crimaptools: a tutorial

Susan Johnston

17 May 2016

This `crimaptools` package creates input files, runs CRI-MAP and parses output files for analyses of linkage mapping and recombination rate estimation in CRI-MAP v2.504. This packages is still in progress in terms of optimisation (particularly dealing with unusual CriMAP syntax), but should run if instructions are followed carefully. To download CRI-MAP, which is currently tirelessly maintained by Jill Maddox, go here. For more information on how to use it from the command line, check out Paris Veltsos's tutorial here.

The package can be installed using devtools:

```
library(devtools)
install_github("susjoh/crimaptools")
```

1. Example dataset.

`crimaptools` requires two inputs to create, run and parse files: * A GenABEL gwaa.data object containing genotype information for IDs * A pedigree object that specifies the individual families used in Crimap, with columns for the ANIMAL, FATHER, MOTHER and FAMILY. An example dataset from Red deer is included, and can be called using `data(deer)`

```
data(deer)

ls()
```

```
## [1] "deer.abel"   "deer.famped" "deer.ped"
```

```
deer.famped
```

```
##    ANIMAL FATHER MOTHER            Family
## 1    4440      0      0 Offspring_Mum_108
## 2    3018      0      0 Offspring_Mum_108
## 3    1169      0      0 Offspring_Mum_108
## 4    1806   1169   3018 Offspring_Mum_108
## 5     108   4440   1806 Offspring_Mum_108
## 6    3049      0      0 Offspring_Mum_109
## 7    1289      0      0 Offspring_Mum_109
## 8     329      0      0 Offspring_Mum_109
## 9    1911    329   1289 Offspring_Mum_109
## 10    109   3049   1911 Offspring_Mum_109
## 11   1303      0      0 Offspring_Mum_110
## 12   1289      0      0 Offspring_Mum_110
## 13    329      0      0 Offspring_Mum_110
## 14   1911    329   1289 Offspring_Mum_110
## 15    110   1303   1911 Offspring_Mum_110
## 16   1920      0      0 Offspring_Mum_111
## 17   1289      0      0 Offspring_Mum_111
## 18    329      0      0 Offspring_Mum_111
## 19   1911    329   1289 Offspring_Mum_111
## 20    111   1920   1911 Offspring_Mum_111
```

2. Creating a CRI-MAP input file.

This is done using the function create_crimap_input. This requires the `deer.abel` and `deer.famped` objects, but also requires additional inputs such as `analysisID`, used as a flag for running CRI-MAP. An ordered list of SNP loci (`snplist =`) *or* a chromosome number `chr =` must be specified. It is also possible to specify the directory to which the output should be written, and `clear.existing.analysisID` will get rid of any previous builds carried out with the same analysisID. Let's run the analysis for chromosome 3, and give it the ID `3a`.

```
library(crimaptools)

create_crimap_input(gwaa.data = deer.abel,
                    familyPedigree = deer.famped,
                    analysisID = "3a",
                    chr = 3,
                    outdir = "crimap",
                    clear.existing.analysisID = TRUE)
```

```
## Recoding alleles to numeric values...

## ...done

## Merging pedigree and genotype information...

## ...done.

## Parsing and writing to crimap/chr3a.gen...

## ...done
```

In the directory `crimap`, an input file `chr3a.gen` will have been created.

3. Run *prepare* and extract and deal with mendelian errors.

The .gen file must now be run through *prepare* to produce the field used for linkage mapping and crossover estimation. This can be done using the function `run_crimap_prepare`. At present this inefficiently has to define the path to the CRI-MAP exectuable (in future versions it will be bundled with the library). At the moment, the exectuable path is specified relative to the .gen file - for example, below, the CRI-MAP executable is in the same directory as the .gen file:

```
run_crimap_prepare("crimap2504.exe", "crimap/chr3a.gen")
dir("crimap")
```

```
## [1] "chr3a.dat"      "chr3a.gen"      "chr3a.loc"      "chr3a.par"
## [5] "chr3a.pre"      "crimap2504.exe" "crimapinput1"
```

