MacVector 17

for Mac OS X

Assembling Plasmids from NGS data using MacVector with Assembler

MacVector, Inc.
Software for Scientists

Copyright statement

This version of the Assembler tutorial was published in May 2019.

Contents

Overview

The tutorial will show you how to use MacVector with Assembler to assemble plasmid sequences from NGS data. This particular example uses a sample set for an anonymous plasmid consisting of a pair of files;

plasmid-1_S5_L001_R1_001.fastq

plasmid-1_S5_L001_R2_001.fastq

These files are each 285.3 MB in size and each contain approximately 1,375,000 reads of 75nt in length for a total number of sequenced residues of approximately 20.6 million. The plasmid is 8,859bp in size, so this means every residue in the plasmid has been sequenced an average of 2,328 times. This massive over-sequencing causes some considerable problems – most of the *de novo* assemblers have been tuned to assemble sequences such as bacterial chromosomes, where you might have a 1 Mb to 10 Mb sequence with an average coverage of just 20-200 fold, or even less.

MacVector uses two different assemblers for *de novo* assembly of NGS reads.

Velvet: We chose this because it is capable of assembling 20-50million+ reads into 10 Mb+ sequences with relatively minimal memory requirements, so it can be used on personal Macintosh computers with as little as 8 GB of RAM. However, the more RAM you have the better.

SPAdes: This is a more recent assembler with an even smaller memory footprint that does a slightly better job getting past repeats, resulting in longer contigs. It also requires less tweaking of parameters to get optimal assemblies. However, it is significantly slower than *Velvet*.

As with most of these *de novo* assemblers, you can supply *Velvet* or *SPAdes* with too much data. When your coverage is only ~100x or less, *Velvet* will ignore the occasional read that has a sequencing error. However, once you get into the 200+x range, the ~5% or so of reads that have errors suddenly can be assembled into their own contigs due to random similar errors. This confuses the assembler as it thinks it is trying to resolve closely related direct repeats. Eventually it gets confused enough that it can't resolve all of the similar contigs and typically gives up and reports only those contigs that have few ambiguities. These often represent just a small portion of the sequenced molecule.

Experiments with the plasmid data set (see below) indicate that only the first 10,000 reads of each file is really required to generate a clean unambiguous sequence for the plasmid. 20,000 x 75nt reads = 169x coverage of the 8.9 kb plasmid.

This tutorial will run through the optimized assembly of the plasmid, using both assemblers, and also showing how you can repeat the assembly in triplicate and confirm the results to be confident the sequence is correct.

Optimization of Velvet and SPAdes Parameters

The plasmid data set was split into a series of smaller files and used in a comprehensive set of assembly experiments using MacVector. The *Velvet* implementation in MacVector can only take a single KMER value, and it should be an odd value between 11 and 255. In general, a value of about two-thirds of the average length of the input reads is a good place to start. During assembly, reads are initially aligned by looking for perfect matches of length KMER, so there is no point to having KMER be longer than the average length of the reads. The *SPAdes* implementation lets you input multiple values and it will try them all, keeping the results for the value it believes to be optimal. In theory, you could enter every potential (odd) KMER value up to the length of the reads, though in practice that results in a very slow assembly. In the example below, *SPAdes* was run with values of "31,41,51,61,71";

pRGN7782 - 8859bp					*VELVET (KMER)*						*SPADES (KMERS)*
Total Reads	***35***	***39***	***43***	***47***	***51***	***55***	***59***	***63***	***67***	***71***	***31,41,51,61,71***
20,000	8,870	8,897	8,901	8,905	8,909	8,830	8,795	8,795	2,754	531	8,930
40,000	8,870	8,897	8,901	8,905	8,909	8,913	8,917	8,921	8,806	1,469	8,930
60,000	8,870	8,897	8,901	8,905	8,909	8,913	8,917	8,921	8,890	2,281	8,930
100,000	7,855	7,886	7,894	8,891	8,895	8,899	8,917	8,921	8,890	6,542	8,930
160,000	7,844	7,886	7,894	7,914	8,909	8,899	8,917	8,921	8,923	8,561	8,930
240,000	7,847	7,855	7,884	7,902	8,348	8,899	8,917	8,921	8,925	8,686	8,930
350,000	7,722	7,855	7,877	7,355	7,363	7,371	8,917	8,921	8,925	8,928	8,930
500,000	7,723	7,873	7,877	7,892	7,900	7,908	8,903	8,921	8,925	8,929	8,930
700,000		7,713	7,716	7,893	7,874	8,889	7,926	8,907	8,925	8,929	8,930
1,000,000			4,634	7,721	7,724	7,705	7,926	8,907	8,925	8,929	8,930
2,740,000				683	8,522	7,519	7,480	7,870	7,546	8,929	4,445

The above table lists the longest contig resulting from each assembly. Those in black could be circularized to create the correct plasmid sequence. Those in red could not.

It is immediately obvious that there is a relationship between the number of reads in the assembly and the optimal *Velvet* KMER. Assemblies using 60,000 reads (2 x 30,000) gave the best results with the broadest range of KMER values. With the multiple KMER values used, *SPAdes* did a much better job of successfully assembling the complete sequence from a wide range of input reads, but even so, it could not assembly the complete set.

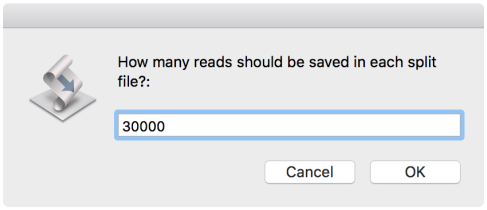
How to Split Fastq Files

MacVector does not have a built-in function to split fastq files. However, over the years we have developed a few utilities to help this. One such utility is "*SplitFastqFile.app*". You can download this from our web site on the macvector.com/Downloads page.

After downloading SplitFastqFile.app.zip, make sure it has been extracted (double-click to extract if it has not) and move *SplitFastqFile.app* to a convenient location e.g. your desktop.

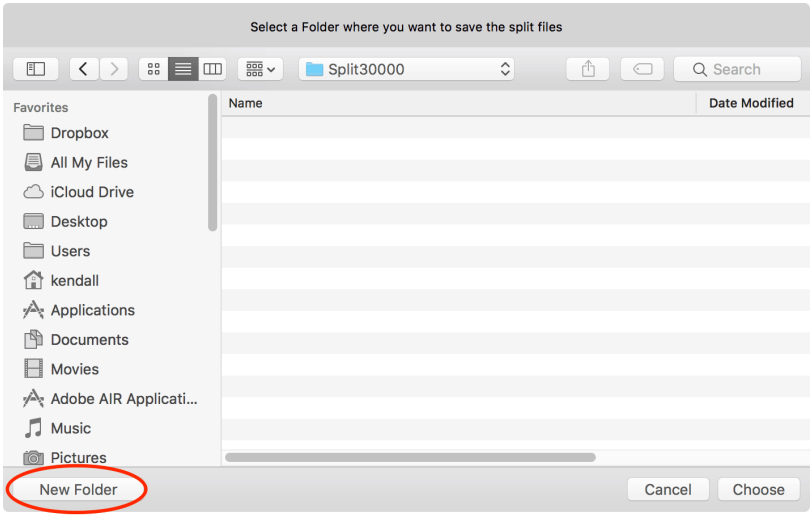
Locate the fastq file(s) you want to split into smaller segments, select them (you can select more than one if you wish), then drag and drop them onto the *SplitFastqFile.app* icon.

You will first be prompted for the number of reads to be saved into each split file;



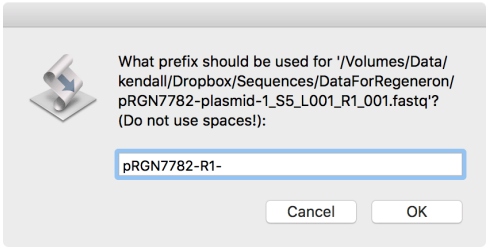
Lets use 30,000 as we will see this is more than enough for this data set and gave the best results in the analysis described above (complete assemblies can be generated with fewer than 10,000 reads from each starting file).

Next you will be prompted for an output folder;



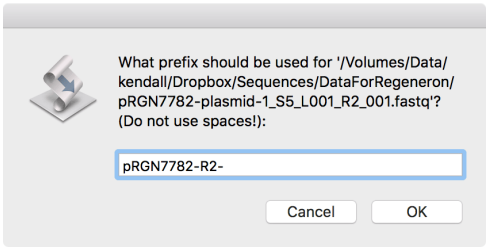
I created a new folder called *Split30000* to save the files into.

Next you will be asked for a "Prefix" for the saved files for the first file (plasmid-1_S5_L001_R1_001.fastq);



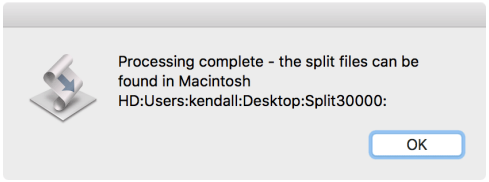
The full name is rather a mouthful so I compacted this to just "plasmid-R1-". The files will be generated with names adding "aaa", "aab", "aac" etc onto the Prefix, so we should get filenames like plasmid-R1-aaa.fastq, plasmid-R1-aab.fastq, plasmid-R1-aac.fastq etc.

