Crosslink - polyploid-aware genetic mapping for outbreeders

Synopsis

Builds genetic maps using codominant biallelic markers derived from the F1 progeny of an outcross between two unrelated non-inbred parents, with additional features suitable for allopolyploid species.

Crosslink was written and tested on the Linux operating system, and designed to allow a high degree of automation by the use of scripting. Cross platform deployment is currently supported by the use of virtual machine and docker images. The source code is also freely available.

Crosslink consists of a number of programs, written in C, and helper scripts written in bash and python. The main programs are:

crosslink_group performs grouping, phasing and approximate marker ordering, and also redundant marker removal, missing value imputation and polyploid-related marker-typing error correction

crosslink_map performs final marker ordering and assigns final map positions, and also, for allopolyploids, detects markers which falsely join homeologous linkage groups

crosslink_pos performs a subset of crosslink_map's functionality, assigning final map positions

crosslink_viewer produces a two dimensional colour-coded plot showing an all-versus-all marker comparison indicating phasing, rf, LOD and map distances for one or more linkage groups at a time

crosslink_graphical shows a graphical representation of genotype values along one or more linkage groups

create_map and **sample_map** allow for creation of simulated marker data

File Formats

genotype file

A file containing marker data in a format similar to the loc file format of JoinMap®, but lacking any header information. Each marker occupies one line as follows:

<name> <type> <phase> <genotype1> [<genotype2>…]

There must be no blank lines or comments. All columns must be separated by a single space. A marker name can be any string, but should not contain any white space characters (space, tab etc.). Marker type must be one of the following three: <lmxll> indicating a biallelic codominant marker heterozygous in the maternal and homozygous in the paternal parent, <nnxnp> a biallelic codominant marker heterozygous in the paternal and homozygous in the maternal parent, or <hkxhk> a biallelic codominant marker heterozygous in both parents. Marker phase must always be specified but can be set arbitrarily to zero where phase is not yet known and will be ignored. For lmxll markers the phase must be either {0-} or {1-}, for nnxnp markers either {-0} or {-1} and for hkxhk markers one of {00}, {01}, {10} or {11} where the first and second digits indicate the maternal and paternal phase respectively. Each marker in the same file must have the same number of genotypes, each of which must consist of exactly two characters. For all marker types missing values must be specified as --, for lmxll markers genotypes must be either lm or ll, for nnxnp either nn or np and for hkxhk one of hh, kk, hk or kh. If the hk genotypes have not been imputed then hk and kh will be treated the same, whereas for imputed genotypes hk is taken to mean allele h was from the maternal parent and kh to mean allele h was from the paternal parent.

Example markers:

```
MK-1234 <lmxll> {0-} lm lm ll lm ll -- ll lm
4567.890 <nnxnp> {-1} nn nn -- np -- -- nn np
abcd <hkxhk> {01} hh kk hh hk hk -- hk kh
```

There is no way to specify groupings of markers into linkage groups within a single file. Initially all markers should be together in the same file. When linkage groups are formed each is put into a separate file, and different versions of the map can be organised into their own subdirectory where required.

map file

crosslink_map outputs one map file per linkage group using a four column format giving the marker name and its centimorgan position in the maternal, paternal and combined map. Where a marker is not present or its position cannot be calculated in one of the maps, its position is listed as NA. The first line of the file contains the linkage group name and the number of markers as follows:

group <linkage_group_name> ; markers <number_of_markers>

Each marker is listed using four whitespace (i.e. tab and space characters) separated columns:

<marker_name> <maternal_map_position> <paternal_map_position> <combined_map_position>

Example map file:

```
group 000 ; markers 126
PHR10-89849276      0.0000     NA   0.0000
NMH21-89792123          NA 0.0000   0.0000
PHR11-89793667      0.0000 0.0000   0.0000
NMH12-89832883      0.6211     NA   0.7819
PHR10-89793765      1.2423     NA   1.5637
NMH01-89892406          NA 2.5322   2.1004
PHR21-89835283          NA 3.7823   3.1373
PHR11-89793784      2.4923 3.7823   3.1373
NMH12-89793368      3.1134     NA   3.6021
```

Installation

Using the Prebuilt VirtualBox Image

Cross-platform installation of Crosslink is only currently supported through the use of virtual machine and docker images. To use the graphical visualisation tools it is recommended to use the virtual machine rather than docker image, as everything should work without any further setup required. The VirtualBox software is currently available from Oracle for Windows, OS X, Linux and Solaris from:

https://www.virtualbox.org

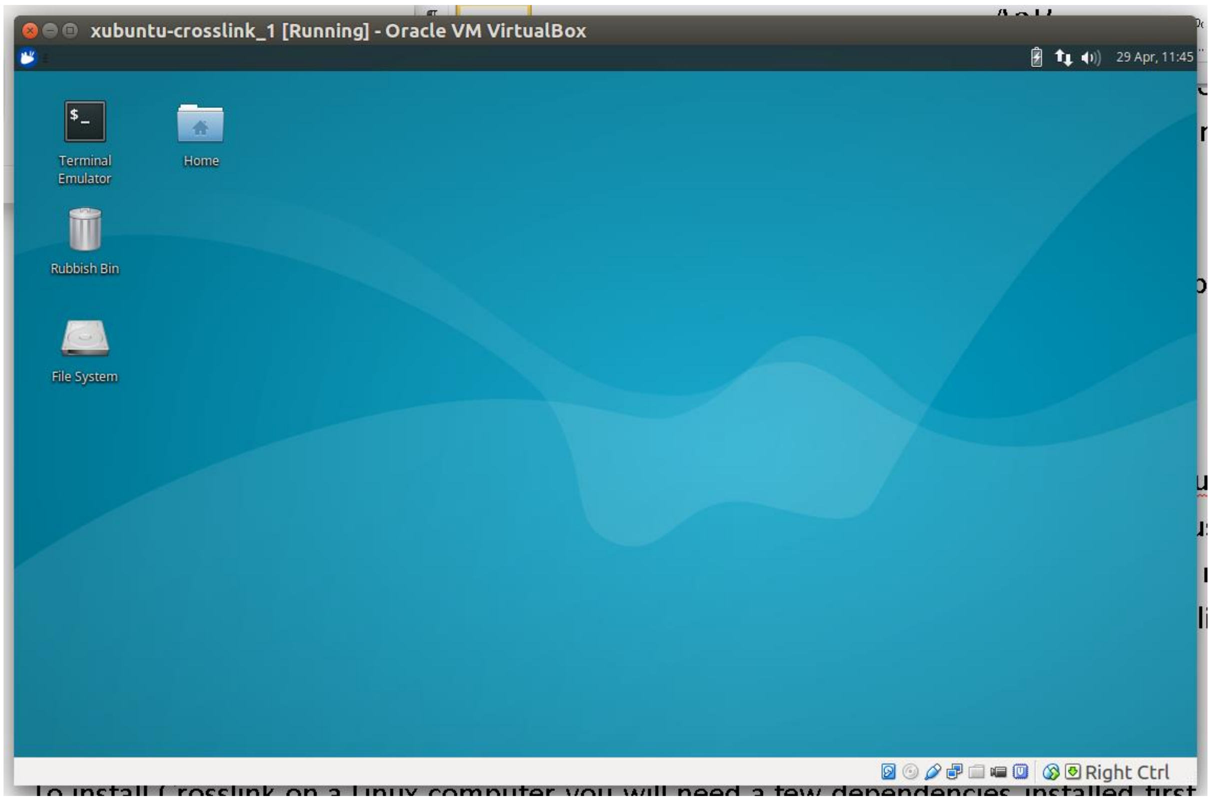
and may also be available for your Linux distribution through its built-in package management system, for example on Ubuntu it can usually be installed by using the following command:

```
sudo apt-get install virtualbox
```

Once VirtualBox is installed, download the virtual machine file xubuntu-crosslink.ova (approximately 2GB) from the latest Crosslink github release:

https://github.com/eastmallingresearch/crosslink/releases

Launch VirtualBox and import the image using File->Import Appliance, then select the xubuntu-crosslink.ova file you just downloaded. When the Appliance Settings dialog appears adjust the amount of RAM and the number of CPUs you which to allocate to the virtual machine if required, then click Import. Once imported select the virtual machine you have just created and click the green Start arrow. The image contains Xubuntu 16.04, a minimalist version of Ubuntu using the Xfce desktop environment, which should appear like this once loaded:



You may wish to enable copy-pasting between the host and guest operating system, if so Devices -> Shared Clipboard -> BidirectionalCrosslink in the VirtualBo menu system.

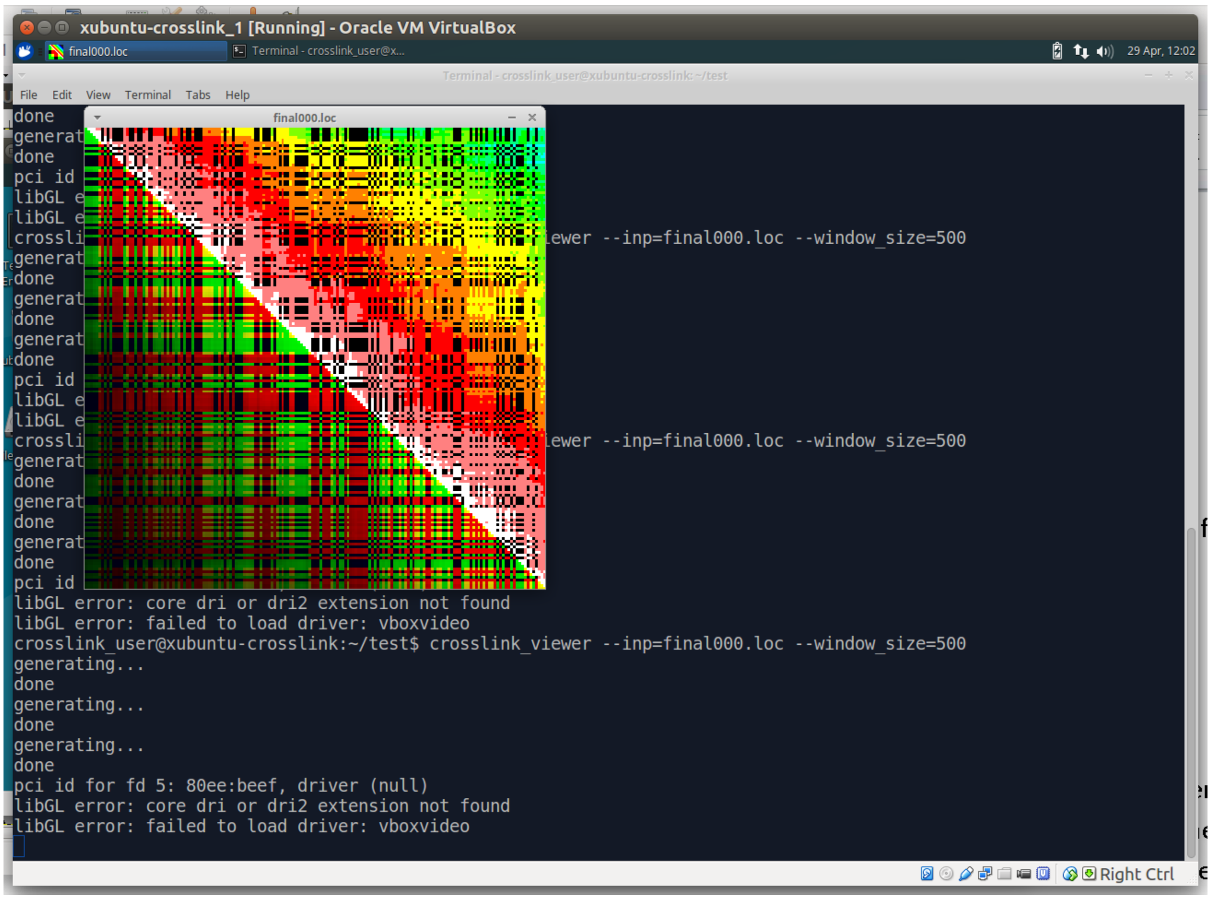
Crosslink is already installed. To test it is working open a terminal by clicking on the black Terminal Emulator icon and run the following commands:

```
mkdir test
cd test
~/crosslink/example_pipelines/simple_simulated.sh
```

