Genome Assembler Project Report

Fall 2023-2024

Abstract:

This project aimed to develop a genome assembler using de Bruijn graphs to reconstruct the genome from a list of reads. The primary objective was to implement an efficient algorithm in C++ capable of accurately assembling the genome from the given reads, which involved the creation and the traversal of de Bruijn graphs.

The implementation involved careful consideration of algorithmic efficiency, memory management, and scalability to accommodate varying dataset sizes.

Key findings of the project include successful genome assembly from simulated and real-world reads, demonstrating the effectiveness of the de Bruijn graph-based approach. Performance metrics, including assembly accuracy and computational efficiency, were assessed to evaluate the assembler's reliability and scalability.

Introduction:

The de Bruijn graph technique stands out as an efficient approach for genome assembly in the constantly evolving area of bioinformatics. This innovative method revolutionizes the assembly process by representing DNA sequences as overlapping k-mers, where the relationships among these k-mers are elucidated through a comprehensive de Bruijn graph. The de Bruijn graph, a complex network of interconnected nodes and edges, is a useful framework for reconstructing entire genomes and captures the intricacy of genomic data. Assembling a genome is like gluing together a massive puzzle, with the puzzle pieces being short segments of DNA generated by sequencing machines. These pieces, which are also called reads, need to be carefully assembled into contigs that represent various parts of the genome. When this assembly puzzle is solved, a thorough understanding of an organism's genetic composition is possible, opening new avenues for research into its biology, evolution, and its uses in anything from agriculture to medicine.

The traditional challenge in genome assembly lies in the complexity of assembling reads that may overlap in intricate patterns due to repetitions and variations in the genome. The de Bruijn graph technique efficiently addresses the problem by breaking down the genome into smaller, overlapping k-mers, simplifying assembly and allowing for efficient representation and traversal of the genomic environment.

This project aims to develop a genome assembler that makes use of computational algorithms to navigate the complex genetic code using the de Bruijn graph approach. This method offers improved accuracy and efficiency in reconstructing genomes, even in the presence of repetitive elements and sequencing errors, by selectively using k-mers as the building blocks of the assembly process. To put the implementation into practice, the de Bruijn graph is constructed from sequenced reads, and then paths representing possible contigs that together make up the assembled genome.

The subsequent sections will detail the Methodology of our project, discuss the implementation, present the results obtained, and engage in a thorough discussion of the implications and limitations of our approach. As we explore the complex world of de Bruijn graph-based assembly, the goal of this project is not only to add to the body of bioinformatics knowledge, but also to provide a useful tool for researchers and scientists seeking to unravel the mysteries encoded in the genetic material of various organisms.

Methodology:

The development of the genome assembler involved a systematic approach, leveraging the de Bruijn graph method implemented in C++. The methodology encompasses the following key steps:

1. Preprocessing: this included generating a vector of reads from the input file and getting the length of each read.
2. De Bruijn Graph Construction: the graph was represented using a map storing the connection between k-mers for quick retrieval of k-mers during graph reconstruction.
3. Graph Traversal: Eulerian walk was employed for efficient traversal of the de Bruijn graph.
4. Genome Reconstruction:

Implementation:

Our genome assembler, implemented in C++ and utilizing the de Bruijn graph methodology, is built upon the following key components:

* De Bruijn graph construction: the graph is efficiently constructed from reads using the kmerifier module. The graph representation employs an adjacency list, facilitated by an unordered map, capturing the connections between k-mers.
* Eulerian Walk Algorithm: a fundamental part of our assembler, is responsible for traversing the de Bruijn graph systematically. A stack-based approach is employed for efficient backtracking, ensuring a comprehensive exploration of graph paths. The algorithm is further enhanced with checks for overlapping regions between consecutive nodes during the reconstruction process.
* Graph Printing and Debugging: for visualization and debugging purposes, the assembler provides a method to print the de Bruijn graph. This feature aids in a clear understanding of the graph's structure during the development and testing phases.