Optimal K-Mer Length for De Bruijn Graph assembly

Christopher Kareem Schmitt - 12.7.2019

Abstract

The research descibed in this paper will explore solutions to the genome assembly problem using De Bruijn Graphs and propose a method for improving assembly time by providing a metric to estimate the minimum required k-mer length to correctly sequence a genome of arbitary size.

1. Introduction

Genome sequencing and whole-genome sequencing are becoming critical tools in a wide range of diverse fields, ranging from pharmacology to family planning. Whole-genome sequencing could, in the future, allow for individualized medicine, where a patient would know in advance what would be an effective treatment. The De Bruijn graph is a powerful tool for solving the genome assembly problem. It does, however, struggle with repeated sequences. This paper will focus on the definition of the assembly problem, solution using the De Bruijn graph, and a proposal to improve assemble-time efficiency. Previous work in the field includes DNA compression methods through De Bruijn graphs [1] and the development of systems that can handle the terrabytes of data that comes with megagenome assemble [2]. The method that this paper will propose aims to simplify this problem by providing an estimate for the k-mer length of the De Bruijn graph, thus reducing memory consumption and assembly time.

2. Definitions

2.1. The Assembly Problem

Genome assembly or sequence assembly refers to the process of taking many short fragments, or reads, of a genome and merging them such that the source genome is readable. Assembling a sequence of any useful length requires terabytes worth of sequencing data [2]. When read lengths are short and less data is available, repetitions in the genome can make propper sequencing nigh impossible [3]. Longer reads increase the likelihood of each read containing some unique feature [4], simplifying the assembly process. Longer reads, however, are both more likely to contain errors and are significantly more challenging to obtain. At current, a typical sequencing read ranges form 100 to 400 basepairs long [4]. Reducing the required read length would both improve assembly time and assembly accuracy (as shorter reads can be more accurate).

2.2. The Read Alignment Problem

The read allignment problem is closely related to the assembly problem. Read alignment refers to the process of taking a number of subsequences and aligning them such that the original sequence. Many older genome assembly systems rely on basic read allignment to solve the assembly problem. This takes enormous amounts of data when sequencing larger genomes and relies heavily on long, accurate reads.

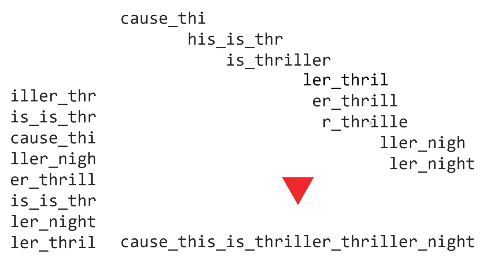

Figure 1. An example of the read alignment problem being solved for a short sequence

2.3. The De Bruijn Graph

A De Bruijn graph is a directed graph that represents overlaps in sequences of symbols. The vertices of a De Bruijn graph consist of every permutation of a fixed, finite alphabet. Each edge corresponds do a difference of one symbol between to sequences A complete De Bruijn graph will have a^b vertices, where a is the number of unique symbols in the alphabet and b is the sequence length[5]. When using a De Bruijn graph to solve the assembly problem, not every permutation will appear. It is also possible that a given permutation will appear more than once. These graphs are, therefore, partial De Bruijn multigraphs. De Bruijn graphs have an extraordinary property when they have a unique Eulerian circuit. The eulerian circuit (the path that uses each edge exactly once) corresponds to the source sequence. Finding the eulerian circuit of a De Bruijn graph yields the source when loops are resolvable.

Algorithm 1 Inserting a read into graph

```
if $prefix \in Nodes$ then
    $leftNode \leftarrow Nodes[prefix]$
else
    $leftNode \leftarrow Nodes[prefix] \leftarrow node(prefix)$
end if
if $suffix \in Nodes$ then
    $rightNode \leftarrow Nodes[suffix]$
else
    $rightNode \leftarrow Nodes[suffix] \leftarrow node(suffix)$
end if
```

Algorithm 2 Compute Eulerian circuit

```
$circuit \leftarrow []$
if $eulerian$ then
    $startNode \leftarrow next(Nodes)$
    while $\mid startNode \mid > 0$ do
        $add(circuit, startNode)$
        $startNode \leftarrow next(startNode)$
    end while
end if
return $circuit$
```

In algorithm one, prefix is defined as the sequence, in order, to be inserted minus the last element. Suffix is defined in a similar manner: as the, sequence, in order, minus the first element. Next is defined as the succeeding item in the order of nodes, or an arbitary node when no ordering exists.