After the command finishes type the 'ls' command and you should see 28 .loc and .map files called group and final. To test a data visualisation command run the following:

```
crosslink_viewer --inp=final000.loc --window_size=500
```

You should see something like this:


xubuntu-crosslink_1 [Running] - Oracle VM VirtualBox
final000.loc
Terminal - crosslink_user@x...
29 Apr, 12:02
Terminal - crosslink_user@xubuntu-crosslink: ~/test
File Edit View Terminal Tabs Help
final000.loc
Right Ctrl

```
done
generat
done
pci id
libGL e
libGL e
crossli                                           ewer --inp=final000.loc --window_size=500
generat
done
generat
done
generat
done
pci id
libGL e
libGL e
crossli                                           ewer --inp=final000.loc --window_size=500
generat
done
generat
done
generat
done
pci id
libGL error: core dri or dri2 extension not found
libGL error: failed to load driver: vboxvideo
crosslink_user@xubuntu-crosslink:~/test$ crosslink_viewer --inp=final000.loc --window_size=500
generating...
done
generating...
done
generating...
done
pci id for fd 5: 80ee:beef, driver (null)
libGL error: core dri or dri2 extension not found
libGL error: failed to load driver: vboxvideo
```

Press 'q' or close the graphical window to quit. Type 'exit' or close the window to exit the terminal. To shut down the virtual machine click on the mouse icon in the top left and click on the logout button at the bottom right of the menu, then select Shutdown.

Installing Directly on Linux

To install Crosslink on a Linux computer you will need a few dependencies installed first. On Ubuntu the apt-get command can be used to installed the required packages. If you use a different distribution the install command and package names will likely be different, but the same dependencies will be required. For Ubuntu the following packages should meet all the dependencies (if you do not have the ability to install packages ask your system administrator to help you):

To compile Crosslink from source code:

gcc

To be able to compile the visualisation tools crosslink_viewer and crosslink_graphical:

libsdl2-2.0-0, libsdl2-dev, libsdl2-image-2.0.0, libsdl2-image-dev

To use some of the python helper scripts:

python-scipy, python-numpy, python-matplotlib

To install these packages use the following:

```
sudo apt-get update

sudo apt-get install gcc git libsdl2-2.0-0 libsdl2-dev libsdl2-image-2.0.0 libsdl2-image-dev\
    python-scipy python-numpy python-matplotlib
```

This should automatically pull in all the additional dependencies these packages require. If you have pip installed and the necessary dependencies you may be able to install the python packages without needing the sudo command as follows:

```
pip install --user numpy scipy matplotlib
```

Next decide where you will put Crosslink's files and change into the parent directory. For example if you wish to put the files in a folder called /home/vicker/programs/crosslink:

```
mkdir -p /home/vicker/programs

cd /home/vicker/programs
```

Modify the path names as appropriate to your chosen install location. Now download the Crosslink files from github. If you have git installed:

```
git clone https://github.com/eastmallingresearch/crosslink
```

will download the files and put them into a new subdirectory called crosslink. Or you can visit https://github.com/eastmallingresearch/crosslink and download the source code as a zip archive and extract it as the folder /home/vicker/programs/crosslink. Now make sure you have the files in the expected place:

```
file /home/vicker/programs/crosslink/src/crosslink_common.h
```

which should respond with:

```
/home/vicker/programs/crosslink/src/crosslink_common.h: C source, ASCII text
```

If it cannot find the file you may have extracted the files into the wrong location. Once you have the files in the correct place, change into the src directory and run the main compilation script:

```
cd /home/vicker/programs/crosslink/src
```

./make.sh

You should now be able to run the main Crosslink programs. Test that crosslink_group is working:

/home/vicker/program/crosslink/bin/crosslink_group --help

which should respond with a help message:

Crosslink Copyright (C) 2016 NIAB EMR see included NOTICE file for details

remove redundant markers... etc

If you have SDL2 installed you can compile the data visualisation programs too:

./viewer_make.sh

And should be able to get the help message from crosslink_viewer:

/home/vicker/programs/crosslink/bin/crosslink_viewer --help

which should respond with:

Crosslink Copyright (C) 2016 NIAB EMR see included NOTICE file for details

presents a graphical view of linkage LOD and map distances... etc

docker

virtual box

amazon?

Quick Start Guide

Simulated Data Without Polyploid-Related Errors

An example of a simple pipeline using Crosslink to generate and map simulated outcross data is included in the file example_pipelines/simple_simulated.sh. To make the script run successfully edit the line near the top which sets the variable CROSSLINK_PATH, ensuring that it specifies the location Crosslink's files were installed on your system. For example if Crosslink is installed to a folder called /home/vicker/programs/crosslink then modify the line as follows:

CROSSLINK_PATH=/home/vicker/programs/crosslink

If you run the example pipeline from the docker or virtual machine image you should not have to edit this line, as it should already be correctly set.

Rather than simply run the script itself we will now step through the simple_simulated.sh pipeline one line at a time, typing (or copy-pasting) the commands into the shell. To begin open a command prompt, create a new directory called test_crosslink and change into it:

```
mkdir test_crosslink
```

```
cd test_crosslink
```

Again set the PATH variable appropriately. Supposing Crosslink's folder is located at /home/vicker/program/crosslink we would enter the following lines at the command prompt:

```
CROSSLINK_PATH=/home/vicker/programs/crosslink
export PATH=${CROSSLINK_PATH}/bin:${PATH}
```

Now invoke the crosslink_group command with the help option to verify that the program runs:

```
crosslink_group --help
```

You should see :

```
Crosslink Copyright (C) 2016 NIAB EMR see included NOTICE file for details
remove redundant markers, form linkage groups, correct marker typing errors, phase, impute
missing values, perform approximate ordering

   --inp=STRING input genotype file (required)...
etc
```

Invoking any of the core Crosslink programs (create_map, sample_map, crosslink_group, crosslink_map, crosslink_pos, crosslink_viewer, crosslink_graphical) with the --help option will print a summary of command line options to the terminal and exit.

Now create a map specification file with the following command, which can be entered as a single line without the backslash characters (\), or across multiple lines retaining the backslashes:

```
create_map --output-file=sample.spec --numb-lgs=28\
        --map-size=2000 --marker-density=2.0\
        --prob-both=0.28 --prob-maternal=0.36
```

A new file called sample.spec should have appeared. Take a look at its contents using:

```
head sample.spec
```

It should begin with something similar to:

```
#markers 4000 lgs 28
M00000a98 <nnxnp> {-1}   0 0.465766
M00000029 <lmxll> {0-}   0 2.513004
```

where each line specifies the marker name, type, phase, linkage group and centimorgan map position.

The options given to create_map define the number of linkage groups (28), the total map length (2000 centimorgans), the average marker density (2.0 markers per centimorgan), the approximate proportion of markers heterozygous in both parents (0.28) and heterozygous in the maternal parent only (0.36) with the proportion heterozygous in the paternal parent being defined implicitly as 1.0 - 0.28 - 0.36 = 0.36. Use create_map --help for more help with the options.

Next we will simulate some genotype data obtained from an F1 cross:

```
sample_map --input-file=sample.spec --output-file=sample.loc\
        --samples=162 --prob-missing=0.007 --prob-error=0.01
```

where we have told sample_map to simulate data from 162 progeny with a missing data rate of 0.007 and a genotyping error rate of 0.01. This program does not generate any allopolyploid-related errors, which are created using the separate create_type_errors.py script.

A new file sample.loc should have appeared, which is a genotype file as described in a previous section, and can be viewed using:

```
less -S sample.loc
```

You will notice that all phase values are set to zero and a few genotypes are set to -- indicating they are missing. The markers are also in a random order with respect to their true linkage group and map position. The marker names do not indicate the true marker order in any way, so that it is not possible for a mapping program undergoing testing to inadvertently cheat by alphabetically sorting the markers. For hkxhk markers, all heterozygous genotype calls are given as hk, so that it is unspecified whether the maternally inherited allele was h or k. Press 'q' to quit the less command.

Next we will split the markers into a separate file for each linkage group:

```
crosslink_group --inp=sample.loc --outbase=group --min_lod=10.0\
          --knn=3
```

where we have told the program to split markers based on a linkage LOD threshold of 10, and to impute missing values using the 3 nearest neighbouring markers. The separate linkage group files should have appeared with names starting with group, and be numbered sequentially from 000 to 027.

If you wish to rerun the grouping at a different LOD threshold, be aware that crosslink_group does not automatically delete any existing group files, so it is important to first remove all the old files, by a command such as (be sure you are familiar with the rm command before deleting any files from the command prompt in this way as it is easy to delete the wrong files by mistake):

```
rm group???.loc
```

With the present data a LOD of 10 ought to split the markers into the correct 28 groups. View the first linkage group using:

```
less -S group000.loc
```

You will notice the phases are now set and there are no longer any missing values (crosslink_map does not support missing values). The markers have also been approximately ordered, so that you may notice consecutive genotypes have similar patterns.

