Carthagène - User Documentation

T. Schiex, S. de Givry, P. Chabrier, M. Bouchez

June 30, 2004

Acknowledgements

The CarthaGène software started out in 1995 from the confrontation of 2 computer scientists and one statistician with a multiple population data-set of backcross-like data. From this initial team, only one computer scientist remains. Thanks to C. Gaspin for respectively incorporating the *genetic algorithm* and B. Mangin for helping us dealing with genetics, pedigree, segregation ratio…

Denis Chenevière entirely implemented a first version of the graphical interface during his CNAM engineer project.

CarthaGène also largely benefited from the interaction will real data (and those that produced the data !). Thanks to to Valérie Laurent (and *Trichogramma brassicae*), Vincent Lombard (and *Brassica napus*), to Karim Gharbi (and *Salmo trutta*), to Jean-Francois Rami (private data - Euralis :-), Jean-Paul Bouchet (Aurige et CarthaGène) for their suggestions and the motivation they created.

We would like to conclude with a very special thank for Denis Milan (and *Sus scrofa*) for inviting us in the radiated hybrid world with ImprH. Some of the commands you may appreciate the most in CarthaGène exist mainly because of the interaction with Denis. Watch out for new commands in the future !

Contents

Introduction

CARTHAGENE is a computer tool for building genetic and/or radiated hybrid maps. For those who don't know, radiation hybrid mapping is a somatic cell technique which is used for ordering markers and estimating a *physical* distances between them (chromosomes are broken by irradiation) expressed in centiRay (instead of centiMorgan).

The main specificity of CARTHAGENE lies in the fact that it can build maximum likelihood "consensus" maps for several data sets corresponding to different populations but it can naturally also handle single population mapping.

The main strengths of CARTHAGENE can be rapidly described as follows:

- it can merge data sets from several types, both genetic and radiated hybrid data, assuming either shared "distances" or not between different populations.
- it always uses maximum multipoint likelihood as the criteria to compare maps, a rigorous well-behaved criteria. With optimized algorithms for backcross and haploid RH data, CARTHAGENE can, with no loss of precision, be one or two order of magnitude faster than alternative algorithms using the same criteria.
- it contains several powerful algorithm for building and validating both comprehensive and framework maps.
- it does not simply return a supposedly optimal map but a set of the k best maps that have been encountered during search. This gives feedback on the reliability of a map ordering.
- it has a graphical interface where maps can be visualised and compared and a text interface with an underlying programming langage (called Tcl) which can be used to automate mapping. If you intend to develop automated mapping procedure, you can get more documentation on the Tcl langage at http://www.tcl.tk/doc/. If you prefer, a Perl interface has also been independently developped by W.M. Snelling (USDA).
- it can be used on several plateforms (unix, Linux, Windows).

This document is the user guide of CARTHAGENE. It is intended for any user that would like to use CARTHAGENE. It essentially describes the textual interface of CARTHAGENE where commands are issued, the graphical interface is considered to be intuitive enough to be understandable once the textual interface is mastered.

For installing CARTHAGENE on your system, please go to CARTHAGENE's home page on the web:

http://www.inra.fr/bia/T/CarthaGene.

Chapter 1

Quick Start

The following section illustrates some of the major commands of CARTHAGENE on small example data sets. This quick start example is done using the shell interface but can also be performed using the graphical interface (either by clicking buttons in most cases, sometimes by issuing the command in the text window). For this QuickStart, we just give here a fast description of CARTHAGENE's main principles, you'll find more information in the sequel of the Documentation.

- Handling Data Sets:
  - Data sets are loaded from files formated as in MapMaker.
  - The current data set used by CarthaGene to compute maps, is always the last data set which has been "created" (loaded or merged).
  - The merging operators (`dsmergen`, `dsmergor`) accept only two data sets. You will have to call them several times to merge more than two data sets.
  - A list of implicitely selected loci is always active. By default this list contains all the markers of the last data set created (loaded or merged). The user can change this selection of loci (`mrkselset`) if he wants to build maps with a specific selection of loci.
- Map Search:
  - Each ordering algorithm sends the maps it has computed to a specific storage area called "the heap".
  - The so-called "heuristics" algorithms start from the data sets and compute maps from scratch..
  - The so-called "Local Search" algorithms, and the verification algorithms start from the best map on the heap and try to improve this map.
- Results exploitation and storage:
  - CarthaGene considers that the result of a map search is the set of the k best maps found during the session,
  - The heap always keep these k best solutions.
  - if the hierarchy of data sets changes, the heap becomes obsolete and is emptied.

To start CARTHAGENE shell, run the shell of Tcl (`tclsh`). If CARTHAGENE is well installed, its functionalities should be available and a message mentioning that CarthaGene has been loaded should appear (the prompt might not be the same, don't worry). Under windows, you can use the graphical interface and directly type your command in the main window.

```
[alta:exemple] tclsh

   CarthaGene version 0.999-LKH, Copyright (c) 1997-2004 (INRA).

   CarthaGene comes with ABSOLUTELY NO WARRANTY.
   CarthaGene is free software. You are welcome to redistribute it,
   under certain conditions. See the License file for information.

Type ’help’ for help.

CG>
```

If you type help, the complete list of the commands of CarthaGene will appear.

```
CG> help
CartaGene commands :
====================
miscellaneous :
---------------
cgout           Save the output to a file.
cgtolerance     Sets EM convergence thresholds.
cgversion       Get the version.
cgrestart       Reset the application.
cgstop          To stop a running command, type ctrl-c.
cglicense       Deprecated command.
cgstat          Get some performance information.
cgnotrobust     Disable reliability test of framework maps.
cg2tsp          Convert current marker selection to TSP.
cgrobustness    Sets reliability threshold for framework maps.
cgsave
cgexport

to manipulate data sets :
-------------------------
dsload          Load a Biological Data Set.
dsmergor        Merge by order two Biological Data Sets.
dsinfo          Summarize the current data sets.
dsmergen        Merge two Biological Data Sets.
dsget           Get a list of the current data sets description.

to manipulate groups :
----------------------
group           Identify linkage groups.
groupget        Get a group by markers ID.

to manipulate loci :
--------------------
mrklod2p        Print the two points LOD matrix.
mrkname         Get the name of a locus.
mrkselget       Return the loci selection into a list.
mrkid           Get the integer id of a locus.
mrkallget       Return all the loci into a list.
mrkfr2p         Print the two points FR/Breaks matrix.
mrklast         Get the integer id of the last known locus.
mrkselset       Set the marker list(by Id).
mrkdist2p       Print the two points distance matrix.
mrkmerge        Merges two compatible markers.
mrkdouble       Identifies pairs of markers with compatible typing.
```

```
mrkinfo         Tell which data set each marker belongs to.
mrkmerget       Retrive a list of merged markers.

to manipulate the heap :
------------------------
heapsizeget     get the size of the heap.
heapsize        Resize the heap.
heapget         Get the heap, in a list.
heaprintd       Display the heap in detail, sorted.
heaprint        Display the heap sorted.
heaprinto       Display the heap of maps by privileging the order of the ma...
heapequiset     enable or disable the equivalence mode of the heap.
heaprintg       Draw the heap(current best maps) into a graphical display.

to manipulate a map :
---------------------
maprint         Print a map.
mapordget       Get a order of loci from a map, in a list.
mapocb          Obligate Chromosome Breaks of a map.
maprintdr       Print a map reverse, in detail.
mapget          Get a map, in a list.
maprintd        Print a map, in detail.

to search for good maps :
-------------------------
greedy          Find good maps using the greedy algorithm.
lkhocbn         Provide a (nice) map, using normalized 2-points Obligate Ch...
ilkhl           Provide a (nice) map, using dynamically updated 2-points LO...
mfmapd          Provide quickly a (nice) map, using the 2-points distances.
buildfw         Build a framework map.
ilkhn           Provide a (nice) map, using dynamically updated normalized ...
mfmapl          Provide quickly a (nice) map, using the 2-points LOD criteria.
algogen         Find good maps using the genetic algorithm.
flips           Try to improve the best map by flipping.
ilkhocb         Provide a (nice) map, using dynamically updated 2-points Ob...
lkh             Provide a (nice) map, using 2-points loglikelihoods and Lin...
ilkh            Provide a (nice) map, using dynamically updated 2-points lo...
quietset        Make the search process quiet.
sem             Compute a Single map (Single EM).
polishtest      Try to improve the map giving with markers by polishing.
nicemapd        Provide quickly a (nice) map, using the 2-points distances.
ilkhocbn        Provide a (nice) map, using dynamically updated normalized ...
annealing       Find good maps using the annealing algorithm.
nicemapl        Provide quickly a (nice) map, using the 2-points LOD criteria.
verbset         Make the search process verbose.
lkhd            Provide a (nice) map, using 2-points distances and Lin-Kern...
polish          Try to improve the best map by polishing.
build           Build maps with two-points informations.
lkhl            Provide a (nice) map, using 2-points LOD criteria and Lin-K...
lkhocb          Provide a (nice) map, using 2-points Obligate Chromosome Br...
ilkhd           Provide a (nice) map, using dynamically updated 2-points di...
lkhn            Provide a (nice) map, using normalized 2-points loglikeliho...

Type ’<command-name> -H’ for more help with a particular command.
Some commands have a default behaviour :
Type ’d_<command-name>’ to use it.
To change the defaults, edit the ’.tclshrc’
```

1.1 Loading data

Now we will begin by loading a dataset. The corresponding command is texttdsload. If you want to learn more about this command, you can use the “-H” flag:

```
CG> dsload -H

Usage : dsload [-h | -H | -u | FileName]

Description : dsload loads a data set. The file format expected for a biolo...
```

To load the first population (it should be available in the `Data/mouse.raw` file[1] :

```
CG> dsload Data/mouse.raw
{1 intercross 308 46 /homes/thomas/CartaGene/dev/test/Data/mouse.raw}
```

Upon loading, CarthaGene computes all the two-points statistics (LOD, distances...) and returns the id of the population, here 1, the type of the population (intercross), the number of individuals (46) and the number of markers (308).

We can now have a look to the the populations loaded using the `dsinfo` (dataset information) command.

```
CG> dsinfo

Data Sets :
----------:
ID        Data Type    markers individuals             filename constraints...
 1       intercross        308          46            mouse.raw
```

We can check that the population number 1, the last loaded, is the current population.

We can now give a look to the markers that have been mentionned in the loaded dataset. To display the loci, their numerical id and the data set they come from, we can use the `mrkinfo` command:

```
CG> mrkinfo
...
```

1.2 Linkage groups and 2-Points estimations

In CarthaGene, all the two points information (LODs, distances, recombination ratios…) is computed at load time. So we can directly try to group the markers in linkage groups. This is done using the `group` command that specifies a distance and a LOD threshold: two markers whose 2 point distance (Haldane/Ray) is below the given distance threshold and whose 2 points LOD is above the LOD threshold will be put in the same linkage group. Users that don’t want to use one of the two thresholds can simply set the threshold they want to ignore to an extreme value (distance threshold set to zero, LOD threshold set to an arbitrary large number).

