Problem Statement:

The Carsonella ruddii bacterium, with its reduced genome due to a symbiotic relationship with insects, offers an opportunity to explore genome assembly techniques. This project aims to simulate error-free read-pairs from the Carsonella ruddii genome and reconstruct it using paired de Bruijn graphs. The goal is to compare this reconstruction with that from a classic de Bruijn graph, where only the reads are known without paired read distance. The objective is to determine the minimum value of d, for various k-mer lengths (k), necessary to successfully reconstruct the genome from its (k, d)-mer composition.

Methodology:

The project aims to reconstruct the Carsonella ruddii genome using De Bruijn graphs and paired De Bruijn graphs. Below is an overview of the key steps involved.

1. Genome Loading

   - The genome is loaded from `genome.txt` and cleaned by removing spaces and newlines.

2. De Bruijn Graph Reconstruction

   - K-mer Composition: The `composition` function generates k-mers of length `k` from the genome.

   - Genome Reconstruction: The `string_reconstruction` function constructs the genome using the k-mers and an Eulerian path through the De Bruijn graph.

The reconstructed genome from the De Bruijn graph is printed as `gen1`.

3. Paired De Bruijn Graph Reconstruction

   - (k, d)-mer Composition: The `k_d_mer_composition` function creates read-pairs from the genome.

   - Paired Graph Creation: The function `de_bruijn_from_pairs` builds the paired De Bruijn graph from the read-pairs.

- Reconstruction: The `reconstruct_from_pairs` function reconstructs the genome by traversing the paired graph using an Eulerian path.

The reconstructed genome from the paired graph is printed as `final_genome`.

4. Hamming Distance Calculation

- The `hamming_distance` function compares the reconstructed genomes (`gen1` and `final_genome`) with the actual genome to evaluate the accuracy of each method.

5. Optimizing (k, d) for Best Reconstruction

- The function `find_best_k_d_reconstruction` iterates over different values of `k` and `d`, generating (k, d)-mer compositions and finding the best (k, d) combination that minimizes the Hamming distance.

The best values of `k`, `d`, and the corresponding score are printed.

6. Plotting Results

- The best `d` values for each `k` are plotted using `matplotlib`, showing how the optimal `d` changes with different `k` values.

Key Functions:

- `composition(k, genome)`: Generates k-mers.

- `string_reconstruction(kmers)`: Reconstructs the genome using a De Bruijn graph.

- `k_d_mer_composition(genome, k, d)`: Creates (k, d)-mer read-pairs.

- `de_bruijn_from_pairs(read_pairs)`: Builds a paired De Bruijn graph.

- `reconstruct_from_pairs(graph, k, d)`: Reconstructs the genome from the paired graph.

- `find_best_k_d_reconstruction(genome, max_k, max_d)`: Finds the best (k, d) combination.

This methodology simulates genome assembly using both De Bruijn graph methods and optimizes parameters to achieve the most accurate reconstruction.

Result:

The optimization of the `(k, d)` parameters to minimize the Hamming distance between the reconstructed and actual Carsonella ruddii genome yielded the following key findings:

- The best combination of k and d was k = 29 and d = 30, with a minimum Hamming distance of 94,335, indicating the most accurate reconstruction.

- For smaller values of `k` (k = 3 to k = 10), optimal `d` values varied significantly, with **k = 3** requiring d = 2 and k = 4 needing d = 4.

- As `k` increased, the optimal `d` values fluctuated, with k = 14 needing d = 30, and k = 30 needing d = 28. Larger values of `k` typically performed better with larger gaps.

- The results suggest that smaller `k` values (3 to 10) often need larger `d` values, while larger `k` values (above 10) converge toward smaller `d` values.

