Oases Manual (version 0.2)

Daniel R. Zerbino & Marcel H. Schulz

December 2, 2011

Contents

NOTE If you are not familiar with using Velvet, read Velvet's manual first, as many references below will point to that document.

1 For impatient people

```
> velveth directory 21,23 data/reads.fa
> velvetg directory_21 -read_trkg yes
> oases directory_21
> ls directory_21

> velvetg directory_23 -read_trkg yes
> oases directory_23

> velveth mergedAssembly 23 -long directory*/transcripts.fa
> velvetg mergedAssembly -read_trkg yes -conserveLong yes
> oases mergedAssembly -merge
```

Or use the python script oases_pipeline.py.

```
> python oases_pipeline.py -m 21 -M 23 data/reads.fa
```

2 Installing Oases

2.1 Getting the code

The recommended installation process for Oases is *git* (www.git.org). After installing *git*, go into the directory of you choice and simply type:

```
git clone git://github.com/dzerbino/velvet.git
git clone git://github.com/dzerbino/oases.git
```

This will create two directories, *velvet/* and *oases/*

2.2 Compiling the code

After compiling Velvet, enter the Oases directory and type:

```
make
```

If the Velvet source directory is not next to the Oases directory, or if it has a different name, you must indicate the location of this directory:

```
make 'VELVET_DIR=/path/to/velvet'
```

Note that this must be an absolute path, without the ~/ shorthand.

If you provided any compilation options when compiling Velvet, you must provide the same when compiling Oases, as in:

```
make 'MAXKMERLENGTH=92'
```

This will produce an executable, *oases*, which you might want to place somewhere on your path for greater convenience.

3 Running Oases

To run Oases it is recommended to run an array of single-k assemblies, for different values of k (i.e. the hash length). These assemblies are then merged into a final assembly.

3.1 Single-k assemblies

Each single-k assembly consists of a simple Velvet run, followed by an Oases run. As in Velvet, the hash length k is an odd integer. Velveth is run normally and velvetg is run with only one option:

```
> velveth directory_k k reads.fa
> velvetg directory_k -read_trkg yes
```

If you have strand specific reads then you have to type:

```
> velveth directory_k k -strand_specific reads.fa
```

Oases is then run on that directory:

```
> oases directory_$k
```

3.2 Oases options

3.2.1 Oases help

In case of doubt, you can always print a help message by typing (interchangeably):

```
> oases
> oases --help
```

3.2.2 Insert length

If using paired-end reads, Oases needs to know the expected insert length (it cannot be estimated reliably automatically). The syntax to provide insert lengths is identical to that used in Velvet:

```
oases directory -ins_length 500
```

3.2.3 Coverage cutoff

Just like Velvet, low coverage contigs are removed from the assembly. Because genes span all the spectrum of expression levels and therefore coverage depths, setting the coverage cutoff does not depend on an expected or median coverage. Instead, the coverage cutoff is set to remove all the contigs whose coverage is so low that *de novo* assembly would be impossible anyways (by default it is set at $3x$). If you wish to set it to another value, simply use the Velvet-like option:

```
oases directory -cov_cutoff 5
```

3.2.4 Edge fraction cutoff

To allow more efficient error removal at high coverage, Oases integrates a dynamic correction method: if at a given node, an edge's coverage represents less than a given percentage of the sum of coverages of edges coming out of that node, it is removed.

By default this percentage is 10% but you can set it manually:

```
oases directory -edgeFractionCutoff 0.01
```

3.2.5 Scaffold filtering

By default the distance estimate between two long contigs is discarded as unreliable if it is supported by less than a given number (by default 4) or read-pairs or bridging reads. If you wish to change this cutoff:

```
oases directory -min_pair_count 5
```

3.2.6 Filtering the output

By setting the minimum transfrag length (by default 100bp), the user can control what is being output in the results files:

```
> oases directory -min_trans_lgth 200
```

3.3 Assembly merging

After running the previous process for different values of k it is necessary to merge the results of all these assemblies (contained in *transcripts.fa* files) into a single, non redundant assembly. To realize this merge, it is necessary to choose a value of K. Experiments have shown that $K = 27$ works nicely for most organisms. Higher values for K should be used if problems with missassemblies are observed.
Assume we want to create a merged assembly in a folder called MergedAssembly.

```
> velveth MergedAssembly/ K -long directory*/transcripts.fa
> velvetg MergedAssembly/ -read_trkg yes -conserveLong yes
> oases MergedAssembly/ -merge
```

NOTE that the transcripts.fa files need to be given as *long*.

3.4 Using the supplied python script

With version 0.2 of Oases comes a new python script that runs the individual single-k assemblies and the new merge module. We highly recommend to use the script for the analyses, but please read the previous subsections 3.2.2-3.2.6 for the Oases params that you need to supply to the script. However, as

the script also runs velvet please consult the velvet manual for more insight. First notice the options for the script below

```
python oases_pipeline.py
Options:
  -h, --help            show this help message and exit
  -d FILES, --data=FILES
                        Velveth file descriptors
  -p OPTIONS, --options=OPTIONS
                        Oases options that are passed to the command line,
                        e.g., -cov_cutoff 5 -ins_length 300
  -m KMIN, --kmin=KMIN  Minimum k
  -M KMAX, --kmax=KMAX  Maximum k
  -s KSTEP, --kstep=KSTEP
                        Steps in k
  -g KMERGE, --merge=KMERGE
                        Merge k
  -o NAME, --output=NAME
                        Output directory prefix
  -r, --mergeOnly       Only do the merge
  -c, --clean           Clean temp files
```

NOTE that the script checks if Oases version at least 0.2.01 and Velvet at least 1.1.7 are installed on your path. Otherwise it will not work.

