BatchMap algorithm for the creation of high density linkage maps in outcrossing species

Bastian Schiffthaler and Carolina Bernhardsson

2017-03-08

Introduction

In general, the reader is encouraged to go through the excellent documentation of the original OneMap package before going through this vignette. An up-to-date version can be found here. The majority of the pipeline still works the same or very similar to the implementations in OneMap (also internally). For those already familiar with the original or those looking for a quick summary, feel free to go on.

NOTE: BatchMap has been written specifically for use in outcrossing species. All OneMap functionality pertaining to back-crosses, f2, ril etc. has been removed for the sake of easier code maintenance. If your use case is not an outcrossing F1 population, turn back now.

Reading data into R

BatchMap keeps with the paradigm and format of the original OneMap data format, but includes a faster function for reading the input file `read.outcross2`. Further, BatchMap ignores all lines following the marker definitions (e.g. phenotypes) as all exploration beyond the construction of the linkage map is not intended to be handled by this package.

```
suppressPackageStartupMessages(library(BatchMap))

input_file <- system.file("example/sim2k.txt",package = "BatchMap")
outcross <- read.outcross2(input_file)
```

```
## Reading data... [.2%..5%..7%..10%..12%..15%..17%..20%..22%..25%..27%..30%..32%..35%..37%..40%..42%..
```

```
outcross
```

```
##   This is an object of class 'outcross'
##     No. individuals:    800
##     No. markers:        2279
##     Segregation types:
##        B3.7: 575
##        D1.10:    880
##        D2.15:    824
##     No. traits:         0
```

Detecting bins and resolving them

High density marker data often has bins of identical markers, which cause problems when estimating recombination fractions, and can in the case of the BatchMap approach make the resulting map worse. OneMap provides functions to detect and resolve such bins. Note the `exact` option to `find.bins()`, which controls wether missing information should be considered when binning data:

```
bins <- find.bins(outcross, exact = FALSE)
outcross_clean <- create.data.bins(outcross, bins)
outcross_clean
```

```
##   This is an object of class 'outcross'
##     No. individuals:    800
##     No. markers:        2085
##     Segregation types:
##        B3.7: 563
##        D1.10:    797
##        D2.15:    725
##     No. traits:         0
```

Note the difference in the number of markers.

Calculating the twopoint table

The function `rf.2pts()` calculates the twopoint table for markers. Note that with very high density datasets, a lot of RAM can be required to hold the twopoint table. As a general rule, this datastructure will require $M * M * 32$ bytes, where M is the number of markers. In our case, with a small dataset of 2,085 markers, we'll need about 133Mb. A large dataset of 20,000 markers will need **>48Gb**. This would be typically run on a server machine (e.g. see some cloud server providers).

```
twopt_table <- rf.2pts(outcross_clean)
```

```
## Computing 2172570 recombination fractions:
##
## [.2%..5%..7%..10%..12%..15%..18%..20%..23%..25%..28%..30%..33%..35%..37%..40%..42%..45%..47%..50%..5
```

```
# Check the size
format(object.size(twopt_table),units = "Mb")
```

```
## [1] "133.6 Mb"
```

Grouping

In order to separate the data into linkage groups, we use the `group()` function:

```
linkage_groups <- group(make.seq(input.obj = twopt_table, "all"),
                        LOD = 12)
```

```
##    Selecting markers:
##    group    1
##     ............................................................
##     ............................................................
##     .......................
##    group    2
##     ............................................................
##     ............................................................
##     ...............................
##    group    3
##     ............................................................
##     ............................................................
##     ............................
##    group    4
##     ............................................................
##     ............................................................
##     ..............
##    group    5
##     ............................................................
##     ............................................................
##     .........
##    group    6
##     ............................................................
##     ............................................................
##     ..............
##    group    7
##     ............................................................
##     ............................................................
##     .......
##    group    8
##     ............................................................
##     ............................................................
##     ......................
##    group    9
##     ............................................................
##     ............................................................
##     ......................
##    group    10
##     ............................................................
##     ............................................................
##     ..........................
##    group    11
##     ............................................................
##     ............................................................
##     ................
##    group    12
##     ............................................................
##     ............................................................
##     ......................
##    group    13
##     ............................................................
##     ............................................................
##     ......
##    group    14
##     ............................................................
##     ............................................................
##     ..........................
##    group    15
##     ............................................................
##     ............................................................
##     ...................
```