To see how many markers were assigned to each group:

```
wc --lines group???.loc
```

The last step is to produce the final marker order and map positions for each linkage group:

```
for fname in group???.loc
do
   echo ${fname}...
   outname=${fname/group/final}
   crosslink_map --inp=${fname} --out=${outname}\
            --map=${outname/loc/map}
done
```

Here we have told crosslink_map to output the final genotype files with names beginning with final, and to produce associated map files for each one. View the first final genotype file using:

```
less -S final000.loc
```

You will notice that hkxhk markers now have some heterozygous genotypes given as kh as well as hk, indicating that crosslink_map has imputed which allele is likely to be from the maternal parent and which from the paternal (the maternal is the first of the two letters). View the first map file using:

```
less final000.map
```

You should see something like:

```
group group000 ; markers 143
M000001da      0.0000      0.0000      0.0000
M00000700         NA      1.8870      2.2589
M00000055         NA      3.7740      4.5178
M00000542         NA      5.0241      6.0142
M000007ae      8.7471      6.2742      7.5106
M00000f0e     13.2663         NA     10.7209
M00000d96     14.5163         NA     11.6090
etc
```

where the four columns are the marker name followed by its positions on the maternal, paternal and combined maps respectively. lmxll and nnxnp markers are only present on the maternal and paternal maps respectively so the other position is listed as NA. To see the combined length of each linkage group:

```
for x in final???.map
do
   echo ${x} $(tail -n 1 ${x} | cut -f 4)
done
```

You should see something like:

```
final000.map 253.2675
final001.map 234.6211
final002.map 234.7441
```

```
final003.map 278.4530
final004.map 265.6859
final005.map 272.5785
final006.map 268.4784
final007.map 288.5128
final008.map 280.2432
final009.map 256.5174
etc
```

Real Data From A Polyploid Outcross

The file sample_data/rgxha.loc.gz contains a real dataset from the allo-octoploid cultivated strawberry *Fragaria x ananassa* from an outcross between Redgauntlet (maternal parent) and Hapil (paternal parent) cultivars. There are 11587 markers genotyped on 162 progeny using the Affymetrix Axiom iStraw90K microarray platform. The markers were called as part of a batch of 6x96 samples (including data from other, unrelated plants) using the Affymetrix Power Tools (APT) and SNPolisher pipeline and further filtered to remove extreme segregation distortion. The marker names indicate the classification assigned to each marker by APT: PHR (polymorphic, high resolution) indicates three clusters were detected across all the samples and assigned as the AA, AB and BB genotypes; NMH (no minor homozygote) indicates only two clusters which were assigned as AA and AB genotypes. The two numbers after the three letter code indicate the genotype code assigned by APT to the maternal and paternal parents using the convention 0=AA,1=AB,2=BB. For example, PHR12 indicates a PHR marker where Redgauntlet was called as AB and Hapil as BB. The rest of the marker name is the iStraw90K probesetid, for example PHR12-89894503 indicates that the original marker probesetid was AX-89894503.

The script example_pipelines/build_rgxha.sh demonstrates a pipeline which builds a map from the sample data, making use of Crosslink's helper scripts. The helper scripts assist in calling the appropriate Crosslink program (crosslink_group, crosslink_map or crosslink_pos) with the appropriate options when only a subset of their full functionality is required, for example, outputting just a list of the names of redundant markers. Most of the scripts require a configuration file specifying Crosslink parameters, and examples of these are provided in sample_data/rgxha_conf which can be copied and modified as needed.

The pipeline has comments explaining each step. Some extra explanations are provided now. To begin, create a new directory and change into it. The script begins by making new working copies of the helper script configuration files and genotype data. An initial exploratory grouping is performed outputting one file per linkage group into a new subdirectory called initgrps. The helper scripts generally deal with individual genotype files or with all the genotype files within a named subdirectory, these being assumed to represent the linkage groups. In most cases passing the name of a subdirectory to a script will cause it to create the directory if required and remove existing files ending with .loc to prevent old files from getting mixed up with new ones:

```
cl_group.sh   all.loc   initgrps   ${MINLOD}
```

This command can be rerun with different LOD values to see how many linkage groups result. If you have graphical output available from your Crosslink install, you can use crosslink_viewer to view the linkage groups. This will be explained in more detail in the next section, but for now we will ensure that we retain all the intermediate results of the pipeline in different subdirectories for later viewing.

Polyploidy appears to cause a few markers to be misclassified. Most commonly this is a NMH marker where the classification of the two clusters as homozygote and heterozygote has been reversed, so that an lmxll marker is called as an nnxnp or vice versa. This error is much less common for PHR markers. Crosslink fixes these errors by detecting unexpected linkage between maternal and paternal markers. Where a group of markers are thus found to be linked above a significant LOD threshold but contain a mixture of maternal and paternal markers, the rarer kind of marker is converted to the more common kind (allowing for some markers to be weighted more strongly than others). This is performed using:

```
cl_fixtypes.sh   all.loc   all.loc   conf/fixtypes.000
```

where conf/fixtypes.000 is the configuration file which contains:

```
CL_GROUP_MINLOD=${CL_GROUP_MINLOD:-6.0}
CL_GROUP_MATPATLOD=${CL_GROUP_MATPATLOD:-10.0}
CL_MATPAT_WEIGHTS=${CL_MATPAT_WEIGHTS:-"01P03"}
CL_GROUP_LOGFILE=${CL_GROUP_LOGFILE:-fixtypes.log}
```

The helper script configuration files, written in bash, contain default values which can be overridden in the calling script. For example MYVARIABLE=${MYVARIABLE:-10} indicates that if no value is already set, MYVARIABLE should be set to 10 (*not* -10), otherwise the existing value is used.

In this case CL_GROUP_MINLOD specifies markers should be grouped at a LOD threshold of 6, CL_GROUP_MATPATLOD that anomalous linkage between maternal and paternal markers in the same group should be considered at a LOD threshold of 10, CL_MATPAT_WEIGHTS indicates that by default markers should be weighted as 01 (i.e. 1), but markers with names starting with the letter P should be weighted as 03 (i.e. considered as 3 times more reliable) when deciding the true marker type. For example if two lmxll PHR-type markers and five nnxnp NMR-type markers are found to be linked together, this will count as 2x3=6 votes for lmxll and 5x1=5 for nnxnp and all the markers will be set as lmxll type (because PHR markers are misclassified more rarely than NMHs). CL_GROUP_LOGFILE names the log file that crosslink_group will output containing the names of the markers whose types were switched.

cl_fixtypes.sh reads markers in from all.loc, processes them using the configuration provided and then outputs to all.loc again, *overwriting* the existing data. Alternatively a different output filename could have been provided to allow retention of the input data. The linkage groups formed internally are not retained and all markers are outputted into the same output file together. This allows different LOD threshold to be used in different steps of the pipeline.

The next step forms linkage groups again, outputting to a new subdirectory. This step is simply to allow the effect of the type error correction to be assessed:

```
cl_group.sh   all.loc   fixgrps   ${MINLOD}
```

Next the pipeline identifies which markers are redundant, containing only genotype calls all exactly the same as another marker(s). Missing values are not counted as being new information.

```
cl_findredun.sh   all.loc   all.redun   conf/findredun.000
```

Input is read from all.loc (which is not modified) and a list of redundant markers output to all.redun in a two column format. Column one gives the name of the redundant marker, column two gives the name of the non-redundant marker that represents it. conf/findredun.000 is the configuration file:

```
CL_GROUP_MINLOD=${CL_GROUP_MINLOD:-6.0}
CL_GROUP_REDUNLOD=${CL_GROUP_REDUNLOD:-20.0}
```

where a CL_GROUP_MINLOD of 6.0 indicates that linkage groups be formed above a LOD threshold of 6, and a CL_GROUP_REDUNLOD of 20 indicates that markers linked above this LOD are explicitly checked to see if one can be flagged as redundant with respect to the other. Only the marker list is output, the grouping is not retained.

The next step is to impute missing genotype calls:

```
cl_knnimpute.sh   all.loc   all.loc   conf/knnimpute.000
```

The configuration file contains:

```
CL_GROUP_MINLOD=${CL_GROUP_MINLOD:-6.0}
CL_GROUP_KNN=${CL_GROUP_KNN:-3}
```

Genotypes are read in from all.loc and once imputed are output to all.loc overwriting the original data. A CL_GROUP_MINLOD of 6 indicates that linkage groups should be formed down to a LOD threshold of 6, a CL_GROUP_KNN of 3 indicates that each missing genotype is imputed from the mean value of the 3 most similar markers in the same linkage group which do not lack the corresponding genotype. The imputed data are output into a single file, the grouping is not retained.

Next just the redundant markers (listed in all.redun) are excluded from the fully-imputed genotypes (all.loc) and saved as a new file (all.uniq). The redundant markers will be used again later and are therefore not overwritten:

```
cl_extract.sh   all.loc   all.redun   all.uniq
```

Next linkage groups are formed from the unique markers and put in a new subdirectory called uniqgrps:

```
cl_group.sh   all.uniq   uniqgrps   ${MINLOD}
```

Because polyploid related errors are causing some markers to be linked strongly to two different (but homeologous) linkage groups, the data do not split into the expected 28 linkage groups. This causes crosslink_group to give up phasing the affected groups, as phasing information will not propagate to all the markers in those groups, therefore we use an additional step to force phasing to complete down to a LOD threshold of zero for each linkage group individually. The phasing between unlinked markers will be meaningless, but at least all genuinely linked markers should have a usable phase. Here we retain the data before and after forced phasing (which just runs crosslink_group with a LOD threshold of zero) for later visualisation:

```
cl_phase.sh   uniqgrps   phasegrps
```

Next, we try to automatically detect all the markers which are incorrectly joining two linkage groups. This step *must* be run on fully phased markers:

```
cl_detect_crosslg.sh phasegrps crosslg_markers conf/detectcrosslg.000
```

The configuration file contains:

```
CL_PARALLEL_JOBS=${CL_PARALLEL_JOBS:-1}
CL_MAP_RANDOMISE=${CL_MAP_RANDOMISE:-0}
CL_GA_SKIPORDER1=${CL_GA_SKIPORDER1:-1}
CL_MAP_CYCLES=${CL_MAP_CYCLES:-5}
CL_GA_ITERS=${CL_GA_ITERS:-150000}
CL_GA_OPTIMISEMETH=${CL_GA_OPTIMISEMETH:-0}
CL_GA_USEMST=${CL_GA_USEMST:-5}
CL_GA_MINLOD=${CL_GA_MINLOD:-3.0}
CL_GA_MSTNONHK=${CL_GA_MSTNONHK:-0}
CL_GA_PROBHOP=${CL_GA_PROBHOP:-0.3333}
CL_GA_MAXHOP=${CL_GA_MAXHOP:-1.0}
CL_GA_PROBMOVE=${CL_GA_PROBMOVE:-0.3333}
CL_GA_MAXMOVESEG=${CL_GA_MAXMOVESEG:-1.0}
CL_GA_MAXMOVEDIST=${CL_GA_MAXMOVEDIST:-1.0}
CL_GA_PROBINV=${CL_GA_PROBINV:-0.5}
CL_GA_MAXSEG=${CL_GA_MAXSEG:-1.0}
CL_GIBBS_SAMPLES=${CL_GIBBS_SAMPLES:-200}
CL_GIBBS_BURNIN=${CL_GIBBS_BURNIN:-5}
CL_GIBBS_PERIOD=${CL_GIBBS_PERIOD:-1}
CL_GIBBS_PROBSEQUEN=${CL_GIBBS_PROBSEQUEN:-0.0}
CL_GIBBS_PROBUNIDIR=${CL_GIBBS_PROBUNIDIR:-1.0}
CL_GIBBS_MINPROB1=${CL_GIBBS_MINPROB1:-0.1}
CL_GIBBS_MINPROB2=${CL_GIBBS_MINPROB2:-0.0}
CL_GIBBS_TWOPT1=${CL_GIBBS_TWOPT1:-1.0}
CL_GIBBS_TWOPT2=${CL_GIBBS_TWOPT2:-1.0}
CL_HOMEO_MINCOUNT=${CL_HOMEO_MINCOUNT:-2}
CL_HOMEO_MINLOD=${CL_HOMEO_MINLOD:-1}
CL_HOMEO_MAXLOD=${CL_HOMEO_MAXLOD:-25}
```

This step uses crosslink_map internally to impute the missing information in the hkxhk markers before detecting those joining two linkage groups. CL_PARALLEL_JOBS=1 indicates to only process one linkage group at a time. Increase the value to utilise more than one CPU core. The many options relating to marker ordering and hk genotype imputation will be described in detail in a later section. The three final options relate directly to detection of cross linkage group markers. The problem appears to be the result of a few hkxhk markers whose maternal information comes from one linkage group while the paternal information comes from one of the homeologous linkage groups. That is to say that the marker's probes seem to be hybridising to two or more homeologous loci, one of which is heterozygous in the maternal genome and another in the paternal genome. These cases are detected by comparing pairs of hkxhk markers, where we expect the markers to be truly linked according to one parent but unlinked according to the other. For a pair of hkxhk markers if one parent indicates a LOD below CL_HOMEO_MINLOD and the other a LOD above CL_HOMEO_MAXLOD then both markers are given one point. Markers accumulate points for each qualifying hkxhk marker pair they are involved in. Those that receive a total number of points greater than or equal to CL_HOMEO_MINCOUNT are flagged as being cross linkage group markers, and listed in the output file (here we have called it crosslg_markers).