Let’s try this with a 30cM distance threshold and a 3.0 LOD threshold:

```
CG> group  0.3 3.0

Linkage Groups :
---------------:
```

---

[1] This file has been extracted from the MapMaker 3 distribution.

```
LOD threshold=3.00
Distance threshold=30.00:

 Group ID : Marker ID List ...
        1 : 275
        2 : 197
        3 : 139
        4 : 128 281 259 178 146
        5 : 86 265 243 192 116
        6 : 54 307 304 295 289 201 177 150 102 96 67
        7 : 47 236 48
        8 : 30 249 248 234 223 210 173 80 172 287 167 120
        9 : 21 303 282 279 242 202 200 185 61 170 26 228 226 224 196 183 16...
       10 : 20 284 277 255 220 239 186 132 99 94 75 85 62 42 38
       11 : 18 306 302 305 272 166 131 187 269 271 250 237 144 176 133 90 1...
       12 : 17 266 260 257 254 231 247 114 207 225 212 199 188 122 55 110 1...
       13 : 15 298 103 27 273 211 208 159 157 115 45 205 164 112
       14 : 13 296 276 292 267 263 230 209 153 141 113 100 22 182 299 175 2...
       15 : 10 204 245 191 190 184 179 240 221 154 127 123 155 92 91 89 57 ...
       16 : 8 41
       17 : 7 138 258 219 34 286 280 203 181 149 107 109 74 37 28 29 63
       18 : 6 268 308 222 291 235 156 105 39 52 32 104 246 194 158 151 126 ...
       19 : 5 262 216 147 117 97 88 12 218 293 300 274 232 171 270 229 206 ...
       20 : 4 283 142 56 129 189 134 87 23 244 238 68
       21 : 3 72 290 251 193 121 84 40 162 137 71 16 50 233 174
       22 : 2 294 297 256 241 143 140 217 252 227 119 136 70 14 11 64
       23 : 1 301 264 165 125 124 35 278 214 213 9 111 66 59 198
23
```

In this case, we get 23 groups. In the sequel we will work on the group 10. To focus on this specific group, we will get the list of the markers in the group using the `groupget` command and then select them using the `mrkselset` command. This can be achieved easyly in the graphical interface with a few clicks. If you use the shell, do as follows:

```
CG> groupget 10
20 284 277 255 220 239 186 132 99 94 75 85 62 42 38
```

To automagically select the markers of the group, one can use the following syntax (using [and]) that simply replaces what appears between [and] with the result of the command in between (this is called a macro-expansion). We can therefore select the group 10 with the following command:

```
CG> mrkselset [groupget 10]
```

Note that the current marker selection is not only a set of markers but also a default markers ordering. We can look at the LOD matrix beetween each pair of markers using the `mrklod2p` command.

```
CG> mrklod2p

             20   284   277   255   220   239   186   132    99    94    75...
           L029  A079  A059  A036  M232  D022  M237  M030  M076  M034  T018...
          -----------------------------------------------------------------...
    L029 |------  4.4   5.5   1.1   4.4   6.3   2.0   4.0   2.0   1.1   3.6...
    A079 |  4.4 ------ 18.4   7.4  16.5   5.7  10.5  21.7  10.5   7.4  14.0...
    A059 |  5.5  18.4 ------  6.2  14.2   6.4   9.0  17.8   9.0   6.2  11.9...
    A036 |  1.1   7.4   6.2 ------  9.0   2.6  13.0   7.7  13.0  21.4   8.6...
    M232 |  4.4  16.5  14.2   9.0 ------  4.8  13.0  16.0  13.0   9.0  17.8...
    D022 |  6.3   5.7   6.4   2.6   4.8 ------  3.2   5.2   3.2   2.6   4.5...
    M237 |  2.0  10.5   9.0  13.0  13.0   3.2 ------ 11.0  19.9  13.0  12.8...
    M030 |  4.0  21.7  17.8   7.7  16.0   5.2  11.0 ------ 11.0   7.7  13.5...
    M076 |  2.0  10.5   9.0  13.0  13.0   3.2  19.9  11.0 ------ 13.0  12.8...
    M034 |  1.1   7.4   6.2  21.4   9.0   2.6  13.0   7.7  13.0 ------  8.6...
    T018 |  3.6  14.0  11.9   8.6  17.8   4.5  12.8  13.5  12.8   8.6 -----...
    T035 | 13.6   7.2   8.8   2.6   7.4   9.6   4.0   6.8   4.0   2.6   6.5...
    L078 | 13.9   6.9   8.4   2.5   7.1   9.8   3.9   6.5   3.9   2.5   6.2...
    L001 |  5.9  16.8  19.9   6.6  15.1   6.4   9.6  16.3   9.6   6.6  12.8...
    L010 | 18.1   4.3   5.5   1.3   4.3   5.4   2.2   3.9   2.2   1.3   3.5...
```

Two-points LOD are also used in the `mrkdouble` command which detects pairs of markers that have *compatible* genotypes on each individuals. Such pairs of markers should be merged in one marker to simplify the search for an optimal map. See the 1.8 section on this topic. In practice, two markers can be compatible on all individuals and nevertheless be unlinked. In this case they should not be merged together.

```
CG> mrkdouble

Possible double markers:

               L029 = L010            [18.1]
               A079 = M030            [21.7]
               A036 = M034            [21.4]
               M237 = M076            [19.9]
               T035 = L078            [21.4]
```

You see that 5 pairs of markers are “double markers”. We will ignore this issue for now and work on the whole group.

We can also have a look to the 2-points distance matrix using the `mrkdist2p` command. Haldane (`h`) or Kosambi (`k`) can be used. Ray distance is automatically selected for radiated hybrid data (whether you specify `h` or `k`).

```
CG> mrkdist2p h

Print two points distance matrices of the loci selection :
---------------------------------------------------------:

Data Set Number  1 :
                  L029  A079  A059  A036  M232  D022  M237  M030  M076  M03...
                 ---------------------------------------------------------...
           L029 |------ 29.9  25.3  61.8  28.2  10.0  43.4  31.1  43.4  61....
           A079 | 29.9 ------  2.2  17.9   3.4  13.2  10.0   0.0  10.0  17....
           A059 | 25.3   2.2 ------ 21.6   5.9  10.6  13.0   2.3  13.0  21....
           A036 | 61.8  17.9  21.6 ------ 13.0  29.8   5.9  16.6   5.9   0....
           M232 | 28.2   3.4   5.9  13.0 ------ 15.9   5.9   3.5   5.9  13....
           D022 | 10.0  13.2  10.6  29.8  15.9 ------ 22.5  13.9  22.5  29....
           M237 | 43.4  10.0  13.0   5.9   5.9  22.5 ------  8.8   0.0   5....
           M030 | 31.1   0.0   2.3  16.6   3.5  13.9   8.8 ------  8.8  16....
           M076 | 43.4  10.0  13.0   5.9   5.9  22.5   0.0   8.8 ------  5....
           M034 | 61.8  17.9  21.6   0.0  13.0  29.8   5.9  16.6   5.9 ----...
           T018 | 30.5   4.9   7.6  12.2   1.2  16.0   4.9   5.0   4.9  12....
           T035 |  6.0  18.4  14.8  40.0  16.7   2.4  27.4  18.9  27.4  40....
           L078 |  5.9  19.6  16.1  41.5  18.0   2.4  28.9  20.3  28.9  41....
           L001 | 23.4   3.4   1.1  19.8   4.6  10.4  11.5   3.5  11.5  19....
           L010 |  0.0  26.3  21.4  51.1  24.3   6.5  36.8  27.4  36.8  51....
```

You can also look to 2-points recombination ratio (breakage ratio for RH data) using `mrkfr2p`:

```
CG> mrkfr2p

Print two points recombination fractions  matrices of the loci selection :
---------------------------------------------------------------------------:

        L029  A079  A059  A036  M232  D022  M237  M030  M076  M034  T018  T...
       --------------------------------------------------------------------...
 L029 |------  0.2   0.2   0.4   0.2   0.1   0.3   0.2   0.3   0.4   0.2   ...
 A079 |  0.2 ------  0.0   0.2   0.0   0.1   0.1   0.0   0.1   0.2   0.0   ...
 A059 |  0.2   0.0 ------  0.2   0.1   0.1   0.1   0.0   0.1   0.2   0.1   ...
 A036 |  0.4   0.2   0.2 ------  0.1   0.2   0.1   0.1   0.1   0.0   0.1   ...
 M232 |  0.2   0.0   0.1   0.1 ------  0.1   0.1   0.0   0.1   0.1   0.0   ...
 D022 |  0.1   0.1   0.1   0.2   0.1 ------  0.2   0.1   0.2   0.2   0.1   ...
 M237 |  0.3   0.1   0.1   0.1   0.1   0.2 ------  0.1   0.0   0.1   0.0   ...
 M030 |  0.2   0.0   0.0   0.1   0.0   0.1   0.1 ------  0.1   0.1   0.0   ...
 M076 |  0.3   0.1   0.1   0.1   0.1   0.2   0.0   0.1 ------  0.1   0.0   ...
 M034 |  0.4   0.2   0.2   0.0   0.1   0.2   0.1   0.1   0.1 ------  0.1   ...
 T018 |  0.2   0.0   0.1   0.1   0.0   0.1   0.0   0.0   0.0   0.1 ------  ...
 T035 |  0.1   0.2   0.1   0.3   0.1   0.0   0.2   0.2   0.2   0.3   0.2 --...
 L078 |  0.1   0.2   0.1   0.3   0.2   0.0   0.2   0.2   0.2   0.3   0.2   ...
 L001 |  0.2   0.0   0.0   0.2   0.0   0.1   0.1   0.0   0.1   0.2   0.1   ...
 L010 |  0.0   0.2   0.2   0.3   0.2   0.1   0.3   0.2   0.3   0.3   0.2   ...
```

1.3 Building maps from scratch

We can now start to try to build maps. All the maps built go in a specific storage structure called the "heap" that remembers the k best maps found during all the map search process. To build a first map in order to get a first map in the heap, we will simply directly ask CarthaGene to assess the quality of the default order specified in the `mrkselset` command. This is done using the `sem` command. This procedure compute true multipoint maximum likelihood of the current order and prints the corresponding map (the markers order used for printing can be reversed w.r.t. the marker selection. See section 2.4.10).

```
CG> sem

Map -1 : log10-likelihood =  -169.89
-------:
 Set : Marker List ...
   1 : L029 A079 A059 A036 M232 D022 M237 M030 M076 M034 T018 T035 L078 L00...
```

To try to build less stupid maps, we can try to use heuristics building procedures. The two simple procedures `nicemapl` and `nicemapd` build reasonably good maps using respectively 2-points LOD and 2-points distances as guide (trying to put strong LOD/small distances close together). The two slightly more complex procedures `mfmapl` and `mfmapd` tend to provide better results. Both classes of heuristics are derived from usual travelling salesman problem heuristics. In all cases, the true multipoint maximum likelihood of the order is then computed and the map printed.

```
CG> nicemapl

Map -1 : log10-likelihood =   -72.96
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 A059 L001 A079 M030 M232 T018 M237 M076 A03...
```

```
CG> nicemapd

Map -1 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 A079 M030 M232 T018 M237 M076 A03...
```