Splittint the data into pseudo testcrosses

In order to calculate a map for each parent and then join them afterwards, we provide a function `pseudo.testcross.split()`, that creates a list of testcrosses. Each list element corresponds to a linkage group and a sequence for markers of type “D1.10” and one for markers of type “D2.15”. Both include all markers of other types.

```
testcrosses <- pseudo.testcross.split(linkage_groups)
testcrosses$LG1.d1.10
```

```
##
## Number of markers: 94
## Markers in the sequence:
## M2 M5 M7 M9 M14 M16 M32 M34 M37 M52 M56 M57 M60 M62 M72 M79 M84 M89 M100
## M113 M128 M126 M130 M136 M138 M143 M144 M151 M156 M157 M160 M161 M167 M174
## M186 M189 M197 M198 M199 M202 M203 M205 M206 M207 M211 M214 M218 M220 M227
## M231 M234 M237 M245 M246 M248 M249 M253 M254 M255 M263 M271 M274 M277 M279
## M280 M285 M290 M309 M312 M324 M335 M338 M351 M355 M360 M376 M377 M380 M385
## M391 M398 M406 M419 M425 M431 M435 M449 M459 M466 M484 M493 M495 M497 M498
##
## Parameters not estimated.
```

Ordering sequences in parallel

Before the map is calculated using the EM model, the sequences need to be ordered by a heuristic. The RECORD algorithm usually performs very well and has desireable characteristics, which make it trivial to parallelize. We use the function `record.parallel()`, which takes a `sequence` as input and we replicate RECORD 10 times (see the `times` argument). We then pick the best of those replicates as our final order. Note that it is rare for `times > 10` to yield any significant improvement. Finally, the `cores` argument defines how many of those RECORD replicates we can process in parallel. Set this to your computers number of CPUs (or maximally the number of the `times` argument).

```
ordered_sequences <- lapply(testcrosses, record.parallel, times = 10, cores = 1)
```

Creating the BatchMaps

With the sequences neatly ordered, we can now go ahead with creating BatchMaps. For this, we define an overall batch size as well as an overlap size and let the function `pick.batch.sizes()` decide on the final size in order to split batches evenly. The `around` argument to the function defines how much smaller or larger the batch size is allowed to be in order to create evenly sized batches. We will work with linkage group 1 from here on to save time:

```
LG1_d1.10 <- ordered_sequences$LG1.d1.10
LG1_d2.15 <- ordered_sequences$LG1.d2.15
batch_size_LG1_d1.10 <- pick.batch.sizes(LG1_d1.10,
                                         size = 50,
                                         overlap = 30,
                                         around = 10)
batch_size_LG1_d2.15 <- pick.batch.sizes(LG1_d2.15,
                                         size = 50,
                                         overlap = 30,
                                         around = 10)
c(batch_size_LG1_d1.10, batch_size_LG1_d2.15)
```

```
## [1] 52 52
```

Now all that's left to do is to call `map.overlapping.batches()`. This function has a great deal of options. For now, take away that `phase.cores` controls the number of parallel threads used to estimate the correct linkage phase between a pair of markers. As there are no more than four possible phases, this should never exceed four. The `size` and `overlap` arguments should match the output of `pick.batch.sizes()` with the given overlap. The `verbosity` option can be set to output different types of progress reports.

```
map_LG1_d1.10 <- map.overlapping.batches(input.seq = LG1_d1.10,
                                         size = batch_size_LG1_d1.10,
                                         phase.cores = 1,
                                         overlap = 30)
```

