Genome Analysis using "Velvet" Algorithm

Abstract:

Velvet is an algorithm package that has been designed to deal with de novo genome assembly and short read sequencing alignments. This is achieved through the manipulation of de Bruijn graphs for genomic sequence assembly via the removal of errors and the simplification of repeated regions. A de Bruijn graph is a compact representation based on short words (k-mers) that is ideal for high coverage, very short read (25–50 bp) data sets. Velvet represents a new approach to assembly that can leverage very short reads in combination with read pairs to produce useful assemblies. This protocol describes how to use Velvet, interpret its output, and tune its parameters for optimal results and also the algorithm used in Velvet.

Keywords*: Genome Sequencing, Velvet Asssembler, de Novo Assembler, de Bruijn Graph*

Introduction

Current DNA sequencing technologies, including NGS, are limited on the basis that genomes are much larger than any read length. Typically, NGS operate with small reads, less than 400 bp, and have a much lower cost per read than previous first generation machines. They are also simpler to operate with higher parallel operation and higher yield.

However, short reads contain less information than larger reads thus requiring a higher assembly read coverage to allow for detectable overlaps. This in turn increases the complexity of the sequencing and significantly increases computational requirements. A larger number of reads also increases the size of the overlap graph, making it more difficult and lengthy to compute. The connections between the reads become more indistinct due to the decrease in overlapping sections leading to a greater possibility of errors.

To overcome these issues, dynamic sequencing programs that are efficient, highly cost effective and able to resolve errors and repeats were developed. Velvet algorithms was designed for this and are able to perform short read de novo sequencing alignment in relatively short computational time and with lower memory usage compared to other assemblers.

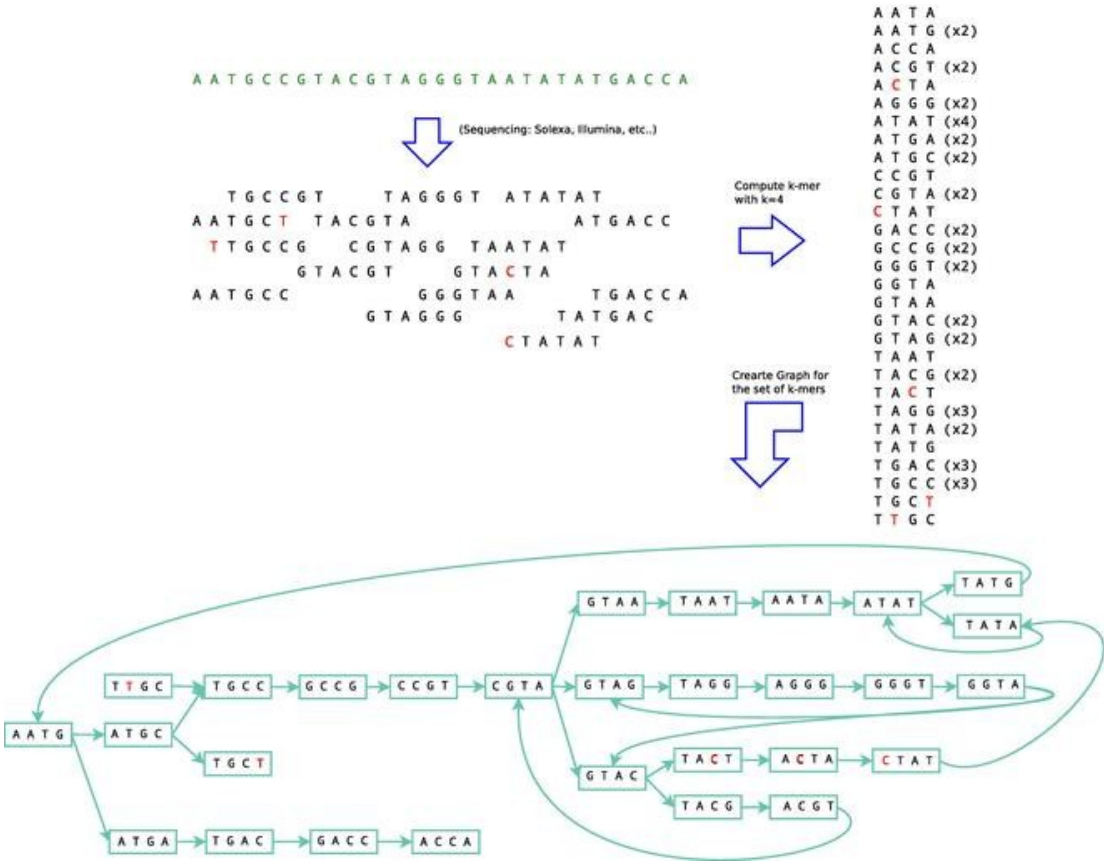
Survey of Mathematical models

- **De Bruijn Graph**

Velvet builds a de Bruijn graph from the reads and removes errors from the graph. It then tries to resolve repeats, based on the available information, whether long reads or paired-end reads. It finishes by outputting an assembly of the reads, along with various statistics.