The function has produced the .pre, .loc, .par and .dat files. The .pre file will contain information on mendelian errors between parents and offspring, which can be extracted using the `parse_mend_err` function. This creates a further file with the extension ".mnd", which has a column for each ID and the problematic locus:

```
parse_mend_err("crimap/chr3a.pre", "crimap/chr3a.gen", familyPedigree = deer.famped)
```

```
## Writing new file crimap/chr3a.mnd
```

```
read.table("crimap/chr3a.mnd", header = T)
```

```
##   ANIMAL             SNP.Name
## 1    111 cela1_red_3_74862326
## 2   1920 cela1_red_3_74862326
```

Information from this file can be used to mask mendelian errors in the .gen file, by rerunning the `create_crimap_input` with `use.mnd = TRUE`:

```
create_crimap_input (deer.abel, deer.famped, analysisID = "3a",
                     chr = 3, outdir = "crimap", clear.existing.analysisID = TRUE,
                     use.mnd = TRUE)
```

```
## Recoding alleles to numeric values...

## ...done

## Merging pedigree and genotype information...

## ...done.

## Parsing and writing to crimap/chr3a.gen...

## ...done
```

```
run_crimap_prepare(crimap.path = "crimap2504.exe", genfile = "crimap/chr3a.gen")

parse_mend_err(prefile = "crimap/chr3a.pre", genfile = "crimap/chr3a.gen", familyPedigree = deer.famped
```

```
## No Mendelian errors detected. No changes made to .mnd file.
```

4. Build a linkage map.

Now the dataset is ready for linkage mapping and characterisation of crossovers. This assumes the order of the deer.abel dataset when chromosome is specified; if snplist was specified in `create_crimap_input` then it assumes the order of the snplist itself. The linkage map can be obtained by running `run_crimap_map`, which may be slow for large numbers of markers/families, and then parsed with `parse_map`, which provides sex specific maps:

```
run_crimap_map(crimap.path = "crimap2504.exe", genfile = "crimap/chr3a.gen")
dir("crimap")
```

```
## [1] "chr3a.dat"      "chr3a.gen"      "chr3a.loc"      "chr3a.map"
## [5] "chr3a.ord"      "chr3a.par"      "chr3a.pre"      "crimap2504.exe"
## [9] "crimapinput1"
```

```
deer.map <- parse_map(mapfile = "crimap/chr3a.map")
head(deer.map)
```

```
##    Order             SNP.Name cMPosition.Female cMPosition.Male Female.r
## 1      0  cela1_sika_3_247473             0.000               0    0.000
## 3      1 cela1_sika_3_1709255             0.000               0    0.211
## 5      2 cela1_sika_3_1763550            22.546               0    0.211
## 7      3 cela1_sika_3_2122831            45.091               0    0.000
## 9      4  cela1_red_3_3089490            45.091               0    0.000
## 11     5  cela1_red_3_3118111            45.091               0    0.000
##    cMdiff.Female Male.r cMdiff.Male analysisID
## 1          0.000      0           0         3a
## 3         22.546      0           0         3a
## 5         22.546      0           0         3a
## 7          0.000      0           0         3a
## 9          0.000      0           0         3a
## 11         0.000      0           0         3a
```

