + kmer.score)
        else if kmer in visited:
            output.add(0)
    return max(output)
```

We tested the efficiency of this with varying kmer size.

The second De-Bruijn graph is used to compute an Eulerian cycle. This graph uses k-1mers as its nodes and kmers as its edges (Figure 2). The Eulerian path is computed using Fleury's algorithm [3].

```
def eulerian_path(start_vertex, graph):
    vertex_order = []
    stack = [start_vertex]
    while len(stack):
        vertex = stack.pop()
        for v in graph[vertex]:
            if not v.visited:
                v.set_visited(True)
                vertex_order += get_edge(vertex, v)
                stack += [v]
                break
    return vertex_order
```

Once the ordering of vertexes are determined, the sequence can be reassembled by concatenating the first letter in each vertex to a string (the last vertex gets its whole vertex sequence concatenated). Below is an example:

```
AAB
 ABB
  BBC
   BCC
    CCD
     CDD
      DDC
       DCD
        CDE
         DEE

Assemebled Sequence:
AABBCCDDCDEE
```

We tested the efficiency of this algorithm as well as the accuracy of this algorithm with varying coverage (within shotgun reads), varying kmer length, and varying sequence length. Additionally, we tested what happens when error is introduced into the shotgun reads.

To test the accuracy of the sequence assembly, the Levenshtein distance between the assembled sequence and the original sequence can be computed. Higher scores indicate smaller Levenshtien distances, and lower scores indicate higher Levenshtien distances.