Velvet works by efficiently manipulating de Bruijn graphs through simplification and compression, without the loss of graph information, by converging non-intersecting paths into single nodes. It eliminates errors and resolves repeats by first using an error correction algorithm that merges sequences together. Repeats are then removed from the sequence via the repeat solver that separates paths which share local overlaps. The combination of short reads and read pairs allows Velvet to resolve small repeats and produce contigs of reasonable length.

As already mentioned Velvet uses the de Bruijn graph to assemble short reads. More specifically Velvet represents each different k-mer obtained from the reads by a unique node on the graph. Two nodes are connected if its k-mers have a k-1 overlap. In other words, an arc from node A to node B exists if the last k-1 characters of the k-mer, represented by A, are the first k-1 characters of the k-mer represented by B. The following figure shows an example of a de Bruijn graph generated with Velvet:

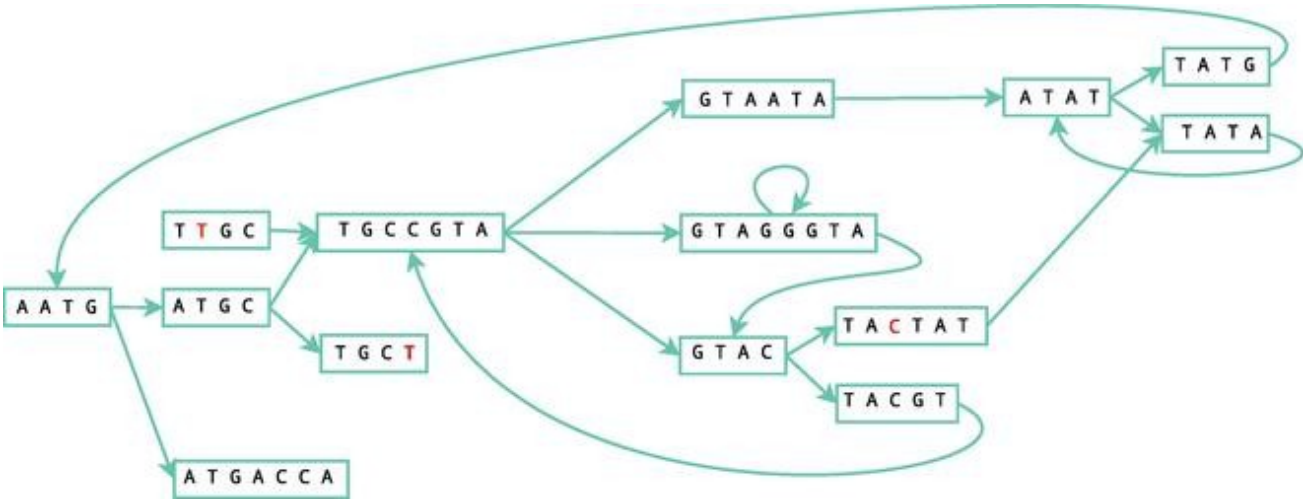


The same process is simultaneously done with the reverse complement of all the k-mers to take into account the overlaps between the reads of opposite strands. A number of optimizations can be done over the graph which includes simplification and error removal.

Algorithm

Step 1: Simplification

An easy way to save memory costs is to merge nodes that do not affect the path generated in the graph, i.e., whenever a node A has only one outgoing arc that points to node B, with only one ingoing arc, the nodes can be merged. It is possible to represent both nodes as one, merging them and all their information together. The next figure illustrates this process in the simplification of the initial example.