The next three steps remove the flagged markers then regroup and rephase the data:

```
cl_removemarkers.sh   all.uniq   filt.uniq   crosslg_markers
cl_group.sh   filt.uniq   filtgrps   ${MINLOD}
cl_phase.sh   filtgrps   filtgrps
```

There are still 27 rather than the expected 28 linkage groups. If you have graphical output available, crosslink_viewer can be used to view each linkage group in turn (explained in detail in the next section), which should make it fairly obvious which group needs to be split. There should be a single hkxhk marker holding the two true linkage groups together. Zoom in using the cursor keys and plus/minus keys to find the marker. Clicking on the marker should print its name to the terminal. We can then manually add it to the list of cross linkage group markers and repeat the filtering and regrouping steps. If you don't have crosslink_viewer available just follow these steps below anyway for now:

```
echo 'PHR11-89834490' >> crosslg_markers
cl_removemarkers.sh   filt.uniq   filt.uniq   crosslg_markers
cl_group.sh   filt.uniq   filtgrps   ${MINLOD}
cl_phase.sh   filtgrps   filtgrps
```

There should now be 28 linkage groups. To produce the final map ordering use the following:

```
cl_order_hkimpute.sh   filtgrps   finalgrps   conf/orderhkimpute.000
```

which will output the final markers orderings with imputed hk genotypes into the new subdirectory finalgrps. The configuration file contains:

```
CL_PARALLEL_JOBS=${CL_PARALLEL_JOBS:-1}
CL_MAP_RANDOMISE=${CL_MAP_RANDOMISE:-0}
CL_GA_SKIPORDER1=${CL_GA_SKIPORDER1:-1}
CL_MAP_CYCLES=${CL_MAP_CYCLES:-5}
CL_GA_ITERS=${CL_GA_ITERS:-150000}
CL_GA_OPTIMISEMETH=${CL_GA_OPTIMISEMETH:-0}
CL_GA_USEMST=${CL_GA_USEMST:-5}
CL_GA_MINLOD=${CL_GA_MINLOD:-3.0}
CL_GA_MSTNONHK=${CL_GA_MSTNONHK:-0}
CL_GA_PROBHOP=${CL_GA_PROBHOP:-0.3333}
CL_GA_MAXHOP=${CL_GA_MAXHOP:-1.0}
CL_GA_PROBMOVE=${CL_GA_PROBMOVE:-0.3333}
```

```
CL_GA_MAXMOVESEG=${CL_GA_MAXMOVESEG:-1.0}
CL_GA_MAXMOVEDIST=${CL_GA_MAXMOVEDIST:-1.0}
CL_GA_PROBINV=${CL_GA_PROBINV:-0.5}
CL_GA_MAXSEG=${CL_GA_MAXSEG:-1.0}
CL_GIBBS_SAMPLES=${CL_GIBBS_SAMPLES:-200}
CL_GIBBS_BURNIN=${CL_GIBBS_BURNIN:-5}
CL_GIBBS_PERIOD=${CL_GIBBS_PERIOD:-1}
CL_GIBBS_PROBSEQUEN=${CL_GIBBS_PROBSEQUEN:-0.0}
CL_GIBBS_PROBUNIDIR=${CL_GIBBS_PROBUNIDIR:-1.0}
CL_GIBBS_MINPROB1=${CL_GIBBS_MINPROB1:-0.1}
CL_GIBBS_MINPROB2=${CL_GIBBS_MINPROB2:-0.0}
CL_GIBBS_TWOPT1=${CL_GIBBS_TWOPT1:-1.0}
CL_GIBBS_TWOPT2=${CL_GIBBS_TWOPT2:-1.0}
```

Again CL_PARALLEL_JOBS controls how many linkage groups will be processed in parallel using multiple CPU cores if available. CL_MAP_CYCLES=5 means to run 5 cycles of marker ordering and hk imputation. CL_MAP_RANDOMISE=0 and CL_GA_SKIPORDER1=1 means to skip the very first round of marker ordering, retaining the approximate ordering of the input data produced previously by crosslink_group and proceed straight to hk imputation. For subsequent rounds marker ordering is performed as normal. CL_MAP_RANDOMISE=1 would cause the program to ignore the order of the markers in the input file and start the first round of ordering from a random initial order (this option cannot be used with CL_GA_SKIPORDER1=1, otherwise randomly ordered markers would be passed to the hk imputation step). The remaining options will be explained in detail in the reference section of this document. In summary, this step produces a final marker ordering for each linkage group and also imputes the likely parental origin of the h and k allele for all hk genotypes. The output genotype files contain the markers in map order with hk genotypes listed explicitly as either hk or kh.

The final three commands produce the final map positions of the markers, and produce a redundant version of the genotype and map file for each linkage group where the redundant markers have been reinserted at the position of their representative marker:

```
cl_mappos.sh   finalgrps   finalgrps
cl_reinsert_loc.sh finalgrps all.loc all.redun finalredun conf/reinsert.000
cl_reinsert_map.sh   finalgrps   all.redun   finalredun
```

The configuration file contains:

```
CL_GIBBS_SAMPLES=${CL_GIBBS_SAMPLES:-200}
CL_GIBBS_BURNIN=${CL_GIBBS_BURNIN:-5}
CL_GIBBS_PERIOD=${CL_GIBBS_PERIOD:-1}
CL_GIBBS_PROBSEQUEN=${CL_GIBBS_PROBSEQUEN:-0.0}
CL_GIBBS_PROBUNIDIR=${CL_GIBBS_PROBUNIDIR:-1.0}
CL_GIBBS_MINPROB1=${CL_GIBBS_MINPROB1:-0.1}
CL_GIBBS_MINPROB2=${CL_GIBBS_MINPROB2:-0.0}
CL_GIBBS_TWOPT1=${CL_GIBBS_TWOPT1:-1.0}
CL_GIBBS_TWOPT2=${CL_GIBBS_TWOPT2:-1.0}
```

and controls the process of imputing the hk genotypes of the reinserted markers.

The version of the map containing only unique markers is now in the finalgrps subdirectory, with the redundant version in finalredun.

Data Visualisation

crosslink_viewer is the main visualisation tool, which creates coloured plots showing linkage LOD and map distances between all the markers. It can display a single linkage group, a subset of a linkage group or multiple linkage groups at once. It can be set to begin centred on a named marker of interest, and clicking on the plot will output information about the markers to the terminal. The plot itself conveys information only through the colour and brightness of the pixels. For this reason it makes full use of the red, green and blue colour channels. An alternative colour scheme is available by editing the src/viewer_make.sh script and uncommenting (i.e. deleting the initial # from) the line AUX=”-DALTCOLSCHEME”, changing into the src directory and running the script as ./viewer_make.sh to recompile. This scheme may be more useful for people with red-green colour blindness. The file src/crosslink_viewer.h can also be edited to change the definition of the function setpixelrgb by simply rearranging the position of the variables r, g and b. The descriptions below assume the default colour scheme.

The visualisation programs can work your computer’s graphical system quite hard, especially for large datasets, and in my experience are more likely to cause crashes or other problem than the rest of Crosslink. For this reason make sure you have saved any other work you are doing before running them.

crosslink_viewer works by loading the marker data, interpreting it as either unphased, phased or imputed depending on the --datatype option, then rendering three versions of the plot: maternal, paternal and combined information. The plots are fully rendered in memory before being displayed on screen, which can cause a considerable delay before anything appears on screen for large datasets. An all-versus-all marker comparison is performed, meaning that for N markers on the order of N^2 memory and time will be required. The plots are created at full resolution, using the graphics card for scaling, panning and zooming. For this reason you may wish to limit the number of markers you load at once, at least to begin with, or use a computer with more video memory.

To illustrate data visualisation we will go through most of the steps of the example_pipelines/build_rgxha.sh pipeline again, but using only one of the linkage group files as an example. Recall that the first step of the Redgauntlet x Hapil example pipeline formed linkage groups using a simple LOD threshold, and that 21 instead of the expected 28 linkage groups were formed. This was caused by some hkxhk markers which had strong linkage to two different linkage groups, resulting in two or more true linkage groups remaining joined together. The example file we will use here is sample_data/000.loc.gz which were created by running cl_group.sh on the sample dataset sample_data/rgxha.loc.gz with a LOD threshold of 7. The file 000.loc grouped correctly at the first try. After running through all three of the quick start guide sections you will have the knowledge to visualise the groups which did not group correctly to begin with and follow them through the steps of the Real Data example pipeline example_pipelines/build_rgxha.sh.

We will begin by processing 000.loc through all the necessary steps. Starting from a clean working directory, make sure the path to Crosslink is correctly set, changing the directory names as required to suit the install location on your machine:

```
echo ${PATH}   #if crosslink is not already in the path then add it as follows

CROSSLINK_PATH=/home/crosslink_user/crosslink #modify as required

export PATH=${CROSSLINK_PATH}/bin:${CROSSLINK_PATH}/scripts:${PATH}
```

Then run the data through the required pipeline steps (note that because the grouping is already correct we do not need to run cl_detect_crosslg.sh or cl_removemarkers.sh at any point):

```
#copy the configuration files

cp -r ${CROSSLINK_PATH}/sample_data/rgxha_conf ./conf

#get a working copy of the sample genotype file

zcat ${CROSSLINK_PATH}/sample_data/000.loc.gz > 000.loc

#run the pipeline

cl_fixtypes.sh   000.loc   000.fix   conf/fixtypes.000

cl_findredun.sh   000.fix   000.redun   conf/findredun.000

cl_knnimpute.sh   000.fix   000.imp   conf/knnimpute.000

cl_extract.sh   000.imp   000.redun   000.uniq

mkdir -p 000.dir

cp 000.uniq 000.dir/000.loc

cl_order_hkimpute.sh   000.dir   000.final   conf/orderhkimpute.000

cl_mappos.sh   000.final   000.final

cl_reinsert_loc.sh   000.final   000.imp   000.redun   000.finalredun   conf/reinsert.000

cl_reinsert_map.sh   000.final   000.redun   000.finalredun
```

To view only the first 100 markers of this linkage group:

```
crosslink_viewer --inp=000.loc --datatype=phased --total=100
```

This plot should display at the default window size of 1000x1000 pixels. If the window is too large or small for your monitor press ‘q’ to quit and restart using the --window_size option to adjust the window size:

```
crosslink_viewer --inp=000.loc --datatype=phased --total=100 --window_size=800
```

The cursor keys pan the plot, plus and minus zoom in and out, and ENTER re-centres the plot. The combined plot is shown first, press ‘m’ or ‘p’ to switch to the maternal or paternal plots and ‘c’ to return to the combined view.