When you click OK, your will see a “Job Running” dialog flash in and out of existence. Behind the scenes the file is being quietly and efficiently split. After a few seconds, you will get prompted for a prefix for the second file;

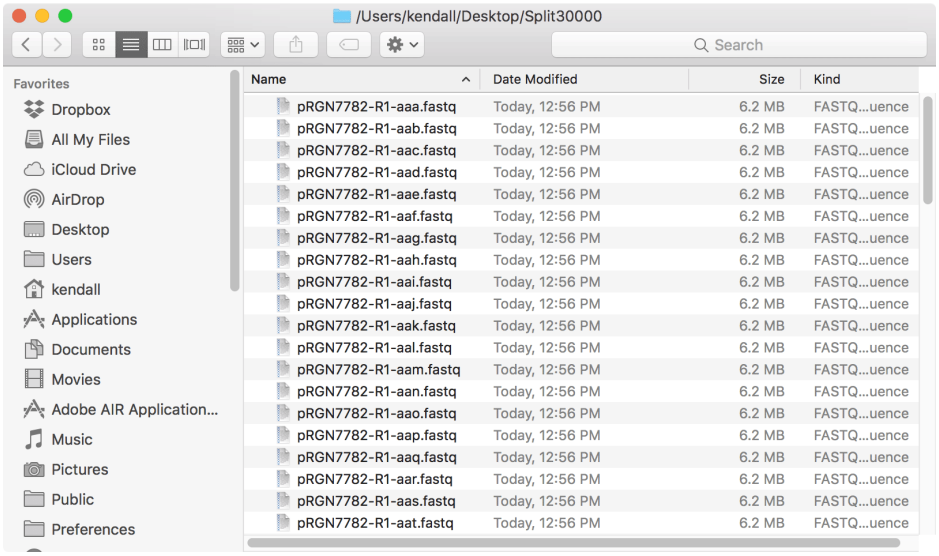


I just named this …R2- to distinguish it from the first.

When the second file has been split you will get a message that the file extensions are being fixed, followed by a message letting you know the processing is complete;



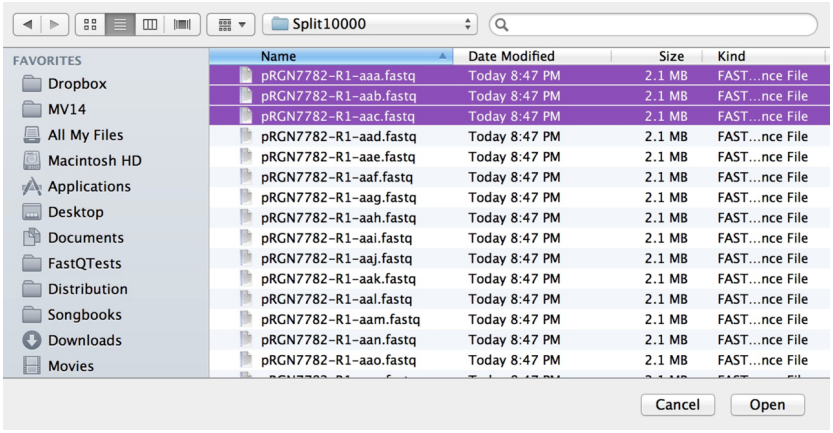
If you look in the output directory you should see all of the split files;



Assembling with Velvet

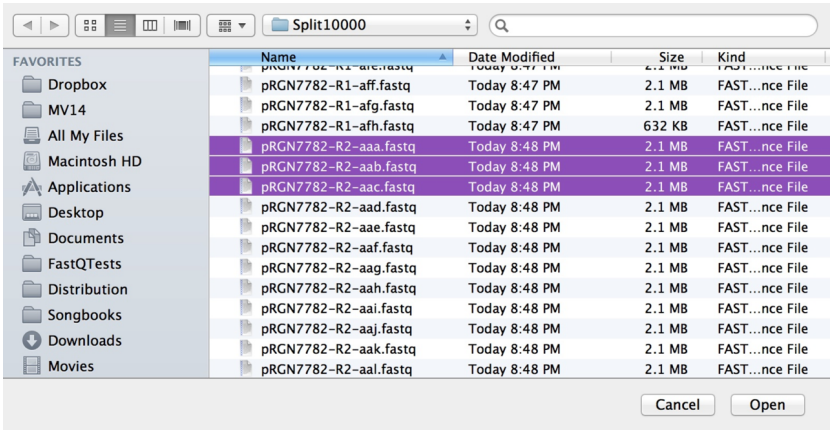
The next phase is to assemble the reads into contigs. We will do this in triplicate – we have lots of split files, so let’s repeat the assembly with three sets of independent data and see if we get the same answer.

Choose **File | New | Assembly Project**. An empty *Assembly Project* window will open. Click on the **Add Seqs** toolbar button and navigate to the folder where you saved the split files;

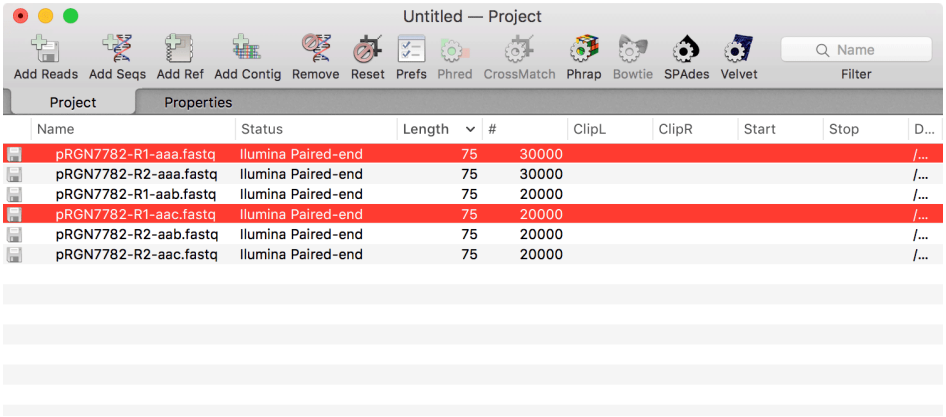


Select the first three “R1” files - …R1-aaa, …R1-aab and …R1-aac and click on the Open button.

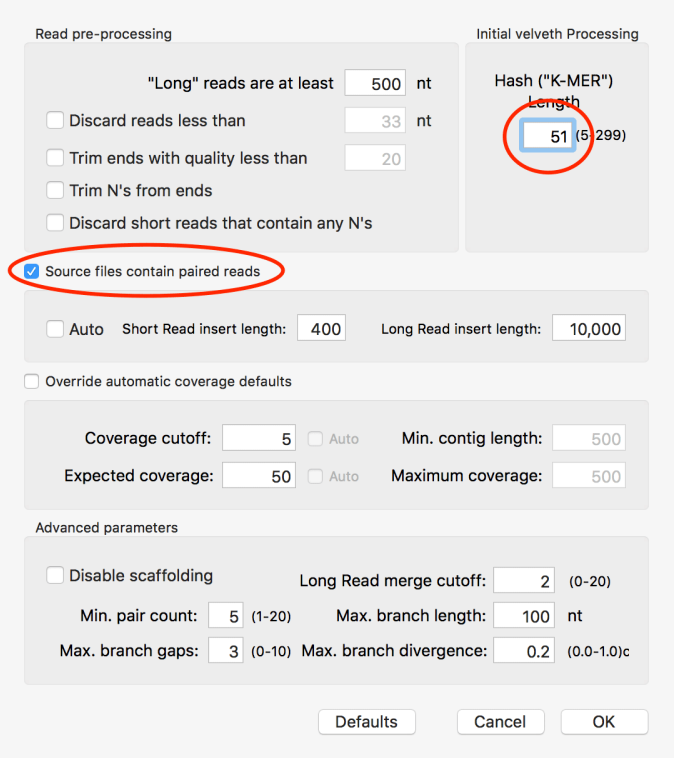
Repeat, but this time, choose the first three “R2” files;



Now select the pair of …aaa.fastq files in the *Assembly Project* window (use the command key to toggle selections on and off) and click on the **Velvet** icon;



A *Velvet* setup dialog appears;

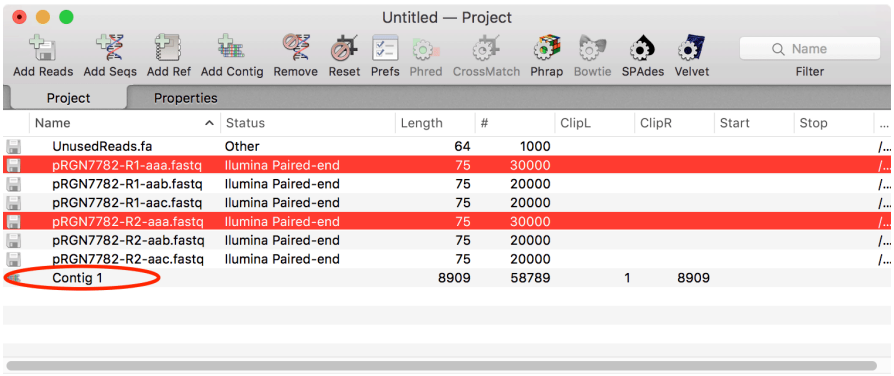


The most critical parameters in this are the **Hash** (K-MER") setting and the checkbox to let *Velvet* know you are using paired-end read files.