Figure 2 shows a short example of the De Bruijn graph of ACTCGTACGA. To simplify the example, each of the reads (shown in red) is of length three, though in practice these reads can be any length. This graph can be read by simply starting at the head node (AC in this case) and adding it to the sequence. Then the edges are followed (respecting their directionality) such that each edge is traveresed exactly once. The last letter of each node (after the first) is added to the sequence. In this example, the graph would be read as: AC-T-C-G-T-A-C-G-A.

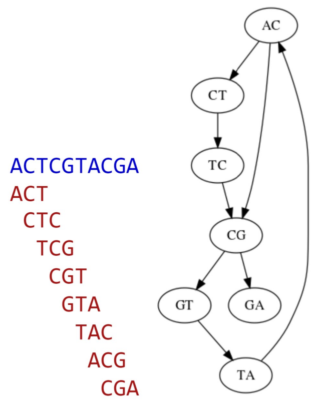

Figure 2. An example graph of the reads of the sequence ACTCGTACGA

In figure 3, the sequence has three copies of the subsequence "AT". This creates an unresolvable loop in the graph. This graph has two distict "loops" in its structure, yielding two valid eulerian circuits. When read length is short, this kind of error is common [4]. This error can be resolved by increasing the length of the reads by one letter. This yields figure 4, which has exactly one eulerian circuit. This issue is resolved becuase the "AT" subsequence that was causing problems now has more unique features associated with it which the graph can use to assemble unambiguiosly.

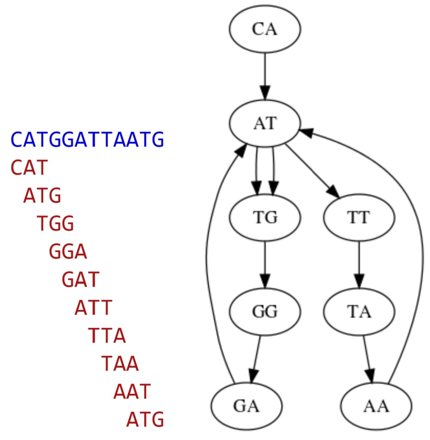

Figure 3. An example of a graph with an unresolvable loop

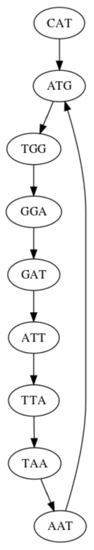

Figure 4. A graph constructed of 4-mers

Figure 4 would be read as CATGGATTAATG, by stating at the first unbalanced node (CAT) and proceding to follow the loop as dictated by the eulerian circuit.

3. Read Length Optimization

3.1. Data

As seen in figures 3 and 4, read length has a direct effect on the resolvability of the De Bruijn graph. In general, longer reads have more unique features useful for aligning [4]. Longer reads, however, require a larger De Bruijn graph. Note that the number of vertices in a complete De Bruijn graph of length b is a^b where a is the number of unique symbols in the alpahbet [5], in this case, four (A, C, T, and G). For this reason, read lengths should be minimized. The optimal read length for this task would be the minimum read length which still yields a single eulerian circuit. figures 5 and 6 plot average and maximum required read length over total genome length. There is a clear logarithmic relationship between required read length and genome sequence length. Figures 7 and 8 again plot the average and maximum required read lengths versus total genome length, and again, the same logarithmic relationship is present. It is clear that, when the genome is small, very small reads will suffice. This is expected because a short sequence will have fewer repeated segments than a longer one. This means that there will be fewer potential loops to resolve. As the size of the genome increases, more unique patterns are nessasary to solve the allignment problem.

