Using the Columbus extension to Velvet

Daniel Zerbino

June 12, 2010

Abstract

Since its 1.0 release, the Velvet short-read assembler contains a module called Columbus which allows the user to provide reference sequences along with mappings of sequencing reads onto those reference sequences, to efficiently assist the assembly process. This short manual describes how to use the Columbus module within the Velvet package. Users unfamiliar with Velvet's use should first refer to the main Velvet Manual.

1 For impatient people

```
> head myRegions.fa
>chr1:123456789-123457789
ATGTGTGTACTAGCTAGCGCGCTAGCTAGTCATGTGTGTACTAGCTAGCGCGCTAGCTAGTC
[etc ...]

> sort myReads.sam > mySortedReads.sam

> velveth my_dir 21 -reference myRegions.fa \
-shortPaired -sam mySortedReads.sam

> velvetg my_dir [etc ...]
```

2 How could it be used?

Breakpoint assembly You sequenced an individual genome by WGS, mapped the reads onto the reference, and detected a structural variant (SV) breakpoint using the algorithm of your choice. As a further validation you wish to assemble the reads which span this breakpoint, and thus obtain an image of the rearrangement down basepair level.

You would then select the two regions which appear to have been joined (allowing for some arbitrary margin on either side), and all the reads which map onto those two regions, along with their mate-pairs. Provided with all this, Velvet would then assemble the breakpoint, using the known references as scaffolds.

If you are confounded by some form of heterzygocity at the breakpoint, then you can be sneaky and consider the assembly to be a problem between short overlapping sequences with different concentrations... a lot like transcripts! You would therefore use the Oases package to deconvolute the different "isoforms" of your "gene" (it sounds pretty roundabout, but from an assembly point of view it actually makes sense).

Assisted transcriptome assembly You sequenced the transcriptome of a new species, strain or individual, and you happen to know the gene sequences of a nearby species, strain or reference individual.

You would then map the reads onto the reference genome, using the short-read mapper of your choice, and provide the alignments along with the known exonic sequences to Velvet. It would rebuild contigs based on the alignments, which could then be used by the Oases package.

3 Overview of the process

1. Map the reads against a set of *target sequences* (typically, an entire reference genome, made up of chromosomal sequences).

2. Select *reference regions* within the target sequence (optionally, a reference region can be an entire target sequence) (e.g.: exon regions or SV regions).

3. Prepare a FASTA file containing the sequences of the reference regions, along with their coordinates within the target sequences.

4. Provide this FASTA file along with the SAM/BAM alignment file (or selected lines thereof).

5. Run *velvetg* as usual.

6. If applicable, run Oases as usual.

4 Providing read mappings

Just like like in Velvet, you can provide paired or unpaired read alignments in SAM or BAM format, as produced by most short read aligners. You should simply remember to sort the reads by read name. If you turn on strand specificity (more on this later), you must sort your reads such that the forward read of each pair comes right before the reverse read of that pair.

Because it is possible to specify regions within the target sequences, the mapping need not be specifically against the selected reference sequences. For example, if you wish to use exons as reference sequences, you can perform an alignment against the whole genome. Velvet will then sort out which alignments are within the desired regions, and ignore the rest.

Beware that some read-aligners (e.g. Tophat) only report aligned reads in their output. To make the most of *de novo* assembly, it is important to ensure that the unmapped reads are also provided to Velvet, whether in a separate file or in the SAM/BAM alignment file. It is generally better to put all the reads in the same file, so that pairings between mapped and unmapped reads are not lost.

5 Reference sequences

5.1 How to choose reference sequences

The reference sequences can be any subset of the target sequences (whether whole genome, contigs, etc) used during the mapping process.

5.2 How NOT to choose reference sequences

- Reference sequences must **NOT** overlap in the target sequence coordinate space

  This is because when reading the alignment files, Velvet must be able to assign a read to at most one reference sequence. Ambiguity in the coordinate space would defeat the purpose of providing alignments. Normally, Velvet will detect overlapping reference coordinates and exit with an error message.

- Reference sequences should **NOT** contain large amounts of repeats

  This is more a question of performance. If you load as reference large chunks of the reference genome (as in 100's of Mb) which contain large amounts of repeats, then Velvet will have to make many loops comparing identical sequences. This will slow down the assembly considerably.

- The coordinate system must **NOT** change between input files.

  This is a very obvious point, but it is important to that the coordinate system in the alignment file must be the same as that in the reference regions' file. If the coordinates are off, then Velvet will be unable to use the alignments (it will still be able to function and use the reference sequences, but not as efficiently as if it had reliable alignments). For example, if you align your reads to the human genome assembly HG19, then the reference regions must also be selected from HG19.

5.3 How to provide reference sequences to Velvet

Reference sequences must necessarily be contained in a FASTA file. If the sequences correspond to complete target sequences used in the mapping process (e.g. when mapping to contigs) then the names in the FASTA headers must be identical to those of the target sequences. If the reference sequences correspond to a subset of one of the target sequences, then its header must correspond to the standard "browser" coordinates (i.e. 1-based, inclusive on both ends, 5' start to 3' end, as illustrated in the quick example above).

5.4 Strand specificity in Columbus

If doing a strand specific analysis on Columbus, then the "-strand_specific" velveth flag must be provided before any option, i.e. right after the hash length.

In this case, the reference sequences are taken from the desired strand. The headers of the regions on the negative strand have their start and end coordinates reversed, i.e. the 3' coordinate comes before the 5' coordinate.

For example, if the positive strand of a toy example is:

```
>chr1:1000-1010
AGTCGATAGA
```

then the reverse complement of this region is provided as:

```
>chr1:1010-1000
TCTATCGACT
```

6 Some scripts I use

Below are a few scripts which I put up to prototype Columbus. They're pretty rough BioPerl code, but they serve their purpose...

enlarge_exons.pl This scripts adds an arbitrary buffer length (default $100bp$) upstream and downstream of each exon. This avoid false positive overlaps just because one reference region happens to end at a homologous region.

merge_gtf_exons.pl This script sorts regions, and merges overlapping ones (it assumes no strand specificity).

gff2fasta.pl Once you have the GFF file you desire, it can be converted into a fasta with the appropriate headers with this script.

Typically, you would prepare the references as such:

```
grep exon my_annotation.gtf | enlarge_exons.pl | merge_gtf_exons.pl \
| gff2fasta.pl my_reference_assembly.fa > my_reference_sequences.fa
```