We will illustrate the usage of the script with the two most common scenarios single-end and paired-end reads.
First assume we want to compute a merged assembly from 21 to 35 with Oases on a single-end data set that is strand specific and retain transcripts of minimum length 100. We want to save the assemblies in folders that start with singleEnd :

```
python oases_pipeline.py -m 21 -M 35 -o singleEnd
-d " -strand_specific yes data/reads.fa "
-p " -min_trans_lgth 100 "
```

The script creates a folder named singleEnd_k for each single-k assembly and one folder named singleEndMerged that contains the merged transcripts for all single-k assemblies in the *transcripts.fa* file.

Now assume we have paired-end reads with insert length 300 and we want to compute a merged assembly from 17 to 40. We want to save the assemblies in the folders that start with pairedEnd

```
python oases_pipeline.py -m 17 -M 40 -o pairedEnd
-d " -shortPaired  data/reads.fa "
-p " -ins_length 300 "
```

Now say we found that using k=17 was a bit too low and we want to look at a different merged assembly range. Instead of redoing all the time consuming assemblies we can re-run the merged module on a different range from 21 to 40 like this:

```
python oases_pipeline.py -m 21 -M 40 -r -o pairedEnd
```

NOTE that it is essential when you re-run the merged assembly for the script to function properly to give the exact same output prefix with the $-o$ parameter.

4 Output files

Oases produces two output files. The predicted transcript sequences can be found in *transcripts.fa* and the file *contig-ordering.txt* explains the composition of these transcripts, see below.

1. new_directory/transcripts.fa – A FASTA file containing the transcripts imputed directly from trivial clusters of contigs (loci with less than two transcripts and Confidence Values = 1) and the highly expressed transcripts imputed by dynamic programming (loci with more than 2 transcripts and Confidence Values <1).

2. new_directory/contig-ordering-highly-expressed.txt – A hybrid file which describes the contigs contained in each locus in FASTA format, interlaced with one line summaries of the transcripts generated by dynamic programming. Each line is a string of atoms defined as: $contig_id:$cumulative_length-($distance_to_next_contig)->next item

   Here the cumulative length is the total length of the transcript assembly from its 5' end to the 3' end of that contig. This allows you to locate the contig sequence within the transcript sequence.

5 FAQ and Practical considerations

I am running out of memory. What should I do?

Depending on the complexity of the dataset it may happen that Velvet and Oases start to use a large amount of memory. In general, the assembly graphs for transcriptome data are smaller as compared to genome data. However, because the coverage is not even roughly uniform it depends on the sample being sequenced how much memory is used. Below we give examples of preprocessing steps that lead to a decrease in memory consumption, sorted in order of importance.

Avoid overlapping reads

In our experience, if insert lengths are so short that paired reads overlap on their 3' ends, i.e., if insert length $< 2 \cdot$ read length, this prevents the early filtering of low coverage reads. This means that even small datasets may require huge amounts of memory. A way to avoid the problem is to join paired-end reads that overlap for example using the software FLASH *(FLASH: fast length adjustment of short reads to improve genome assemblies, Magoc T, Salzberg SL. Bioinformatics 2011)*. Otherwise it is possible to clip reads so that their overlap is strictly shorter than the hash length, that might lead to a huge loss of data however.

Avoid bad quality reads

Reads with many errors create new nodes in the de Bruijn graph, all of which are later discarded by the error removal steps. Removing the errors in the reads *a priori* is a good way to decrease the memory consumption. A very common approach is to remove bad quality bases from the ends of read sequences or remove entire read sequences, numerous packages exist to do that but differ in functionality and definition of bad quality (in no particular order):

1. FASTX-toolkit - `http://hannonlab.cshl.edu/fastx_toolkit/` (diverse functionality, command line, Galaxy support)

2. ea-utils - `http://code.google.com/p/ea-utils/wiki/FastqMcf` (diverse functionality, command line)

3. ShortRead (R-package) - see here for long version `http://manuals.bioinformatics.ucr.edu/home/ht-seq` or short version `http://jermdemo.blogspot.com/2010/03/soft-trimming-in-r-using-shortread-and.html`

4. Condetri - `http://code.google.com/p/condetri/` (only read trimmer good documentation, command line)

5. bwa - `http://bio-bwa.sourceforge.net/bwa.shtml` (command line *aln* option)

NOTE that when you use paired-end sequences and reads are discarded from the file you need to make sure that the fasta/fastq file that you give to velveth contains the paired-end reads in correct pairing.

Trim sequencing adapters

Depending on your sequencing experiment, some of the reads may have adapters that were used, e.g, during library preparation. These adapters should be trimmed. You can use the Fastx-toolkit or ea-utils to do that, among many others, see above.

Remove duplicate reads

Especially for paired-end datasets it happens that the exact same read sequence is found a large number of times. Again you can use some of the tools mentioned above to remove these duplicates.

Oases does not give an expression level. How can I get that?

In general you need to align reads to the transcripts with your favorite aligner, one that allows for multi mappings !! Although few results exist about the difficulties and problems of quantitation of *de novo* transcripts' expression levels, here are a few pointers to specialized tools:

1. eXpress - `http://bio.math.berkeley.edu/eXpress/` (command line, Windows support)

2. RSEM - `http://deweylab.biostat.wisc.edu/rsem/` (command line)

I want to predict coding regions. How can I do that?

All the transcripts reported in a locus should have the same directionality, but that is only true if there is no contamination. Standard approaches are to use blastp and tools alike. Alternativley use gene prediction tools like GlimmerHMM, SNAP or AUGUSTUS.