The result of `map.overlapping.batches()` has a data member `$Map`, which corresponds to the final map:

```
map_LG1_d1.10$Map
```

```
##
## Printing map:
##
## Markers           Position           Parent 1       Parent 2
##
##   1 M2                0.00           a |   | b       a |   | b
##   3 M5                0.52           a |   | b       a |   | a
##   4 M7                1.03           a |   | b       a |   | a
##   5 M9                1.40           a |   | b       a |   | a
##   6 M14               2.83           b |   | a       b |   | a
##   7 M16               3.14           b |   | a       a |   | a
##   9 M32               6.52           a |   | b       a |   | a
##  10 M34               7.44           a |   | b       a |   | b
##  11 M37               8.34           a |   | b       a |   | a
##  13 M52              12.18           a |   | b       a |   | b
##  15 M57              12.78           b |   | a       a |   | b
##  14 M56              12.78           b |   | a       a |   | a
##  17 M62              14.42           a |   | b       b |   | a
##  16 M60              14.42           b |   | a       a |   | a
##  21 M72              16.44           a |   | b       a |   | b
##  24 M84              18.69           b |   | a       a |   | a
##  25 M89              19.26           a |   | b       a |   | a
##  22 M79              19.84           b |   | a       a |   | b
##  26 M100             21.78           a |   | b       a |   | a
##  30 M113             25.55           a |   | b       a |   | a
##  37 M136             27.72           b |   | a       a |   | b
##  34 M130             27.72           b |   | a       a |   | a
##  33 M126             28.75           b |   | a       b |   | a
##  32 M128             28.75           a |   | b       a |   | b
##  38 M138             29.58           b |   | a       a |   | a
##  41 M144             31.19           b |   | a       a |   | a
##  40 M143             31.19           a |   | b       a |   | a
##  42 M151             32.59           a |   | b       b |   | a
##  45 M156             33.32           b |   | a       a |   | b
##  46 M157             33.55           a |   | b       b |   | a
##  47 M160             33.65           b |   | a       a |   | a
##  48 M161             33.65           a |   | b       b |   | a
##  49 M167             34.42           b |   | a       a |   | a
##  50 M174             35.22           b |   | a       a |   | b
##  51 M186             38.89           b |   | a       b |   | a
##  52 M189             39.08           b |   | a       b |   | a
##  55 M197             39.84           a |   | b       a |   | a
##  57 M199             40.35           a |   | b       b |   | a
##  56 M198             40.47           a |   | b       a |   | b
##  58 M202             40.71           a |   | b       a |   | a
##  62 M207             41.48           b |   | a       a |   | a
##  59 M203             41.63           b |   | a       a |   | a
##  60 M205              41.78           b |   | a       a |   | b
##  61 M206              41.78           b |   | a       b |   | a
##  63 M211              42.71           a |   | b       a |   | b
##  64 M214              43.43           b |   | a       a |   | a
##  65 M218              44.50           b |   | a       a |   | a
##  66 M220              45.14           b |   | a       b |   | a
##  69 M227              46.47           b |   | a       a |   | a
##  70 M231              46.96           b |   | a       a |   | a
##  71 M234              48.40           a |   | b       a |   | b
##  72 M237              49.04           b |   | a       b |   | a
##  76 M248              51.40           b |   | a       a |   | a
##  74 M245              51.40           a |   | b       a |   | a
##  75 M246              51.53           a |   | b       a |   | a
##  77 M249              51.67           b |   | a       a |   | a
##  78 M253              52.38           b |   | a       b |   | a
##  79 M254              52.91           b |   | a       a |   | a
##  80 M255              53.18           a |   | b       b |   | a
##  82 M263              54.03           a |   | b       a |   | a
##  83 M271              55.39           b |   | a       a |   | a
##  87 M280              56.53           b |   | a       b |   | a
##  86 M279              56.53           b |   | a       a |   | a
##  84 M274              57.08           b |   | a       a |   | b
##  85 M277              57.17           a |   | b       b |   | a
##  88 M285              57.94           b |   | a       a |   | a
##  89 M290              59.49           b |   | a       a |   | a
##  93 M309              61.64           b |   | a       a |   | a
##  94 M312              62.14           b |   | a       a |   | a
##  97 M324              64.79           b |   | a       a |   | b
## 100 M335              65.80           b |   | a       b |   | a
## 102 M338              67.37           b |   | a       b |   | a
## 103 M351              70.55           a |   | b       a |   | a
## 104 M355              70.71           a |   | b       a |   | a
## 106 M360              72.44           b |   | a       a |   | a
## 111 M376              74.61           b |   | a       b |   | a
## 112 M377              74.72           a |   | b       a |   | b
## 121 M391              77.22           b |   | a       a |   | a
## 117 M385              78.38           b |   | a       b |   | a
## 114 M380              78.97           a |   | b       a |   | a
## 125 M398              81.40           b |   | a       a |   | b
## 126 M406              83.31           b |   | a       a |   | a
## 129 M419              85.17           b |   | a       b |   | a
## 130 M425              85.51           a |   | b       a |   | a
## 131 M431              86.63           a |   | b       a |   | a
## 132 M435              87.25           b |   | a       b |   | a
## 133 M449              89.83           a |   | b       a |   | a
## 136 M466              90.83           a |   | b       b |   | a
## 134 M459              90.97           b |   | a       a |   | a
## 137 M484              96.27           a |   | b       a |   | a
## 139 M493              97.53           b |   | a       a |   | b
## 143 M498              98.04           a |   | b       b |   | a
## 141 M495              98.04           a |   | b       b |   | a
## 142 M497              98.04           b |   | a       a |   | b
##
## 94 markers            log-likelihood: -8012.925
```

