**CRASS Protocol April 2022**

*Made by Josué Rodríguez-Ramos and Rebecca Daly*

Step 1: Run Crass

* Crass runs on the server and takes as input either a) a combined .fasta or .fastq file of all your reads or b) a forward and reverse set of reads.
* It does not matter which you do, as Crass does not use the “paired” information.

**Commands:**

1. If using forward / reverse trimmed reads:

Feed it a loop with a list of your reads and run the following:

for element in $(<$1)

do

mkdir "$element"\_crass

cd "$element"\_crass

crass "$element"\_R1.fastq "$element"\_R2.fastq

1. If using combined file of R1/R2:

crass all\_trimmed\_reads.fa

Step 2: Extract information from CRASS output.

* *Crass output breakdown*: The output from CRASS will be several hundred “.fa” files that will contain the Group ID, and spacer sequence which should be somewhere around 30-40bp in the file name. Inside each file, are the reads and read ID’s that contain hits to that spacer. There are other several hundred .gv files, which can be visualized using graphviz, but in the beginning it’s not too useful (depends on your ecosystem) so will ignore for now. Crass will also output a file called crass.crispr that contains the direct repeat sequence for each group, and the spacer sequences near the direct repeats.
* To extract the information we need, we use a custom script made by Josué (crasspipeliner.sh) that uses crisprtools and a Rich script.

Josué’s script does the following:

First, it pulls out the group ID’s from the file names. This will get you a comma separated list that crisprtools needs to then summarize. Then it splits up these files into separate files with up to 20 different ID’s. Crisprtools extract is incredibly strange and only allows for ~25-30 ID’s to be extracted at a time. After prepping the files, it runs crisprtools to extract the info from the crass.crispr file based on the ID’s the script extracted. It takes in a comma separated list that gets made by cr\_asspipeliner.sh from the above file in integer form. The final output will be a concatenated crispr spacer file and concatenated direct repeats file for that specific sample.

Crasspipeliner.sh is below:

#!/bin/bash

#Loop made by Josué Rodríguez-Ramos. Instructions: Simply run using a bash call in the directory that CRASS output can be found.

ls \*spacers.gv | awk -F "\_" '{ print $2}' > splitfiles.txt #Make gv files into a line separated list.

mkdir crass\_pipeliner

cd crass\_pipeliner

split -l 20 ../splitfiles.txt split\_ #Split comma separated list into individual files of 20 spacers each.

ls split\_\* > split\_file\_paths.txt #Make the directory for final sed commands.

while read x; do

sed '$!s/$/,/' "$x" > "$x"\_comma

sed '{:q;N;s/\n//g;t q}' "$x"\_comma > "$x"\_comma\_list

done <split\_file\_paths.txt

cat \*\_comma\_list > comma\_list\_concatenated.txt #make a concatenated file of each of the split files where each file content is a single line. Then we feed each of these lines as the argument for the crisprtools stat command.

#run the crisprtool stats

while read j; do

crisprtools stat -g "$j" -apH ../crass.crispr > crisprtools\_"$j"\_stats\_sampleID.out

done <comma\_list\_concatenated.txt

#now run the crisprtools extract

while read j; do

crisprtools extract -o "crisprtools\_extract" -g "$j" -s -xC -d -f ../crass.crispr

cat crisprtools\_extract/\*\_direct\_repeats.fa > All\_direct\_repeats.fa

cat crisprtools\_extract/\*\_spacers.fa > All\_spacers.fa

done <comma\_list\_concatenated.txt

After you run crasspipeliner.sh on all your CRASS outputs, concatenate all the direct repeats and all the spacers that you found. Now, we need to blast to our MAG and vMAG databases. You only need to do this once, to the file of concatenated direct repeats from all samples and concatenated spacers from all samples. If you have many samples, just run the crasspipeliner.sh as a loop and iterate across your crass outputs. To concatenate all my files, I made a simple copy loop:

#!/bin/bash

#Pull out each set of reads and rename them and move them to crass output folder while renaming them to prevent multiple IDs.

for element in $(<$1)

do

cp /home/projects-wrighton-2/GROWdb/WHONDRS\_2018/metaG/2018\_rivers/erpe/erpe\_JRR/surface\_and\_pore\_viral\_db\_analyses/crass/"$element"/crass\_pipeliner/All\_direct\_repeats.fa /home/projects-wrighton-2/GROWdb/WHONDRS\_2018/metaG/2018\_rivers/erpe/erpe\_JRR/surface\_and\_pore\_viral\_db\_analyses/crass/all\_crass\_output

mv /home/projects-wrighton-2/GROWdb/WHONDRS\_2018/metaG/2018\_rivers/erpe/erpe\_JRR/surface\_and\_pore\_viral\_db\_analyses/crass/all\_crass\_output/All\_direct\_repeats.fa /home/projects-wrighton-2/GROWdb/WHONDRS\_2018/metaG/2018\_rivers/erpe/erpe\_JRR/surface\_and\_pore\_viral\_db\_analyses/crass/all\_crass\_output/"$element"\_All\_direct\_repeats.fa