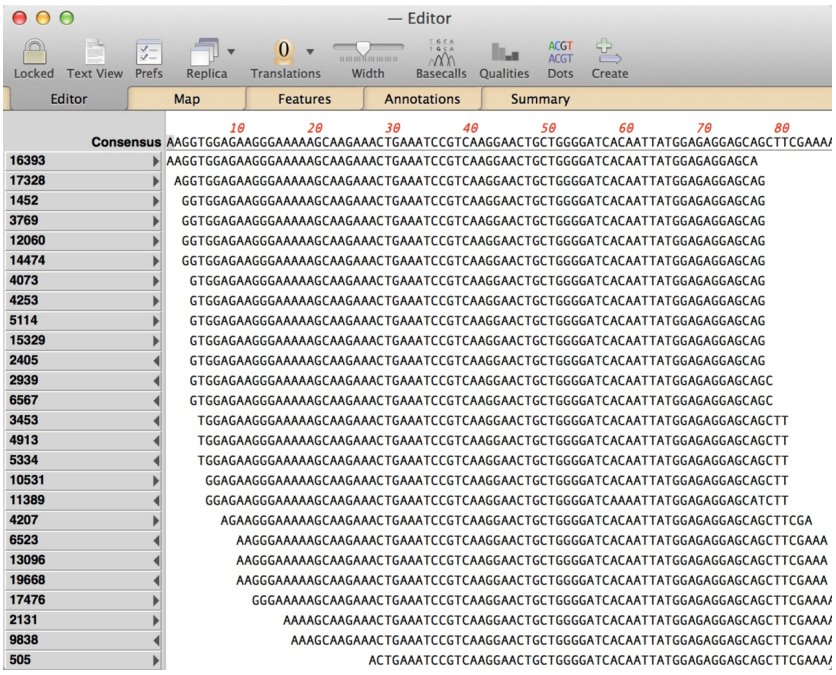
The K-MER value is generally the first parameter you should consider changing. It should have an odd value between 11 and 255. For *Illumina* reads like these, a good starting point is to aim at two-thirds of the read length. So, we will try 51.

The remainder of the parameters we will leave as the default. Click **OK**.

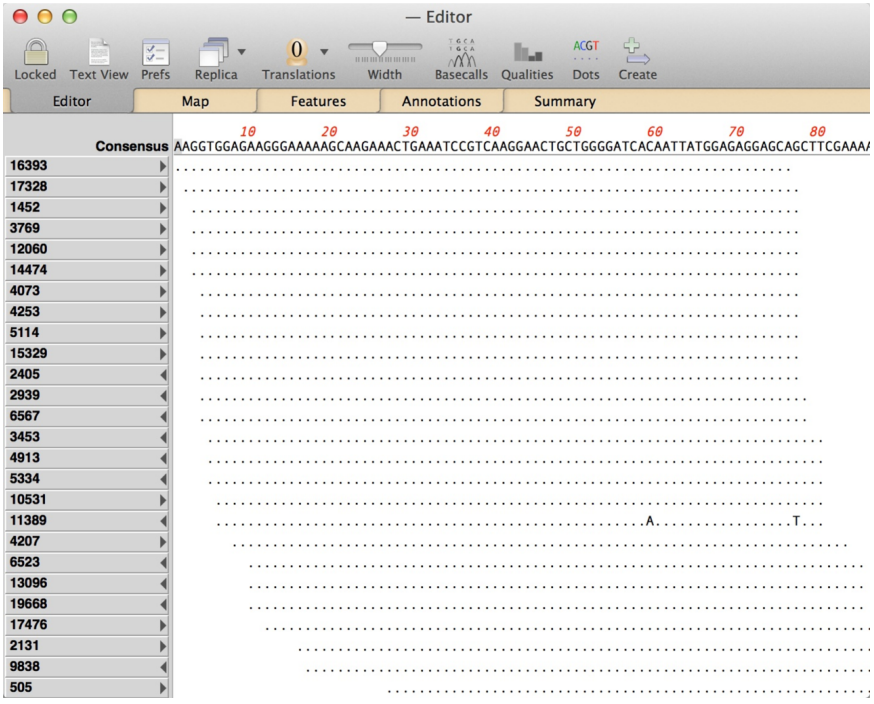
A progress dialog will appear. The entire assembly should take just a few seconds, then a new "*Contig 1*" entry appears in the *Assembly Project* window;



The fact that a single contig appears is a good thing. It is also in the size range we would expect for our plasmid. Double-click on *Contig 1* to open up the *Contig Editor* window;

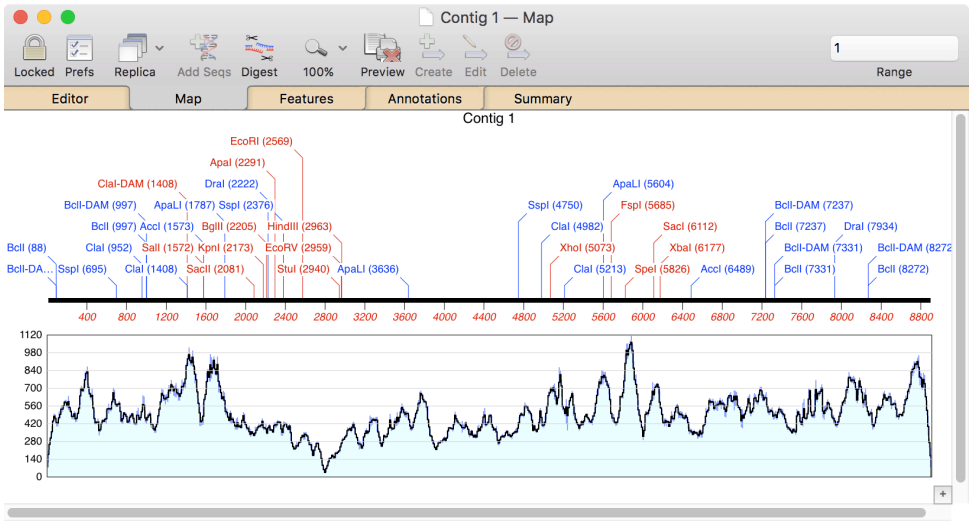


The **Editor** tab shows the aligned reads with the consensus across the top. Note that *Velvet* renames the reads, so unfortunately the names in the left hand margin do not correspond to names in the original fastq file. Unlike ABI contig assemblies, this assembly is NOT editable. The displayed alignment is for informational purposes only. You can click on the **Dots** toolbar item to get a feel for the quality of the alignment;

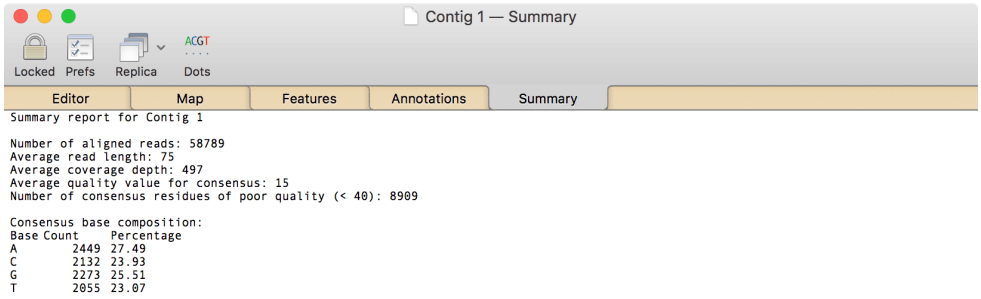


In this case, the alignment is pretty good, with just a couple of mismatched residues in one of the reads.

Perhaps more informative is the **Map** tab;



Here we can see a coverage map of the reads across the assembly. You can see that while the average is around 400x, there are some regions with over 1000x coverage and one region, centered around 2800, that has only about 30x coverage. The **Summary** tab has more information on the coverage;



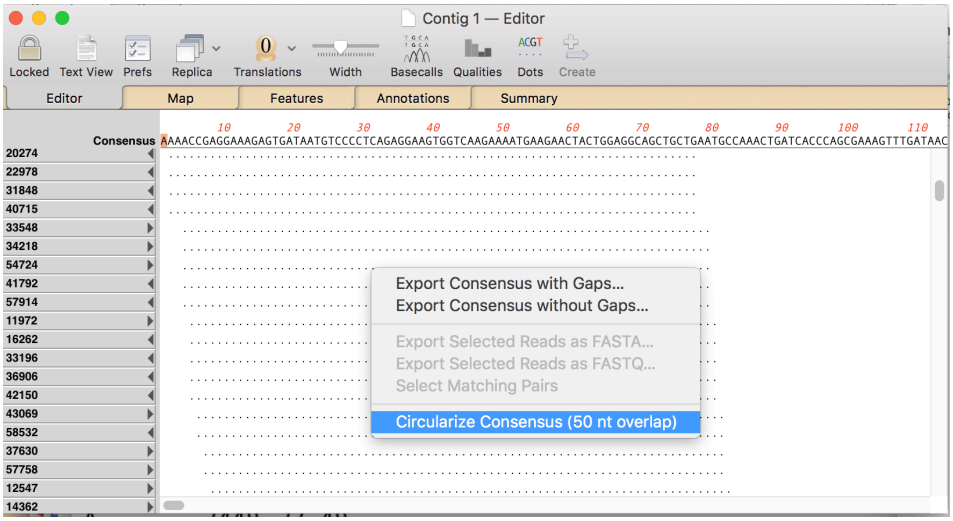
One thing to note about the **Summary** output is that *Velvet* does not assign quality values to the consensus sequence (in contrast to *phrap*, which generates quality values for every base in the consensus). Thus the **Summary** will always report that all of the residues are of poor quality – this can be ignored.