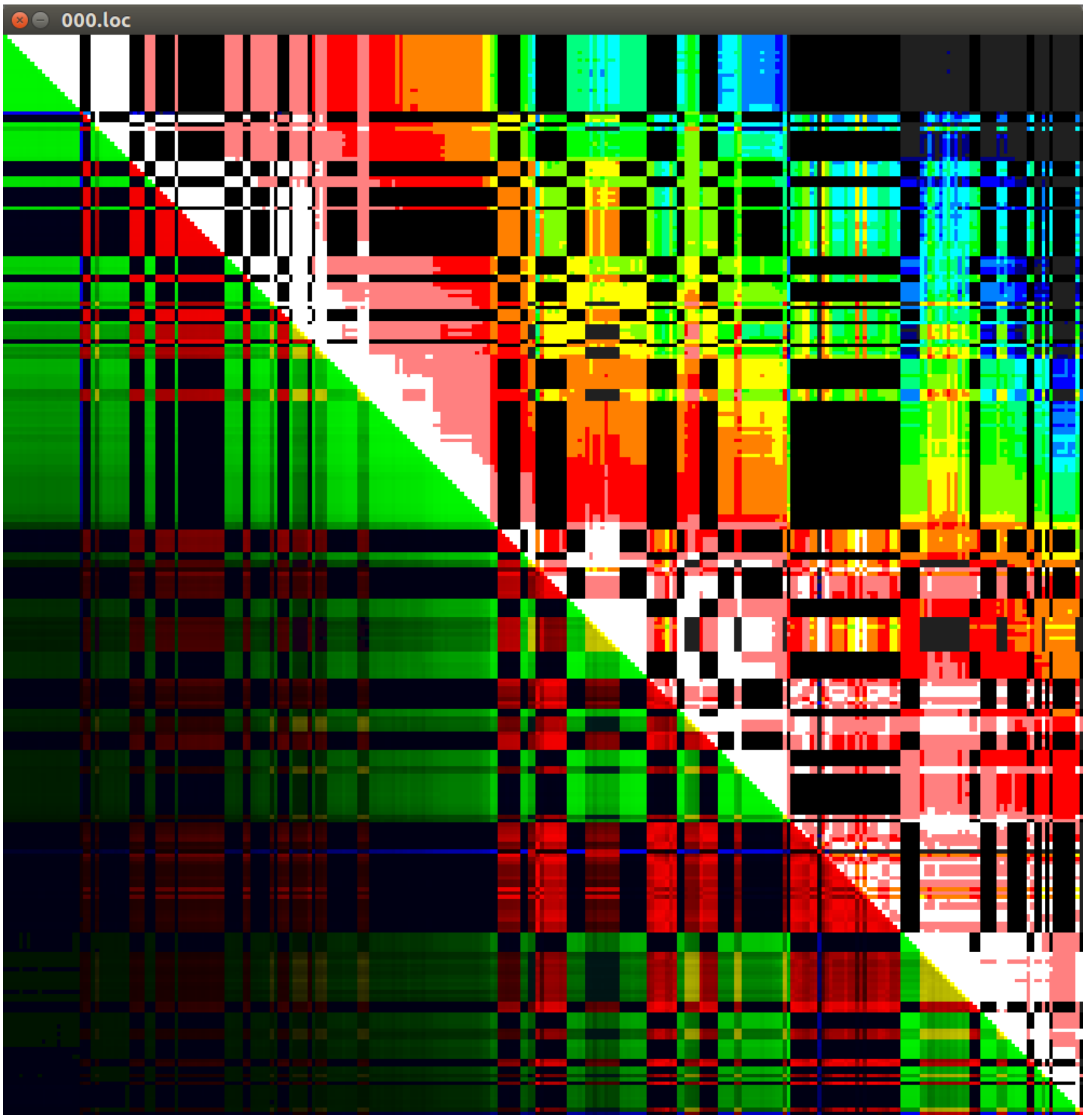
To view the next 100 markers in the linkage group we would use:

crosslink_viewer --inp=000.loc --datatype=phased --skip=100 --total=100

To view all the markers at once:

crosslink_viewer --inp=000.loc --datatype=phased

which should look like this:



The markers are represented as coloured squares along the main diagonal with the first at the top left and the last at bottom right. Maternal (lmxll) markers are red, paternal (nnxnp) green and shared (hkxhk) yellow. These squares are always shown at full brightness. Click on a marker, making sure to click only on one of the squares along the main diagonal (you may need to zoom in to be sure you have clicked on the marker you intend) and information about the marker will appear in the terminal, for example:

Xaxis: (0)NMH01-89891327 nnxnp {-0}...
Yaxis: (0)NMH01-89891327 nnxnp {-0}...
Comp: pat:N=162 rf=0.00000 lod=48.77

This tells us the linkage group (0), the marker name (NMH01-89891327), type (nnxnp) and phase ({-0}). In this case we get the same information twice, because, as we clicked on the main diagonal, both the row and column we clicked on correspond to the same marker. We therefore get a comparison of the marker with itself, telling us the paternal information (pat) has N=162 non missing genotype values with an rf of zero (of course) and a LOD of 48.77. There is no maternal information as this is an nnxnp marker.

Now click away from the main diagonal. We get information comparing the markers corresponding to the row and column that we clicked on, for example:

Xaxis: (0)NMH01-89891327 nnxnp {-0}...
Yaxis: (0)PHR11-89818133 hkxhk {00}...
Comp: pat:N=76 rf=0.03947 lod=17.39

This indicates that there were N=76 genotypes available for the comparison, (before hk genotypes have been imputed they are not available and must be treated as missing), showing an rf of 0.03947 corresponding to a LOD of 17.39.

Below the main diagonal the square colour shows information about the rf, LOD and phasing between the two markers. In general, to find the square showing the rf/LOD/phasing information between two markers, find the first (higher) of the two markers and move down its column until you reach the row corresponding to the second of the two markers. This square's red channel shows the maternal information and the green channel the paternal. The LOD is represented by the brightness, so a brighter square means a larger LOD. Squares corresponding to a pair of lmxll markers therefore indicate the LOD between them as the red brightness, squares corresponding to a pair of nnxnp markers as the green brightness and squares corresponding to a pair of hkxhk markers as the red and green brightness, resulting in a yellow square. Comparing an lmxll with an hkxhk allows only the maternal information of the hkxhk to be used, therefore the square will be red, likewise an nnxnp and an hkxhk comparison can only use the paternal information and is represented as a green square.

Comparing an lmxll with an nnxnp marker does not normally make sense, as the markers should not show any linkage. However, due to the polyploid nature of the strawberry, some markers are misclassified as lmxll when they are actually nnxnp and vice versa. To visualise this phenomenon crosslink_viewer *does* perform a linkage analysis between the maternal information of all lmxll and the paternal information of all nnxnp markers. These squares show the linkage LOD with the blue brightness, making it easier to pick out these likely errors.

Blue is also used to indicate likely phasing errors (for comparisons other than lmxll versus nnxnp). When treating the data as phased by passing option --datatype=phased, if the rf value between two markers is in disagreement with their declared phases, this may indicate a phasing error. For example, if two linked lmxll markers have been assigned different phases (i.e. one is {0-} and the other is {1-}) then we expect the rf to be greater than 0.5, conversely if they have the same phase we expect the rf to be less than 0.5. If the rf is found to be in agreement with the phase of the two markers then the usual colour channel is used, whereas if a disagreement is found the colour channel is switched to blue. Therefore an lmxll to lmxll square would be blue instead of red and an nnxnp to nnxnp blue rather than green. For an hkxhk to hkxhk comparison there are two independent phases. If both checks fail the square changes from yellow to blue, if only the maternal phase check fails the red channel changes to blue but the green channel stays green resulting in a cyan colour, if only the paternal check fails the green channel changes to blue resulting in a purple colour.

If you zoom in a little and pan along the main diagonal you should notice that three of the markers are showing bright blue squares against some of the other markers. Click on some of the blue squares below the main diagonal and see if you can work out what is going on from the rf, LOD and marker type information displayed to the terminal. Three lmxll markers are showing a strong linkage to many of the nnxnp markers, causing the corresponding squares to appear bright blue, but lack any strong linkage to other lmxll markers, causing the corresponding squares to appear almost black, all of which indicates they are likely misclassified nnxnp markers. There are no other bright blue squares, suggesting that the remaining markers do not have any obvious phasing errors.

Summary of crosslink_viewer colour scheme (if compiled using default colour mode) when viewing phased data, for the bottom left half of the plot which shows rf/LOD/phasing information. This applies to combined mode (press ‘c’ to switch to this mode). Increasing brightness indicates increasing linkage LOD.

Marker 1	Marker 2	Correct Phasing	Incorrect Phasing
lmxll	lmxll	red	blue
nnxnp	nnxnp	green	blue
hkxhk	hkxhk	both correct => yellow	**maternal error => cyan** **paternal error => purple** **both in error => blue**
lmxll	hkxhk	red	blue
nnxnp	hkxhk	green	blue
If *one* of the two markers has been incorrectly typed:			
lmxll	nnxnp	blue	blue

If we are viewing fully hk imputed data using the --datatype=imputed option the colour scheme is the same as the above. If we are viewing unphased data using the --datatype=unphased option then the colour scheme is interpreted slightly differently. Because there are no declared phases to be check now the blue channel is simply used to represent where the rf value is greater than 0.5 indicating a likely repulsion phase. So in the table above we simply change the meaning of “correct phasing” to “coupling phase” and “incorrect phasing” to “repulsion phase”. So for an hkxhk to hkxhk comparison a yellow square would indicate coupling phase in both maternal and paternal genomes for example.

If viewing in maternal or paternal mode (by pressing ‘m’ or ‘p’) then only information about the respective parent will be shown so, for example, in maternal mode an hkxhk to hkxhk square will appear as red, showing only information about the maternal part of the genotypes.

The bottom left half of the plot therefore shows information about rf, LOD and phasing, and is useful for detecting grouping, phasing and marker typing errors. However the implied map distance between two markers is not linearly related to the LOD, making subtle problems with marker ordering difficult to spot. Above the diagonal a colour scheme is used to represent the centimorgan distance between the markers which is designed to make problems with the marker ordering easier to see. The rf value is converted into a centimorgan distance using the Haldane mapping function and displayed on a heatmap scale where white represents the shortest distances and grey the longest, with black indicating no information (between lmxll and nnxnp markers).

Colour scheme used in the top right half of the crosslink_viewer plot, which displays map distance information, calculated using the Haldane mapping function.