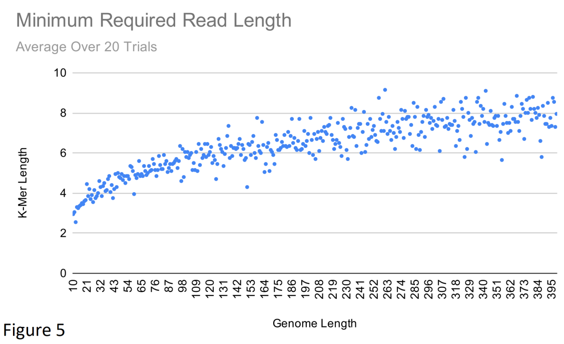

Figure 5

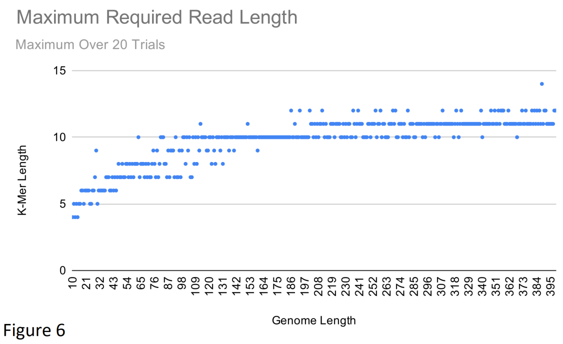

Figure 6

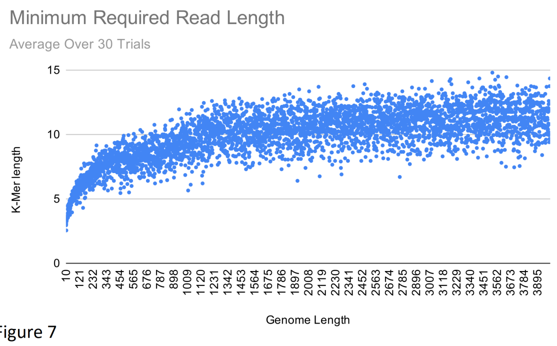

Figure 7

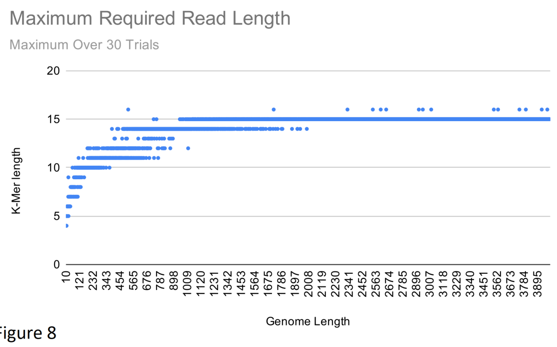

Figure 8

3.2. Method

These data where collected by running the De Bruijn graph sequencer implementation on random segments of the HIV-1 virus [6]. To start a random segment of length k, where k is "Genome Length" on the x-axis, is taken from the refrence genome. The program then simulates "shotgun" reads, where random parts of the segment are read. These reads are then shortened down to a length of two. These shortened reads are fed into the De Bruijn graph assembler, which attempts to sequence the genome segment. If the sequencer is unable to produce a sequence, or it produces a sequence with an error, the result is thrown out and length of the shortened reads is increased by one. This process reapeats until the sequencer can correctly sequence the genome segment. This test is repeated twenty times for each genome length from ten to four-hundred. These data are aggregated in figures 5 and 6. Figures 7 and 8 show the same process over thirty trials from k-mer length 10 to k-mer length 4000.

3.3. Conclusion

In this paper we have shown, that for genome lengths between 10 and 4000, there exists a logarithmic relationship between the length of the genome and the minimum avgerage read length. Applying a logarithmic regression to these data yields $1.51 \ln x$ with $R^2 = 0.641$. We can say that, for genome sizes between 10 and 4000, the function $\mathscr{K}$ which takes a genome length and returns the optimal read length is $\mathscr{K}(x) \approx 1.51 \ln x$. In the future, metrics like the one proposed in this paper could be used to greatly reduce both memory and time costs of sequencing an entire genome. Combined with tequinques like De Bruijin graph gene compression [1], better scaling assemblers [3] and other techniques like edge minimization [7] personalized medicine and personal gene ownership could become a reality.

References

[1] Diego Díaz-Domínguez, Travis Gagie, and Gonzalo Navarro. Simulating the dna string graph in succinct space, 2019.

[2] Evangelos Georganas, Rob Egan, Steven Hofmeyr, Eugene Goltsman, Bill Arndt, Andrew Tritt, Aydin Buluc, Leonid Oliker, and Katherine Yelick. Extreme scale de novo metagenome assembly, 2018.

[3] Ben Langmead, Christopher Wilks, Valentin Antonescu, and Rone Charles. Scaling read aligners to hundreds of threads on general-purpose processors. *Bioinformatics*, 35(3):421–432, 07 2018.

[4] Ben Langmead. De bruijn graph assembly lectures.

[5] Wikimedia Foundation. De bruijn graph, Nov 2019.

[6] ncbi.nlm.nih.gov. Hiv-1, complete genome - nucleotide - ncbi.

[7] Uwe Baier, Thomas Büchler, Enno Ohlebusch, and Pascal Weber. Edge minimization in de bruijn graphs, 2019.