MICRO8657 – Lecture 12 - The Velvet Assembler

The next assembler we will consider is the Velvet assembler. This assembler is one of the most famous and well-accepted assemblers that specializes on Illumina data. Sadly, it has not been maintained since 2012, but even so, it is still one of the most widely used assemblers even to this day. Here, we will install the assembler and use it to assemble the *Yersinia pestis* KIM dataset we used for the Celera assembler.

The easiest way to install the velvet assembler is to utilize the built in repositories that we have used previously. If you are on Ubuntu on Windows 10, you can simply type the following at the command prompt:

sudo apt-get install velvet

If you are on a Mac, you can open up the terminal and type:

brew install brewsci/bio/velvet

This will go ahead and install velvet on both platforms. Velvet includes two programs that are run sequentially: velveth and velvetg. The program velveth sets up the assembly by creating kmers, while the program velvetg does the assembly itself. A description of the programs and how to run them can be found in the velvet manual locate here: http://www.ebi.ac.uk/~zerbino/velvet/Manual.pdf.

So, now that we have the assembler installed, we can go ahead and run the program on the *Y. pestis* KIM data we used for the Celera assembler. To do this, go into the READ-sra folder in the MICRO875 folder you created. Here, there are two files that we extracted from the .sra folder: SRR133640_1.fastq and SRR133640_2.fastq. Recall that these two files refer to the forward and reverse paired ends of the sequence data for this bacterium. To run velveth you need to specify a few things, including an output directory where all of the output files will be stored, the input files, and the kmer value for which you'd like the program to use. The value for kmers must be odd (e.g. 21), but if you enter an even number, it will use the closest odd number. The general format is as follows, as an example:

velveth Ypestis_velvet21 21 –fastq –shortPaired –separate SRR133640_1.fastq SRR133640_2.fastq

which indicates that we want the output to placed into the folder Ypestis_velvet21, a kmer size of 21, the fact that we are providing our input in fastq format (-fastq), that our sequence data is short paired-end data (-shortPaired), the fact that our files are separated into two different files (-separate) and then the names of the two input files.

This will then run velveth and place a number of files in the new folder it creates called Ypesists_velvet21. Next, we can run velvetg on the output folder of velveth and it will assemble the sequences. To do this, the general format is:

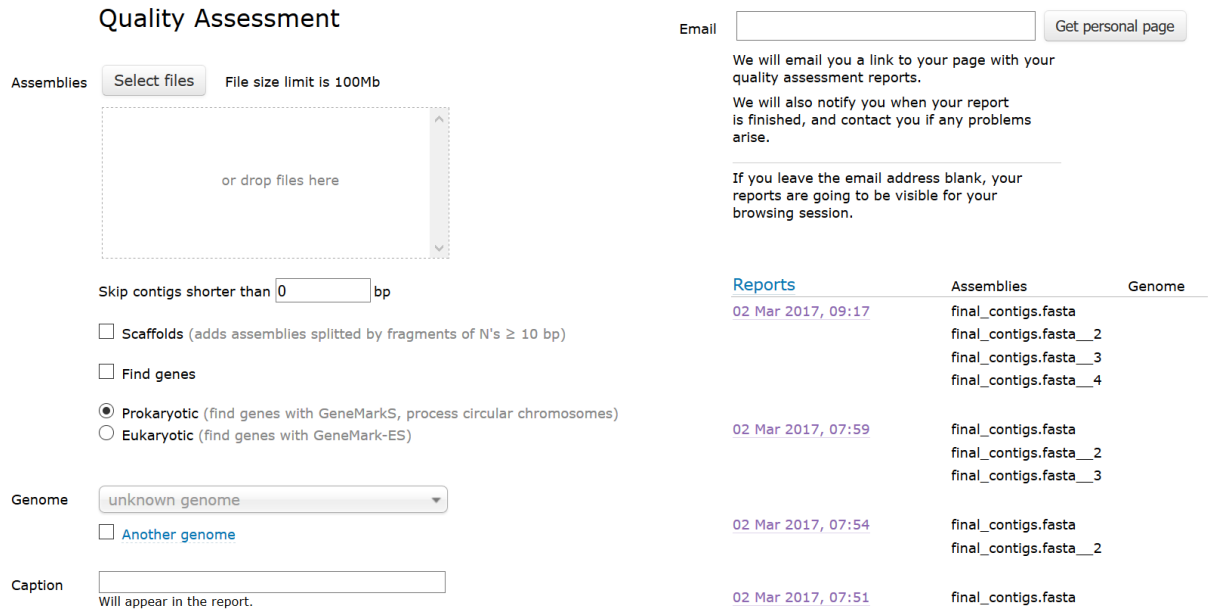
velvetg Ypestis_velvet21 –exp_cov auto –cov_cutoff auto