Analyzing the Consensus Sequence

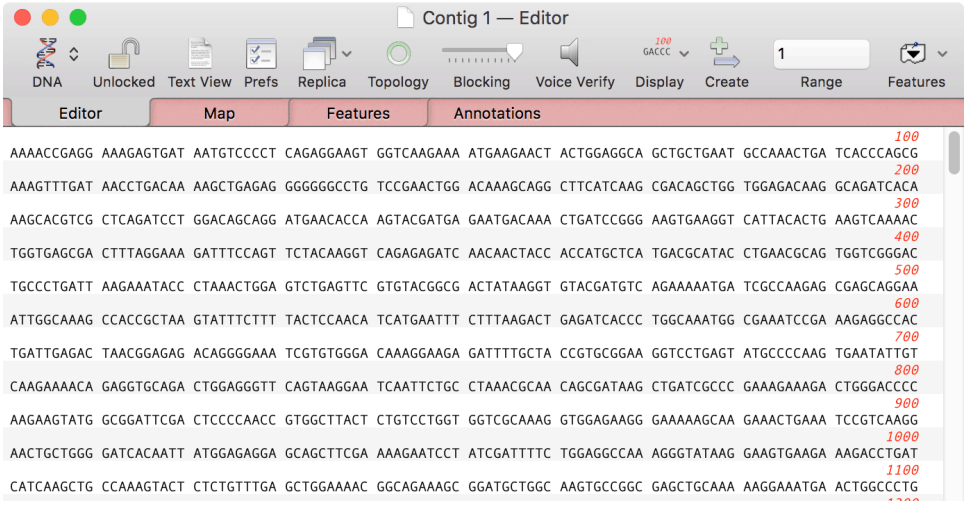
Circularizing Using a Right-Click Menu

In common with other *de novo* assemblers, *Velvet* will not automatically circularize sequences, so we have to do that manually. Luckily, MacVector

provides several functions to simplify this. By far the easiest to use is a new feature added in MacVector 16.0.1. In the **Editor** tab of the *Contig* window, right-click (<ctrl>-click if you don't have a right mouse button) and a popup context-sensitive menu will appear;



The menu item we are interested in here is the one at the bottom. If the contig has direct repeats at the ends, the item will be active and will have text similar to "**Circularize Consensus (50 nt overlap)**", where it will report the actual length of the overlap. This needs to be at least 15 nt. In the absence of direct repeats, the menu item will be disabled and have the text "**Cannot Circularize Consensus**". If you select the menu item, a new circular sequence document window appears.



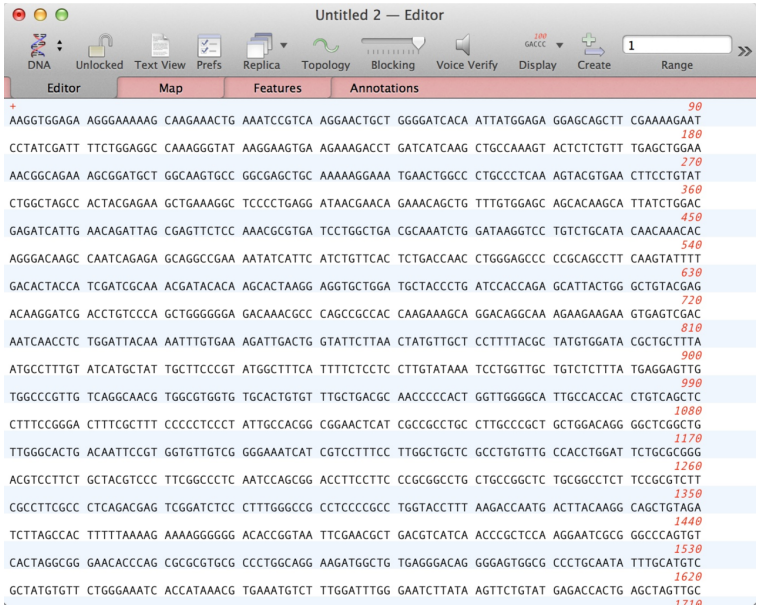
Save this sequence as *plasmid-1.nucl*.

Circularizing Using a Dot-plot Analysis

The right-click menu approach works fine most of the time. But, if you do not have a perfect repeat at the ends of the contig, you may need to manually circularize the molecule. Here's how to do that so you can explore any similarities

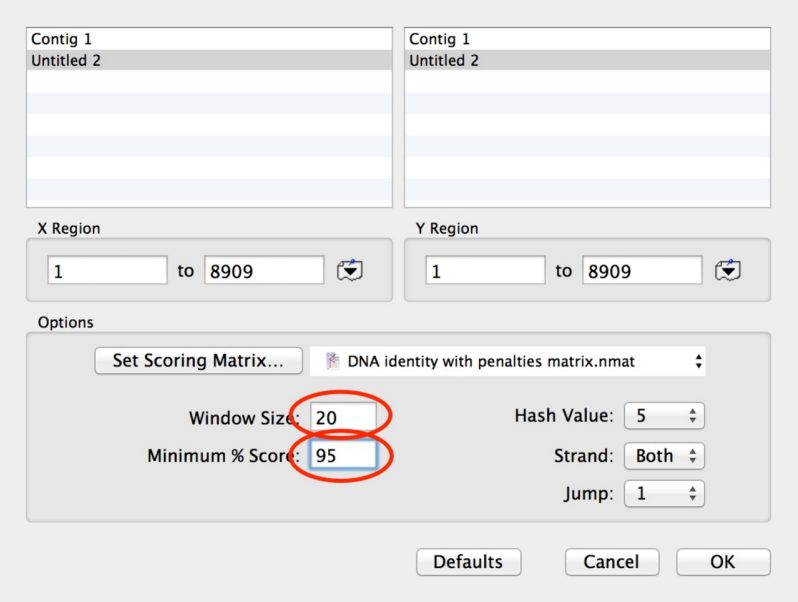
at the ends and potentially spot and fix sequencing errors that might prevent circularization.

First, to simplify the analysis, lets retrieve the consensus sequence from *Contig 1*. There are several ways to do this (e.g. you can **File | Export** as a single MacVector Nucleic Acid sequence), but perhaps the simplest is to use the right-click menu again and select the **Export Consensus Without Gaps**... item;

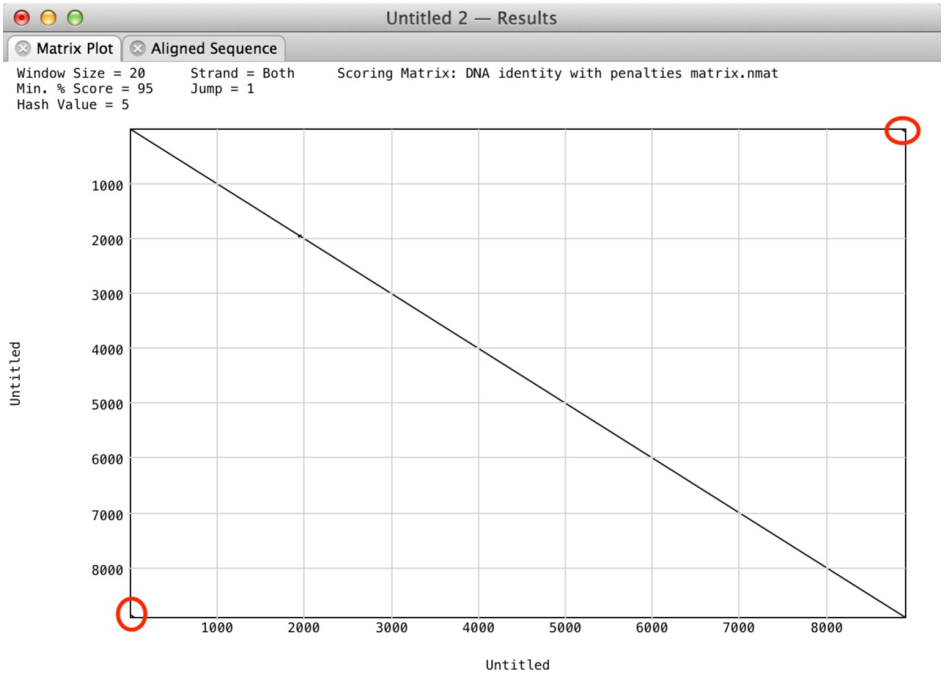


The next step is to circularize the molecule. But first we have to work out if there is an overlap between the beginning and the end of the sequence. One important thing to know about *Velvet* is that when it assembles a circular molecule, it tends to produce duplications at the beginning and end of approximately the value of K-MER. You may remember we used a value of 51 for this, so we are going to expect a duplication of approximately that value. This is just a rough guide – it is never longer than this, but frequently it is shorter, particularly with low coverage assemblies.