The loglikelihoods of the maps found using these two heuristics are -72.96 and -70.86 respectively. We can have a closer look to the maps build up to this point by asking for a detailed view of all the maps stored in “the heap”. This is achieved using the `heaprintd` command:

```
CG> heaprintd

Map  0 : log10-likelihood =  -169.89, log-e-likelihood =  -391.19
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

  1  20 L029          29.9 cM     29.9 cM     24.3 cM    22.5 %%     4.4
  2 284 A079           2.2 cM     32.2 cM      2.2 cM     2.2 %%    18.4
  3 277 A059          21.6 cM     53.7 cM     18.3 cM    17.5 %%     6.2
  4 255 A036          13.0 cM     66.8 cM     11.7 cM    11.5 %%     9.0
  5 220 M232          10.1 cM     76.9 cM      9.2 cM     9.1 %%     4.8
  6 239 D022          13.9 cM     90.8 cM     12.4 cM    12.2 %%     3.2
  7 186 M237           8.7 cM     99.4 cM      8.0 cM     7.9 %%    11.0
  8 132 M030           8.7 cM    108.1 cM      8.0 cM     7.9 %%    11.0
  9  99 M076           5.9 cM    114.0 cM      5.6 cM     5.6 %%    13.0
 10  94 M034          12.2 cM    126.3 cM     11.0 cM    10.9 %%     8.6
 11  75 T018          19.3 cM    145.6 cM     16.6 cM    16.0 %%     6.5
 12  85 T035           0.0 cM    145.6 cM      0.0 cM     0.0 %%    21.4
 13  62 L078          14.5 cM    160.0 cM     12.8 cM    12.6 %%     9.0
 14  42 L001          19.4 cM    179.4 cM     16.6 cM    16.1 %%     6.0
 15  38 L010        ----------              ----------
                     179.4 cM                156.8 cM


       15 markers, log10-likelihood =  -169.89
                   log-e-likelihood =  -391.19

Map  1 : log10-likelihood =   -72.96, log-e-likelihood =  -167.99
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

  1  20 L029           0.0 cM      0.0 cM      0.0 cM     0.0 %%    18.1
  2  38 L010           5.9 cM      5.9 cM      5.6 cM     5.6 %%    13.1
  3  62 L078           0.0 cM      5.9 cM      0.0 cM     0.0 %%    21.4
  4  85 T035           2.8 cM      8.7 cM      2.7 cM     2.7 %%     9.6
  5 239 D022          12.7 cM     21.4 cM     11.4 cM    11.2 %%     6.4
  6 277 A059           1.1 cM     22.5 cM      1.1 cM     1.1 %%    19.9
  7  42 L001           3.4 cM     25.9 cM      3.3 cM     3.3 %%    16.8
  8 284 A079           0.0 cM     25.9 cM      0.0 cM     0.0 %%    21.7
  9 132 M030          3.4 cM     29.3 cM      3.3 cM     3.3 %%    16.0
 10 220 M232          1.1 cM     30.4 cM      1.1 cM     1.1 %%    17.8
 11  75 T018          4.7 cM     35.1 cM      4.5 cM     4.5 %%    12.8
 12 186 M237          0.0 cM     35.1 cM      0.0 cM     0.0 %%    19.9
 13  99 M076          5.9 cM     41.0 cM      5.6 cM     5.6 %%    13.0
 14 255 A036          0.0 cM     41.0 cM      0.0 cM     0.0 %%    21.4
 15  94 M034        ----------             ----------
                     41.0 cM                38.6 cM


       15 markers, log10-likelihood =   -72.96
                   log-e-likelihood =  -167.99

Map  2 : log10-likelihood =   -70.86, log-e-likelihood =  -163.17
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

  1  20 L029          0.0 cM      0.0 cM      0.0 cM     0.0 %%    18.1
  2  38 L010          5.9 cM      5.9 cM      5.6 cM     5.6 %%    13.1
  3  62 L078          0.0 cM      5.9 cM      0.0 cM     0.0 %%    21.4
  4  85 T035          2.5 cM      8.5 cM      2.5 cM     2.5 %%     9.6
  5 239 D022         11.5 cM     19.9 cM     10.4 cM    10.2 %%     6.4
  6  42 L001          1.1 cM     21.0 cM      1.1 cM     1.1 %%    19.9
  7 277 A059          2.2 cM     23.3 cM      2.2 cM     2.2 %%    18.4
  8 284 A079          0.0 cM     23.3 cM      0.0 cM     0.0 %%    21.7
  9 132 M030          3.4 cM     26.7 cM      3.3 cM     3.3 %%    16.0
 10 220 M232          1.1 cM     27.8 cM      1.1 cM     1.1 %%    17.8
 11  75 T018          4.7 cM     32.5 cM      4.5 cM     4.5 %%    12.8
 12 186 M237          0.0 cM     32.5 cM      0.0 cM     0.0 %%    19.9
 13  99 M076          5.9 cM     38.4 cM      5.6 cM     5.6 %%    13.0
 14 255 A036          0.0 cM     38.4 cM      0.0 cM     0.0 %%    21.4
 15  94 M034        ----------             ----------
                     38.4 cM                36.2 cM


       15 markers, log10-likelihood =   -70.86
                   log-e-likelihood =  -163.17

        EM calls:
           Set  1 : 36 (33,0)
        CPU Time (secs): 0.15
        Maps within -3.0: 2
```

So far there are only our 3 maps in the heap. We can try to build other maps using a smarter heuristics procedure called `build`. This command incrementally includes markers, always choosing the best loglikelihood and the best insertion point. Because it is too “greedy” in its choices, this procedure can be performed in parallel on several maps, always keeping the k best maps. So, the command takes one parameter k to specify the number of map built at the same time.

```
CG> build 10

Build(10) : |||||||||||||||

Map  5 : log10-likelihood =   -70.86
```

```
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 A079 M030 M232 T018 M237 M076 A03...
```

No better map was found. We may now shift to so-called "improving" methods. These methods cannot start from scratch and are dedicated at improving available maps.

1.4 Map improving methods

As a first trial, we can use the local search algorithms. In this example, we will use the "simulated annealing" algorithm embodied in the `annealing` command. The command takes some parameters to specify how the search takes place (and how much cpu-time it will consume). Here we use 100 moves per level (the larger the better the search and the more expensive it is), an initial temperature of 50 (this is automatically reajusted), a final temperature of 0.1 (the smaller, the better the search and the more expensive too) and a cooling factor of 0.8 (always strictly less than 1, the closer to 1, the better the search and the more expensive too). This a very fast search (use other parameters in practice).

```
CG> annealing 100 50 0.1 0.8

Map -1 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 A079 M030 M232 T018 M237 M076 A03...

50.00? :
40.00? :
32.00? :
25.60? :
20.48? :
16.38? :
13.11? :
10.49? :
8.39? :
6.71? :
5.37? :
4.29? :
3.44? :
2.75? :
2.20? :
1.76? :
1.41? :
1.13? :
0.90? :
0.72? :
0.58? :
0.46? :
0.37? :
0.30? :
0.24? :
0.19? :
0.15? :
0.12? :
```

When the annealing process improves over the best known solution, a "+" is printed. In our case, no better map could be found. Now we would like to see the best maps found (which are stored in the heap), ordered by loglikelihood:

```
CG> heaprint

Map  0 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L010 L029 L078 T035 D022 L001 A059 A079 M030 M232 T018 M237 M076 M03...

Map  1 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 T035 L078 D022 L001 A059 M030 A079 M232 T018 M076 M237 M03...

Map  2 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 T035 L078 D022 L001 A059 M030 A079 M232 T018 M237 M076 M03...

Map 14 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 T035 L078 D022 L001 A059 A079 M030 M232 T018 M237 M076 M03...

Map  3 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 T035 L078 D022 L001 A059 A079 M030 M232 T018 M237 M076 A03...

Map  4 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 T035 L078 D022 L001 A059 A079 M030 M232 T018 M076 M237 A03...

Map  6 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 T035 L078 D022 L001 A059 A079 M030 M232 T018 M076 M237 M03...

Map  7 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 M030 A079 M232 T018 M237 M076 M03...

Map  8 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 M030 A079 M232 T018 M076 M237 A03...

Map 13 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 M030 A079 M232 T018 M076 M237 M03...

Map  5 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 M030 A079 M232 T018 M237 M076 A03...

Map 11 : log10-likelihood =   -70.86
```

```
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 A079 M030 M232 T018 M076 M237 M03...

Map 10 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 A079 M030 M232 T018 M237 M076 M03...

Map  9 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 A079 M030 M232 T018 M076 M237 A03...

Map 12 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 A079 M030 M232 T018 M237 M076 A03...
```

The best map is always the last map. We see that on this data set there are several maps with different orders and a close (or equal) likelihood. This is not surprising if you remember that the `mrkdouble` command detected several “double” markers. Usually, swapping 2 “double markers” in a map will not change the loglikelihood.

Another local search algorithm is the “taboo search” algorithm embodied in the `greedy` command. The command takes some parameters to specify how the search takes place (and how much cpu-time it will consume). Here we use 3 loops (the larger the better the search and the more expensive it is), an additional number of iterations of 0, the minimum size of the taboo list is 1% of the neighborhood and the maximum size of the list is 15%.

```
CG> greedy 3 1 1 15 0

Map -1 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 A079 M030 M232 T018 M076 M237 A03...
Run number 0
-*-*----*--*----*----*-----*----*---*-----*--*-*-----*---Run number 1
---**-----*----*-----*-----*------*----*-------*-------*---Run number 2
------*----*---*-----*--*-*------------*----------*--------
```

The algorithm prints a ’-’ when the map could not be improved, a “+” when the best map has improved and a “*” to say that the serach process has detected that it is stuck in a region it has already explored. In this case, it starts from another random order. Here, again, no improvement was obtained. Note that the map improving commands can be used as map validating commands using the `cgrobustness` command (see 2.5.18).

1.5 Order validating methods

We can try to test how good this map is using a “verification algorithm”. The usual method of flipping markers inside a window (`ripple` command in mapmaker) is called `flips` in CarthaGene. It takes 3 parameters: the size of the flipping window, a printing threshold on the difference of loglikelihood with the best map and a last flag that says if the flips command must be reiterated if a better map is found. Here we will use a window of 4 markers, all maps whose loglikelihood is better or within 1.0 LOD unit of the loglikelihood of the best map will be printed and the flips command will be reiterated on the new improved map if such a map is found.

```
CG> flips 4 1 1

Repeated Flip(window size : 4, threshold : 1.00).


Map -1 : log10-likelihood =   -70.86
-------:
 Set : Marker List ...
   1 : L029 L010 L078 T035 D022 L001 A059 A079 M030 M232 T018 M076 M237 A03...

         2   2 2 1 2     1 2
 2 3 6 8 3 4 7 8 3 2 7 9 8 5 9  log10
 0 8 2 5 9 2 7 4 2 0 5 9 6 5 4    -70.86
[1 0 3 2]- - - - - - - - - - -     -0.00
[- - 3 2]- - - - - - - - - - -     -0.00
[1 0 - -]- - - - - - - - - - -     -0.00
 - - - - -[- - 3 2]- - - - - -     -0.00
 - - - - - - - - -[- - 3 2]- -      0.00
 - - - - - - - - - - -[1 0 3 2]      0.00
 - - - - - - - - - - -[- - 3 2]      0.00
```

Here we see that by flipping markers in the original best map, we have not improved the loglikelihood. We also see that several alternative orders exist. As said before, this is due to the existence of double markers. For example, a line such as `[1 0 - -]- - - - - - - - -- - -0.00` says that by swapping the first and second markers of the map we don't change the likelihood. These are markers 20 and 38 (top rows). We can ask for the name of these 2 markers:

```
CG> mrkname 20
L029
CG> mrkname 38
L010
```

These two markers were effectively detected as double by the `mrkdouble` command. Ideally, all the previous searches should be repeated after merging the double markers which are strongly linked.

Another validation procedure is the `polish` command. It takes each marker successively and tries to insert it in all possible intervals. The variation in loglikelihood is reported for each marker and each destination interval.

```
CG> polish

Local map analysis:

        L029  L010  L078  T035  D022  L001  A059  A079  M030  M232  T018  M...
      ---------------------------------------------------------------------...
 L029 |-----   0.0  12.6   9.5  10.0  27.3  27.3  32.3  26.6  29.5  26.9  3...
 L010 |  0.0 -----  10.5   7.8   9.0  25.2  25.5  30.8  25.0  27.9  25.6  3...
 L078 |  7.6  10.5 -----   0.0   3.3  24.0  24.5  29.9  24.0  26.9  25.2  3...
 T035 |  7.8  10.5   0.0 -----   3.3  22.7  23.4  28.9  23.0  25.9  24.3  3...
 D022 |  4.4   6.8   1.9   3.3 -----  10.4   9.9  14.1  11.1  14.5  12.4  1...
 L001 | 14.6  26.4  19.5  22.7  10.4 -----   2.1   9.0   5.0  10.4  10.9  2...
 A059 | 15.9  28.2  21.3  24.8  12.5   2.1 -----   6.9   5.4  13.3  13.4  2...
 A079 | 17.9  31.5  24.8  28.7  16.7   7.5   6.9 -----   0.0   9.6  10.5  2...
 M030 | 17.7  31.4  24.8  28.7  16.7   7.5   6.9   0.0 -----   9.6  10.4  1...
 M232 | 16.0  29.5  22.8  26.5  16.0   7.2  10.9   8.1   9.6 -----   2.4  1...
 T018 | 15.1  29.2  22.5  26.3  16.8   8.7  13.0  10.2  11.9   2.4 -----  1...
 M076 | 17.8  34.0  27.9  32.7  22.7  15.7  21.2  18.8  20.9  12.9  11.9 --...
 M237 | 17.8  34.0  27.9  32.7  22.7  15.7  21.2  18.8  20.9  12.9  11.9   ...
 A036 | 20.3  37.5  31.7  37.0  27.4  21.6  28.4  26.1  28.8  21.6  21.8  1...
 M034 | 20.3  37.5  31.7  37.0  27.4  21.6  28.4  26.1  28.8  21.6  21.8  1...
      --------------------------------------------------------------------...
```

We can see again the ‘double” markers effect (with 0.0 LOD differences). Otherwise, all markers seems to be relatively firmly placed with high LODs, all above 2.1.