5. Characterising recombination events.

The recombination events can be obtained by running `run_crimap_chrompic`, which again may be slow for larger numbers of markers/families. A sex averaged linkage map can be parsed with `parse_map_chrompic` and crossovers can be extracted with `parse_crossovers`, which also requires the family pedigree:

```
run_crimap_chrompic(crimap.path = "crimap2504.exe", genfile = "crimap/chr3a.gen")

deer.cmpmap <- parse_map_chrompic(chrompicfile = "crimap/chr3a.cmp")
head(deer.cmpmap)
```

```
##    Order             SNP.Name cMPosition     r cMdiff analysisID
## 1      1  cela1_sika_3_247473      0.000 0.000  0.000         3a
## 3      2 cela1_sika_3_1709255      0.000 0.211 22.546         3a
## 5      3 cela1_sika_3_1763550     22.546 0.211 22.546         3a
## 7      4 cela1_sika_3_2122831     45.091 0.000  0.000         3a
## 9      5  cela1_red_3_3089490     45.091 0.000  0.000         3a
## 11     6  cela1_red_3_3118111     45.091 0.000  0.000         3a
```

```
deer.xovers <- parse_crossovers(chrompicfile = "crimap/chr3a.cmp", familyPedigree = deer.famped)
deer.xovers[1:2,]
```

```
##
## 1 ---1-----1--------1--1----1---1-1--1------11-1-1111--111-----1-----------------------------------
## 2 -0-----111--------1---1---1-11--1---1-1-1---1--1-----1-------1--1------1--1--------11--1--------
##   ANIMAL RecombCount parent            Family No.Inf.Loci First.Inf.Order
## 1    108           1 MOTHER Offspring_Mum_108         480               4
## 2    109           3 MOTHER Offspring_Mum_109         421               2
##   Last.Inf.Order FATHER MOTHER RRID analysisID
## 1           1845   4440   1806 1806         3a
## 2           1786   3049   1911 1911         3a
##                           UniqueID
## 1 3a_Offspring_Mum_108_1806_MOTHER
## 2 3a_Offspring_Mum_109_1911_MOTHER
```

Column `data` is the inheritance pattern from grandparents, with each character representing an ordered SNP. 0 is grandmaternal, 1 is grandpaternal. `RRID` is the parent in which the meiosis took place.

6. Investigating doubles crossovers.

Genotyping and/or phasing errors can lead to erroneous calls of double crossovers. These can be investigated by running `check_double_crossovers` on the parsed crossovers:

```
deer.doubles <- check_double_crossovers(parsed.xovers = deer.xovers)
```

```
## Splitting chromosome into segments of shared grandparental origin
```

```
head(deer.doubles)
```

```
##   Phase StartPos StopPos StartSpan StopSpan InfCount Segment Segment.Count
## 1     1        4      62         1      117       19       1             2
## 2     0      117    1845        62     1845      461       2             2
## 3     0        2       2         1        8        1       1             4
## 4     1        8     395         2      402       96       2             4
## 5     0      402     402       395      403        1       3             4
## 6     1      403    1786       402     1786      323       4             4
##    Type                          UniqueID            Family RRID parent
## 1 First 3a_Offspring_Mum_108_1806_MOTHER Offspring_Mum_108 1806 MOTHER
## 2  Last 3a_Offspring_Mum_108_1806_MOTHER Offspring_Mum_108 1806 MOTHER
## 3 First 3a_Offspring_Mum_109_1911_MOTHER Offspring_Mum_109 1911 MOTHER
## 4   Mid 3a_Offspring_Mum_109_1911_MOTHER Offspring_Mum_109 1911 MOTHER
## 5   Mid 3a_Offspring_Mum_109_1911_MOTHER Offspring_Mum_109 1911 MOTHER
## 6  Last 3a_Offspring_Mum_109_1911_MOTHER Offspring_Mum_109 1911 MOTHER
##   analysisID Singleton
## 1         3a        no
## 2         3a        no
## 3         3a       yes
## 4         3a        no
## 5         3a       yes
## 6         3a        no
```

This takes the phasing information and returns information on the Phase fragments per chromosome (i.e. runs from a single grandparent. Phase: 0 = grandmaternal, 1 = grandpaternal; StartPos and StopPos are the first and last informative positions of the fragment, StartSpan and StopSpan are the closest informative SNPs on either side of the fragment (if first or last, then is first or last SNP); InfCount is the number of informative SNPs in the fragment; Segment is the order of the fragment, numbered 1:N; Type is whether the fragment was the first, last or occured in the middle of the chromosome; RRID is the individual in which the meiosis occurred.