One quick way to look for direct repeats in a DNA molecule in MacVector is to use the **Dot Plot** functionality. With the new sequence window active, choose **Analyze | Create Dot Plot | Pustell DNA Matrix**;

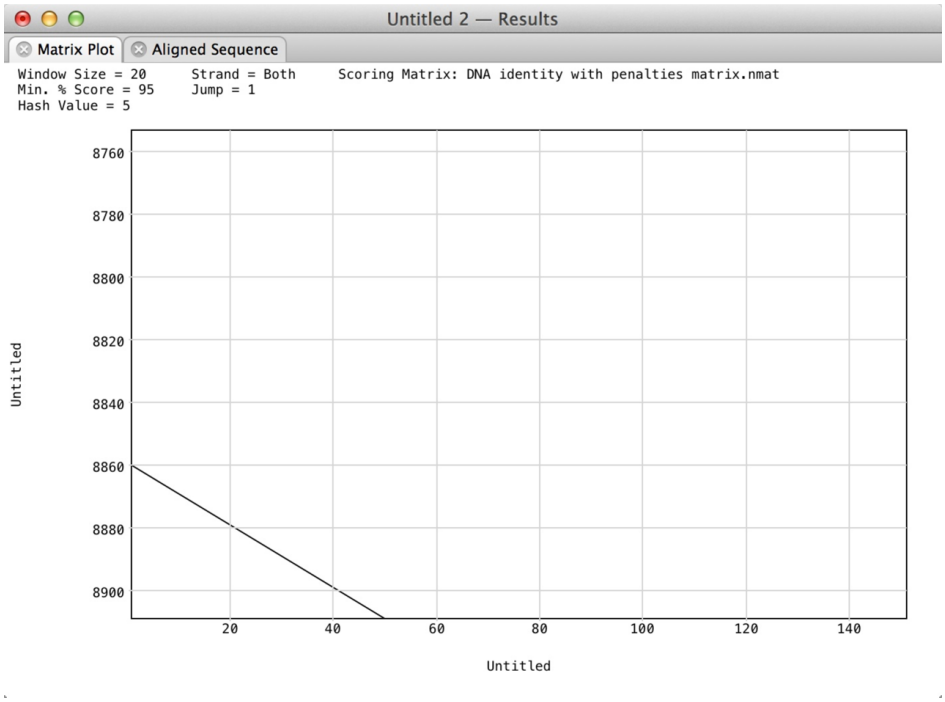


Set the **Window Size** to 20 (or a value that is significantly less than the expected overlap between the ends) and **Minimum % Score** to 95%, as we are expecting essentially identical overlaps at the ends. Make sure you are comparing the new sequence (*Untitled 2* in this case) against itself. Click OK.

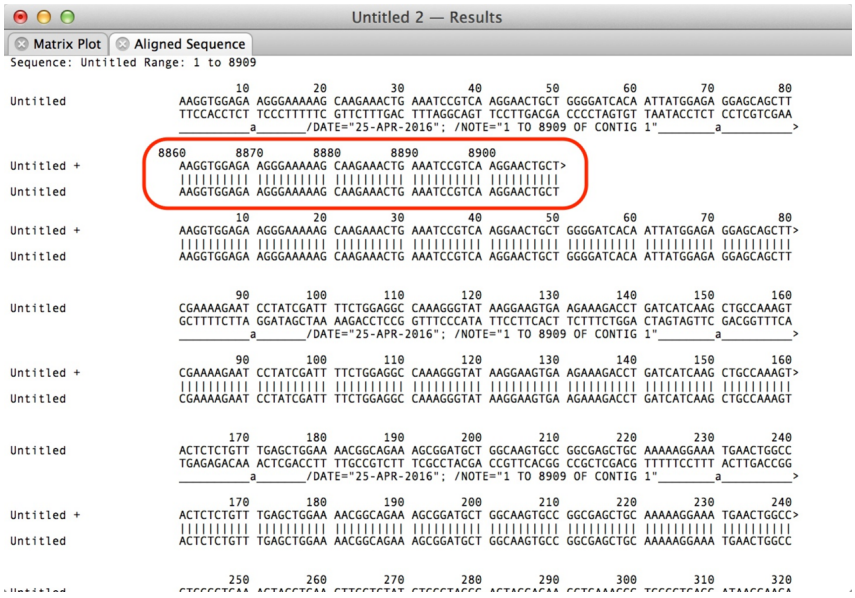


The **Matrix Plot** usually gives a good overview of the similarities between two sequences, or of direct repeats within a sequence. The long black diagonal line indicates the expected end-to-end identity when you compare a sequence to itself. In this case we are hoping to see direct repeats at the ends of the sequence – unfortunately, these are hard to see at the fully zoomed out resolution, though if

you look carefully you can see small “blips” at the circled corners. You can click and drag in the corners to “zoom” into those regions;



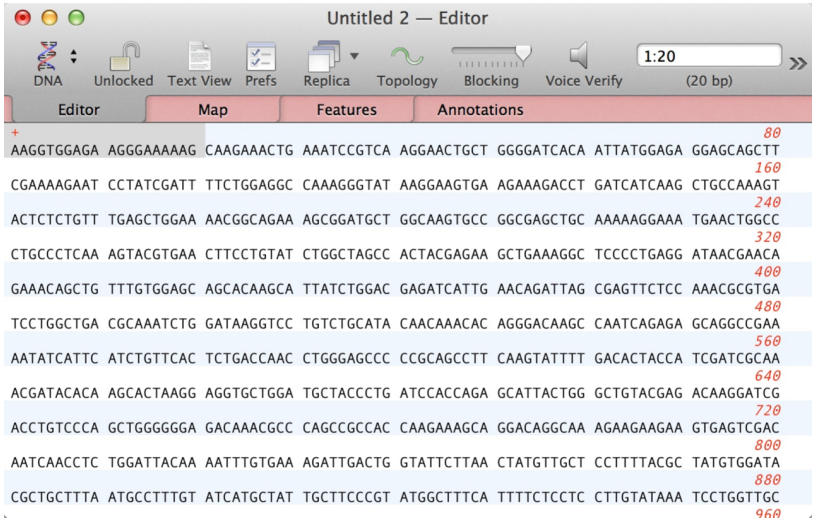
Once zoomed in. the diagonal representing the repeat is much easier to see. Even more clear, is to simply click on the **Aligned Sequence** tab;



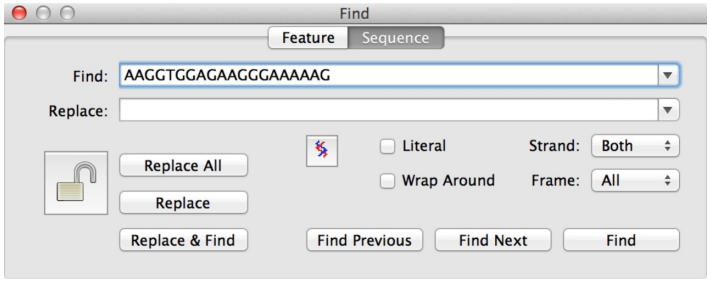
The repeat from the end of the sequence is now clearly shown aligned to the beginning. We can see that it does look to be right about 50nt in length.

So, now we know that we do have the expected repeat, indicating that we have truly sequenced a circular sequence that is “wrapping around, we need to truncate the sequence to remove the duplicated segment. By far the easiest way to do this

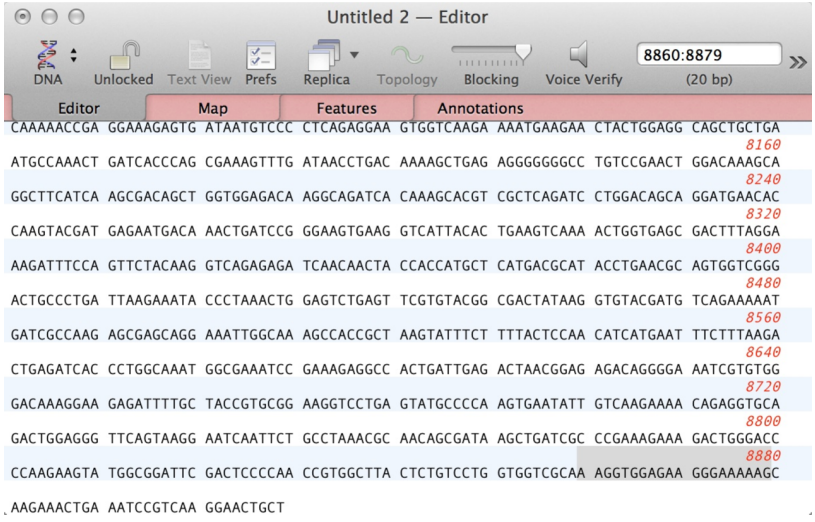
is to use MacVector's **Find** functionality. Close the *Dot Plot* result window and select the first ~20 nt in the sequence. If there was a mismatch in the alignment, you should make sure that residue is not included in the selection.



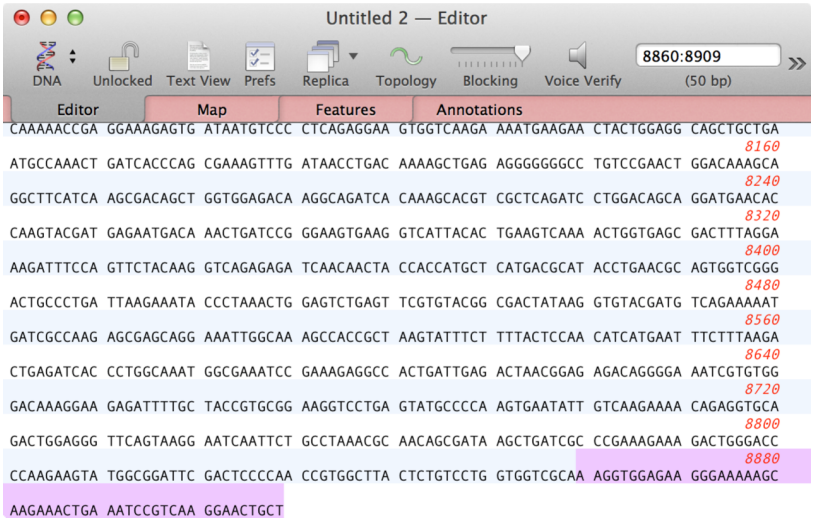
Choose **Edit | Copy** to copy the sequence, then choose **Edit | Find | Find...** Paste the copied sequence into the *Find* edit box;



Click on the **Find** button – it will simply re-select the first ~20nt in the sequence window. Then click **Find Next**. You should see that it finds a match close to the end of the sequence;



This is the start of the duplicated region at the end. To extend the selection to the end of the sequence, hold down the <shift> key and click just beyond the last character;



