Analysis of Assembly Algorithm and Implementation of De Bruijn Graph to generate the longest possible consensus sequence

Ashwani Kumar*

Department of Microbiology, Miami University, Ohio

ABSTRACT

De novo genome assembly has become one of the fundamental and computationally intensive tasks at hand. This is because of the nature of genomic sequences ranging from microorganism to higher organism. Thus it is very important to develop an efficient algorithmic approach, which could give us results using less resources and time. De Bruijn graph has proved to be a promising approach in the constructing the correct path and hence correct order of nucleotide. However with the ever-increasing in-flow of data from sequencing require more efficient data structure and algorithms. This report looks into the currently available assembly algorithms and implements the De Bruijn graph to find the correct order of nucleotide. Two data structure proposed by Ye *et al*. (2012) and Bowe *et al*. (2013) will be implemented in python. The results after implementation will be compared with ABySS, to test memory usage and correctness of implemented data structure.

Keywords: De novo assembly, De Bruijn Graph, nucleotide, microorganism, assembly algorithms.

Index Terms: Reads: Sequenced DNA fragments

Contigs: Reads assembled into contiguous sequence

1 INTRODUCTION

The first sequenced genome belongs to a phiX174 bacterial phage (virus that infects bacteria). Frederick Sanger completed sequencing in 1977, for which he was awarded the Nobel Prize. Later Human genome sequencing was also done using Sanger method of sequencing. But this took scientist around 7 years to complete. Eventually, with technological advancements in sequencing platforms sequencing time was greatly reduced.

Even with new technologies in market we are faced with computational challenge in determining the order of nucleotides. The general method followed for sequencing genome involves the following steps (Figure 1). Experimental Biologists take DNA samples and extract DNA from it. The extracted DNA is then fed into sequencing machines (Illumina, 454, Pac Bio). These machines generate millions of small sequenced fragments called reads. Next job is to put together all of the generated reads so that we could get sequenced order of nucleotide. This process is called genome assembly. Major challenge while assembling genome is that researchers have no clue of the origin coordinates of reads. Further DNA is double stranded and there is no way of knowing a priori which strand a read belongs to. The repeats in genome are another source of error. In addition to this, sequencing platforms are not perfect and are error prone as they miss lot of reads.

Most of the genome assemblers [1,2,3,4] uses De Bruijn graph [2,3,4,5] over other methods like overlap graph. It can be defined as a directed graph with two node connected by an edge of k-mer length. Nodes share an exact (k-1) overlap. It also forms the basis of different steps in genome assembly. Due to DNA sequence of varying size, millions of nodes and edges are generated, requiring large memory and space. Thus developing an efficient assembly algorithm is current one of the fundamental question being addressed. Further, from a biological point of view knowing the correct order of nucleotide is knowing the gene sequence of an organism which forms the basis of research in various areas like cancer, Alzheimer's, diseases caused by pathogens etc.

This report is an analysis of various assembly algorithms and implementation of De Bruijn graph to find the longest consensus sequence. A detailed analysis of Hamiltonian path in the overlap graph and Eulerian path in De Bruijn graph has been presented. Towards the later part, an overview of data structure [5,6] has been described. The coding part involves the implementation of de Bruijn graph and finding an optimal path of longest sequence. The major aim of this project is to analyze the available efficient algorithms for genome assembly and address the bubble problem, which arises during the assembly. Due to complexity of genomes of different organism small input set will be considered. Errors due to sequencing will not be considered. The report closes with the discussion section, where an overview of whole report has been presented.

2 THEORY

In the last few years, research to find efficient data structure and algorithms to assemble short DNA fragments (reads) into a consensus sequence has taken new turns. Most of the assembler uses either overlap graph by Hamiltonian path or de Bruijn graph by Eulerian path to put nucleotides in correct order. Genome assembly problem could be related to the String Reconstruction problem, where input is an integer k and a collection of pattern of *k*-mers. The expected output is a string text with *k*-mer composition equal to input patterns. Hence our goal is to reconstruct a string, given a collection of *k*-mer.