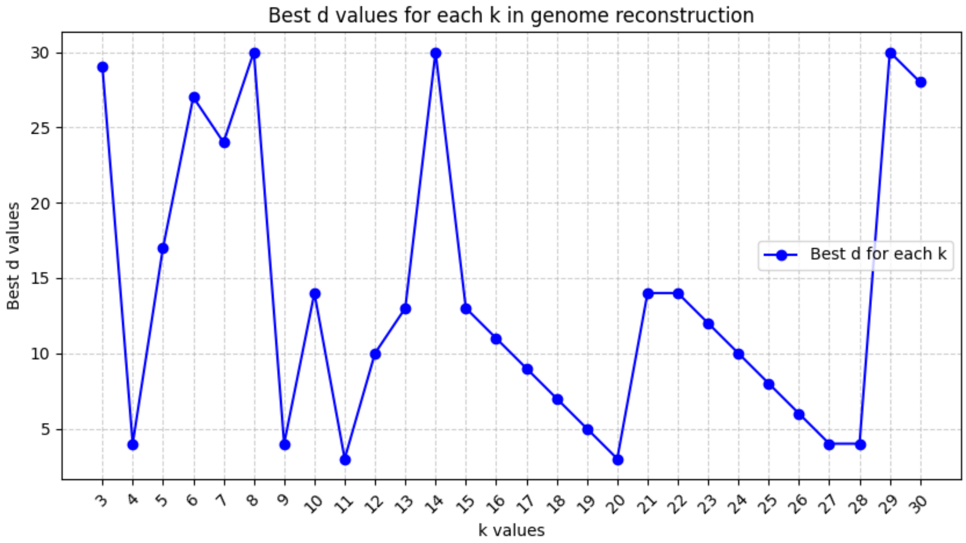
These findings emphasize the impact of `k` and `d` on genome assembly quality and the need for fine-tuning these parameters to optimize reconstruction accuracy.

Additionally, when comparing the Hamming distances of the two reconstruction methods:

- The **De Bruijn Graph Reconstruction** initially had a higher Hamming distance of **108,920**.
- The **Paired De Bruijn Graph Reconstruction** had a slightly higher Hamming distance of **109,090**.

While both methods showed improvement as the optimal parameters were fine-tuned, the **paired De Bruijn graph's**score steadily decreased as parameters were optimized, indicating that it performed better over time compared to the initial reconstruction. In contrast, the simple De Bruijn graph method started with a higher Hamming distance, which gradually decreased but remained higher than that of the paired method.

These findings emphasize the impact of `k` and `d` on genome assembly quality and the need for fine-tuning these parameters to optimize reconstruction accuracy.



Optimizations:

Here are several methods for genome assembly to achieve accurate results, including the use of machine learning (ML) and artificial intelligence (AI):

1. De Bruijn Graph-Based Assembly: This method builds a graph of k-mers and reconstructs the genome using Eulerian paths. Experimenting with different values of `k` and `d` helps optimize assembly accuracy by minimizing Hamming distance.

2. Greedy Algorithm: Iteratively joins the most overlapping reads to form contigs. It is fast but prone to errors in repetitive regions.

3. Overlap-Layout-Consensus (OLC): Builds a graph from all overlapping reads and creates a consensus sequence. It's highly accurate but computationally expensive.

4.String Graph Assembly: Uses a variation of the De Bruijn graph for better handling of repetitive regions in larger genomes.

5. Paired-End and Mate-Pair Assembly: Utilizes paired-end or mate-pair reads to improve assembly accuracy by resolving ambiguities in repetitive regions.

6. Hybrid Assembly: Combines short and long reads to leverage the accuracy of short reads and the long-range information from long reads, improving overall assembly.

7. Iterative Refinement: Involves refining the initial assembly through multiple rounds of error correction, improving quality over time.

8. Optimization of k and d: Testing different `k` and `d` values systematically helps find the best parameters for accurate genome reconstruction.

9. Machine Learning (ML) Approaches: ML models, like random forests, deep learning, and support vector machines, can predict and correct errors in genome assembly. These methods analyze patterns in sequencing data to optimize assembly, improve read correction, and detect misassemblies.

10. Artificial Intelligence (AI) for Error Correction: AI algorithms, such as reinforcement learning, can be used to refine genome assembly by identifying the best strategies for gap filling, correcting sequencing errors, and handling complex genomic structures, improving overall accuracy and assembly quality.

By combining traditional methods with ML and AI-based approaches, genome assembly can be significantly enhanced, leading to more accurate and efficient genome reconstruction.