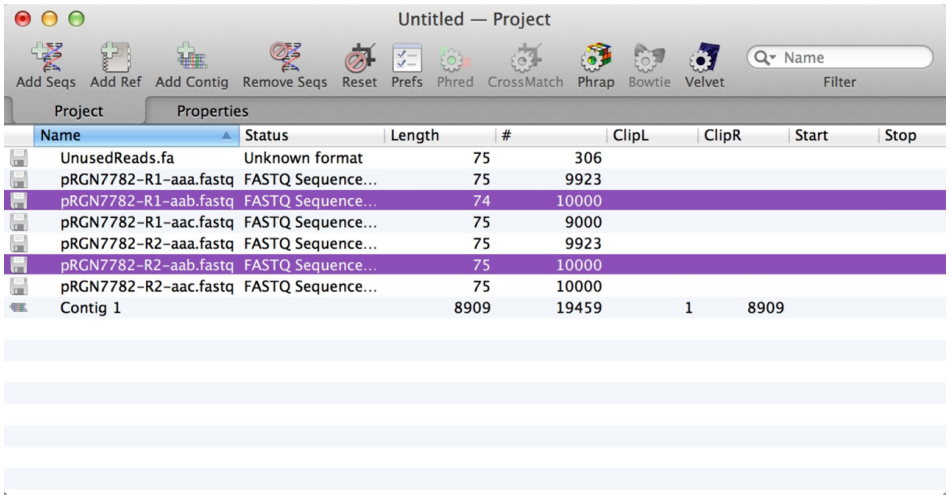
Now press the <delete> key to remove the duplication. Finally, click on the **Topology** button to circularize the sequence. then **File | Save** it under a suitable name (e.g. *pRGN7782-1*). We have the sequence of our plasmid!

Of course, if you had a mismatch in the repeat region, you would now need to confirm the sequence at the ends. We will look at some ideas on how to do that later on.

Repeating the Assembly

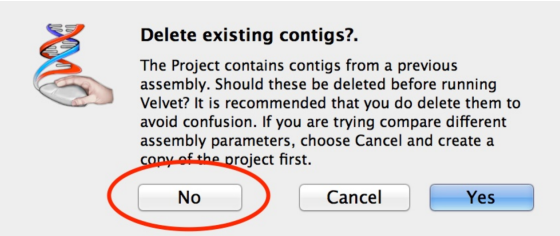
Lets double-check that the sequence is correct. We imported 3 pairs of files into our original *Assembly Project*. So we can repeat the assembly using the aab and aac pairs.

Select the aab pair of files and click the Velvet button;



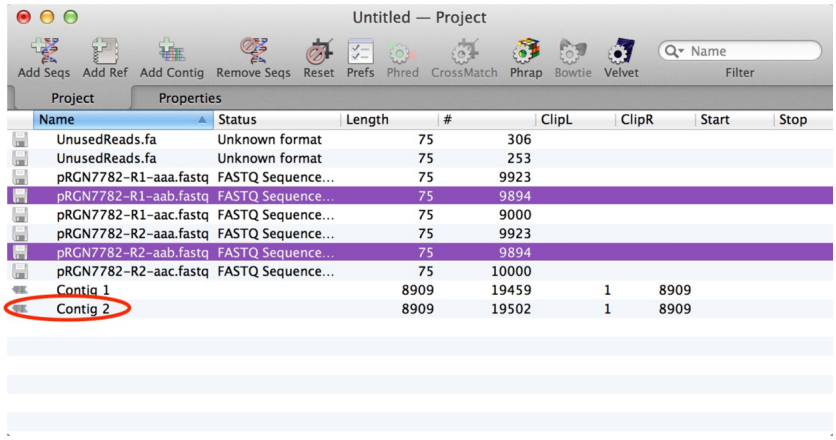
Name	Status	Length	#	ClipL	ClipR	Start	Stop
UnusedReads.fa	Unknown format	75	306				
pRGN7782-R1-aaa.fastq	FASTQ Sequence...	75	9923				
pRGN7782-R1-aab.fastq	FASTQ Sequence...	74	10000				
pRGN7782-R1-aac.fastq	FASTQ Sequence...	75	9000				
pRGN7782-R2-aaa.fastq	FASTQ Sequence...	75	9923				
pRGN7782-R2-aab.fastq	FASTQ Sequence...	75	10000				
pRGN7782-R2-aac.fastq	FASTQ Sequence...	75	10000				
Contig 1		8909	19459	1	8909		

You will get a warning dialog about contigs already having been created;



In this case, we know we are assembling a different pair of files, so we can safely click the **No** button as we don't want to delete *Contig 1*.

After assembly, we see a new *Contig 2* item.



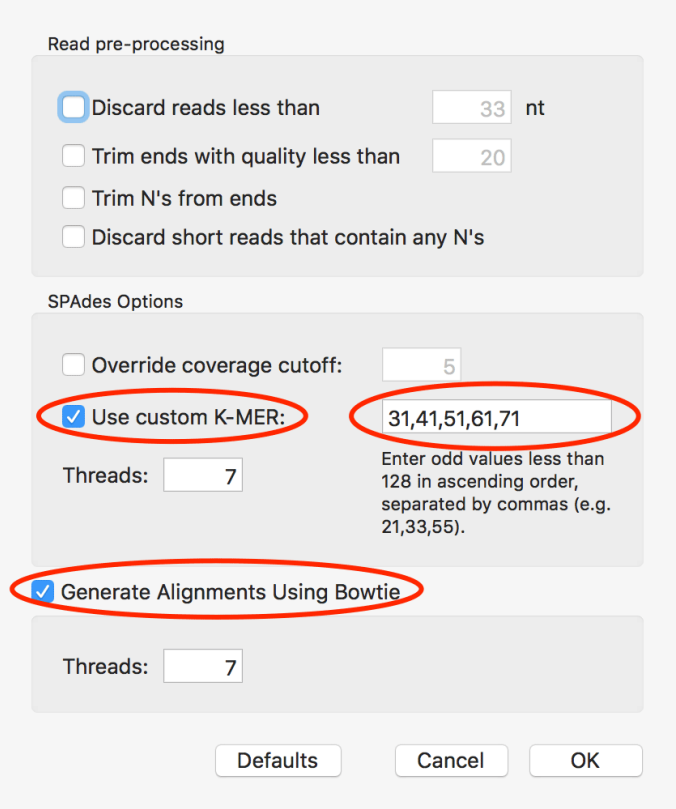
Repeat the previous steps to circularize the sequence and save it as *pRGN7782-2*.

Assembly using SPAdes

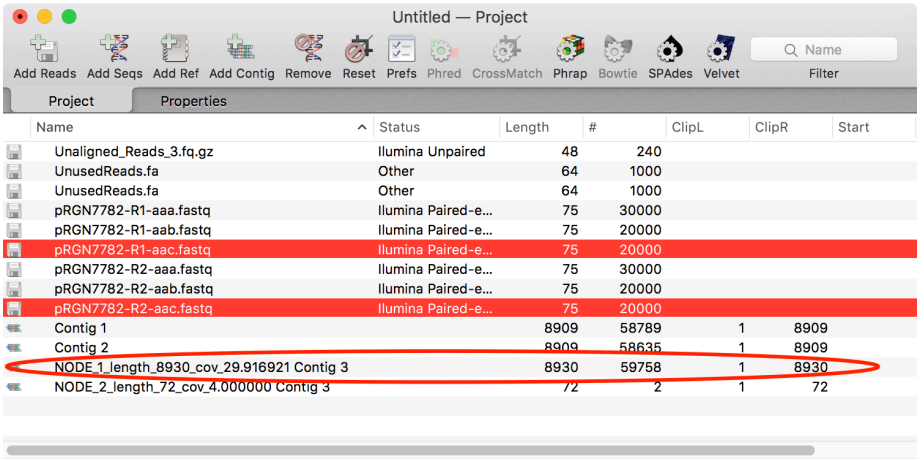
For the third set of files we will try using the *SPAdes* assembler. Select the pair of *…aac.fastq* files and click on the **SPAdes** toolbar button.

Note that we are going to use a custom K-MER – type the values 31,41,51,61,71 into the box. The values must be odd and must be separated by commas. You must also provide at least 3 values. You can add odd values up to 127.

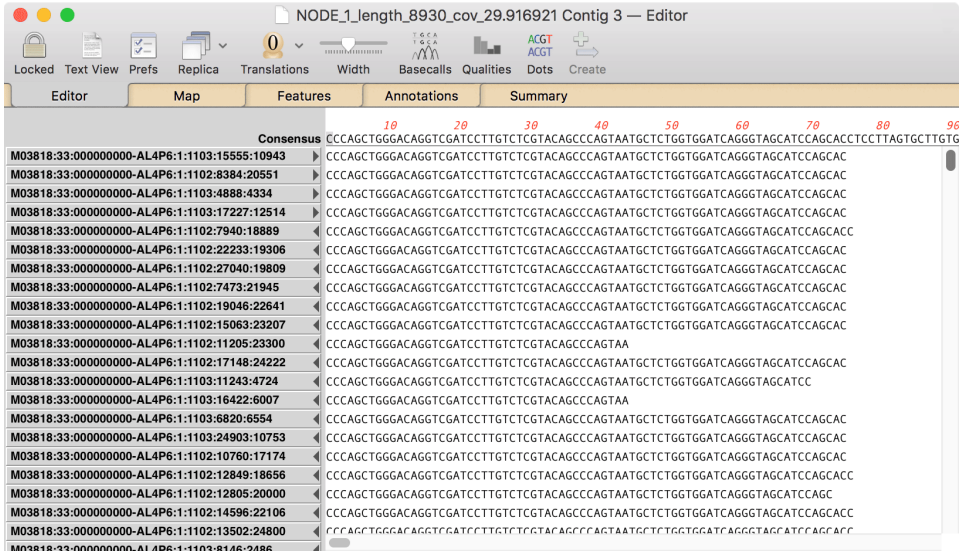
Unlike *Velvet*, *SPAdes* does not create alignments, it merely generates a list of consensus sequences. If you are interested in looking at the actual alignments, then MacVector offers the option of running a *Bowtie* alignment on each of the consensus sequences. This is very useful for visualization, so we will select that option too.