## V. FIGURES

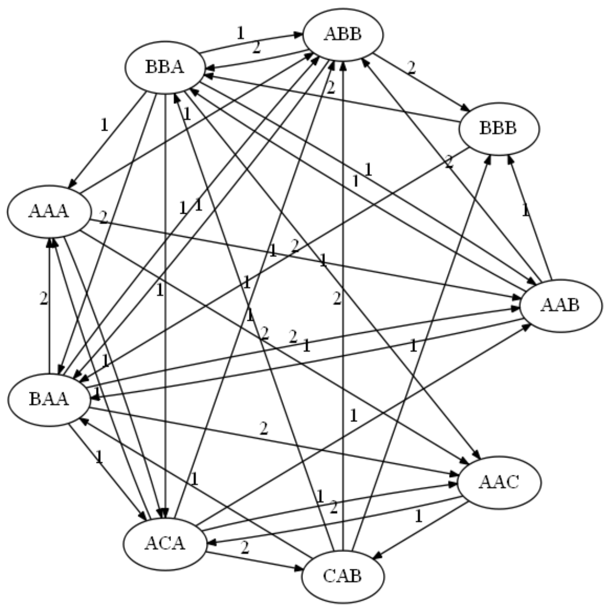

Fig. 1. The Hamiltonian graph with k=3

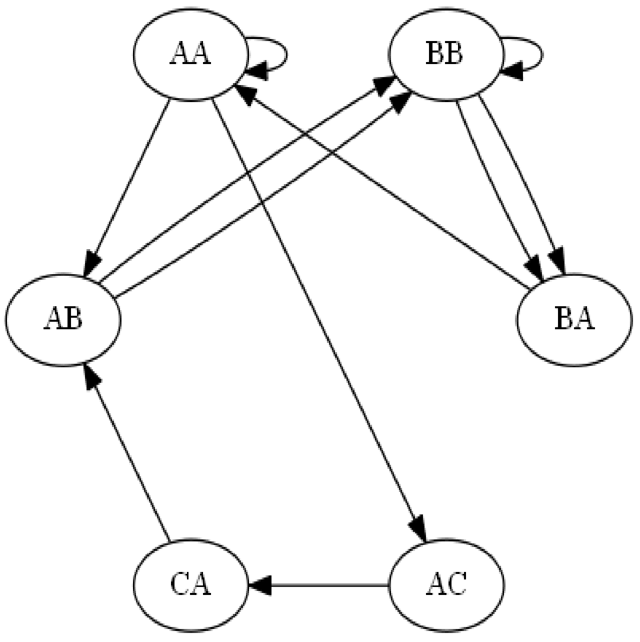

Fig. 2. The Eulerian graph with k=3

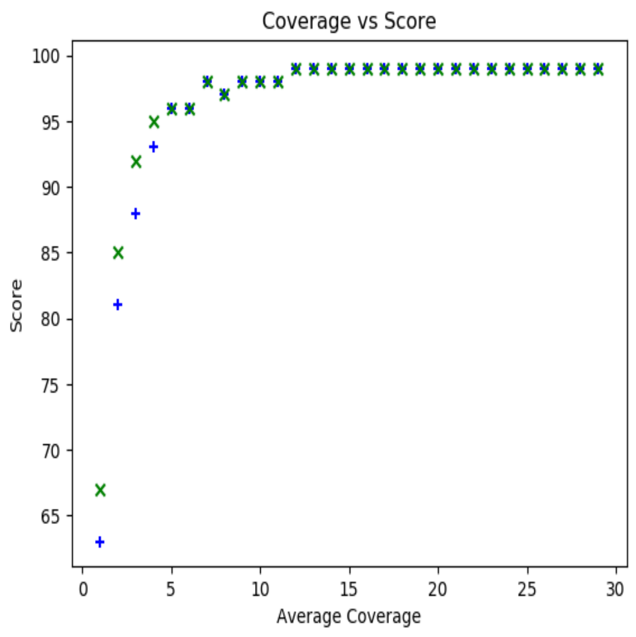

Fig. 3. Coverage vs Score. 31-mers on 1000 base sequence.

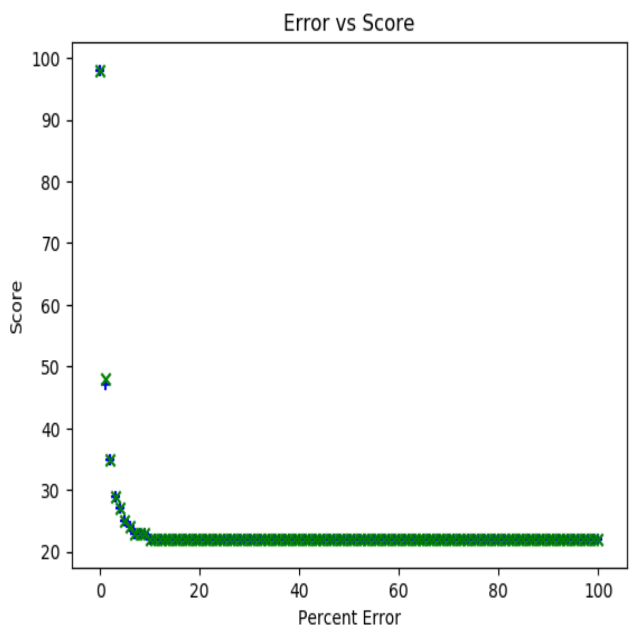

Fig. 4. Error vs Score. 31-mers with 10x coverage.

## ACKNOWLEDGMENT

- Alex - Building Graph class and De-Bruijn Graph generation, Testing
- Cameron - Eulerian Circuit, Reconstruction of Eulerian, Simulating Error, Code Optimizations, Testing
- Nick - Hamiltonian Circuit, Visualization, Organizing Slides
- Josh - Convert Sequencing to Shotgun, Simulate Errors in Shotgun, Breakdown in K-Mers, Performance Optimizations, Testing

## REFERENCES

[1] Jackman, S. D., Vandervalk, B. P., Mohamadi, H., Chu, J., Yeo, S., Hammond, S. A., … Birol, I. (2017). ABySS 2.0: resource-efficient assembly of large genomes using a Bloom filter. Genome research, 27(5), 768–777. doi:10.1101/gr.214346.116

[2] Miller, J. R., Koren, S., Sutton, G. (2010). Assembly algorithms for next-generation sequencing data. Genomics, 95(6), 315–327. doi:10.1016/j.ygeno.2010.03.001

[3] Sutaria, V. Hamiltonian and Eulerian Cycles. International Journal of Trend in Research and Development, 3.

[4] Tammi, M., Arner, E., Kindlund, E., and Andersson, B. (2003). Correcting errors in shotgun sequences. Nucleic Acids Research, 4663-72.