In below subsections overlap graph and de Bruijn graph is described.

2.1 Overlap Graph and Hamiltonian Path

Some of the earlier assemblers used the concept of overlap graph to assemble genomes. Given a set of patters of *k*-mers, the output should be a directed graph with nodes as *k*-mer patterns and edges connecting two nodes, if suffix of *k*-mer in one node is equal to prefix of *k*-mer in other (Figure 2).

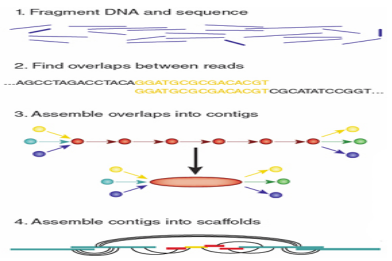
1. Fragment DNA and sequence
2. Find overlaps between reads
...AGCCTAGACCTACAGGATGCGCGACACGT
GGATGCGCGACACGTCGCATATCCGGT...
3. Assemble overlaps into contigs
4. Assemble contigs into scaffolds

Figure 1: Michael Schatz, Cold Spring Harbor, Steps in Genome Assembly

Genome assembly stitches together a genome from short sequenced pieces of DNA. Once we have the overlap graph; we try to find a path by visiting each of the nodes. A path visiting each of the nodes in a graph exactly once constitutes Hamiltonian path.

2.2 De Bruijn Graph and Eulerian Path

The concept of De Bruijn graph began to be used in most of the assembler towards the later part. The main reason was its efficiency when compared to overlap graph. We describe De Bruijn graph as follows.

Given a set of *k*-mers patterns as input. We are expected to output a directed graph such that edges are *k*-mers pattern and nodes share (k-1)-mers occurring as a prefix or suffix in *k*-mers pattern. Once we have the De Bruijn graph, a path is constructed by visiting all the edges instead of nodes.

One of the most important feature of De Bruijn graph is that if, the nodes are same we could glue those nodes together, thus reducing the number of nodes. The number of edges remains same. This is important because genomes of almost every organism have repeat sequences. Presence of repeat sequences makes it difficult to look ahead while constructing the path during assembly. Gluing together repeated nodes helps in constructing better and optimal path.

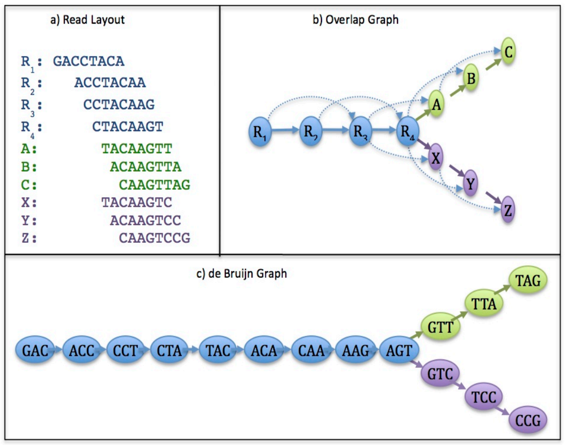

Figure 2

Assembly of large genomes using second-generation assembly Salzberg *et al*. (2010), Genome Research 20, 1165-73.

2.3 Errors in Genome Assembly

This report looks into one of the errors encountered during the genome assembly. A source of common error during sequencing is substitution, deletion, duplication of bases in the reads being generated. This further leads to the erroneous k-mers generation. Hence we have a situation where two paths could be generated and then decision has to be made as to which path to follow to get the correct order of DNA sequence.
The case has been described below in figure.

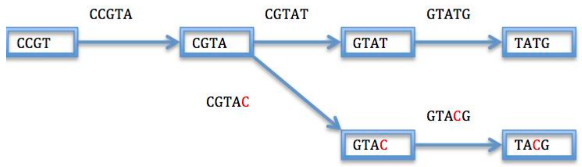

Figure 3. Generation of bubble during assembly

3 Data Structure

One of the fundamental tasks in creating de Bruijn graph is the choice of an efficient data structure. There has been a lot of research in this field. One of the first assembler [6] used an open-addressing hash table. The idea was to store *k*-mers of the graph in the keys. The edges need not be stored as it could be inferred from nodes.