which indicates that velvetg should operate on the files in Ypestis_velvet21, that it should use the default settings for the expected coverage, and that it should use the default setting (auto) for the coverage cutoff value of the contigs it generates. These two settings provides important information to the assembler so that it can expect a target coverage (this will be roughly based on the number of raw sequences generated divided by the size of the genome – the "auto" setting tells the assembler to estimate the coverage), and what cutoff coverage value it should use as a quality control for keeping a

contig in the assembly (e.g. if we know our coverage is say 33X, you may not want to keep contigs that have a coverage of only 2X. The default “auto” is to indicate that it should only keep those contigs that have a coverage greater than 50% of the expected coverage).

Once this is done, you will see that it provides some basic stats, namely the number of nodes in the final assembly and the N50. The nodes refers to the fact that it constructs a de Bruijn graph and then trims down the graph to the shortest path between nodes, which refers to the best assembly. However, this doesn’t actually tell us how many contigs are in the assembly. To do this, we will need to use a second program.

There are numerous programs for telling us the statistics of an assembly. One of my favourites is the website QUAST, for which you can upload your contigs file. Note that your contigs file will be located in the output directory in a file called contigs.fa. So, you can go to the QUAST website located here: http://quast.bioinf.spbau.ru/.



You can essentially drag the contigs.fa file into the upload box where you will see it upload. I keep every parameter as default, except I change the “Skil contigs shorter than” value from 500 to 0, so we capture all contigs. You can then run the analysis by clicking the “Evaluate” button, and a report link will show up under the Reports section on the right hand side. Clicking on this will report the details of the assembly, as shown:

All statistics are based on contigs of size >= 0 bp, unless otherwise noted (e.g., "# contigs (>= 0 bp)" and "Total length (>= 0 bp)" include all contigs.)

Suggestion: assembly contigs21 contains continuous fragments of N's of length >= 10 bp. You may consider rerunning QUAST using --scaffolds (-s) option!

Statistics without reference	contigs21
# contigs	815
# contigs (>= 0 bp)	815
# contigs (>= 1000 bp)	222
# contigs (>= 5000 bp)	153
# contigs (>= 10000 bp)	122
# contigs (>= 25000 bp)	67
# contigs (>= 50000 bp)	21
Largest contig	119 695
Total length	4 477 655
Total length (>= 0 bp)	4 477 655
Total length (>= 1000 bp)	4 404 687
Total length (>= 5000 bp)	4 225 241
Total length (>= 10000 bp)	3 984 445
Total length (>= 25000 bp)	3 041 073
Total length (>= 50000 bp)	1 414 353
N50	36 461
N75	20 678
L50	41
L75	81
GC (%)	47.37
Mismatches	
# N's	39 153
# N's per 100 kbp	874.41

From here, you can see that the assembler generate 815 contigs, with an N50 of 36,461bp, which is not bad. However, this is not as good as the Celera assembler, which generated 288 contigs, although the N50 is substantially better. However, here we did a first pass assembly using a rather small kmer value of 21. In general, using a larger kmer value will get you a better assembly, and one rule of thumb is that a kmer value that is roughly 50% of your read length gets you the best results. In this case, since our data is 2x100bp reads, we could choose a kmer value of something like 49 or 51 and see what the results look like. However, if we tried to run this command:

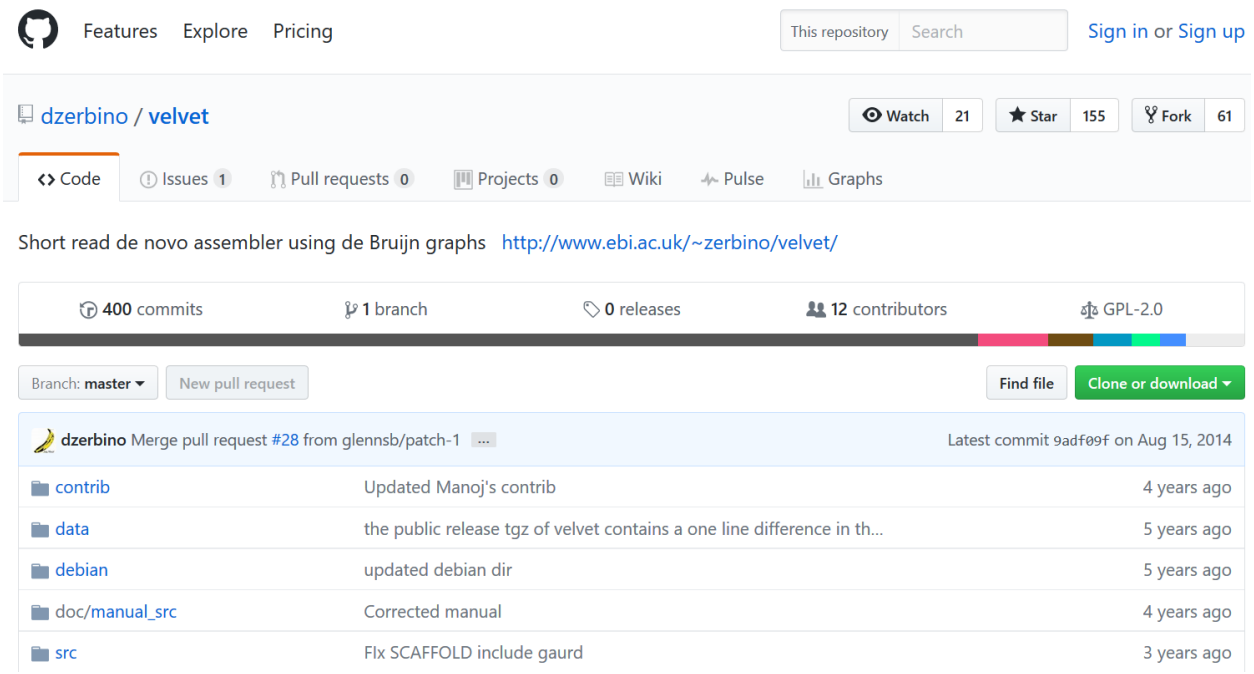
velveth Ypestis_velvet51 51 –fastq –shortPaired –separate SRR133640_1.fastq SRR133640_2.fastq

we actually get an error indicating that velvet was compiled with a max kmer value of 33, and that it will thus default to that kmer value. This presents a problem, as we would ideally like to use a higher kmer value. So to solve this, we will actually have to go and recompile velvet with a higher kmer value and try this again. So, in order to do this, we will have download the sourcecode and compile it from scratch in a similar manner to what we did for the Celera assembler. However, unlike the Celera assembler, we have one trick up our sleeves that we can use to make this a lot easier.