Colour	Distance Range (cM)
white	< 10
pink	10 - 20
red	20 - 30
orange	30 - 40
yellow	40 - 50
yellow-green	50 - 60
green	60 - 70
green-cyan	70 - 80
cyan	80 - 90
cyan-blue	90 - 100
blue	100 - 110
dark blue	110 - 120
grey	> 120
black	no information

000.fix contains the same markers after typing errors were fixed using cl_fixtype.sh, view it now using:

crosslink_viewer --inp=000.fix --datatype=phased

which should look the same as the previous plot except that the erroneously typed markers have now been recoded as the correct type. To check this view the uncorrected plot, click on an affected marker to get its name, then view the corrected file using the above command but add the --marker option to specify that the affected marker should appear in the centre of the view, e.g.:

crosslink_viewer --inp=000.fix --datatype=phased --marker=NMH10-89914030

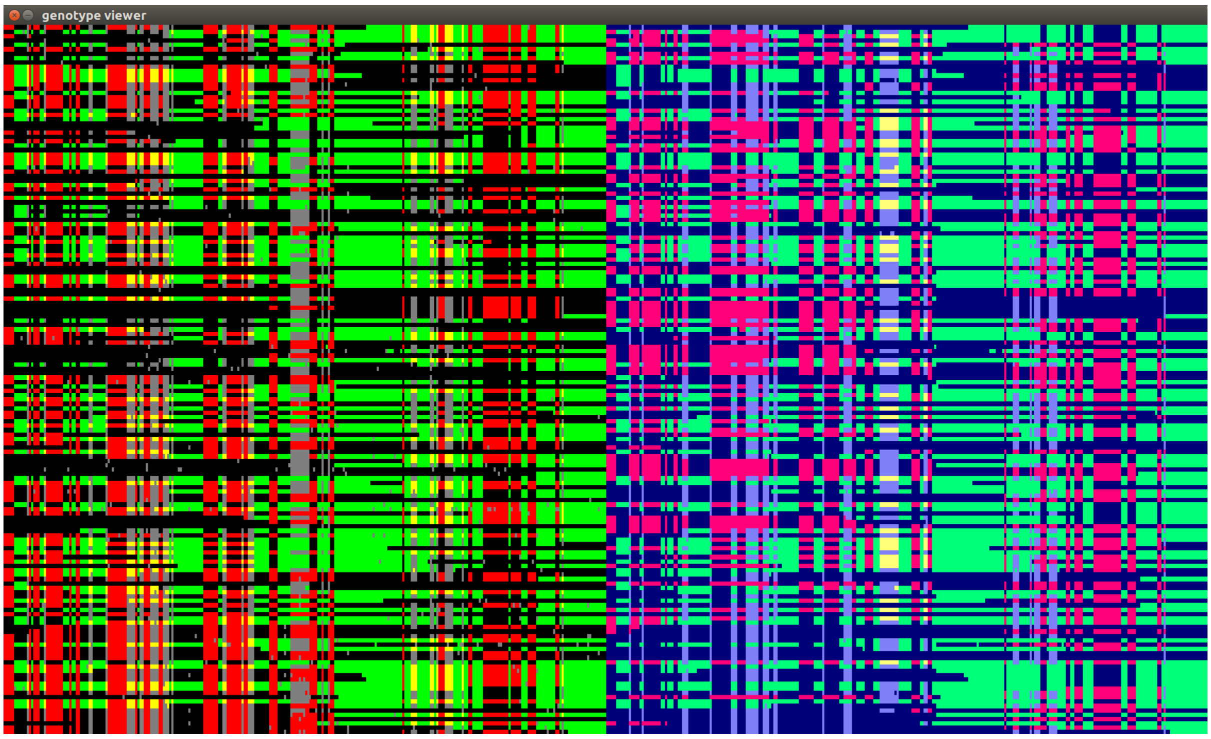
The next steps identified redundant markers (cl_findredun.sh), imputed missing genotype calls (cl_knnimpute.sh) and extracted the imputed versions of the non-redundant markers (cl_extract.sh) producing 000.uniq which contains 126 instead of 284 markers, and contains no missing calls. View it using:

crosslink_viewer --inp=000.uniq --datatype=phased

To compare the data before and after missing calls have been imputed we can use the crosslink_graphical command, which displays a simple colour coded representation of the genotype values. The following command will open 000.fix and 000.imp (the data before and after missing value imputation) side-by-side in a window 1700x1000 pixels. You may wish to adjust the size to fit your monitor:

crosslink_graphical --inp="000.fix 000.imp" --datatype=phased --sizex=1700 --sizey=1000

which should look like this (this figure shows the default combined-phased mode):



Each row represents a progeny and each column a marker. The slight blue colouration on the right indicates that these columns are the data from the second genotype file (000.imp). Use 'm', 'p' and 'c' to change between maternal, paternal and combined views, and 'z' to which between phased and unphased views. In *unphased* view mode a red or green colour represents an m or p allele for an lmxll or nnxnp marker respectively, whereas a black or dark blue rectangle represents the complementary allele (l or n); and for hkxhk markers yellow, red, green and black/dark blue represent the genotypes kk, kh, hk and hh respectively. For data lacking imputed hk information, as in the present example, grey is used to represent hk/kh genotypes. In *phased* mode red and green represent a phase 0 allele m or p or a phase 1 allele l or n. For hkxhk markers the red and green channels represent the maternal and paternal information, so the red channel is activated if there is a maternal phase 0 k or phase 1 h, and the green channel activated if there is a paternal phase 0 k or phase 1 h. If red and green are active a yellow colour results, if neither are active a black / dark blue colours results. Grey also represents any missing genotype calls.

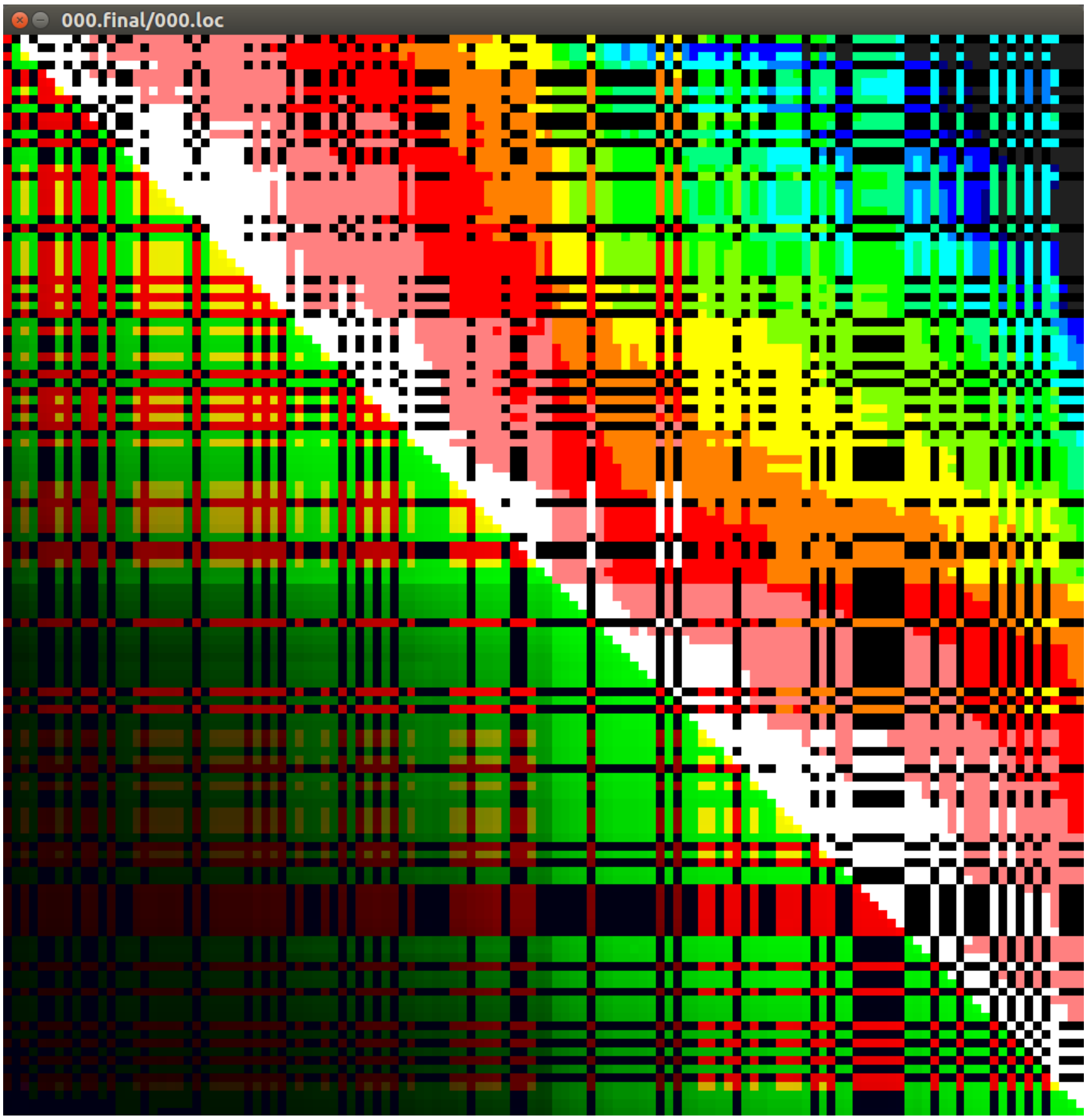
As with crosslink_viewer plus/minus perform zooming, cursor keys pan and ENTER recentres the view. Shift combined with a cursor key can be used to shrink or grow the plot in one direction only. You should be able to see a few grey squares in the left plot (000.fix), these represent the missing values which have all been imputed in the right plot. Identifying markers/progeny by clicking on them is not yet implemented in this program.

As a quirk of the way that the helper script pipeline works by calling crosslink_group internally, the process of imputing missing values actually also repeats the process of phasing and approximate ordering. (The advantage of breaking the crosslink_group functionality down into separate steps is that the full dataset can be used to impute missing values at the same time as identifying redundant markers in the non-imputed data, so that markers with less missing data are retained preferentially). You will therefore be able to see in the above figure that the maternal information in 000.fix is phased using the opposite convention to those in 000.imp (i.e. phase 0 in 000.fix is called phase 1 in 000.imp), whereas the paternal markers have been assigned the same phasing convention in both files (you may have experienced a different outcome when you ran the pipeline, as a small degree of randomness can arise in imputing missing values). The markers are also in a slightly different order. The get around this problem press ‘m’ to switch to maternal-only view, then press ‘z’ to switch to unphased view. Now both datasets should look almost the same, except for the slight difference in ordering.