Let us have a look in detail to the best map, the map 12.

```
CG> maprintd 12

Map 12 : log10-likelihood =   -70.86, log-e-likelihood =  -163.17
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

  1  20 L029           0.0 cM      0.0 cM      0.0 cM     0.0 %%    18.1
  2  38 L010           5.9 cM      5.9 cM      5.6 cM     5.6 %%    13.1
  3  62 L078           0.0 cM      5.9 cM      0.0 cM     0.0 %%    21.4
  4  85 T035           2.5 cM      8.5 cM      2.5 cM     2.5 %%     9.6
  5 239 D022          11.5 cM     19.9 cM     10.4 cM    10.2 %%     6.4
  6  42 L001           1.1 cM     21.0 cM      1.1 cM     1.1 %%    19.9
  7 277 A059           2.2 cM     23.3 cM      2.2 cM     2.2 %%    18.4
  8 284 A079           0.0 cM     23.3 cM      0.0 cM     0.0 %%    21.7
  9 132 M030           3.4 cM     26.7 cM      3.3 cM     3.3 %%    16.0
 10 220 M232           1.1 cM     27.8 cM      1.1 cM     1.1 %%    17.8
 11  75 T018           4.7 cM     32.5 cM      4.5 cM     4.5 %%    12.8
 12 186 M237           0.0 cM     32.5 cM      0.0 cM     0.0 %%    19.9
 13  99 M076           5.9 cM     38.4 cM      5.6 cM     5.6 %%    13.0
 14 255 A036           0.0 cM     38.4 cM      0.0 cM     0.0 %%    21.4
 15  94 M034         ----------              ----------
                      38.4 cM                 36.2 cM


       15 markers, log10-likelihood =   -70.86
                   log-e-likelihood =  -163.17
```

This map will be kept as the best comprehensive map of the group in this tutorial. In practice, longer local search algorithm must be performed and other verification procedures should be used.

1.6 Building framework maps

Framework maps are maps that are supposedly firmly ordered (weakly ordered markers are not included). Such maps can be build by CarthaGene using a stepwise marker insertion method called `buildfw`.

This command uses an incremental insertion method. It tries to insert non inserted markers in a current map. For a given marker that could be inserted, it tries to insert the marker at all possible position of the map. The difference in loglikelihood between the best insertion and the second best insertion is used to qualify the marker. The marker inserted in practice is the marker that maximizes this difference and such that this difference is larger than a first threshold (Adding Threshold). The marker is inserted at its optimal position. However, if there are orders whose difference in loglikelihood with this best position is less than a second threshold (the Keep Threshold), they are also kept as possible new starting point for the next marker to be inserted.

Note that the build process may start from an empty map or from an existing order (that must contain at least 3 markers). Finally, when all insertable markers have been inserted, the map is kept aside and the remaining marker are tentatively inserted, one by one, in all possible positions in this map and then removed. For each such marker, the command reports how the loglikelihood changes wrt to the best inseryion point. The best position is marked with a "+". The difference in loglikelihood between the best position and each position are printed out if this difference is less than a threshold. When the last parameter of `buildfw` is set to 0, this last step (testing all remaining loci) is not performed. More complex postprocessing are possible, see the reference user documentation.

Let us test this command on our data. We will use a Keep and Adding threshold of 3.0 LODs, starting the build process from the empty map and trying to position remaining markers.

```
CG> buildfw 3 3 {} 1

BuildFW, Adding  Threshold = 3.00, Saving Threshold = 3.00.

>>> Delta = 5.23 :

Map  0 : log10-likelihood =   -44.30
-------:
 Set : Marker List ...
   1 : T035 L001 M237

>>> Delta = 5.34 , Id = 220,  Locus = M232 :

Map  0 : log10-likelihood =   -47.13
-------:
 Set : Marker List ...
   1 : T035 L001 M232 M237

>>> Delta = 4.01 , Id = 255,  Locus = A036 :

Map  0 : log10-likelihood =   -55.54
-------:
 Set : Marker List ...
   1 : T035 L001 M232 M237 A036

>>> Delta = 3.53 , Id = 284,  Locus = A079 :

Map  0 : log10-likelihood =   -59.57
-------:
 Set : Marker List ...
   1 : T035 L001 A079 M232 M237 A036

>>> Delta = 3.21 , Id = 20,  Locus = L029 :

Map  0 : log10-likelihood =   -67.59
-------:
 Set : Marker List ...
   1 : L029 T035 L001 A079 M232 M237 A036

BuildFW, remaining loci test :
       |       2 2 1 2  |
       | 2 8 4 8 2 8 5  |     Lod2pt         Dist2pt
       | 0 5 2 4 0 6 5  |  Left<-M->Right Left<-M->Right | 0->N    N->M | W...
     --|----------------|--------------------------------|--------------|--...
```

```
  A059 |    2 +         |  19.88   18.40     1.1   2.2   |   20.3   1.1 |  ...
  D022 |  2 +           |   9.64    6.37     2.4  10.4   |    6.3   2.6 |  ...
  M030 |      0 +       |  21.67   15.97     0.0   3.5   |   23.7  0    |  ...
  M076 |          + 0   |  12.97   19.87     5.9   0.0   |   33.1  0    |  ...
  M034 |            + 0 |  12.97   21.37     5.9   0.0   |   39.0  0    |  ...
  T018 |        2 +     |  17.79   12.82     1.2   4.9   |   27.1   1.1 |  ...
  L078 |  + 0           |  13.87   21.37     5.9   0.0   |    6.3  0    |  ...
  L010 |0 +             |  18.06   12.85     0.0   4.1   |    0.0  0    |  ...
```

Here the map produced contains only 7 markers. The additional uninserted markers list (A059, D022, M030, M076, M034, T018, L078, L010) contain all the 5 “possible double markers” identified by the `mrkdouble` command which is good news. The final map produced is automatically used as the best map. It is supposed to be firmly ordered as we can check with a `flips` command:

```
CG> flips 6 3.0 1

Repeated Flip(window size : 6, threshold : 3.00).


Map -1 : log10-likelihood =   -67.59
-------:
 Set : Marker List ...
   1 : L029 T035 L001 A079 M232 M237 A036

       2 2 1 2
 2 8 4 8 2 8 5  log10
 0 5 2 4 0 6 5    -67.59
```

which finds no better order at 3.0 LOD units.

We can check with a `polish` command:

```
CG> polish

Local map analysis:

        L029  T035  L001  A079  M232  M237  A036
      ------------------------------------------
 L029 |-----   3.2  19.9  21.1  19.9  23.2  12.3
 T035 |  3.2 -----  16.7  18.4  18.1  23.0  14.2
 L001 | 13.0  16.7 -----   3.5   7.2  15.7  12.3
 A079 | 13.9  19.6   3.5 -----   4.1  13.3  10.9
 M232 | 14.6  20.2   5.7   4.1 -----  10.0  10.0
 M237 | 15.0  24.0  12.6  13.7  10.0 -----   4.0
 A036 | 11.9  22.4  12.9  15.8  13.2   4.0 -----
      ------------------------------------------
```

which finds no better order at 3.0 LOD units.

To check the quality of the map even more thoroughly, we can use map improving commands such as `annealing` or `greedy`. In the process of trying to improve the current framework map, these commands may identify a map whose loglikelihood, if not better than the current map’s loglikelihood, may still be close enough to prove that the current map is not a valid framework map. In order to stop the search process as soon as such a map is found, we can use the `cgrobustness` command. The threshold given to the command indicates that every search should stop as soon as a map is found whose loglikelihood is within the threshold of the best map’s loglikelihood.

```
CG> cgrobustness 3.0
```

We now try to improve the map using the greedy command.

```
CG> greedy 1 0 5 30

Map -1 : log10-likelihood =   -67.59
-------:
 Set : Marker List ...
   1 : L029 T035 L001 A079 M232 M237 A036
Run number 0
-----**********-*****-*--*
```

which does not find any counter-example that proves that our map is not a framework map. We now remove the previous threshold using the `cgnorobust` command:

```
CG> cgnotrobust
```

1.7 Merging data sets

Now, we want to merge two different dataset (populations) into one. Two types of merging are available:

- `dsmergen`: merges the two dataset completely. For a given pair of successive markers in the order, the probabilistic model for the data has one single recombination ratio estimation for the 2 datasets. This produces consensus maps with consensus distances[2].
- `dsmergor`: merges two datasets by order only. For a given pair of successive markers in the order, the probabilistic model for the data has one single recombination ratio estimation for each of the 2 datasets. This produces consensus orders with per data set distances.

In this tutorial, we will merge two data sets by “order” which is the most complex case. The merge by order process is used because we want to merge genetic recombination data and radiated hybrid data and there is no simple way of relating the 2 distances (RH data being essentially physical). Actually, genetic and RH data nicely complement each other for *ordering* markers. Genetic data leads to myopic ordering: set of close markers cannot be reliably ordered because usually no recombination can be observed between them. On the contrary RH data leads to hypermetropic ordering: set of closely related markers can be reliably ordered but distant groups are sometimes difficult to order because too many breaks occurred between them. The data used here is coming from a single group representing one porcine chromosome.

First, let us examine each dataset independently and see what the `buildfw` command can produce from each of these. We first reinitialize CarthaGene using the `cgrestart` command (to keep markers numerical Ids low) and load the new data-set:

```
CG> cgrestart

CG> dsload Data/bc1.cg
{1 f2 backcross 17 208 /homes/thomas/CartaGene/dev/test/Data/bc1.cg}
CG> buildfw 3.0 3.0 {} 1

BuildFW, Adding  Threshold = 3.00, Saving Threshold = 3.00.
```

---

[2] When used on two data sets of the same type, this merging process corresponds to the addition of information on new individuals (on new or known markers). It must not be used when new markers are types on existing individuals. In this case, the data-set file must directly be edited by adding new lines for the new markers (and changing the number of markers in the header).

```
>>> Delta = 10.84 :

Map  0 : log10-likelihood =  -114.56
-------:
 Set : Marker List ...
   1 : MS4 MS9 MS12

>>> Delta = 9.73 , Id = 17,  Locus = MS1 :

Map  0 : log10-likelihood =  -155.67
-------:
 Set : Marker List ...
   1 : MS12 MS9 MS4 MS1

>>> Delta = 10.78 , Id = 9,  Locus = MS7 :

Map  0 : log10-likelihood =  -166.66
-------:
 Set : Marker List ...
   1 : MS12 MS9 MS7 MS4 MS1

>>> Delta = 13.30 , Id = 10,  Locus = MS2 :

Map  0 : log10-likelihood =  -178.48
-------:
 Set : Marker List ...
   1 : MS12 MS9 MS7 MS4 MS2 MS1

>>> Delta = 7.08 , Id = 7,  Locus = MS16 :

Map  0 : log10-likelihood =  -203.02
-------:
 Set : Marker List ...
   1 : MS16 MS12 MS9 MS7 MS4 MS2 MS1

>>> Delta = 11.51 , Id = 5,  Locus = MS11 :

Map  0 : log10-likelihood =  -207.54
-------:
 Set : Marker List ...
   1 : MS16 MS12 MS11 MS9 MS7 MS4 MS2 MS1

>>> Delta = 8.25 , Id = 16,  Locus = MS19 :

Map  0 : log10-likelihood =  -246.26
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS12 MS11 MS9 MS7 MS4 MS2 MS1

>>> Delta = 8.23 , Id = 13,  Locus = MS15 :

Map  0 : log10-likelihood =  -250.99
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS15 MS12 MS11 MS9 MS7 MS4 MS2 MS1

>>> Delta = 7.07 , Id = 2,  Locus = MS5 :
```

```
Map  0 : log10-likelihood =  -253.69
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS15 MS12 MS11 MS9 MS7 MS5 MS4 MS2 MS1

>>> Delta = 4.07 , Id = 8,  Locus = MS8 :

Map  0 : log10-likelihood =  -262.35
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS15 MS12 MS11 MS9 MS8 MS7 MS5 MS4 MS2 MS1

BuildFW, remaining loci test :
       |                          |
       | 1   1 1   1         1 1  |     Lod2pt         Dist2pt
       | 6 7 3 4 5 2 8 9 2 1 0 7  |  Left<-M->Right Left<-M->Right | 0->N  ...
     --|--------------------------|--------------------------------|-------...
  MS13 |      0 +                 |  18.66   11.91     0.0   8.4   |   43.3...
   MS6 |              3 +         |  17.32   15.41     1.6   4.5   |   70.3...
  MS17 |2 +                       |  14.06    6.31     7.4  27.2   |    0.0...
   MS3 |                    + 2   |  35.05   47.62     5.7   0.6   |   79.2...
  MS20 |+ 1                       |    -     34.83      -    2.9   |   -   ...
```