In the bioinformatics field, there is a central repository that can be easily accessed called “github” (github.com). Github is a file manager that allows users to “clone” entire folders containing programs for easily compiling. Installing git is both easy and free. On a Mac, it is native and does not require any installation. On Ubuntu on Windows 10, it can be installed using the following command:

sudo apt-get install git

Once installed, you can essentially find the folder containing the velvet sourcecode (do a google search for “git velvet”), which is shown here:



The key thing to notice is that there is a “Clone or Download” button that you can click and copy the URL to your clipboard. Once you do this, you can then invoke the git clone command to copy the entire contents of the folder to your computer. To do this, go to the MICRO875 folder in the Terminal and then type the following:

git clone https://github.com/dzerbino/velvet.git

where the URL is the one you copied to your clipboard. This will then invoke the git clone command a new folder called “velvet” will be created containing all of the files.

So, to compile velvet with a higher kmer value, you can look in the manual to see how this is done. A quick read will show that you can simply invoke the make command with an appended MAXKMERVALUE that you specify. To do this go into the velvet folder, and type the following:

make MAXKMERVALUE=150

This will compile velvet with a maximum kmer value of 150, which means that you can specify a kmer value up to 150. Once compilation is complete, it will generate two new files: velveth and velvetg. So, now we have our new binaries, we need to replace our old binaries. The cleanest way to do this is to remove our old installation, and copy the new binaries we just compiled into their binary source. To do this, you can run the following command if you are on a Mac:

brew uninstall velvet

or the following if you are on Ubuntu on Windows 10:

sudo apt-get remove velvet

Both commands will remove our old versions of velvet. We can then do the following to copy our new binaries into a Path accessible folder:

sudo cp velveth /usr/local/bin

sudo cp velvetg /usr/local/bin

This will copy the two files that we just compiled into /usr/local/bin, which is PATH accessible. Now, we can go back to our READS-sra folder, and rerun our analysis with a kmer value of 51

velveth Ypestis_velvet51 51 –fastq –shortPaired –separate SRR133640_1.fastq SRR133640_2.fastq

We can run velvetg on these new data:

velvetg Ypestis_velvet51 –exp_cov auto –cov_cutoff auto

which will generate a new assembly. Comparison of this assembly with our previous assembly (kmer = 21) shows a substantial improvement:

Statistics without reference	**contigs_K51**	**contigs21**
# contigs	266	815
# contigs (>= 0 bp)	266	815
# contigs (>= 1000 bp)	168	222
# contigs (>= 5000 bp)	133	153
# contigs (>= 10000 bp)	106	122
# contigs (>= 25000 bp)	71	67
# contigs (>= 50000 bp)	29	21
Largest contig	120 060	119 695
Total length	4 475 577	4 477 655
Total length (>= 0 bp)	4 475 577	4 477 655
Total length (>= 1000 bp)	4 446 543	4 404 687
Total length (>= 5000 bp)	4 350 983	4 225 241
Total length (>= 10000 bp)	4 141 702	3 984 445
Total length (>= 25000 bp)	3 586 092	3 041 073
Total length (>= 50000 bp)	2 073 101	1 414 353
N50	48 664	36 461
N75	27 454	20 678
L50	33	41
L75	63	81
GC (%)	47.39	47.37
Mismatches		
# N's	10 966	39 153
# N's per 100 kbp	245.02	874.41

We are now down to 266 contigs, with a nice N50 of 48,664. So we now know how to do some basic assemblies using Velvet!

One last thing, if you wish to step-through a number of kmer values, velveth will let you do this easily. For example, if you wanted to explore the kmer space around 51 to see if you can improve the assembly, you could run the following velveth command:

velveth Ypestis_velvet 45,53,2 –fastq –shortPaired –separate SRR133640_1.fastq SRR133640_2.fastq

which will essentially run velveth on your data using the kmer values between 45 and 53, using a stepup of 2. This means that it will run velveth with a kmer value of 45, 47, 49, 51, and 53, and drop those into individual subfolders in the folder you specify (Ypestis_velvet in this case). You can then utilize velvetg on each of these directories to determine if you can get a better assembly.