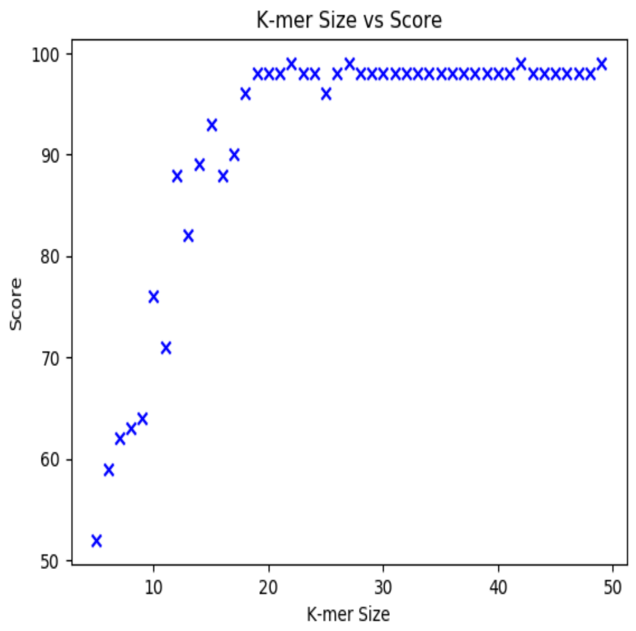

Fig. 5. Kmer Size vs Score. 1000 base sequence with 10x coverage.

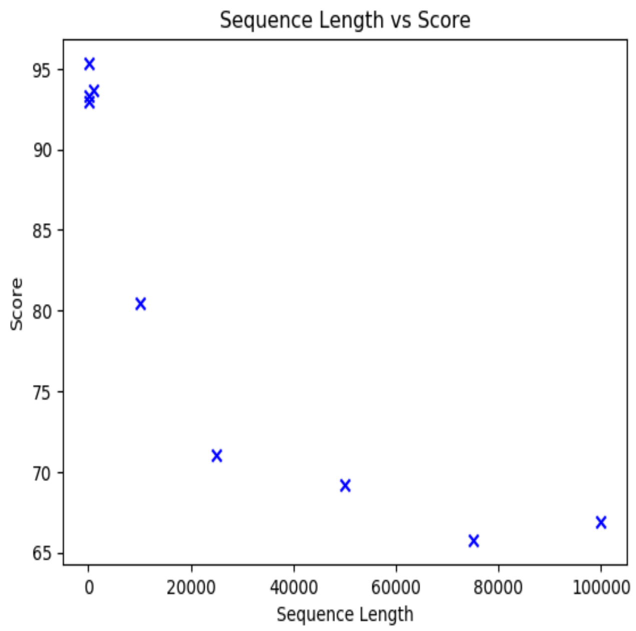

Fig. 6. Sequence Length vs Score. 31-mers with 10x coverage.

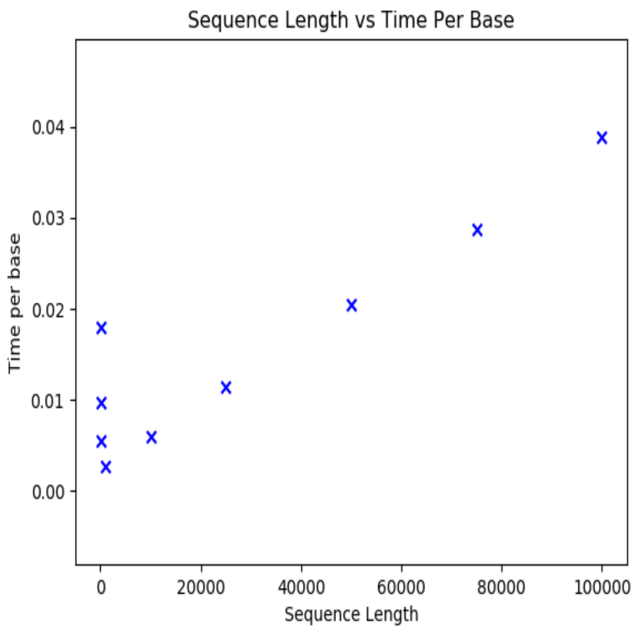

Fig. 7. Sequence Length vs Time. 31-mers with 10x coverage.