Crass: The CRISPR assembler (v0.3.1)

Connor Skennerton and Michael Imelfort

3rd October 2012

Contents

1 Quick Start

```
[$ ./autogen.sh]
$ ./configure
$ make
$ make install

$ ./crass [options] input_file{1, n}
$ ./crass-assembler [--velvet | --cap3] [options]
```

2 Installation

Crass has been developed with the GNU build system for ease of configuration on the system. However, because of this Crass requires a UNIX operating system and has been tested on both 64-bit Linux (Ubuntu) and MacOSX personal computers with intel processors and servers with 64-bit Opteron processors.

2.1 Prerequisites

2.1.1 Computer Resources

Crass should run on Linux or MacOSX with 64-bit architectures with gcc installed (note that other compilers have not been tested). Crass successfully compiles with gcc 4.2 and gcc 4.4 other versions of gcc have not been tested.

2.1.2 Pre-installed Software and Libraries

Crass requires that libcrispr and zlib be installed for compilation. In turn, libcrispr requires that Xerces-c is also present on the system. Therefore make sure that zlib and Xerces-c are installed, then install libcrispr and finally Crass on the system.

2.1.3 Optional Packages and Programs

There are a number of optional features of Crass that require additional software. These programs are not required for compilation but add extra features that you might find useful. It is possible to create image files of the graphs that are produced by Crass if the Graphviz package is installed. By default Crass outputs a .gv file containing Graphviz source code, so even if your running Crass on a server that doesn't have it installed you can download the .gv file to another computer for visualization. Testing for the Graphviz package occurs during configuration. Even if you have Graphviz installed you still need to opt-in to graph rendering (see Configure/Compiling Options for the correct options) as rendering images for a large number of graphs can take alot of time. If you enable rendering during the configuration process you can still set user options during run-time to prevent image rendering. (see User Flags).

Crass can also be run as a wrapper for some common assembly algorithms to assemble CRISPRs into contigs. Currently there are two assembly wrappers, one for `velvet` and `cap3`. To activate this feature you must have the executables for either of these two

programs in your `$PATH` environment variable. If at least one of them is in your `$PATH` then the assembly wrapper will be compiled.

2.2 Compiling

On a GNU system simply:

```
[$ ./autogen.sh]
$ ./configure
$ make
$ [sudo] make install
```

NOTE: running `./autogen.sh` may be necessary to generate the `configure` program

2.2.1 Configure/Compiling Options

Crass supports a number of compile-time options as well as the common options:

Option	Definition
`--enable-verbose_logging`	This option is different from enable-debug as it changes what information is printed in the log statments. enabling verbose logging will add the source file name, the function name and the line number for every log statement.
`--enable-rendering`	Set this option will allow Crass to output rendered images of the final spacer graphs. Setting this option does not garentee that Crass will produce images, it will also need to detect at least one of the executables of the Graphviz package in your PATH environmental variable.
`--disable-assembly`	Disables the assembly wrapper even if the Xerces library and suitable genome assemblers can be found. Disabling the assembly wrapper willl change which souce files are compiled and as a result reduce the size of the executable.
`--with-xerces`	Set the path where the Xerces header and library objcets can be found
`--with-libcrispr`	Set the path where libcrispr header and library objects can be found

3 Running Instructions

Crass has two separate commands for finding and for assembling CRISPRs:

```
$ ./crass [options] input_file
$ ./crass assemble ASSEMBLER [options]
```

3.1 Finding CRISPRs

Once installed the most minimal comand is

```
$ ./crass input_file
```

which leaves all of the user options at their default values.

3.1.1 User Flags

Crass has a large munber of user options which may seem overwhelming to a new user. Luckily most of the options are considered 'advanced' and should never have to be changed from their default values. Of course they are in there as user options in case you want to experiment or you suspect that your sample contains a very unusual CRISPR which may require special attention.

Option	Definition
`-a STRING` `--layoutAlgorithm STRING`	When enable-rendering is set and you have Graphviz installed this option will become available and allow you to change the Graphviz layout engine. The full range of layout engines is: neato, dot, fdp, sfdp, twopi, circo
`-b INT` `--numBins INT`	sets the number of colour bins used in the output spacer graph for visualising the coverage of spacers in a dataset. By default the number of bins is equal to the range of the highest and lowest coverage for a CRISPR
`-c STRING` `--graphColour STRING`	Changes the colour range for the output spacer graph. There are four colour scales: red-blue, blue-red, green-red-blue, red-blue-green with the default being red-blue
`-d INT` `--minDR INT`	The lower bound considered acceptable for the size of a direct repeat. The default is 23bp
`-D INT` `--maxDR INT`	The upper bound considered acceptable for the size of a direct repeat. The default is 47bp