sed "s/>/>"$element"\_/g" "$element"\_All\_direct\_repeats.fa > renamed\_"$element"\_direct\_repeats.fa

cp /home/projects-wrighton-2/GROWdb/WHONDRS\_2018/metaG/2018\_rivers/erpe/erpe\_JRR/surface\_and\_pore\_viral\_db\_analyses/crass/"$element"/crass\_pipeliner/All\_spacers.fa /home/projects-wrighton-2/GROWdb/WHONDRS\_2018/metaG/2018\_rivers/erpe/erpe\_JRR/surface\_and\_pore\_viral\_db\_analyses/crass/all\_crass\_output

mv /home/projects-wrighton-2/GROWdb/WHONDRS\_2018/metaG/2018\_rivers/erpe/erpe\_JRR/surface\_and\_pore\_viral\_db\_analyses/crass/all\_crass\_output/All\_spacers.fa /home/projects-wrighton-2/GROWdb/WHONDRS\_2018/metaG/2018\_rivers/erpe/erpe\_JRR/surface\_and\_pore\_viral\_db\_analyses/crass/all\_crass\_output/"$element"\_All\_spacers.fa

sed "s/>/>"$element"\_/g" "$element"\_All\_spacers.fa > renamed\_"$element"\_all\_spacers.fa

done

Now we need to make sure that the direct repeats are not the same sequence! You can import them into Geneious and then remove duplicates. Make sure the IDs are also not insanely long – as can have issues. Unfortunately, if you have too many DR’s, Geneious will yell at you and cannot dereplicate. You can however do this using a mmseqs command on the server at 100% ID and 100% sequence coverage (I had to do this for Erpe):

mmseqs easy-linclust ./all\_erpe\_repeats\_renamed.fa 100per\_100cov\_clust tmp --min-seq-id 1 -c 1 --alignment-mode 3 --max-seqs 30000 --threads 20

I set 30,000 max because I have 28,400 sequences and want to make sure we don’t leave any out. You should check you’re not above this number. Min seq id of 100%. -c 1 refers to 100% coverage of sequence. Alignment mode 3 is defined as the number of identical aligned residues divided by the number of aligned columns including internal gap columns, or, by default, defined by a highly correlated measure, the equivalent similarity score of the local alignment (including gap penalties) divided by the maximum of the lengths of the two locally aligned sequence segments.

Now that we have the concatenated crispr spacers, direct repeats, and our vMAG and MAG scaffolds, we can blast. Note, you can also do this to unbinned scaffolds or even whole assemblies. But, it is best to run on MAGs/vMAGs as that is what can be used to get ecologically meaningful interactions.

Starting with direct repeats.

First – make sure that all your MAG scaffolds are unique / identifiable. I.e., make sure that if you want to find Bin.X in your concatenated MAG scaffold file, all the scaffolds within genome Bin.X are named something along the lines of Bin.X\_scaffold\_X. Otherwise, you won’t be able to distinguish towards the end!

Second - make blast db of MAG scaffolds. This will generate 3 files: all\_erpe\_125\_MAGs\_no2019\_renamed.fasta.nhr, all\_erpe\_125\_MAGs\_no2019\_renamed.fasta.nin, all\_erpe\_125\_MAGs\_no2019\_renamed.fasta.nsq

makeblastdb -in all\_erpe\_125\_MAGs\_no2019\_renamed.fasta -dbtype nucl

Then blast the direct repeats to the MAGs:

blastn -db all\_erpe\_125\_MAGs\_no2019\_renamed.fasta -query erpe\_repeats\_100per\_100cov\_clust\_rep\_seq.fasta -out DR\_to\_erpe\_MAGs\_blastn -outfmt 6 -num\_threads 10 -evalue 1e-8

Can also use this outfmt to tell it to give you % query coverage.

-outfmt "6 qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore qcovs qcovhsp"

From this output, we need to filter it. For DRs, we only take the blast hit if it is 100% “pident”, 0 “mismatch”, and 0 “gapopen” on top of the 1e-8 evalue we used in the command. Filter out the columns from the outfmt 6 based on that. Reminder, header format for outfmt 6 is:

qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore

I did this in excel and made file: DR\_to\_erpe\_MAGs\_blastn\_worked\_100id\_0mismatch\_0gapopen

After we make the filtered file, we need to extract some information from that, so we print the 2nd column of the blast file to a text file so we can pull out headers / sequences:

awk '{print $2 }' DR\_to\_erpe\_MAGs\_blastn\_worked\_100id\_0mismatch\_0gapopen > DR\_to\_erpe\_MAGs\_blastn\_nomismatch\_nogap\_scaffold\_ids.txt

Then we extract the actual sequences based on the headers from original DR file (these will be the “good hits” that we are keeping. We make a file of the extracted sequences that were good and another file with the bad filtered sequences:

pullseq\_header\_name.py -i all\_erpe\_125\_MAGs\_no2019\_renamed.fasta -o erpe\_scaffolds\_BactDR.fa -n DR\_to\_erpe\_MAGs\_blastn\_nomismatch\_nogap\_scaffold\_ids.txt -e F

#This gave me 826 DR sequences. Keep in mind, that the way this was done means that the multiple DR’s each MAG can have get consolidated into a single “hit”. The way around this / to fix would be to call genes and blast to the nucleotides of those calls for the MAGs. For me, this is fine since I’m just trying to confirm any/all CRISPR linkage.