The final version of the non-redundant markers was produced using cl_order_hkimpute.sh to refine the map order and impute the hk genotype missing information. View this version of the file using:

crosslink_viewer --inp=000.final/000.loc --datatype=imputed

which should look like this:

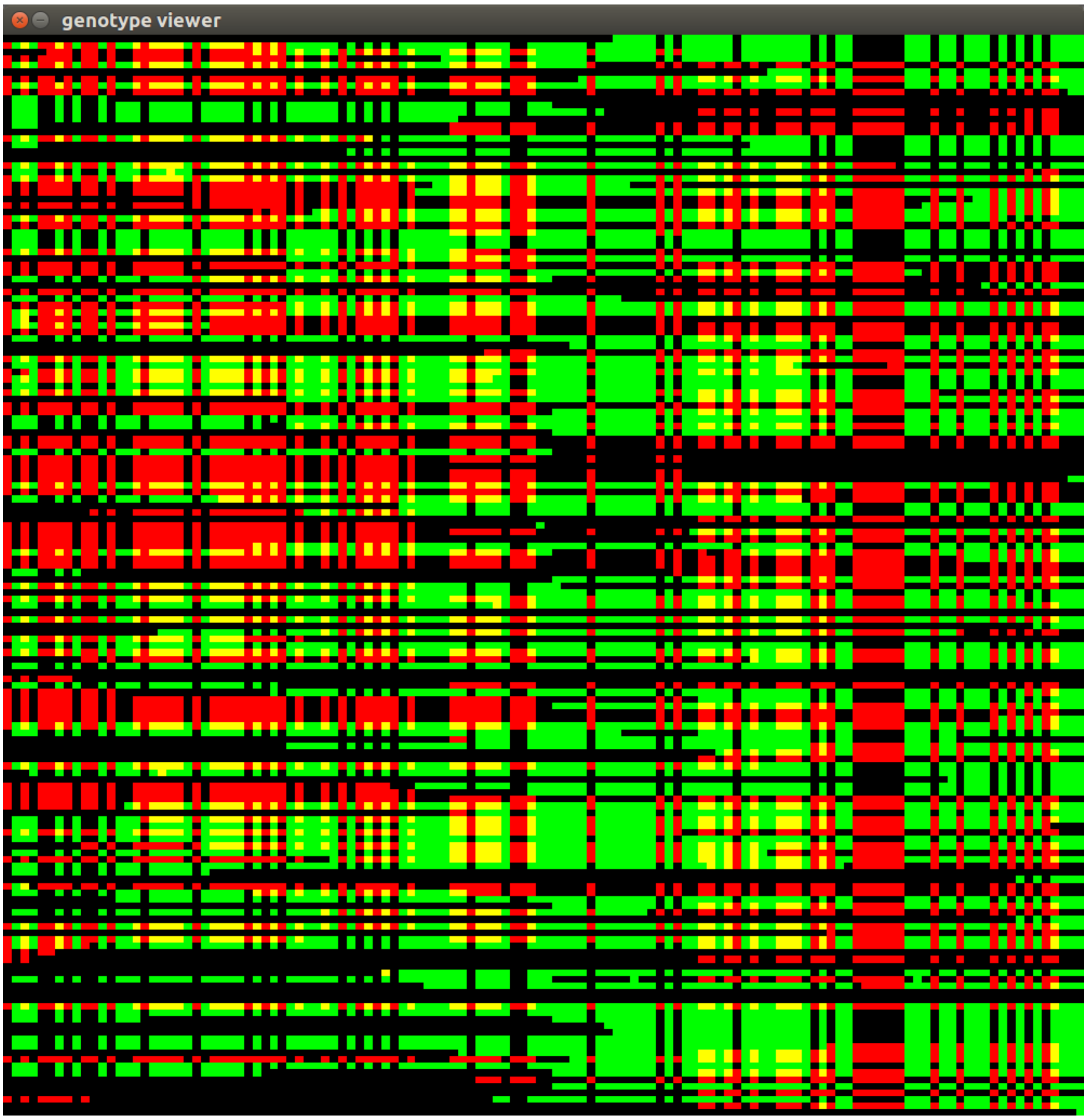


To check the ordering switch to maternal-only view by pressing ‘m’ and check that the map distances increase monotonically away from the main diagonal, then repeat this process for the paternal map by pressing ‘p’.

View the same data in crosslink_graphical using:

```
crosslink_graphical --inp=000.final/000.loc --datatype=imputed
```

which should look like this:



Switch between combined, maternal and paternal views (by pressing ‘c’, ‘m’ and ‘p’) to check that the ordering and hk imputation seem correct (there will be a few remaining errors as Crosslink does not currently support error correction).

Finally the redundant markers were reinserted into the ordering producing the redundant version of the final ordering, viewable using:

crosslink_viewer --inp=000.finalredun/000.loc --datatype=imputed

As well as viewing single files, crosslink_viewer can display multiple genotype files at the same time. To view all the linkage groups from the build_rgxha.sh pipeline at the same time (be aware this will use a lot more video memory than the previous single linkage group examples) we would use a command like this:

crosslink_viewer --inp=”$(echo initgrps/.loc)” --datatype=phased*

To view each linkage group one at a time:

```
for x in initgrps/*.loc
do
        crosslink_viewer --inp=${x} --datatype=phased || break
done
```

In this example pressing ‘q’ quits the current linkage group and moves on to the next one whereas pressing ‘escape’ breaks out of the for loop without viewing anymore. This works because crosslink_viewer returns an exit status (accessible in bash as the $? variable) of 0 if ‘q’ is pressed, but a status of 100 if ‘escape’ is pressed. If you have a large batch of file to view and want to automatically process each one differently depending on a decision made by a user viewing the files, crosslink_viewer also exits when any of the number keys 0-9 are pressed with an exit status of ten plus the number of the key pressed.

Export To JoinMap® Compatible Files

Commands and Options

crosslink_group

This program, implemented in C, performs the first part of the mapping pipeline, consisting of the following steps:

(1) Two-point rf and LOD Calculation (all markers)

This step estimates rf and LOD between all pairs of markers using the so-called two-point rf method (i.e. by treating all hk genotypes of hkxhk markers as missing). Consider each of n markers as a vertex in a graph. Proceed to create a fully connected graph where each of n^2 edges stores the estimated rf and LOD between the two associated vertices. To avoid consuming O(n^2) memory a minimum LOD threshold option, --min_lod, is provided. Any LOD below this threshold causes the edge to be dropped. As outlined above, hkxhk to hkxhk rf estimation sometimes fails to provide complete phasing information (cxr and rxc phase cannot be distinguished), but given sufficient marker density, phasing can likely be worked out indirectly via other markers. Therefore, an option, --ignore_cxr, is provided which causes such ambiguous edges to be dropped. In a very dense map nearby markers will often have identical genotype calls. To avoid retaining these redundant markers an option, --redundancy_lod, specifies a LOD threshold above which two markers are checked to see if they contain exactly identical genotype calls, or if one of them is identical except for missing values. If so one redundant marker is deleted, along with any associated edges, and no further rf/LOD values are estimated against it. To deal with lmxll markers incorrectly classified as nnxnp and vice versa an option, --matpat_lod, is provided. If enabled, rf and LOD values are calculated between the maternal information of lmxll markers and the paternal information of nnxnp markers, and an edge is retained if the LOD meets the specified threshold value (which must be greater than --min_lod), and will be used in steps (2) and (3) below. See supplement for rf and LOD formulae.

(2) Form Linkage Groups (all markers)

This step forms linkage groups consisting of all markers connected (directly or indirectly) by edges from step (1) (i.e. via LOD values above --min_lod or, if linkage between lmxll and nnxnp markers was enabled, above --matpat_lod). Consider each of the (non-redundant) markers initially each belonging to its own linkage group. Now perform union-find operations (union-find or disjoint-set is an efficient method to track elements partitioned into non-overlapping subsets [Algorithms book]) between each pair of markers linked by an edge from step (1). If they are found to belong to different linkage groups, the two groups are joined. Edges are processed in order of decreasing LOD (by first applying a quicksort to the edge list) so the minimum LOD which caused a change in the grouping can be reported in the log file if required. Edges are already filtered to the required LOD threshold so the ordering does not affect the final grouping obtained. If the --ignore_cxr option is disabled then cxr and rxc linkage between pairs of hkxhk markers contributes to linkage group formation even though it will not directly provide phasing information. Markers are split into their respective linkage groups which are processed separately for the remaining steps.

(3) Fix Marker-Typing Errors (per linkage group)

We encountered markers incorrectly classified as lmxll when they were actually nnxnp and vice versa, and found that markers of the PHR class were more reliable than those from the NMH class. The --matpat_lod option when set activates an error correction method to resolve this issue, and allows markers to receive a different weight based on the first letter of the marker name using the --matpat_weights option. Markers are initialised using a union-find structure with each assigned to a separate set, hkxhk markers are ignored completely. All edges from step (1) are used to perform join operations as was done with step (2). Because hkxhk markers and related edges are ignored this should result in separate sets for maternal and paternal markers. There may be more than one set for each marker type since the missing hkxhk markers may have broken the linkage group into smaller pieces. For each connected set a weighted sum of the marker types is calculated and all markers are converted to the type with the highest score.

(4) Phasing (per linkage group)

lmxll and nnxnp markers require one phase each whereas hkxhk markers require a maternal and paternal phase. The phases calculated here are purely relative, so that the same marker phase value could be assigned as 0 or 1, but should always be assigned the same phase relative to the other markers in its linkage group. Phasing can be estimated between any pair of markers that show linkage (except for cxr and rxc linkage between hkxhk markers) based on the rf value being greater than or less than 0.5, but is more reliable between markers with a high LOD value. Maternal and paternal phasing is performed separately, so that the same algorithm is run twice for each linkage group, hkxhk markers taking part in both runs.

Each marker in the linkage group is assigned as a disconnected vertex in a graph, and the first marker encountered is marked as the (arbitrary) starting point for phasing. The edge list from step (1) is quick-sorted into decreasing order by LOD, except that cxr and rxc edges (if permitted) are given lower priority than all other edges. Using Kruskal's algorithm, markers are joined into an MST (minimising the chances of a phasing error) using union-find operations starting with the highest priority edge, until all markers are joined into a single tree connecting all those requiring phasing for the current parent (maternal or paternal). A warning is given if producing a single tree required the use of any cxr/rxc type edges. Phasing is aborted with a warning if a single tree was not produced after all edges were used. The starting point marker phase is assigned to 0 and phasing is propagated throughout the tree using a depth-first search, propagating the same phase value across edges whose rf is less than 0.5 and switching to the other phase value where rf is greater than 0.5. If cxr/rxc edges are encountered the phase information is incomplete and is simply guessed as always being cxr (i.e. coupling in the maternal and repulsion in the paternal), which will be incorrect on average 50% of the time. If cxr/rxc linkage is required to correctly form linkage groups at a stringent LOD threshold but causes incorrect phasing, crosslink_group can be run a second time on each linkage group separately to rephase them excluding cxr/rxc linkage with a lower LOD threshold (helper scripts are provided to facilitate this process).

(5) Impute Missing Values (per linkage group)

Missing genotype calls are imputed using the k nearest neighbour algorithm, whereby missing calls are set to the mean value of the k most similar markers which do not lack the call. Only markers connected by edges from step (1) are considered as potential neighbours. Similarity is defined using a lower bound of the rf rather than the maximum likelihood rf to guard against overestimation between marker pairs with many missing values. Specifically the rf is estimated assuming missing values differ with a probability of 0.5 (mimicking unlinked markers). If no neighbours are found or there is a tie the value will be imputed to a random value and a warning written to the log file. With polyploid genomes often the loss of whole chromosomes can be tolerated, meaning some progeny could be aneuploid. If so all the marker calls for one chromosome could be missing, and would be filled with random values, which should be obvious from the log file or from examining the linkage group using crosslink_graphical. In such a case the linkage group should have the effected progeny removed and then be reprocessed.