`-e --noDebugGraph`	When the DEBUG preprocessor symbol is defined this option will become available. When set it prevents the output of any of the debugging .gv files being produced
`-f INT` `--covCutoff INT`	This variable sets the minimum number of spacers allowed for a putative CRISPR to be considered real and for the assembly to be attempted. The default is 3
`-g --logToScreen`	Does not produce a log file but instead prints the contents to screen.
`-G --showSingletons`	Set this flag if you would like to see unconnected singleton spacers in the final graph.
`-h --help`	Print the basic usage and version information.
`-H --removeHomopolymers`	This is an experimental feature of Crass where the search algorithms attempt to correct for homopolymer errors in reads.
`-k INT` `--kmerCount INT`	Sets the number of kmers that need to be shared between putative direct repeats for them to be clustered together after the find stage. Clustered direct repeats are eventually concatenated to form a 'true' direct repeat for a CRISPR; putative repeats that cannot be clustered are removed from consideration. Change this variable if you feel that the clustering is too stringent and is breaking appart one CRISPR into multiple types. The default number of kmers is 6, however the value should not be set below 6 as this would not be stringent enough; a higher value would split closely related direct repeats apart
`-K INT` `--graphNodeLen INT`	Crass makes a graph by cutting kmers on either side of the direct repeat and then joining these together. The length of the kmer will dictate how connected the graph will be. A smaller number will increase the chances of new conections being formed, however it also increases the chances of false positives. The default value is 9.

`-l INT` `--logLevel INT`	Sets the verbosity of the log file. Under most circumstances the log level cannot go higher than 4, unless the enable-debug option is set during configuration, which will increase the maximum value to 10. Note that above a level of 4 alot of the information will not be understandable to the user as most of these messages are specifically for us, the developers to track down bugs.
`-L --longDescription`	This changes the names of the nodes in the spacer graph to include the sequence of the spacer. The default is to just use the spacer ID
`-n --minNumRepeats`	Used only for long reads, sets the minimum number of repeats that must be identified in a read for it to be considered part of a CRISPR [default: 3]
`-o STRING` `--outDir STRING`	Sets the output directory for files produced by Crass. The default is the current directory
`-r --noRendering`	When the RENDERING preprocessor symbol is defined this option will become available. When set it prevents the generation of rendered images from the intermeadiate debugging graphs (if DEBUG preprocessor symbol is set) and the final graphs.
`-s INT` `--minSpacer INT`	The lower bound considered acceptable for the size of a spacer sequence. Default is 26bp.
`-S INT` `--maxSpacer INT`	The upper bound considered acceptable for the size of a spacer sequence. Default is 50bp.
`-V --version`	Preints out program version information.
`-w INT` `--windowLength INT`	When using the long read search algorithm, changes the window length for finding seed sequences; can be set between 6 - 9bp. The default value is 8bp.

`-x DECIMAL` `--spacerScalling DECIMAL`	Overide the default scalling of the spacer bounds (`-sS`) set by `--removeHomopolymers`. The default is 0.7, i.e. the size of the spacer bounds is reduced by 30% when removing homopolymers in sequences. The value must be a decimal.
`-y DECIMAL` `--repeatScalling DECIMAL`	Overide the default scalling of the direct repeat bounds (`-dD`) set by `--removeHomopolymers`. The default is 0.7, i.e. the size of the direct repeat bounds is reduced by 30% when removing homopolymers in sequences. The value must be a decimal.
`-z --noScalling`	This turns off the effects of (`-x` or `-y`) so that the bounds of the direct repeat and spacer (`-dDsS`) given on the command line are interpreted literally when the `--removeHomopolymers` option is set.

3.1.2 Output From Crass

Once the CRISPR finder is done there will be a number of output files. For every CRISPR type that Crass finds it will produce a sequence file in fasta format that contains reads with that CRISPR type and a .gv file containing Graphviz source code for the arrangment of spacers. There will also be a file called `crass.crispr` file containing information about all the found CRISPRs in an XML format (see The CRISPR File (.crispr) for information about the .crispr specification) and a log file (if you specified for one) that contains all the information about the run. Note that at this stage the CRISPR has not been assembled into contigs, however the reads have been ordered and the arragnement of spacers will be known. You can obtain alot of information about the CRISPRs identified from the `crass.crispr` file by using a toolkit called crisprtools that has been developed for this purpose. The two main tools that get used most often are `crisprtools stat` and `crisprtools extract`. The `stat` command produces a tabular summary of all the CRISPRs that were identified returning stats such as the number of spacers and the number of reads that each CRISPR was found in. The other tool is `extract`, which can be used to get the DNA sequences of the direct repeats, spacers or flankers in fasta format from the crispr file.

3.1.3 Visualizing Graphs

The graph file (.gv) is important for you to visualise all of the spacer arrangments in your CRISPRs. If your input was a metagenomic sample then it is likely that there will be multiple arrangments of spacers. Each individual section will be numbered and this numbering is important to the Crass assembler. The graph can be visualized using a number of programs, the ones that I use are Graphviz (`www.graphviz.org`) and Gephi (`www.gephi.org`). Both of these programs can take a .gv file as input; Graphviz will

produce a single static image whereas Gephi creates an interactive graph that can be manipulated.