With such a small data set, *SPAdes* will still complete in just a few seconds. However, with larger genomic data sets, this can take several hours.



Note that *SPAdes* uses a different naming convention for the contigs, calling them "*NODE_x_length_<len>_cov_<coverage>*" which MacVector honors in the display. Again, you can double-click on the longest contig to open up the *Contig Editor* window. Note that because *Bowtie* retains the read names, you can see the original read names in the assembly.



Again, you can right-click in the **Editor** tab to see if the contig can be circularized. In this case we have a 71 nt overlap. Create the new circular sequence and save it as *pRGN7782-3*.

Confirming the Correct Sequence using Three Contigs

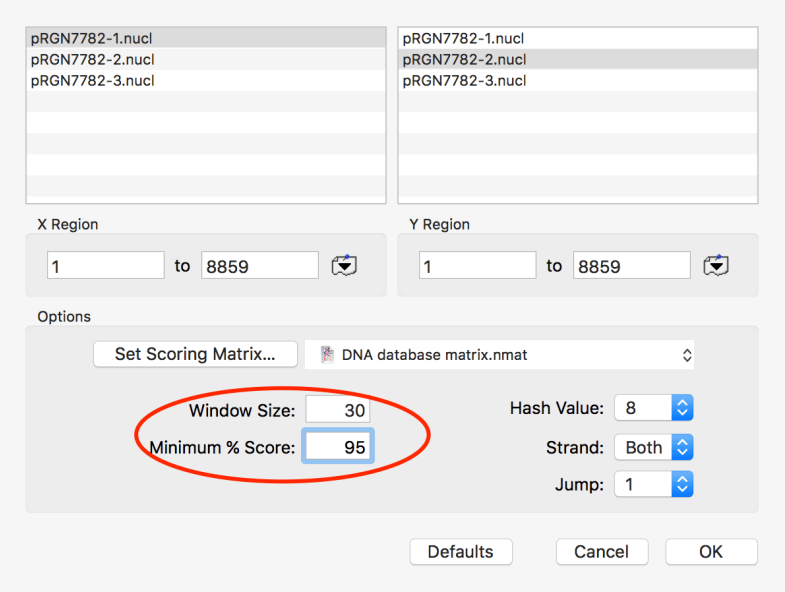
So, now we have our three circular contig sequences, each generated from a different set of input reads. However, before we can compare them to make sure they are identical, there are two additional factors to consider;

(a) **Orientation** – contigs can be created in one of two orientations. We need to make sure that all three are in the same orientation before we can do the comparison.

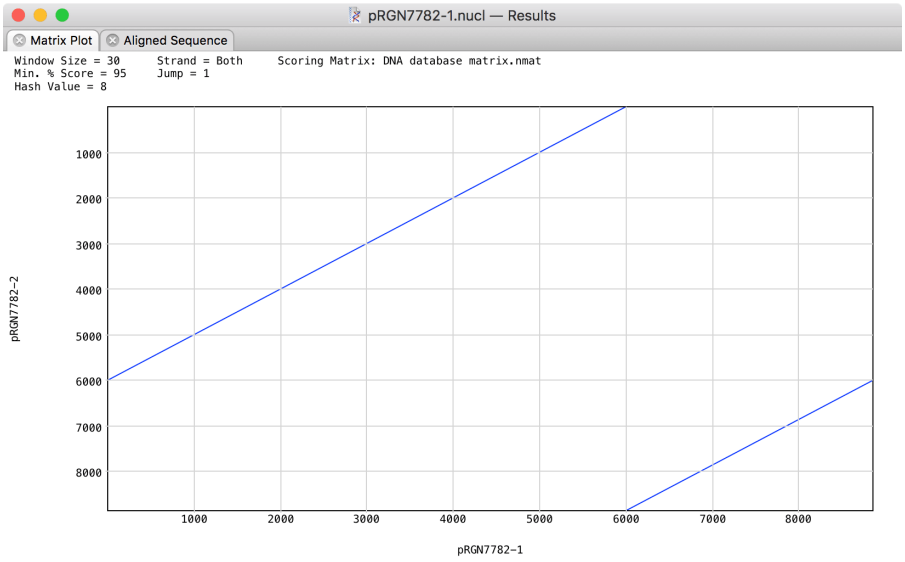
(b) **Circular Origin**. - the assemblers essentially choose a random location for the start of the assembly. Thus, when we circularize, the contigs are likely to be "split" in a different location. We need a way to ensure they are all split at the same point.

Determining Orientation

The easiest way to determine orientation is to use MacVector's dotplot analysis function. Close all open windows except for the three circular plasmids, *plasmid-1*, *-2* and *-3*. Choose **Analyze | Create Dot Plot | Pustell DNA Matrix**, select *plasmid-1* in the left column and *plasmid-2* in the right. Set the **Minimum % Score** to 95;

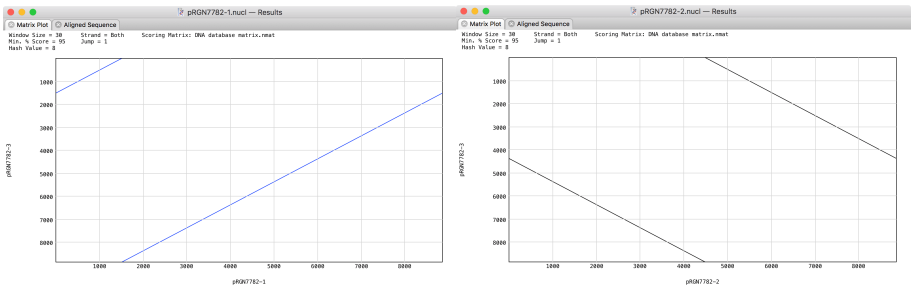


Run the analysis and take a look at the **Matrix Plot** result tab;



In this plot (your results may vary) there are two diagonal lines of identity. Because the plots are blue and travel from lower left to upper right, this indicates that this is an inverted match i.e. the two contigs are in different orientations. The fact that there are two lines, indicates that they have different circular origins.

Repeat the dot plot with the other two combinations – 1 vs 3 and 2 vs 3;

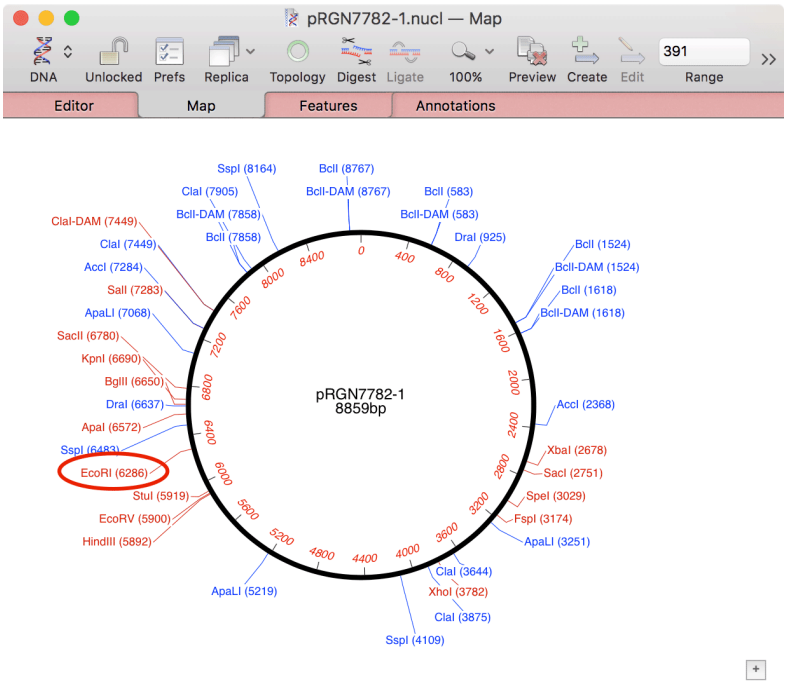


In this case, 1 vs 3 (left) indicates that -1 and -3 are also in the opposite orientation, but 2 vs 3 (right) are in the same orientation as indicated by the black plots travelling from top left to bottom right.

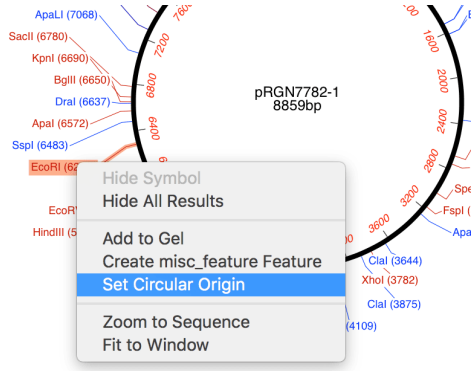
The easiest thing to do is "flip" the orientation of *plasmid-1* so that it matches the other two. To do this, simply bring that plasmid to the front, use **Edit | Select All** to select the entire sequence, then choose **Edit | Reverse & Complement** to "flip the sequence. Don't forget to save it!