(6) Approximate Marker Ordering (per linkage group)

The markers are approximately ordered using a modified version of the MST method employed by [MSTMap]. This involves building a map-distance minimising spanning-tree, extracting a longest path through the tree as the map backbone, then collapsing any remaining side branches onto the backbone at their point of attachment. This overall procedure is performed separately for maternal and paternal maps, which will then be merged in the next step. The MST is built using Kruskal's algorithm using the edge list from step (1) sorted into ascending order by inter-marker map distance (option --map_func selects either the Haldane or Kosambi function). As hk genotypes must be treated as missing at this stage in the mapping process, a simple implementation of the MSTMap's approach would cause the backbone to form along a path where missing hk values are masking the largest possible number of recombination events, and therefore likely distort the true ordering. To counter this Kruskal's algorithm is run with the edge list sorted so that edges connected to one or two hkxhk markers appear below those involving only lmxll and nnxnp markers, as the latter edges cannot mask recombination events. Since the linkage groups were formed using both maternal and paternal markers there is no guarantee that an lmxll-hkxhk or nnxnp-hkxhk only MST will span the full linkage group. If this is the case the approximate ordering is abandoned (crosslink_map can still be used to produce a final ordering). Once the MST is built a longest path is extracted using two depth-first searches. The first search starts at an arbitrary marker and finds the most distant marker from the starting point. The second search starts at this most distance marker and finds the marker most distant from it. The map backbone path is extracted by backtracking from this second marker to the first, and is traversed to assigned map positions to backbone markers. Finally, a depth-first search from each backbone marker of any side branch markers assigns them the same position as the associated backbone marker.

(7) Combined Map Positions (per linkage group)

A combined map is produced using the maternal and paternal maps created in the previous step, following the method used by [JoinMap]. In order to produce a useful combined map there must be at least two hkxhk markers otherwise it is not possible to work out the relative orientations of the two maps. A warning is written to the log file if any linkage group has less than 2 hkxhk markers. Otherwise, the covariance between the maternal and paternal positions of all hkxhk markers is found, and the paternal map positions are inverted if the covariance is negative. hkxhk marker combined positions are set to the average of their maternal and paternal map positions. lmxll and nnxnp marker combined positions are interpolated between the nearest two flanking hkxhk markers, or extrapolated from the nearest hkxhk if they are not flanked by two. Finally, the markers are sorted into the order defined by their combined map position.

crosslink_map

This program, implemented in C, performs the second part of the mapping pipeline, which produces the final map ordering, imputes the missing information for hk genotypes and calculated final map positions. It loads only one linkage group at a time. Multiple instances can therefore be run in parallel for a multi-linkage group map. The two main steps are ordering the markers using a GA and imputing the hk information using a Gibbs sampler. These steps are alternated as many times as required to let the state settle into a final order. Option --ga_gibbs_cycles controls the number of cycles of GA+Gibbs that are run, --ga_iters controls how many iterations of the GA are run each cycle, and --gibbs_burnin, --gibbs_samples and --gibbs_period control the iterations of the Gibbs sampler (see below). Final map positions are calculated following the method of [JoinMap], by averaging the maternal and paternal positions of the shared hkxhk markers, and interpolating/extrapolating the positions of lmxll and nnxnp markers based on the nearest hkxhk marker(s).

The --ga_skip_order1 option allows the first round of GA ordering to be skipped, so that the approximate ordering from crosslink_group will be used directly by step (2) before any further ordering changes are made. Conversely, if a completely new ordering is desired the option --randomise_order will cause marker order to be randomised before running the first round of ordering. If not skipped, the first round of ordering will use two-point rf values (treating hk genotypes as missing). All subsequent ordering rounds use multipoint rf values from the imputed hk information. Option --ga_use_mst causes ordering rounds (excluding the first) to begin with an approximate ordering of the markers using the MST method employed by crosslink_group before the ordering improvement algorithm is run. Because the ordering and imputing steps each depend on the other, some trial and error may be required to find the best parameter settings for a given dataset. The initial ordering given to the imputation step must be good enough that imputation can provide better genotype information than the two-point method, which will in turn allow the order to be further improved, and so on.

(1) Improve Marker Ordering With A Genetic Algorithm

The genetic algorithm works by scoring the optimality of the current marker ordering, making a random change to the ordering, then reassessing the ordering score. If the score improves or stays the same the new ordering is accepted, otherwise the change is rejected. The idea behind accepting neutral changes is that they may allow the ordering to escape from local optima [netcrawler algorithm]. This approach roughly follows [JoinMap], [tmap] and [MSTMap]. There are three available scoring methods, all of which take account of both maternal and paternal information simultaneously: (i) minimise the sum of recombination events in maternal and paternal maps (option --ga_optimise_meth=0), (ii) minimise the combined map lengths of the maternal and paternal maps, quantised to 100ths of a centimorgan (option --ga_optimise_meth=1), (iii) maximise a metric which rewards small recombination fractions between markers which are also nearby in the current ordering (option --ga_optimise_meth=2). The third method provides more of a "global" picture of the ordering optimality by comparing all against all markers (with a LOD threshold) instead of just adjacent ones, but is more computationally intensive for this reason.

The first two scoring methods make use of several optimisations to increase execution speed. Before ordering begins the current genotype values, including the current imputed hk information, are compressed into a bitstring representation (disabled using --bitstring=0) so that 64 progeny can be represented by a single integer value and processed simultaneously using bitwise operations such as XOR and popcount. Missing values are represented by a corresponding bitmask. When the score of the initial ordering is calculated the recombination events or map distances are calculated by scanning along the ordering and comparing adjacent markers. Each time a pair of markers are compared the calculated recombination count or map distance is cached (disabled using --ga-cache=0) so that next time those two markers are compared the value will not need to be recalculated. Two copies of the ordering are maintained called elite and mutant, both start with the same ordering. Mutation events are always defined on a block of (one or more) consecutive markers: by the first and last marker of the block, a flag specifying whether the block will be inverted and the position they will finally occupy after the mutation is completed. Before the markers are rearranged, the locations where the ordering will be broken are examined to determine how many recombination events or quantised map distance units will be lost. The rearrangement is carried out by copying blocks of (pointers to) markers from elite to mutant. Finally, the locations where new markers have been brought together are examined to determine how many new recombinations/distance units have been added. The new score is found from the previous score by subtracting the lost units and adding the gained units, and avoids having to scan the entire ordering for each mutation. If the mutation is rejected the affected portion of the ordering is copied from elite to mutant otherwise the affected portion is copied back from mutant to elite.

For the global scoring method, at the start of each GA round a list of rf and LOD values are calculated by comparing all markers against all others, those with LOD values lower than option --ga_minlod are dropped, the otherwise the rf values are converted into map distances. To find the overall score for a map ordering each map distance stored in the list is given a score which disproportionately rewards short map distances which are also nearby in the ordering. Longer map distances and/or larger separations in the ordering reduce the score, (see the Supplement for the equation used). The best scored is obtained by having as many short map distances correspond to nearby ordering positions as possible. This will penalise any strongly linked markers which are widely separated by the ordering even where all immediately adjacent pairs have low recombination counts. The overall score for the ordering is the sum of all the individual scores.

Three types of mutation operations are provided: single marker hop, segment move and segment inversion, chosen with probabilities controlled by options --ga_prob_hop and --ga_prob_move. A single marker hop moves a single marker a random distance up to a proportion of the total marker count controlled by option --ga_max_hop. A segment move moves a randomly chosen block of markers with a size from 1 up to a proportion of the total marker count controlled by option --ga_max_mvseg, a random distance up to a proportionate maximum controlled by option --ga_max_mvdist, and inverts the block with a probability --ga_prob_inv. A segment inversion always inverts a randomly chosen block of markers with a size of 2 up to a proportionate maximum controlled by option --ga_max_seg.

(2) Impute hk Information By Gibbs Sampler

For each hkxhk marker genotype called as hk it is unknown whether the h allele came from the maternal and the k allele from the paternal parent or vice versa. Without this information the rf values cannot be known as accurately, as all hk genotypes must be treated as missing. Using the convention of listing the maternal allele first, the two possible imputed forms of the genotype call will be written as HK and KH, using uppercase to indicate they have been imputed. The marker is present in both the maternal and paternal maps, therefore both need to be considered simultaneously. The most likely choice between HK or KH depends on the current marker ordering, whether either imply a recombination before and/or after the marker and the current recombination fractions between the previous and next markers in the maternal and paternal maps. The problem then is to find the most probable set of choices for all the hk genotypes considered jointly. A Gibbs sampler can address this problem by initially randomising all hk genotypes, and then resampling each individually in a random order many times. The final state of each is then set to the most common during resampling. Resampling an hk genotype entails assuming the correct state is unknown but that all other states are known and correct, calculating the probabilities of HK and KH given the neighbouring marker states, and assigning the state to either HK or KH with the respective probabilities.

Three implementations are provided all based on the basic Gibbs sampler algorithm: (A) a Gibbs sampler which resamples hk genotypes in a completely random order, (B) a Gibbs sampler which resamples states sequentially along the current map order one progeny at a time, treating the progeny in a random order, and in either forward or reverse map order at random (C), a simplified heuristic which resamples sequentially per progeny in the same manner as method B but only takes account of the preceding marker states as it progresses.

Each Gibbs cycle begins by randomising all hk genotype states. Then, during a burn-in period, the states are resampled a given number of times (controlled by option --gibbs_burnin) using one or more of methods A-C. A counter associated with each hk genotype is then increment if the state is currently HK or decremented if it is KH. Then the states are resampled again a given number of times (controlled by option --gibbs_period) and the counters updated again. This is repeated until the required number of samples (controlled by option --gibbs_samples) have been obtained. For example if --gibbs_burnin is 20, --gibbs_period is 1 and --gibbs_samples is 300 then every state will be resampled 20 times during the burn-in before the first counter update, then exactly once between all subsequent counter updates until each counter has been updated a total of 300 times. Therefore each Gibbs iteration consists of updating each hk state exactly once.

Each iteration the method of resampling is chosen from the three available methods (controlled by options --gibbs_sequential, the probability of using method B, and --gibbs_unidir, the probability of method C, the probability of A being defined implicitly by subtraction from 1), such that any one method can be used exclusively or a random mixture of some or all.

The initial burn-in period is needed to allow the states to converge from their initial random values towards a more optimal part of the state space. Some additional options are provided to fine tune the resampling process during burn-in. Firstly, to prevent the state getting stuck prematurely in a suboptimal part of state space, options --gibbs_min_prob_1 and --gibbs_min_prob_2 control the minimum probability that a resampled hk state will change to the other value. For the very first iteration of burn-in the minimum probability is --gibbs_min_prob_1, for the last burn-in iteration it is --gibbs_min_prob_2, and for intermediate iterations the probability changes linearly from the former to the latter. After burn-in, during normal iterations, the minimum probability remains at --gibbs_min_prob_2. At the start of burn-in, multipoint rf values are available immediately, but are initially determined from the randomly chosen hk states, consequently the two-point rf values may be more accurate until some point through the burn-in period. For this reason, when resampling hk states a weighted average of the multipoint and two-point rf values is used to calculate the state probabilities. Two control parameters are provided, --gibbs_twopt_1 and --gibbs_twopt_2, the weighting given to two-point values being changed linearly from the first to the second over the course of the burn-in period and remaining at the second value during normal iterations.

(3) Calculate Final Map Positions

This step follows the same procedure as the final step of the crosslink_group program.

crosslink_graphical

This program

Glossary

rf - recombination fraction, the proportion of progeny showing recombination between two markers

LOD - the (linkage) logarithm of odds, a measure of the statistical significance of the linkage between two markers, specifically the base-10 logarithm of the ratio of probabilities that the markers are truly linked versus unlinked