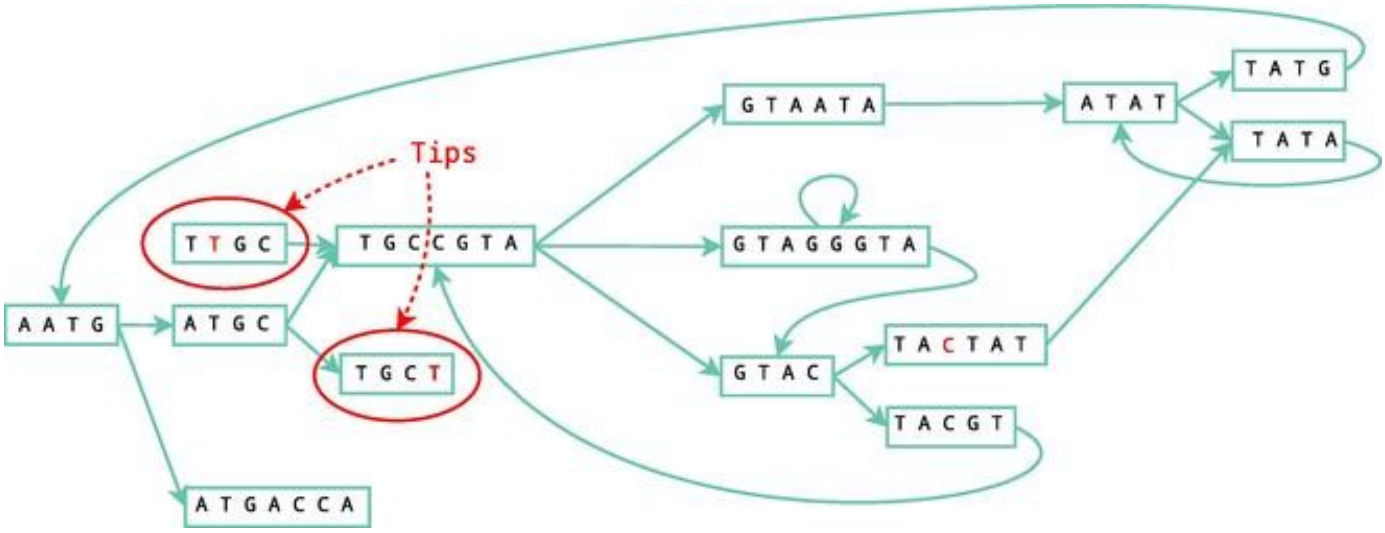
Step 2: Error Removal

Errors in the graph can be caused by the sequencing process or it could simply be that the biological sample contains some errors (for example polymorphisms). Velvet recognizes three kinds of errors: tips; bubbles; and erroneous connections.

Step 3: Tips

A node is considered a tip and should be erased if it is disconnected on one of its ends, the length of the information stored in the node is shorter than 2k, and the arc leading to this node has a low multiplicity (number of times the arc was found during the construction of the graph) and as a

result cannot be compared to other alternative paths. Once these errors are removed, the graph once again undergoes simplification.



Step 4: Bubbles

Bubbles are generated when two paths start and end at the same nodes. Normally bubbles are caused by errors or biological variants. These errors are removed using the Tour Bus algorithm, which is similar to a Dijkstra's algorithm, a breadth-first search that detects the best path to follow and determines which ones should be erased. A simple example is shown in figure.

Step 5: Erroneous connections

These are connections that do not generate correct paths or do not create any recognizable structures within the graph. Velvet erases these errors after completion of the Tour Bus algorithm, applying a simple coverage cut-off that must be defined by the user.

IMPLEMENTATION

Necessary Resources:

1. Hardware:

A system with as much physical memory as possible (12 GB or more) is recommended. Velvet can in theory function in a 32-bit environment, but such systems have memory limitations which might ultimately be a constraint for assembly. A 64-bit system is therefore strongly recommended.

2. Software:

- velveth: This command helps to construct the data set (hashes the reads) for velvetg and includes information about the meaning of each sequence files.
- velvetg: This command builds the de Bruijn graph from the k-mers obtained by velveth and runs simplification and error correction over the graph. It then extracts the contigs.

After running velvetg a number of files are generated. Most importantly, a file of contigs contains the sequences of the contigs longer than 2k, where k is the word-length used in velveth.

3. Files:

Sequence files in FASTA. FASTQ

Running velveth:

Velveth helps you construct the dataset for the following program, velvetg, and indicate to the system what each sequence file represents.

> ./velveth output_directory hash_length [[-file_format][-read_type] filename]

The hash length, also known as k-mer length, corresponds to the length, in base pairs, of the words being hashed.

Running velvetg:

Velvetg is the core of Velvet where the de Bruijn graph is built then manipulated. Note that although velvetg saves some files during the process to avoid useless recalculations, the parameters are not saved from one run to the next.

This means you can freely play around with parameters, without re-doing most of the calculations:

> ./velvetg output-directory/ -cov_cutoff 4

> ./velvetg output-directory/ -cov_cutoff 3.8

> ./velvetg output-directory/ -cov_cutoff 7

> ./velvetg output-directory/ -cov_cutoff 10

Guidelines for understanding results:

- Output Files: In its output directory (as specified on the command line of velveth and velvetg), Velvet produces a number of files, including:

  1. *contigs.fa*: Contig sequences in FASTA format.

  2. *stats.txt*: A tab-separated table with statistics on the contigs.

  3. *velvet asm.afg*: Assembly file (compatible with AMOS, see Advanced Parameters for more information on how to create and use it).

  4. *Sequences*: a modified FASTA file which contains the original sequence names (as they appear in the input files) and the corresponding Velvet read ID numbers.

Future Enhancement

Velvet algorithm is a serial algorithm. Parallelization of the algorithm without hampering the time and space complexities is the future scope in genome sequencing.

References

1. K-mer Mapping and de Bruijn graphs: The case for velvet fragment assembly (IEEE: ISBN: 978-1-5090-1612-9)

2. Velvet: Algorithms for de novo short read assembly using de Bruijn graphs (EMBL-European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SD, United Kingdom)

3. Velevet Wikipedia