3.2 Assembling CRISPRs

CRISPRs from metagenomic data can/will have many spacer arrangements. This makes them difficult for a regular genome assembler to resolve as they look for a single route through the graph. The Crass assembler is a wrapper for other popular genome assemblers; currently there are wrappers for Velvet and Cap3. I premise is that you, the user will look at the output from Crass to determine which of the paths to take through the spacer graph, which are listed as different 'contigs'. To perform the assembly you need to tell Crass the group number of the CRISPR from the finder stage (this will be in the file name), the .crispr file from Crass and the a comma separated list of the segments/contigs IDs for your group of interest. The segments are listed in the names of the spacers in the graph file. For example a spacer with the name 'sp_445_17_C74' means this is spacer with id 445, in segment (contig) 17 with a coverage of 74.

The most minimal command for the assembler is shown below:

```
$ ./crass-assembler [--velvet | --cap3] [options]
```

If for example you run Crass of a metagenome and get 5 groups and you want to assemble contigs 1, 2, 5, 6, 10 from group 3 using velvet

```
$ ./crass-assembler --velvet -g 3 -x crass.crispr -s 1,2,5,6,10
```

This will extract all of the reads that contain spacers from those contigs and use them as input to velvet which will do the assembly in the backgroud. This is a 'dumb' process in the sense that if you provide contig IDs for multiple pathways the assembler will still try and succeed but this will undoubtibly cause breaks.

4 The CRISPR File (.crispr)

The .crispr file is an XML format to describe all aspects of a CRISPR loci. The bulk of the file specification will not be discussed in this manual, however the basics will be talked about to give you some idea of what it is and how Crass uses it.

4.1 Specification Overview

Each .crispr file contains a number of <group> tags, each one containing data about an individual CRISPR locus. There are three sections for each group:

The Data Section This section gives you a list of all of the direct repeats (<drs>), spacers (<spacers>) and flanking (<flankers>) that crass found. Spacers will also have a coverage associated with them.

The Metadata Section This section lists all the other files that are associated with the CRISPR locus described in the group. These include links to image files, and files containing raw sequence data.

The Assembly Section This represents all of the links between each spacer to all others. Reading this section essentially equates to reading a graph of the spacer arrangement of the CRISPR locus

5 Practical Considerations

Crass has been designed primarily with use of Illumina data, however there is nothing stopping you from using it of data produced by other sequencing technologies such as 454, Sanger or Ion Torrent. There are a few practical considerations when dealing with other data types.

5.1 Read Type and Quality

Crass relies on exact string searching algorithms to find direct repeats in reads, which means that reads that are low quality will not produce good results. Crass does contain a number of heuestic algorithms which try to overcome certain types of sequencing errors such as mismatches, although homopolymers require special attention (see below). Although Crass can read fastq files it currently does not utilise quality data. This is partly due to the inacuracy of quality scores of Illumina data, which was the original data that Crass was developed to process.

5.1.1 Sanger Data

Crass does work for Sanger data and should give good results due to the long read length and high quality. It is advised that you quality trim the data to at least Q20. The reason is one primarily of speed of execution time as longer reads will take longer to process.

5.1.2 Homopolymers

Due to the underlying algorithms used in Crass, homopolymers pose a real challenge when finding direct repeats. To try and overcome this the --removeHomopolymers option tells Crass to correct these types of errors using a run-length encoding algorithm. Basically all homopolymers are removed from the read as a preprocessing step and the 'squeezed' read is then used for direct repeat finding. This feature is EXPERIMENTAL and at this stage does not provide optimal results. The main issue is the rate of false positive matches is drastically increased when correcting for homopolymers so it is up to you whether you want to shift through the dud matches at the end to get out the CRISPRs that you want.

5.2 Bird's Nests - fixing overly connected graphs

When running Crass some CRISPRs may end up looking like a bird's nest due to over connectidness in the graph. This basically makes the inteptretion of the graph impossible. You can try and resolve this by rerunning Crass on just that particular CRISPR and increasing the value of -K option. -K sets an internal parameter when Crass builds the graph that determines how easily spacers can be joined together. By default it is set to 11 however raising it up to a higher value may resolve the issues. You may like to set the -G option as well to see all the extra singleton spacers that get generated. Be warned though that in some instances Graphviz will consume significant amounts of memory when -G is set on large graphs with lots of singletons.

5.3 Assembling

The Crass assembler is a simple wrapper for other genome assembly algorithms. Currently there are only two wrappers available: cap3 and velvet. The choice of assembler should be governed by the input data so for example if Crass was processing Sanger data, it would make more sense to use cap3 as the assembler. This isn't to say that you shouldn't experiment with different assemblers to see what's best with your data. Finally if you want to add a new wrapper to the list just send in a bug report.