The BC dataset is a good quality dataset. Few markers could not be placed in the framework map. The “quality” of this map should naturally be checked more thoroughly using other “validating” and “improving” commnds. We now perform the same analysis on RH data:

```
CG> dsload Data/rh1.cg
{2 haploid RH 13 118 /homes/thomas/CartaGene/dev/test/Data/rh1.cg}
CG> buildfw 3.0 3.0 {} 1

BuildFW, Adding  Threshold = 3.00, Saving Threshold = 3.00.

>>> Delta = 5.13 :

Map  0 : log10-likelihood =   -68.70
-------:
 Set : Marker List ...
   2 : MS4 G40 G39

>>> Delta = 6.14 , Id = 18,  Locus = G36 :

Map  0 : log10-likelihood =   -77.85
-------:
 Set : Marker List ...
   2 : MS4 G36 G40 G39

>>> Delta = 3.91 , Id = 4,  Locus = MS6 :

Map  0 : log10-likelihood =  -101.52
-------:
 Set : Marker List ...
   2 : MS6 MS4 G36 G40 G39

>>> Delta = 3.65 , Id = 12,  Locus = MS9 :

Map  0 : log10-likelihood =  -125.81
```

```
-------:
 Set : Marker List ...
   2 : MS9 MS6 MS4 G36 G40 G39

BuildFW, remaining loci test :
       |              |
       | 1     1 2 1  |     Lod2pt         Dist2pt
       | 2 4 1 8 1 9  |  Left<-M->Right Left<-M->Right | 0->N    N->M | Wei...
     --|--------------|--------------------------------|--------------|----...
   MS5 |  2 +         |   7.44    3.64    54.3  90.1   |   81.4  29.4 |    ...
   MS8 |+ 0           |    -     18.48       -   16.5   |   -     16.5 |    ...
   MS7 |  + 3         |   5.44   12.79    72.3  29.5   |    0.0  57.8 |    ...
   MS3 |          + 0 |   9.84    9.69    45.2  45.9   |  205.2  23.5 |    ...
  MS15 |+           1 |    -      0.99       -  159.0   |   -    159.0 |    ...
   MS1 |3           + |   3.76     -      89.6    -    |  252.7  89.6 |    ...
   G37 |        3 +   |  15.04    7.23    24.9  59.2   |  205.2  14.0 |    ...
```

Since radiated hybrid data usually have very few missing information, it is possible to use specific ordering methods to build very good maps. Several commands in CarTHaGene can directly translate a RH dataset into a specific travelling salesman problem instance such that the optimal solution of this TSP instance will also yield the optimal map. This translation [BDCP00] is faithful when there is no missing information but is still very effective when few missing exists. CarTHaGene automatically reevaluates all the maps generated using multi-point maximum log-likelihood. Here we will try to reorder the framework map we built using the `lkhn` command (lk stands for Lin-Kernighan heuristics [LK73, Hel00], a famous algorithm used to solve travelling salesman problem instances.):

```
CG> lkhn 1 1
Best map with log10-likelihood = -125.81
TSP: optimum= 127.355000 lowerbound= 127.355000 gap= 0.000000% totaltime= 0.01

Map -1 : log10-likelihood =  -125.81
-------:
 Set : Marker List ...
   2 : MS9 MS6 MS4 G36 G40 G39
Optimum found, equal to 127355! The minimum 1-tree is a tour.
```

The same map with a -125.81 loglikelihood is found which is a very good indication that the order is indeed optimal.

The RH dataset is less order informative than the BC dataset. However, merging the two data sets will help in ordering the markers. Because we are merging populations with not directly related parameters (RH and genetic distances), we will use the `dsmergor` command. The command takes two parameters that specify the numerical id of the two populations to be merged. To see the numerical ids, we can use `dsinfo`.

```
CG> dsinfo

Data Sets :
----------:
ID        Data Type    markers individuals             filename constraints...
 1     f2 backcross         17         208               bc1.cg
 2       haploid RH         13         118               rh1.cg

CG> dsmergor 1 2
{3 merged by order 21 326}
CG> dsinfo

Data Sets :
```

```
----------:
ID        Data Type    markers individuals             filename constraints...
 1     f2 backcross         17         208               bc1.cg
 2       haploid RH         13         118               rh1.cg
 3  merged by order         21         326                                ...
```

To better see how markers are shared between the 2 populations, we can use the `mrkinfo` command:

```
CG> mrkinfo
...
```

We can now try to build a framework map for the hybrid genetic/RH data set.

```
CG> buildfw 3.0 3.0 {} 1

BuildFW, Adding  Threshold = 3.00, Saving Threshold = 3.00.

>>> Delta = 11.54 :

Map  0 : log10-likelihood =  -202.47
-------:
 Set : Marker List ...
   1 : MS7 MS3 MS1
   2 : MS7 MS3 MS1

>>> Delta = 14.11 , Id = 1,  Locus = MS4 :

Map  0 : log10-likelihood =  -232.07
-------:
 Set : Marker List ...
   1 : MS7 MS4 MS3 MS1
   2 : MS7 MS4 MS3 MS1

>>> Delta = 11.64 , Id = 14,  Locus = MS12 :

Map  0 : log10-likelihood =  -266.52
-------:
 Set : Marker List ...
   1 : MS12 MS7 MS4 MS3 MS1
   2 :      MS7 MS4 MS3 MS1

>>> Delta = 13.07 , Id = 12,  Locus = MS9 :

Map  0 : log10-likelihood =  -302.10
-------:
 Set : Marker List ...
   1 : MS12 MS9 MS7 MS4 MS3 MS1
   2 :      MS9 MS7 MS4 MS3 MS1

>>> Delta = 11.18 , Id = 2,  Locus = MS5 :

Map  0 : log10-likelihood =  -325.87
-------:
 Set : Marker List ...
   1 : MS12 MS9 MS7 MS5 MS4 MS3 MS1
   2 :      MS9 MS7 MS5 MS4 MS3 MS1

>>> Delta = 7.07 , Id = 7,  Locus = MS16 :
```

```
Map  0 : log10-likelihood =  -350.40
-------:
 Set : Marker List ...
   1 : MS16 MS12 MS9 MS7 MS5 MS4 MS3 MS1
   2 :           MS9 MS7 MS5 MS4 MS3 MS1

>>> Delta = 11.49 , Id = 5,  Locus = MS11 :

Map  0 : log10-likelihood =  -354.94
-------:
 Set : Marker List ...
   1 : MS16 MS12 MS11 MS9 MS7 MS5 MS4 MS3 MS1
   2 :                MS9 MS7 MS5 MS4 MS3 MS1

>>> Delta = 8.25 , Id = 16,  Locus = MS19 :

Map  0 : log10-likelihood =  -393.66
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS12 MS11 MS9 MS7 MS5 MS4 MS3 MS1
   2 :                     MS9 MS7 MS5 MS4 MS3 MS1

>>> Delta = 8.23 , Id = 13,  Locus = MS15 :

Map  0 : log10-likelihood =  -424.96
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS15 MS12 MS11 MS9 MS7 MS5 MS4 MS3 MS1
   2 :           MS15           MS9 MS7 MS5 MS4 MS3 MS1

>>> Delta = 6.52 , Id = 4,  Locus = MS6 :

Map  0 : log10-likelihood =  -438.73
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS15 MS12 MS11 MS9 MS7 MS6 MS5 MS4 MS3 MS1
   2 :           MS15           MS9 MS7 MS6 MS5 MS4 MS3 MS1

>>> Delta = 6.44 , Id = 21,  Locus = G40 :

Map  0 : log10-likelihood =  -454.74
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS15 MS12 MS11 MS9 MS7 MS6 MS5 MS4     MS3 MS1
   2 :           MS15           MS9 MS7 MS6 MS5 MS4 G40 MS3 MS1

>>> Delta = 5.18 , Id = 20,  Locus = G37 :

Map  0 : log10-likelihood =  -463.88
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS15 MS12 MS11 MS9 MS7 MS6 MS5 MS4         MS3 MS1
   2 :           MS15           MS9 MS7 MS6 MS5 MS4 G40 G37 MS3 MS1

>>> Delta = 6.99 , Id = 18,  Locus = G36 :

Map  0 : log10-likelihood =  -473.02
```

```
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS15 MS12 MS11 MS9 MS7 MS6 MS5 MS4             MS3 MS1
   2 :           MS15           MS9 MS7 MS6 MS5 MS4 G36 G40 G37 MS3 MS1

>>> Delta = 6.88 , Id = 19,  Locus = G39 :

Map  0 : log10-likelihood =  -493.00
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS15 MS12 MS11 MS9 MS7 MS6 MS5 MS4             MS3     MS1
   2 :           MS15           MS9 MS7 MS6 MS5 MS4 G36 G40 G37 MS3 G39 MS1

>>> Delta = 4.96 , Id = 8,  Locus = MS8 :

Map  0 : log10-likelihood =  -513.77
-------:
 Set : Marker List ...
   1 : MS19 MS16 MS15 MS12 MS11 MS9 MS8 MS7 MS6 MS5 MS4             MS3    ...
   2 :           MS15           MS9 MS8 MS7 MS6 MS5 MS4 G36 G40 G37 MS3 G39...

BuildFW, remaining loci test :
       |                                    |
       | 1   1 1   1           1 2 2 1 1 1  |     Lod2pt         Dist2pt
       | 6 7 3 4 5 2 8 9 4 2 1 8 1 0 1 9 7  |  Left<-M->Right Left<-M->Righ...
     --|------------------------------------|------------------------------...
  MS13 |      0 +                           |  18.66   11.91     0.0   8.4 ...
  MS17 |2 +                                 |  14.06    6.31     7.4  27.2 ...
   MS2 |                     2 2 2 2 + 0    |  47.62    0.00     0.6   0.0 ...
  MS20 |+ 1                                 |    -     34.83      -    2.9 ...
```

Thanks to RH data, a new genetic marker that could not previously be inserted is now sufficiently strongly ordered to be inserted. Furthermore, all new RH markers have been inserted. The final insertion map also shows that MS2 is replaced by MS3. The two markers are probably close one to the other and cannot both be inserted,

As mentioned earlier, further work should include a thorough validation of this order.

1.8 Dealing with “double markers”

In one of the previous examples (section 1.1), we noted that several markers were identified as “double markers” by the `mrkdouble` procedure. These double markers corresponds to pairs of markers such that the observed genotypes are compatible with the assumption that the 2 markers are the same. There are two cases where such a situation may occur:

- the two markers are effectively closely related (or the same) on the chromosome. In this case it is a good idea to try to merge them in one since this will simplify the ordering process.
- the two markers are totally unrelated markers in two different populations/panels. In this case, it is not a good idea to merge them. Such markers can often be placed using multipoint maximum likelihood.

To distinguish between these two cases, `mrkdouble` indicates the 2-points LOD between the pair of potential double markers. If this LOD is small, then the two markers should not be merged in one.