Physical map infomation for the markers can also be added by specifying map positions in a data frame with headers SNP.Name, Position, Order and analysisID:

```
library(GenABEL)
```

```
## Loading required package: MASS
```

```
## Loading required package: GenABEL.data
```

```
physmap <- data.frame(SNP.Name = snpnames(deer.abel)[chromosome(deer.abel) == 3],
                      Position = map(deer.abel)[chromosome(deer.abel) == 3],
                      Order = 1:length(which(chromosome(deer.abel) == 3)),
                      analysisID = "3a")


deer.doubles <- check_double_crossovers(parsed.xovers = deer.xovers, physical.map = physmap)
```

```
## Splitting chromosome into segments of shared grandparental origin
```

```
head(deer.doubles)
```

```
##   Phase StartPos StopPos StartSpan StopSpan InfCount Segment Segment.Count
## 1     1        4      62         1      117       19       1             2
## 2     0      117    1845        62     1845      461       2             2
## 3     0        2       2         1        8        1       1             4
## 4     1        8     395         2      402       96       2             4
## 5     0      402     402       395      403        1       3             4
## 6     1      403    1786       402     1786      323       4             4
##    Type                         UniqueID            Family RRID parent
## 1 First 3a_Offspring_Mum_108_1806_MOTHER Offspring_Mum_108 1806 MOTHER
## 2  Last 3a_Offspring_Mum_108_1806_MOTHER Offspring_Mum_108 1806 MOTHER
## 3 First 3a_Offspring_Mum_109_1911_MOTHER Offspring_Mum_109 1911 MOTHER
## 4   Mid 3a_Offspring_Mum_109_1911_MOTHER Offspring_Mum_109 1911 MOTHER
## 5   Mid 3a_Offspring_Mum_109_1911_MOTHER Offspring_Mum_109 1911 MOTHER
## 6  Last 3a_Offspring_Mum_109_1911_MOTHER Offspring_Mum_109 1911 MOTHER
##   analysisID StartPos.GenomePos StopPos.GenomePos StartSpan.GenomePos
## 1         3a            2122831           7067364              247473
## 2         3a           10792068         121375733             7067364
## 3         3a            1709255           1709255              247473
## 4         3a            3155501          29013193             1709255
## 5         3a           29656063          29656063            29013193
## 6         3a           29689850         117062248            29656063
##   StopSpan.GenomePos PosLength SpanLength Singleton
## 1           10792068   4944533   10544595        no
## 2          121375733 110583665  114308369        no
## 3            3155501         0    2908028       yes
## 4           29656063  25857692   27946808        no
## 5           29689850         0     676657       yes
## 6          117062248  87372398   87406185        no
```

This outputs additional columns with the genome positions for Pos and Span values, and also PosLength and SpanLength, which is the difference between the start and stop positions.

The user then has some choice of which lines may be erroneous. For example, singletons may be removed.

```
deer.remove <- subset(deer.doubles, Singleton == "yes")

deer.xovers.clean <- revise_double_crossovers(parsed.xovers = deer.xovers, removesections = deer.remove)
```

```
## [1] "Fixing Problem 1 of 3"
```

```
deer.xovers.clean
```

```
##
## 1 ---1-----1--------1--1----1---1-1--1------11-1-1111--111-----1-----------------------------------
## 2 -------111--------1---1---1-11--1---1-1-1---1--1-----1-------1--1------1--1--------11--1---------
## 3 -1-----111----------------1-10--0---0-0-0---0--0-0---0---00--0--0------0-------0---00--0---------
## 4 -0-----000--------1---1---------1----------1----1---1---11--1--1------1--1----1---11--1---------
##   ANIMAL RecombCount parent            Family No.Inf.Loci First.Inf.Order
## 1    108           1 MOTHER Offspring_Mum_108         480               4
## 2    109           0 MOTHER Offspring_Mum_109         421               2
## 3    110           2 MOTHER Offspring_Mum_110         421               2
## 4    111           1 MOTHER Offspring_Mum_111         417               2
##   Last.Inf.Order FATHER MOTHER RRID analysisID
## 1           1845   4440   1806 1806         3a
## 2           1786   3049   1911 1911         3a
## 3           1786   1303   1911 1911         3a
## 4           1778   1920   1911 1911         3a
##                           UniqueID
## 1 3a_Offspring_Mum_108_1806_MOTHER
## 2 3a_Offspring_Mum_109_1911_MOTHER
## 3 3a_Offspring_Mum_110_1911_MOTHER
## 4 3a_Offspring_Mum_111_1911_MOTHER
```