Changing Circular Origin

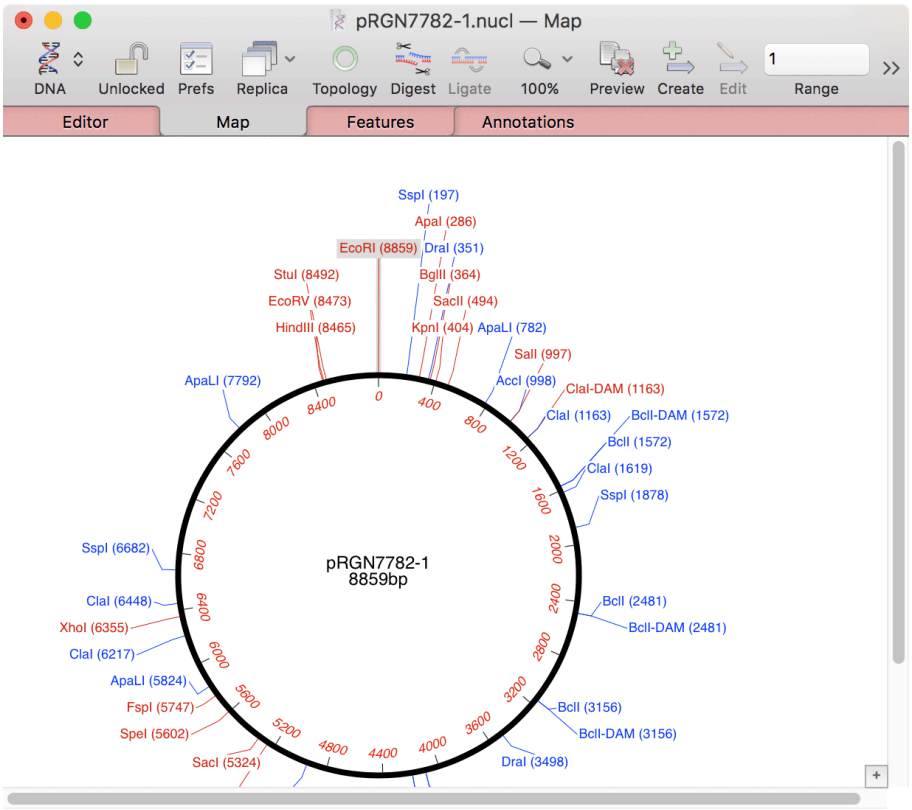
If we look at the **Map** tab of *pRGN7782-1*, the unique restriction enzymes are easily identifiable as they appear in red;



In this case, there is a unique *EcoRI* site at (6286). Let's make this the new circular origin. Click on the site to select it, then right-click (or <ctrl>-click) to view the context-sensitive menu;



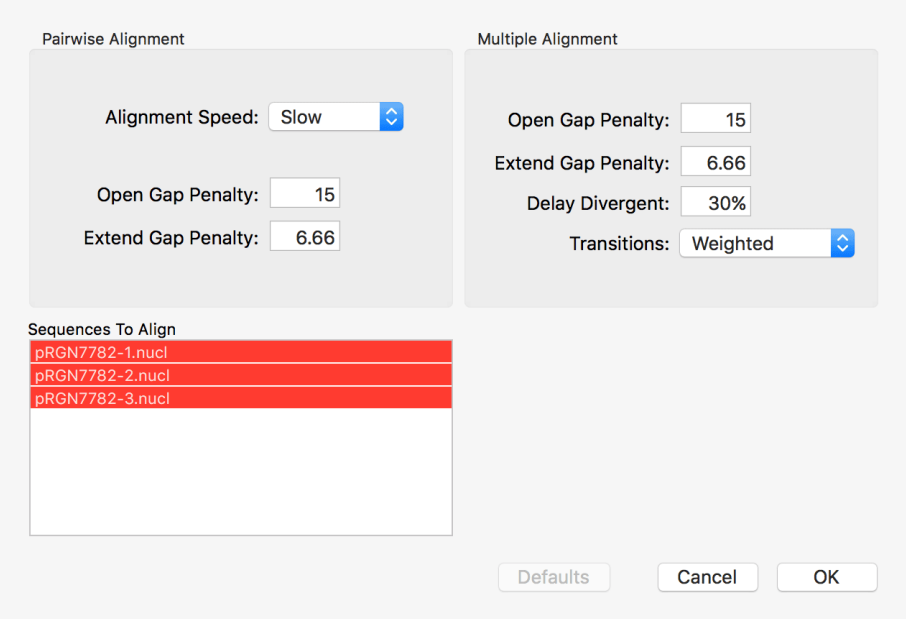
Choose the **Set Circular Origin** item and the **Map** refreshes to show the plasmid rotated with the *EcoRI* site now at the 12 o'clock position.



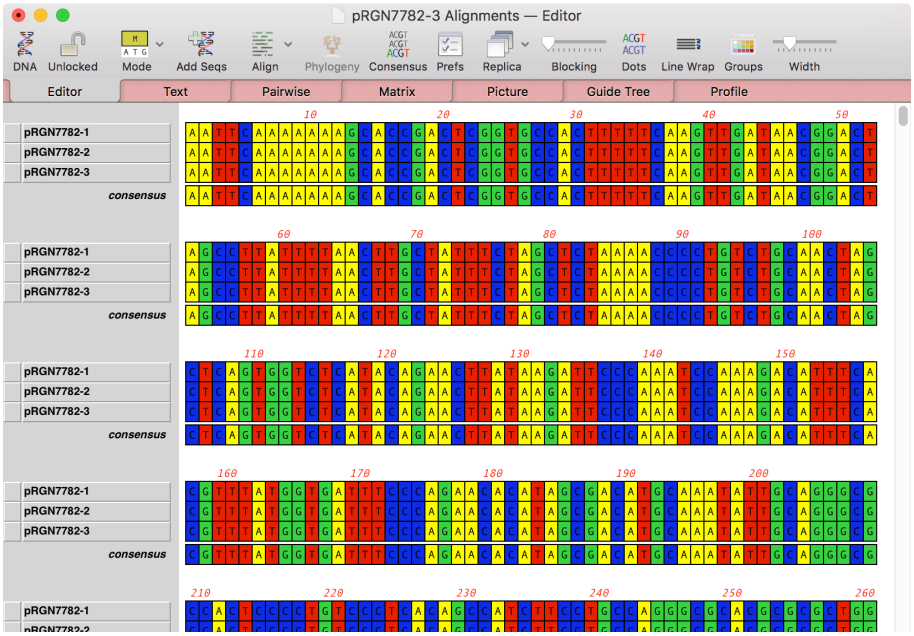
Save the file and repeat with the *EcoRI* sites in *pRGN7782-2* and *pRGN7782-3*.

Confirm Identity using Multiple Sequence Alignment

With the three rotated plasmid sequences open, choose **Analyze | Align Multiple Sequences Using | ClustalW**;



Make sure you have all three plasmids selected, then click **OK**. After the alignment has completed you should have a window like this;



You can scroll through the alignment if you wish, looking for any mismatches, but the most useful tabs from this perspective are the **Pairwise** tab and the **Matrix** tab. The **Pairwise** tab shows each sequence aligned with each other sequence. The header for each alignment lists the gaps and identities between the pair of sequences;

```
1. pRGN7782-1 vs. pRGN7782-2

   Aligned Length = 8859   Gaps = 0
   Identities = 8859 (100%)

pRGN7782-1      1 AATTCAAAAAAAGCACCGACTCGGTGCCACTTTTTCAAGTTGATAACGGACTAGCCTTAT   60
pRGN7782-2      1 AATTCAAAAAAAGCACCGACTCGGTGCCACTTTTTCAAGTTGATAACGGACTAGCCTTAT   60
                  ************************************************************

pRGN7782-1     61 TTTAACTTGCTATTTCTAGCTCTAAAACCCCTGTCTGCAACTAGCTCAGTGGTCTCATAC  120
pRGN7782-2     61 TTTAACTTGCTATTTCTAGCTCTAAAACCCCTGTCTGCAACTAGCTCAGTGGTCTCATAC  120
                  ************************************************************

pRGN7782-1    121 AGAACTTATAAGATTCCCAAATCCAAAGACATTTCACGTTTATGGTGATTTCCCAGAACA  180
```

Here it's clear that *plasmid-1* and *plasmid-2* are identical with no gaps.

The **Matrix** tab simple summarizes the combinations of pairwise alignments;

```
ClustalW multiple sequence alignment

3 Sequences Aligned           Processing time: 9.3 seconds
Gaps Inserted = 0             Conserved Identities = 8859
Score = 53045

Pairwise Alignment Mode: Slow
Pairwise Alignment Parameters:
    Open Gap Penalty = 15.0   Extend Gap Penalty = 6.7

Multiple Alignment Parameters:
    Open Gap Penalty = 15.0   Extend Gap Penalty = 6.7
    Delay Divergent = 30%     Transitions: Weighted

          ** Identity Scores (%) **
              pRGN778   pRGN778   pRGN778
              2-1       2-2       2-3
pRGN7782-1     100.0     100.0     100.0
pRGN7782-2     100.0     100.0     100.0
pRGN7782-3     100.0     100.0     100.0
           ** Similarity Scores (%) **

**Similarity Scores(s) are shown below the diagonal (x) with Identity Scores(I) above**
  a b c d e
a x i i i i
b s x i i i
c s s x i i
d s s s x i
e s s s s x
```

Here we can clearly see that all three plasmids share 100% identity with each other. With this confirmation of the sequence from three different sets of reads, we can be extremely confident that we have accurately determined the sequence of our plasmid.