In the `mouse.raw` dataset, all pairs of double markers have strong LODs. Let us reload that file, perform grouping and select group 10 has we previously did:

```
CG> dsload Data/mouse.raw
...
CG> group  0.3 3.0
...
CG> mrkselset [groupget 10]
...
```

If you issue the `mrkdouble` command, you should see the following:

```
CG> mrkdouble

Possible double markers:

               L029 = L010            [18.1]
               A079 = M030            [21.7]
               A036 = M034            [21.4]
               M237 = M076            [19.9]
               T035 = L078            [21.4]
```

Let's merge `A079` and `M030` together. This can be achieved using the mrkmerge command. The command actually merges the obervation of both markers into one and leaves the other unchanged. The command takes the numerical id of the 2 markers as arguments. If you don't know this numerical id, the command mrkid can be used as in the example below.

```
CG> mrkselget
41 305 298 276 241 260 207 153 120 115 96 106 83 63 59
CG> mrkmerge [mrkid A079] [mrkid M030]
Markers 305 and 153 merged in 305.
```

Automatically, the marker 132 has been removed from the list of selected markers:

```
CG> mrkselget
41 305 298 276 241 260 207 120 115 96 106 83 63 59
```

The same can be done with each pair of markers detected as double markers. Note that it is possible that after merging two markers, some previously detected double markers become separated (eg. on BC data, one individual, if marker M1 is typed "1", marker M3 is type "0" and marker M2 is typed "-", then M1-M2 and M2-M3 are two pairs of "double markers" but once M1 and M2 are merged, then M3 cannot be merged with the (m1-M2) pair). Although this is an unlikely situation, CarthaGene checks this in practice and will check compatibility before merging. In the example, all pairs can be merged without problem.

```
CG> mrkmerge [mrkid L029] [mrkid L010]
Markers 41 and 59 merged in 41.

CG> mrkmerge [mrkid A036] [mrkid M034]
Markers 276 and 115 merged in 276.

CG> mrkmerge [mrkid M237] [mrkid M076]
Markers 207 and 120 merged in 207.

CG> mrkmerge [mrkid T035] [mrkid L078]
Markers 106 and 83 merged in 106.

CG> mrkselget
41 305 298 276 241 260 207 96 106 63
```

```
CG> mrklod2p

             41   305   298   276   241   260   207    96   106    63
           L029  A079  A059  A036  M232  D022  M237  T018  T035  L001
          ------------------------------------------------------------
    L029 |------  4.4   5.5   1.1   4.4   6.3   2.0   3.6  13.6   5.9
    A079 |  4.4 ------ 18.4   7.4  16.5   5.7  10.5  14.0   7.2  16.8
    A059 |  5.5  18.4 ------  6.2  14.2   6.4   9.0  11.9   8.8  19.9
    A036 |  1.1   7.4   6.2 ------  9.0   2.6  13.0   8.6   2.6   6.6
    M232 |  4.4  16.5  14.2   9.0 ------  4.8  13.0  17.8   7.4  15.1
    D022 |  6.3   5.7   6.4   2.6   4.8 ------  3.2   4.5   9.6   6.4
    M237 |  2.0  10.5   9.0  13.0  13.0   3.2 ------ 12.8   4.0   9.6
    T018 |  3.6  14.0  11.9   8.6  17.8   4.5  12.8 ------  6.5  12.8
    T035 | 13.6   7.2   8.8   2.6   7.4   9.6   4.0   6.5 ------  9.3
    L001 |  5.9  16.8  19.9   6.6  15.1   6.4   9.6  12.8   9.3 ------
```

We can now try to build a comprehensive map. In this simple case, we just try the `nicemapd` command followed by a `flips 5 0 0`.

```
CG> nicemapd

Map -1 : log10-likelihood =   -68.30
-------:
 Set : Marker List ...
   4 : L029 T035 D022 L001 A059 A079 M232 T018 M237 A036

CG> flips 5 0 0

Single Flip(window size : 5, threshold : 0.00).


Map -1 : log10-likelihood =   -68.30
-------:
 Set : Marker List ...
   4 : L029 T035 D022 L001 A059 A079 M232 T018 M237 A036

   1 2   2 3 2   2 2
 4 0 6 6 9 0 4 9 0 7  log10
 1 6 0 3 8 5 1 6 7 6    -68.30


CG> heaprintd
...
```

The best map found can be printed using `heaprintd`. The best map has number 7. We can later visualize it using the `maprintd` command:

```
CG> maprintd 7

Map  7 : log10-likelihood =   -68.30, log-e-likelihood =  -157.26
-------:

Data Set Number  4 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

--- L010
```

```
  1  41 L029           4.0 cM      4.0 cM      3.8 cM     3.8 %%    13.6
--- L078
  2 106 T035           2.5 cM      6.5 cM      2.5 cM     2.5 %%     9.6
  3 260 D022          11.5 cM     18.0 cM     10.4 cM    10.2 %%     6.4
  4  63 L001           1.1 cM     19.1 cM      1.1 cM     1.1 %%    19.9
  5 298 A059           2.2 cM     21.3 cM      2.2 cM     2.2 %%    18.4
--- M030
  6 305 A079           3.4 cM     24.8 cM      3.3 cM     3.3 %%    16.5
  7 241 M232           1.1 cM     25.9 cM      1.1 cM     1.1 %%    17.8
  8  96 T018           4.7 cM     30.5 cM      4.5 cM     4.5 %%    12.8
--- M076
  9 207 M237           5.9 cM     36.5 cM      5.6 cM     5.6 %%    13.0
--- M034
 10 276 A036        ----------              ----------
                      36.5 cM                 34.5 cM


       10 markers, log10-likelihood =   -68.30
                   log-e-likelihood =  -157.26
```

You can see that the merged markers are indicated on the map. Looking onto the heap for alterated ordering, the situation has much improved with respect to the previous situation (where several orders with identical log-likelihood existed). We can have a look to the situation using the `heaprinto` command:

```
CG> heaprinto 3 0 0
Loci Id  ..........   1 2   2 3 2   2 2
                  : 4 0 6 6 9 0 4 9 0 7
                  : 1 6 0 3 8 5 1 6 7 6
Loci Pos .......... | | | | | | | | | |
Map Id : log10    : | | | | | | | | | |  (Delta lod per set)
     7 :   -68.30 : 1 2 3 4 5 6 7 8 9 10  (        4 )
    13 :     2.03 =   2 1                (     2.03 )
     8 :     2.09 =       4 3            (     2.09 )
    10 :     2.37 =             7 6      (     2.37 )
     9 :     2.54 = 2 0 1                (     2.53 )
    12 :     2.88 = 1 0                  (     2.87 )
     4 :     2.95 = 2   0                (     2.95 )
    14 :     3.49 =       5   3          (     3.49 )
    11 :     4.08 = 1 2 0                (     4.07 )
     5 :     4.11 =   2 1 4 3            (     4.11 )
     6 :     4.37 =                 9 8  (     4.37 )
     2 :     4.47 =       4 3   7 6      (     4.46 )
     3 :     4.62 = 2 0 1 4 3            (     4.61 )
     1 :     4.86 = 1 0   4 3            (     4.86 )
     0 :     4.92 = 2   0 4 3            (     4.92 )

CG>
```

and see that the best alternate map found is at 2 LOD units and is obtained by swapping two markers.

Chapter 2

User and reference documentation

When you use CarThaGene, there are three implicit objects that are manipulated by the software that are worth mentionning: these are the current data-set, the current list of selected markers and the current set of the best known maps (also referred to as *the heap*).

Despite the fact that CarThaGene can load several data-sets to work with, there is only one data-set active at a given moment in CarThaGene. This data-set may be a single population data set or a multiple population (or so-called “merged”) data-set. This active data set is always the last data-set that has been created (loaded or merged) . All further computations that will be done will be done with respect to this data set, later referred to as the *active* dataset. Note that the name of the markers in the data-sets are very important since they are used by CarThaGene to determine if two markers in two different data sets are identical or not. Besides its name, each markers will receive a so-called numerical id that is often used by CarThaGene. One can easily go between marker names and marker ids using the `mrkname`/`mrknames` and `mrkid`/`mrkids` commands.

Once the populations you intend to work with have been loaded and merged as you want, mapping is essentially a problem of choosing and ordering a set of markers. At any time when you use CarThaGene there is an implicit *list of selected markers* that specify the set of markers you intend to work with. This list is also used as a default markers ordering for some commands (eg. the `sem` command, see section 2.5.2). When a data set is created (loaded or merged), all the markers of this data set are selected. You can visualize and modify this selection (see the `mrkselset` command, section 2.3.13 and the `mrkadd`, `mrksub` and `mrkdel` commands).

The last important object in CarThaGene is the so-called *heap*. The heap is simply a bounded-size container for all the best maps for the current marker selection that have been encountered by CarThaGene during the session. Basically, this means that each time a map likelihood is evaluated by CarThaGene, either inside a complex ordering strategy or following a direct user request, it is considered as a candidate for being kept in the heap. If there is room in the heap or if new map has a better likelihood than the worst map in the heap, then the new map will be stored in the heap for further analysis. Eventually, when enough alternative orders have been considered, the heap can be examined to see how strongly the best order is supported by the data by giving alternative sub-optimal orders that were found during search. The size of the heap is user configurable and defaults to 15 maps. Because the heap is implemented using a nice and efficient data-structure, it can be fairly large (eg. more than 1000 maps) without slowing down the software. So don’t hesitate to change this default of 15 if needed (see the `heapsize` command, section 2.4.2).

The heap has another implicit use: several CarThaGene commands that need a marker ordering (or map) to start with will automatically take the best map of the heap as a starting point. This means that if the heap is empty because you haven’t yet computed the likelihood of any map, these procedures will complain of having no starting point. It is up to you to fill the heap using any map building command (see section 2.5).

2.1 Formatting the data

Essentially, CarthaGene uses a format inspired by MapMaker [LGA+87] for datasets. All datasets are described using two header lines followed by the data itself, one marker being described per line.

The first line indicates the type of the data: backcross, RIL, F2 intercross, outbreds, haploid or diploid radiated hybrids... See the following sections for more details.

The second line indicates the size of the data set: number of individual/hybrids typed followed by the number of markers. For compatibility with MapMaker, it may contains 2 further numbers which will be ignored by CarthaGene and aliases declaration that will be used by CarthaGene to analyze the last section of the file that contains typing data. An alias declaration specify that a given character (eg `0`) will be used instead of another default one (eg. `A`) to specify marker typing data. Such an alias is specified by the statement `0=A` in the second line.

The last section contains maker typing data. Each line describe typing data for one given marker, one individual (hybrid) after the other. A line begins with a star character (`*`) immediately followed by the marker name separated from the rest of the line using either tabulations or space characters. The rest of the line indicates typing data for the marker for each individual (hybrid), one after the other. The number of individual reported on each line must match the number of individuals indicated in the second line of the file and the number of markers (lines of data) must not be lower than the number of markers indicated in the second line of the file (if extra lines are there, they will be ignored).

The following is an example of an haploid radiation hybrid dataset with 4 markers and 40 hybrids were the default `A` (Absent) and `H` (Here) typing characters have been respectively aliased to `0` and `1`.

```
data type radiated hybrid
40 4 0 0 1=H 0=A
*N7b 110110110-101100001011011111111000111010
*A11 0101001101101100001011011001111000101110
*N4 11010011011011000010110110011110001111101
*rA20 1101001101101100101011010001111111101010
```

The next section specify which header line and which typing conventions must be used to encode marker data depending on the dataset type.

2.1.1 Genetic data sets

Backcross data

This is a MapMaker compatible format. CarthaGene uses a dedicated boosted EM for backcross data which may be one or two orders or magnitudes faster than a standard EM algorithm without any loss of precision [SCBM01]. All backcross data file must start with the following header line:

```
data type f2 backcross
```

In the case of backcross data, each locus can either be homozygous or heterozygous. These two situations are encoded using the `H` and `A` characters respectively. Loci with an unknown status are encoded as `-`. This encoding can be redefined using aliasing in the second header line (see the beginning of the section).

Intercross F2 data

This is a MapMaker compatible format. All F2 (intercross) datasets must start with the following header line:

```
data type f2 intercross
```

Depending on the dominance or codominance of each loci, several situations can be encoded. If we call `A` and `B` the two alleles of an heterozygous individual, the descendance can be either:

- homozygous `A`: this is encoded by the character `A`.
- homozygous `B`: this is encoded by the character `B`.
- heterozygous: this is encoded by the character `H`.
- know to be not homozygous `A`: this is encoded by the character `C`.
- know to be not homozygous `B`: this is encoded by the character `D`.
- unknown: this is encoded as `-`.

This encoding can be redefined using aliasing in the second header line (see the beginning of the section).

Recombinant inbred lines (RIL) data

These are MapMaker compatible formats. Both self and sib RIL data are handled by CarTHaGene. Note that since these data types are internally handled as pure backcross data (recombination frequencies being adequately corrected to take into account the fact that the data represent self/sib RIL), it is impossible to completely merge (both on order and recombination frequencies, see the `dsmergen` command, section 2.2.2)) such data with other genetic data. Use the `dsmergor` command in this case.