Now doing the spacers, make blast db of vMAG scaffolds. This will generate 3 files: all\_viruses\_1250\_erpe\_2018\_swpw\_ABF\_95-85\_manual\_or\_keep.fna.nhr, all\_viruses\_1250\_erpe\_2018\_swpw\_ABF\_95-85\_manual\_or\_keep.fna.nin, all\_viruses\_1250\_erpe\_2018\_swpw\_ABF\_95-85\_manual\_or\_keep.fna.nsq

makeblastdb -in all\_viruses\_1250\_erpe\_2018\_swpw\_ABF\_95-85\_manual\_or\_keep.fna -dbtype nucl

Then blast the spacers to the vMAGs:

blastn -db all\_viruses\_1250\_erpe\_2018\_swpw\_ABF\_95-85\_manual\_or\_keep.fna -query all\_erpe\_spacers\_renamed.fa -out spacers\_to\_erpe\_vMAGs\_blastn -outfmt 6 -num\_threads 10 -evalue 1e-8

This is where it gets a little stranger. Blastn reports local alignments, which means that the query length is not necessarily the actual query sequence full length. For spacers, we want to take ONLY hits that are exactly identical, and this is only true if a) the full sequence length matches a query sequence exactly at 100% ID, 0 mismatch, and 0 “gapopen, b) the full sequence length matches a query sequence at 1 mismatch and 0 gapopen, or c) the query sequence length is 1 nucleotide different, but the hit has a mismatch of 0, and a gapopen that is 0. We do the same as we did above for direct repeats with filtering to meet these metrics on top of the 1e-8 evalue we used in the command. Reminder, header format for outfmt 6 is:

qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore

To get the lengths, we can use a Rich script and an awk command.

Rich script is here:

“/home/projects-wrighton-2/GROWdb/WHONDRS\_2018/metaG/2018\_rivers/erpe/erpe\_JRR/surface\_and\_pore\_viral\_db\_analyses/crass/spacers\_blastn/lengths\_all/count\_sequence\_residues.pl”

And the run is simply:

Perl count\_sequence\_residues.pl spacers.fa

This will give output.txt. Then you use awk below to pull out ID’s and lengths only to make your file more manageable in size.

awk '{print $1, $2 }' output.txt > lengths\_and\_names.txt

Now that we have the lengths, I just did the rest in excel by vlookup and subtracting the “length” column from the blast command from the actual length we just calculated. If your lengths file is too large, you can to a pullseq command to just pull out those that you have blast hits for. IF THE “SPACER”IS LONGER THAN 60bp, THROW IT OUT. It is a CRASS error.

Now that we have done the first “level” of filtering, we have to do an additional step and it is what is called the “query cover” from Reb. The rules are:

- Query cover column: Query Length - Query End + 1

- If result in query cover = 0 or 1, always keep

- If value is 2, keep it because it means 100% match except 1 nucleotide mismatch at the end.

- If value is negative, you must manually inspect this hit.

With all these criteria, I made file: spacers\_to\_erpe\_vMAGs\_blastn\_worked\_only\_good\_hits\_filtered.txt

Now that’s it! That’s all the information you need to make CRASS linkages. Go into excel now, and pull out the Group ID’s from the spacer names, and from the direct repeat names. This is the “G###” portion. In the spacer, it says “G###SP##” and for DR it says “G###DR##”. You ONLY pull out the “G###” part of it, and then you match the ID across the Spacer and DR tables. If the groups are shared, you have a link!

We can further extract some information from these hits, so we print the 2nd column of the good, filtered out hits from the blast file to a text file so we can pull out headers of vMAGs and their subsequent sequences:

awk '{print $2 }' final\_list.spacers,txt > final\_spacer\_scaffold\_ids.txt

Then we extract the actual sequences based on the headers from original spacers file (these will be the “good hits” that we are keeping. We make a file of the extracted sequences that were good and another file with the bad filtered sequences. \*\* This pulls out the entire vMAG. If you want only spacers, replace the “$2” in the command above with “$1” and pull from the spacer file.\*\*

pullseq\_header\_name.py -i /home/projects-wrighton-2/GROWdb/WHONDRS\_2018/metaG/2018\_rivers/erpe/erpe\_JRR/surface\_and\_pore\_viral\_db\_analyses/crass/all\_erpe\_spacers\_renamed.fa -o erpe\_spacers\_withHits.fasta -n list\_of\_spacers.txt -e F