The maps were simulated to be 100cM, which we come very close to. However, the markers in the simulated map are also ordered by their name, so M1 -> M2 -> M3 et cetera. We can spot some errors in the results, which can be improved in the next section.

BatchMap with ripple to improve order

As we saw at the end of the previous section, the markers still have some order error. While we can probably never recover the true map, we can expend resources (CPU time) to improve the current order. To do this, we can supply an ordering function to `map.overlapping.batches()` using the `fun.ord` argument. Currently there exists an umbrella function called `ripple.ord()` that should be supplied to this argument. This function will go through sliding windows within each batch and test alternative orders according to a given rule set. If an order improves the map likelihood, it is kept. The default and recommended ruleset is called "one", and will test each **pairwise** marker swap within a window. Further, a number of alternative orders can be considered in parallel. This is controlled by the `ripple.cores` argument. Note that the total number of threads used, will be `ripple.cores` * `phase.cores`.

How many cores will I need?

Depending on the rule set and window size that `ripple.ord()` uses, the number of comparisons can be calculated. Let the w be the window size:

- "one": $\frac{w*(w+1)}{2}$
- "all": $\frac{w!}{2}$

The rule set "random" can be supplied with the number of desired alternative orders. Let's consider a window size of 4 for our dataset. We will need to test $\frac{4*5}{2} = 10$ alternative order per window. On a machine with 16 threads available, a good combination would be `phase.cores=3` and `ripple.cores=5`, as often no more than two phases are plausible and even considered in the model. I am writing this vignette on a laptop with four cores available, which I will all use for `ripple.cores`, setting `phase.cores` to one. The rule set used by `ripple.ord()` is controlled by the `method` argument, the window size by the `ws` argument. Even with only about 100 markers, this function can take some time, it is advised you don't run it here.:

```
rip_LG1_d1.10 <- map.overlapping.batches(input.seq = LG1_d1.10,
                                         size = batch_size_LG1_d1.10,
                                         phase.cores = 1,
                                         overlap = 30,
                                         fun.order = ripple.ord,
                                         ripple.cores = 4,
                                         method = "one",
                                         min.tries = 1,
                                         ws = 4)
```

We can evaluate the number of mistakes in the order, because the true order is known in the simulated dataset:

```
err_rate <- function(seq)
{
  # Get the marker position
  s_num <- seq$seq.num
  # If the sequence is reverse, turn it around
  if(cor(s_num, 1:length(s_num)) < 0)
    s_num <- rev(s_num)
  # Get the number of misorders and divide by the total length
  sum(order(s_num) - 1:length(s_num) != 0) / length(s_num)
}
```

```r
c("BatchMap" = err_rate(map_LG1_d1.10$Map),
  "RippleBatchMap" = err_rate(rip_LG1_d1.10$Map))
```

```
## [1] 0.3723404 0.2234043
```