Ye *et al* (2012) [7] proposed sparse k-mer graph approach. This approach greatly reduces the memory requirement by skipping some fraction of k-mers (m). For example with overlaps (C, D) and (D, E), D will be skipped. The number of skipped bases or less could be stored as neighboring bases on each side of k-mer. The k-mer storage is thus reduced to 1/g. The memory requirement using this approach corresponds to Ω (k/g).

Bowe *et al*. (2013) [8] proposed a data structure, which uses Burrows-Wheeler transformation and rank, select functions to store the final nodes and traverse an optimum path of correct nucleotide order.

4 Implementation

As stated above the aim of this project is to present a detailed analysis of efficient algorithm present for genome assembly. The data structure as suggested by Ye *et al. (2012)*, and Bowe *et al. (2013)* will be implemented to generate De Bruijn graph and hence the optimal path. The path should represent correct order of nucleotides.

The bubble error due to sequencing will be solved by Breadth first search. All the codes will be written in python.
Algorithm:

1. Given a Text string and integer, get a collection of all possible *k*-mers.
2. Construct a path PATH_OF_GRAPH, with edges as k-mers and two nodes sharing (k-1)-mer prefix and suffix.
3. If the above condition hold true, glue all the repeating nodes.
4. Start visiting every edge exactly once and construct the optimal path.
5. Return path.

Less complex genomes will be used to reduce the complexity.

5 Analysis and Performance

The correctness of algorithm will be tested with reference to ABySS assembler. For implementing the data structure, the genome of known microorganism will be used as an input set. This is because; order of nucleotide generated could then be verified against the known order of genomic sequence.

* kumara3@miamioh.edu

Further to measure the correctness of output sequence generated, BLAST [9] will be performed with query as output sequence and target as genomic databases.

6 Conclusion

This report aims at analyzing some of the efficient assembly algorithms available. It has been shown by various researchers that implementation of De Bruijn graph has reduced the memory usage and running time. ABySS [6], Velvet [2], SOAPdenovo [3] and ALLPATHS [4] uses the concept of De Bruijn graph.

The report is an attempt to test and analyze the behavior of suggested data structure in implementing De Bruijn graph. The report also considers the analysis of running time and memory usage by comparing the results to ABySS assembler. The big picture of this project work could be seen as an additional insight into various assemblers, their errors and generation of ideas to develop more efficient ones.

References

[1] Myers, E. W. *et al.* A whole-genome assembly of *Drosophila*. *Science* **287**, 2196–2204 (2000)

[2] Zerbino, D. R. & Birney, E. Velvet: algorithms for *de novo* short read assembly using de Bruijn graphs. *Genome Res.* **18**, 821–829 (2008)

[3] Li, R. *et al. De novo* assembly of human genomes with massively parallel short read sequencing. *Genome Res.* **20**, 265–272 (2010).

[4] Butler, J. *et al.* ALLPATHS: *de novo* assembly of whole-genome shotgun microreads.*Genome Res.* **18**, 810–820 (2008).

[5] Compeau P, Pevzner P, Tesler G (2011) How to apply de Bruijn graphs to genome assembly. Nat Biotechnol **29**:987–991.

[6] Simpson JT, et al. (2009) ABySS: A parallel assembler for short read sequence data. Genome Res**19**:1117–1123

[7] Ye C, Ma ZS, Cannon CH, Pop M, Yu DW: Exploiting sparseness in de novo

[8] http://alexbowe.com/succinct-debruijn-graphs/#fnref:8

[9] Altschul,S.F., Gish,W., Miller,W., Myers,E.W. and Lipman,D.J. (1990) J. Mol. Biol., 215, 403–410.