Depending on the RIL type (self/sib), the first header line of the format file must be respectively:

```
data type ri self
```

or

```
data type ri sib
```

The characters used to encode RIL data are the same as for backcross data (see section 2.1.1). This encoding can be redefined using aliasing in the second header line (see the beginning of the section).

Phase known outbred data

CarTHaGene can handle outbred data as far as phases are fixed (either known or fixed to the most probable phases). Such phase known outbred datasets can be handled using different strategies. A first simple method (which ignores part of the information) consists in projecting the information on each parent side: this gives two backcross datasets which can be merged using either the `mergen` or `mergor` command. The first case will aim at computing a consensus map (with consensus distances) while the second one will aim at computing a consensus order with different recombination ratio on each parent. We will not detail this strategy here although it has the advantage of relying on our “Boosted EM” algorithm which means that it will be a lot faster than the approach below. In this section, we describe a more complex encoding which does not ignore any information. All outbred datasets must start with the following header line (same as for intercross data:

```
data type f2 intercross
```

Because the ability to handle outbred data has evolved from a classical intercross situation using Mapmaker syntax, the syntax used to encode such data is currently rather clumsy. This may change one of these days but you'd better not count on it…

At one locus, consider the cross of $F_0|F_1 \times M_0|M_1$ where F_0, F_1, M_0 and M_1 stand for the alleles on each haplotype of the father and the mother respectively. The genotype of the child obtained may be either $F_0|M_0$, $F_1|M_0$, $F_0|M_1$ or $F_1|M_1$. Depending on the heterozygocity of the parents, or the number of different alleles, on the dominance or codominance of the markers, on the observations available on the child's phenotype, only a subset of these 4 possibilities is possible. For example, in the "usual" F2 intercross situation, the two parents are heterozygous and bear the same pair of alleles: $F_0 = M_0 = A$ and $F_1 = M_1 = B$. In this simple case, observations on the phenotype of a child may lead to different situations:

- the child is typed A and the allele A is not dominant. In this case the only possible genotype is $A|A$ or $M_0|F_0$. This is encoded by the character `A` in MapMaker.
- the child is typed A and A is dominant. Then the set of possible genotypes is $\{A|A, A|B, B|A\}$, i.e, $\{F_0|M_0, F_0|M_1, F_1|M_0\}$. This is encoded by the character `D` in MapMaker.
- the child is typed B and the allele B is not dominant. In this case the only possible genotype is $B|B$ or $M_1|F_1$.This is encoded by the character `B` in MapMaker.
- the child is typed B and B is dominant. Then the set of possible genotypes is $\{A|B, B|A, B|B\}$, i.e, $\{F_0|M_1, F_1|M_0, F_1, M_1\}$. This is encoded by the character `C` in MapMaker.
- the child is typed AB (the marker is codominant) then the set of possible genotypes is $\{A|B, B|A\}$, i.e, $\{F_0|M_1, F_1|M_0\}$. This is encoded by the character `H` in MapMaker.
- the child is untyped at this locus. The set of possible genotypes includes all possible genotypes i.e $\{F_0|M_0, F_0|M_1, F_1|M_0, F_1, M_1\}$. This is encoded by the character `-` in MapMaker.

In order to be able to cope with all phases known situations, including cases where one parent is homozygous, when 3 or 4 different alleles are present, Carthagène actually enables the user to express any subset of the 4 different possibilities. In order to do so, these 4 possibilities are associated with a number:

- genotype $F_0|M_0$ is associated with 1
- genotype $F_0|M_1$ is associated with 2
- genotype $F_1|M_0$ is associated with 4
- genotype $F_1|M_1$ is associated with 8

and the user will be able to tell Carthagène which set of genotype is actually possible at a given locus by:

- adding up the numbers associated with each possible genotypes at the locus given the observations
- converting this number to hexadecimal (base 16 where one counts 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f).

The following tables recapitulates all possible codes from `1` to `f` and the corresponding set of possible genotypes at the locus.

Notation	Synonym	Possible Genotypes
`1`	`A`	$F_0\|M_0$
`2`		$F_0\|M_1$
`3`		$F_0\|M_0$, $F_0\|M_1$
`4`		$F_1\|M_0$
`5`		$F_0\|M_0$, $F_1\|M_0$
`6`	`H`	$F_0\|M_1$, $F_1\|M_0$
`7`	`D`	$F_0\|M_0, F_0\|M_1$, $F_1\|M_0$
`8`	`B`	$F_1\|M_1$
`9`		$F_0\|M_0$, $F_1\|M_1$
`a`		$F_0\|M_1$, $F_1\|M_1$
`b`		$F_0\|M_0$, $F_0\|M_1$, $F_1\|M_1$
`c`		$F_1\|M_0$, $F_1\|M_1$
`d`		$F_0\|M_0$, $F_1\|M_0$, $F_1\|M_1$
`e`	`C`	$F_0\|M_1$, $F_1\|M_0$, $F_1\|M_1$
`f`	`-`	$F_0\|M_0$, $F_0\|M_1$, $F_1\|M_0$, $F_1\|M_1$

Here is an example of a small outbred dataset:

```
data type f2 intercross
40 5 0 0
*M1 1118822228821414212414281812248128422488
*M2 1418822228821414212414281882248128422488
*M3 4418828288821414242414281881148122422488
*M4 4412288228411814211444881884248222124488
*M5 8412288224412814811484881281848822184188
```

Let see how this can be used in some practical phase-known outbred situations according to the segregation ratio of the marker at the current locus.

Segregation ratio 1:2:1 This is the usual case in F2 intercross with codominance. In this case, the parents $F_0|F_1$ and $M_0|M_1$ are such that typically $F_0 = M_0 = A$ and $F_1 = M_1 = B$ and AB codominate. The usual observations on the child are either phenotype A, B or AB. this lead to the following encoding:

- A: The only possible genotype is $A|A = M_0|F_0$: this is coded by `1` (the synonym `A` used in MapMaker can also be used).
- B: The only possible genotype is $B|B = M_1|F_1$: this is coded by `8` (the synonym `B` in MapMaker can also be used).
- AB: The only possible genotypes are $A|B = M_0|F_1$ and $B|A = M_1|F_0$: this is coded by `6`=4+2. The synonym `H` used in MapMaker can also be used.

When missing data occurs, it is still possible to give partial information to Carthagène. Eg., if the child is typed A but the other allele is not known. The child's genotype can be either $A|A = M_0|F_0$ or $A|B = M_0|F_1$ or $B|A = M_1|F_0$. This is encoded by the character `7`=1+2+4 (synonym `D`)

The character `e` = 14 = 8+4+2 (synonym `C`) encodes a situation where the child has been typed B but the other allele is not known. In this case, the child's genotype can be $B|B = F_1|M_1$ or $A|B = F_0|M_1$ or $B|A = F_1|M_0$.

Segregation ratio 3:1 This type of segregation ratio occurs when dominance appears. Imagine A dominates B, then it is impossible to distinguish between $A|A$, $A|B$ and $B|A$. Precisely, if the child is typed B, then the character 8 (or the synonym `B` of MapMaker) will be used. Else the character `7` = 1+2+4 (or the synonym `D` of MapMaker) will be used to represent the fact that we simply know that the child genotype is either $A|A = F_0|M_0$, $A|B = F_0|M_1$ or $B|A = F_1|M_0$.

Conversely, if B dominates A, the code `1` (resp. `e`) will be used if the child is typed A (resp. B). The respective synonyms `A` and `C` of MapMaker can also be used.

Segregation ratio 1:1:1:1 This occurs when different alleles appear in the father and in the mother. For example in $A|B \times C|D$. In this case the child is either $A|C$ or $A|D$ or $B|C$ or $B|D$ and the corresponding codes are respectively `1` (or `A`), `2`, `4` and `8` (or `B`).

When some data is missing, it is still possible to give information to Carthagène. Imagine for example that the child is typed A then the second allele is unknown. Because we have 4 different alleles, we know for sure that the first allele of the children is A. Therefore, the only possible genotypes are $A|C$ (code `1`) or $A|D$ (code `2`) and the corresponding code is `3`= 1+2.

Imagine that instead of having 4 alleles, we have 3 alleles in $A|B \times A|C$. Then children may be $A|A$, $A|C$, $B|A$ or $B|C$ i.e., there is still a 1:1:1:1 segregation ratio. If again the children is just typed A then there is more indetermination at hand: the child may be either $A|A$ (code `1`), $A|C$ (`2`) or $B|A$ (code `4`). In this case, the corresponding code is `7` = 1+2+4 (or the synonym `D`).

Segregation ratio 1:1 Imagine the second parent is homozygous at current locus i.e., we cross $A|B$ with $C|C$ (a backcross like situation), then the children genotypes may either be $A|C$ or $B|C$ If we observe $A|C$ (or A simply), it is not know where does the C come from i.e, the children may be $A|C_0$ or $A|C_1$. This case is encoded by `3` = 1+2. This is the sum of `1` and `2` which corresponds to the two possible cases: $A|C$ with C coming from one grand-parent (code `1`) or the other (code `2`).

Similarly, if we observe B, then we know that the first allele is B, the second is C but we don't where this allele comes from. The code will be `c` = 12 = 4+8.

If the homozygocity appears on the first parent ($A|A \times B|C$) instead of the second one and if we observe B we get the code `5`, the sum of `1` and `4` corresponding to the fact that the origin of the first allele is unknown. If we observe C, we get the code `a` which in hexadecimal corresponds to 10 which is the sum of `8` and `2`.

2.1.2 Radiated Hybrid data sets

CAR$^{\mathrm{T}}_{\mathrm{H}}$AGENE can handle both haploid and diploid radiated hybrid panels data using the so-called "equal retention model" [LBLC95]. CAR$^{\mathrm{T}}_{\mathrm{H}}$AGENE uses a specific boosted EM [SCBM01] algorithm for handling *haploid* data. This algorithm may run one or two orders of magnitude faster than a standard EM algorithm without any loss of precision. We therefore strongly recommend you to use the haploid model for all RH data and to shift to the diploid model only for final estimation of distances.

Depending of the ploidy of the model you intend to use (haploid or diploid), the first header line of the dataset should be respectively set to:

```
data type radiated hybrid
```

or

```
data type radiated hybrid diploid
```

The character used to encode RH data are repectively:

- `A` for missing (Absent) marker.
- `H` for present (Here) marker.
- `-` for unknowns (eg. inconsistent typing when double typing is performed)

2.1.3 Constraints data sets

In some cases, you may want to incorporate ordering information that comes from another source of information (already published maps...) for which you don't have the data. CarTHaGene can try to take advantage of such information using so called "triplet constraints". A triplet constraint is defined by a triple of markers (m_1, m_2, m_3) and a loglikelihood penalty p (expressed in base 10 logarithm). The "semantics" of such a triplet constraint is that any marker ordering inconsistent with the triplet ordering (i.e., an order that places m_2 outside of the $m_1 - m_3$ interval) will see its loglikelihood penalized by the amount p. Use this facility with extreme precaution since it breaks the good property of the maximum likelihood criteria used by CarTHaGene. The most obvious and reasonable use of this facility is to answer a question such as: is there a map that places the three markers as indicated in the triplet and whose loglikelihood is not worse than the loglikelihood of an optimal map by more than p.

A constraint dataset is simple composed by a set of such triplet constraints after a single header line:

```
data type constraint
```

A triplet constraint is simply written by puting the name of the 3 markers in the triplet in sequence followed by the penalty. An example of such a dataset is given below:

```
data type constraint
MS4 MS5 MS13 3.0
MS5 MS13 MS6 3.0
```

The markers mentionned in the triplet must be already know to CarTHaGene, i.e, one or several datasets which contain data about these markers must be already loaded.

2.2 Loading and Merging data sets

To load a new dataset in CarTHaGene, the only command is the `dsload` command which must be followed by the name of the file to load. The dataset loaded will get a numerical id (reported when the dataset is loaded and which can be requested later using the `dsinfo` command).

To do multiple populations mapping, two merging commands can be used. Each of these commands takes as arguments the numerical id of the 2 datasets merged (if you want to merge more than 2 datasets together, simply use the commands repeatedly).

