

Whole Genome Shotgun Assembly

General outline

How to reconstruct a genome sequence from scratch based on a set of whole genome shotgun sequencing data? Traditionally, we differentiate between two strategies for DNA sequence assembly (i) the overlap layout based methods, and (ii) the de Bruijn graph based methods (Fig. 1). **Overlap**


layout consensus approaches are rather easy to explain. Each read represents a  consecutive stretch of the sequenced genome. From this follows that two overlapping reads, i.e. reads where the end of one read resembles the start of the second read, can be joined to represent a longer segment of the sequenced genome. The concept of **de Bruijn graph based approaches** is fundamentally different. These algorithms aim at reconstructing the shortest superstring that can be built from a list of words of length k , so called **kmers**. Sequencing reads, in the first approximation, serve only to identify kmers occurring in the genome of interest, and are, in principle, no longer considered as representations of consecutive stretches of the genome under study. This is of course an oversimplification, but it may help to initially understand the difference between an overlap consensus based assembler and a de Bruijn graph based assembler.



Figure 1: General approaches to the read assembly problem. a) Sequence reads are generated from a circular (or linear) genome. b) Overlap layout based assemblers generate an overlap graph from reads whose terminal overlaps display a minimal length and sequence similarity. The overlap is considered as evidence that the reads partly cover the same section of the template genome. Word (kmer) based approaches use the sequence reads as resources to extract words that occur in the genome sequence. From the word lists graphs can be reconstructed using either the kmers as nodes (c), or more commonly as edges (d). In the latter case, an Eulerian cycle (or path in the case of linear sequences) can be identified that reconstructs the template sequence. Figure taken from Compeau et al. (2011).

de Bruijn graph based approaches

Functional concepts

The general procedure of a de Bruijn graph based assembly is pretty similar across various approaches. Initially, the user has to decide on a length (or a range) for k . The software then compiles a kmer catalogue from the read set. The number of reads a given kmer is represented in is then called the kmer coverage (Figs. 2 and 3).



Figure 2: A typical kmer coverage histogram for a sequencing experiment of a single species. The large number of kmers with a low coverage are introduced by sequencing errors. Genomic kmers have a higher coverage with a mean of 40 for this plot. The subtle bump at a kmer coverage around 85 represents kmers that correspond to words that occur in repeat regions.



Figure 3: The kmer coverage histogram depends on the chosen value of k . The left plot shows the histogram resulting from a fungal-algal metagenome for a k of 151. The genomic kmers from the alga overlap in frequency with the kmers introduced by the sequencing error. They are hence ignored during genome assembly. It is for this reason that the assembly covers only 40% of the metagenome. The right plot gives the kmer histogram for the same data, this time using a k of 51. The algal kmer frequency is now clearly separated from the kmers introduced by the sequencing error, and are thus used for the genome reconstruction. The resulting assembly, though having a lower contiguity ($N50 = 22$ Kb), covers now almost the entire metagenome.

Such kmers with a low coverage – and also those with a **kmer coverage** below the expectation given the read coverage – will be considered as sequencing errors. To save memory, they will either be removed from the list, or alternatively are subjected to an inherent error correction. Subsequently, the initial de Bruijn graph will be generated from the remaining kmers. K-1 mers will serve as nodes in the graph, and a directed edge is drawn between two nodes once their k-2 overlap of prefix and suffix result in an observed kmer (Fig. 4). This pre-graph is then subsequently simplified by correcting for sequencing errors and repeats both resulting in reticulations of the graph, and subsequent nodes connected by unambiguous paths through the graph are condensed into longer nodes. All long nodes with lengths above a user-defined minimum contig length are finally output as contigs. Note, in individual assemblers a scaffolding procedure is directly implemented. It is, thus, advisable to make sure at what stage of the assembly procedure your program ends.



Figure 4: A simple de Bruijn graph using a kmer size of 3. k-1 mers serve as nodes in the graph, and an edge is drawn to connect two nodes if their k-2 terminal overlap forms a kmer that is observed in the data. It is an inherent assumption that each kmer occurs only once in the template genome.

Method selection

Meanwhile a plethora of different WGS assemblers exist, and it is hard to decide a priori which assembler performs best for a given genome and WGS data set. However, determining how good an assembly is, can be very difficult and there's even a competition – the Assemblathon – which intends to benchmark current state-of-the-art methods in genome assembly (Earl, et al. 2011; Bradnam, et al. 2013). Still the problem exists, to what extent the insights from these benchmarks can be generalized to any particular assembly problem. Given the complexity of the assembly problem, it is easily conceivable that an algorithm that performs non-optimal on any of the benchmark data sets happens to be superior for your particular assembly problem. It is, thus, that separate benchmarks are generated for particular subsets of genomes (e.g. Abbas, et al. 2014). As an alternative, Greshake et al. (2016) recently proposed the idea of simulated twin sets. The idea here is, to simulate a WGS read set that closely resembles that of the actual sequencing experiment, hence its name 'twin set'. Assemblers can then be custom-benchmarked on the twin sets resulting in a more informed assembler choice.

Task list

- [Task list](#)

From:
<https://aplbio.biologie.uni-frankfurt.de/teaching/wiki/> - **Teaching**

Permanent link:
<https://aplbio.biologie.uni-frankfurt.de/teaching/wiki/doku.php?id=general:bioseqanalysis:shotgunassembly>

Last update: **2021/10/19 21:48**