- `mergen`: assumes that the markers order and the distance/recombination ratios are the same in the two datasets merged. This command cannot be used to merge radiated hybrid data with genetic data (for obvious reasons) and also cannot merge RIL self/sib data with other genetic data (for mathematical reasons). On such datasets, a single consensus map is reported by all mapping commands.
- `mergor`: assumes only that the markers order is the same in the two populations merged. Population specific distances will be estimated. Any pair of types of dataset can be merged by this command. On such dataset, two maps are reported by all mapping commands. When few markers are common to the merget datasets, many equivalent maps may populate the heap and limit the efficiency of the `buildfw` command. See the `heapequiset` command.

All mapping/map validating commands applied to merged datasets still compute exact multipoint maximum likelihood.

These two commands can be used together and repeatedly. Imagine for example that you have 2 radiated hybrid panels with the same irradiation level along with 2 genetic datasets available. You can "mergen" the 2 RH datasets together (assuming that the same irradiation level will lead to the same distances) and also "mergen" the 2 genetic datasets. This will give you two merged datasets which can be again merged using the `mergor` command. This gives a single dataset that merges all 4 datasets. All mapping commands on this dataset will report two maps: one RH consensus map and one genetic consensus map with the guarantee that they use a consistent markers order.

2.2.1 `dsload`

Loads a data-set from a file.

Synopsis: The `dsload` command is invoked either as:

- `dsload` *Options*
- `dsload` *FilePath*

Description: The `dsload` command loads the data-set from the file whose path is indicated in the argument *FilePath*.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *FilePath*: the path to a valid CarthaGene data-set.

Returns: a list summarizing some basic information on the data-set loaded:

- numerical Id of the data-set.
- type of the data-set.
- number of markers in the data-set.
- number of individuals/hybrids in the data-set.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> dsload Data/rh.cg
{2 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG>
```

See also:

- `dsinfo` (2.2.5)
- `dsget` (2.2.6)
- `dsmergor` (2.2.3)
- `dsmergen` (2.2.2)

2.2.2 `dsmergen`

Merges two data-sets in a consensus data-set, sharing order and distances.

Synopsis: The `dsmergen` command is invoked either as:

- `dsmergen` *Options*
- `dsmergen` *DsId1 DsId2*

Description: The `dsmergen` command merges two data-sets of compatible types in a single consensus data-set. If more then 2 data-sets need to be merged, this command must be called repeatedly. The consensus data-set built uses all the information available in the two data-sets for eg. maximum likelihood multipoint estimations. Only data-sets of "compatible" types can be merged using this merging operator:

- radiated hybrid data can only be merged with other RH data (and this assumes a similar level of irradiation)
- f2 (backcross/intercross) and outbred data can all be merged together using this operator,
- RIL data of a given type (Sib or Self) can only be merged with data-sets of the same type (Sib/Self respectively).

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *DsId1 DsId2*: the numerical If of the two data sets to merge.

Returns: a list summarizing some basic information on the merged data-set:

- numerical Id of the data-set.
- type of the data-set (i.e., `merged genetic`)
- number of markers in the data-set.
- number of individuals/hybrids in the data-set.

The number of markers is usually strictly less than the number of markers in each of the merged data-set (identical markers are detected by their identical name).

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> dsload Data/bc1.cg
{2 f2 backcross 17 208 /homes/thomas/CartaGene/dev/test/Data/bc1.cg}
CG> dsmergen 1 2
{3 merged genetic 20 416}
CG> mrkinfo

Markers :
--------:
Num                Names : Sets         Merges
  1                  MS1 :   1   2   3
  2                  MS2 :   1   2   3
  3                  MS3 :   1   2   3
  4                  MS4 :   1   2   3
  5                 MS5 :   1   2   3
  6                 MS6 :   1   2   3
  7                 MS7 :   1   2   3
  8                 MS8 :   1   2   3
  9                 MS9 :   1   2   3
 10                MS10 :   1       3
 11                MS11 :   1   2   3
 12                MS12 :   1   2   3
 13                MS13 :   1   2   3
 14                MS14 :   1       3
 15                MS15 :   1   2   3
 16                MS16 :   1   2   3
 17                MS17 :   1   2   3
 18                MS18 :   1       3
 19                MS19 :   1   2   3
 20                MS20 :   1   2   3
```

See also:

- `dsmergor` (2.2.3)
- `dsload` (2.2.1)
- `dsinfo` (2.2.5)
- `dsget` (2.2.6)

2.2.3 dsmergor

Merges two data-sets in a consensus data-set, sharing order but with separate parameter estimations (data-set specific recombination/breakage/retention ratio).

Synopsis: The `dsmergor` command is invoked either as:

- `dsmergor` *Options*
- `dsmergor` *DsId1 DsId2*

Description: The `dsmergor` command merges two data-sets of compatible types in a single data-set for which an overall maximum likelihood marker ordering is sought. If more then 2 data-sets need to be merged, this command must be called repeatedly. The merged data-set built uses all the information available in the two data-sets but independent maximum likelihood multipoint parameter estimations are performed on each data-set. This can be used to eg. determine sex-specific racombination ratio.

There is no restriction on the type of data-sets that can be merged using this operator (including merging of radiated hybrid and genetic recombination data).

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *DsId1 DsId2*: the numerical If of the two data sets to merge.

Returns: a list summarizing some basic information on the merged data-set:

- numerical Id of the data-set.
- type of the data-set (i.e., `merged by order`)
- number of markers in the data-set.
- number of individuals/hybrids in the data-set.

The number of markers is usually strictly less than the number of markers in each of the merged data-set (identical markers are detected by their identical name).

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> dsload Data/rh.cg
{2 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> dsmergor 1 2
{3 merged by order 59 326}
```

See also:

- `dsmergen` (2.2.2)
- `dsload` (2.2.1)
- `dsinfo` (2.2.5)
- `dsget` (2.2.6)

2.2.4 heapequiset

Activates or desactivates the detection of “equivalent” orders on datasets merged using the `dsmergor` command.

Synopsis: The `heapequiset` command is invoked either as:

- `heapequiset` *Options*
- `heapequiset` *on/off flag*

Description: When two datasets are merged, it is often the case that the two data sets only share few markers. When the two such datasets are merged using `dsmergor`, many different orders correspond to equivalent situations on the two datasets and they will get the same log-likelihood. Imagine for example that the dataset 1 is informative on markers A, B, D and E and that the dataset 2 is informative on markers A, C and E. Then the orders $ABCDE$, $ACBDE$ and $ABDCE$ are equivalent and will get the same loglikelihhod since they both correspond to the two same orders: $ABDE$ on dataset one and ACE on the dataset 2.

This phenomenon will populate the heap with many equivalent maps and also severely limit the efficacy of the `buildfw` process we uses the heap structure internally to detect if two orders with close enough likelihoods exist. At saome point, It may be impossible to insert a marker in the framework map because equivalent orders (therefore with the same likelihood) exist in the heap. To avoid this, activate the `heapequiset` command. It is important to mention that as soon as the mode is activated, “equivalent” orders are expunged from the heap. The command is still in alpha stage.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *level*: the on/off flag (0 for off or 1 for on).

Returns: nothing.

Example:

```
# we load and merge two datasets
CG> dsload Data/bc.cg
...
CG> dsload Data/rh.cg
...
CG> dsmergor 1 2
...
# we evaluate two different but equivalent orders
CG> mrkselset {17 29 30 18 39 19}

CG> sem

Map -1 : log10-likelihood =  -164.94
-------:
 Set : Marker List ...
   1 : MS17        MS18     MS19
   2 : MS17 G9 G10      G19 MS19

# we move the marker 18 which is only defined in the first dataset
CG> mrkselset {17 29 18 30  39 19}

CG> sem

Map -1 : log10-likelihood =  -164.94
-------:
 Set : Marker List ...
   1 : MS17    MS18         MS19
   2 : MS17 G9      G10 G19 MS19

# the two orders have entered the heap
CG> heaprint

Map  0 : log10-likelihood =  -164.94
-------:
 Set : Marker List ...
   1 : MS17        MS18     MS19
   2 : MS17 G9 G10      G19 MS19

Map  1 : log10-likelihood =  -164.94
-------:
 Set : Marker List ...
   1 : MS17    MS18         MS19
   2 : MS17 G9      G10 G19 MS19

# we now activate the detection of equivalent orders
CG> heapequiset 1
```

```
# the equivalent order is expunged
CG> heaprint

Map  0 : log10-likelihood =  -164.94
-------:
 Set : Marker List ...
   1 : MS17        MS18     MS19
   2 : MS17 G9 G10      G19 MS19

# we evaluate a third equivalent order
CG> mrkselset {17 18 29 30  39 19}

CG> sem

Map -1 : log10-likelihood =  -164.94
-------:
 Set : Marker List ...
   1 : MS17 MS18            MS19
   2 : MS17      G9 G10 G19 MS19

# it did not entered the heap
CG> heaprint

Map  0 : log10-likelihood =  -164.94
-------:
 Set : Marker List ...
   1 : MS17        MS18     MS19
   2 : MS17 G9 G10      G19 MS19

CG>
```

See also:

- `dsmergor` (2.2.3)

2.2.5 `dsinfo`

Displays a summary of information on all the data-sets that are currently available.

Synopsis: The `dsinfo` command is invoked either as:

- `dsinfo` *Options*
- `dsinfo`

Description: The `dsinfo` command display a table of information on all the datat-sets currently available in the software. For each data-set, a line summarizes:

- the numerical Id of the data-set.
- the type of the data-set.
- the number of markers that appear in the data-set.

- the number of individuals (hybrids) that appear in the data-set.
- the number of order constraints in the data-set.
- if the data-set is a merged data-set, which data-set it merges.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> dsload Data/rh.cg
{2 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> dsmergor 1 2
{3 merged by order 59 326}
CG> dsinfo

Data Sets :
----------:
ID        Data Type    markers individuals             filename constraints...
 1     f2 backcross         20         208                bc.cg
 2       haploid RH         53         118                rh.cg
 3  merged by order         59         326                                  ...
```

See also:

- `dsget` (2.2.6)
- `dsload` (2.2.1)

2.2.6 dsget

Displays a summary of all the data-sets that are currently loaded.

Synopsis: The `dsget` command is invoked either as:

- `dsget` *Options*
- `dsget`

Description: The `dsget` command returns a list of descriptions of all the data-sets that have been loaded since the beginning of the session. For each data-set, the numerical id of the data-set, its type, the number of markers that appears in it and the numbers of individuals (hybrid) typed int it are reported in a list.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: a list of list of information on the available data-sets.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> dsload Data/rh.cg
{2 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> dsmergor 1 2
{3 merged by order 59 326}
CG> dsget
{1 "f2 backcross" /homes/thomas/CartaGene/dev/test/Data/bc.cg 20 208} {2 "r...
```

See also:

- `dsinfo` (2.2.5)
- `dsload` (2.2.1)
- `mrkinfo` (2.3.5)

2.3 Markers, linkage groups and 2-points estimations

For a given dataset, CarTHaGene performs all 2-points LOD calculations when the data set is loaded. This explains why loading large datasets can be (relatively) slow for expensive pedigree (eg. outbreds). All 2-points information can be requested using the `mrklod2p`, `mrkfr2p` and `mrkdist2p` commands. The `mrkdouble` command reports all pairs of markers that have compatible typing (i.e., markers which may be identical) along with their 2-points LOD. This can occur in two cases:

- the two markers have never been typed on a single common individual. These 2 markers should not be merged and this is clearly indicated by the fact that their 2-points LOD is equal to 0.
- the two markers have been typed on a possibly large number of individuals and no sure recombination (or breakage for RH) exists between the two markers. The 2 markers can be merged and this is indicated by the usually strong 2-points LOD between the 2 markers.

When two markers are merged using the `mrkmerge` command, no information is lost. All the information is merged in the first marker and the second marker is deselected. The number of markers to map is lowered and the information stronger. This is always a good thing.

Several other commands can be used to get information about markers. The most useful ones are probably the commands that can convert marker names to their numerical Id (`mrkid` and `mrkids` commands) and vice-versa (`mrkname` and `mrknames` commands).

The 2-points estimations are the basis of the linkage group computation process performed using the `group` command. This command puts in the same linkage group any two markers such that:

- their 2-points LOD is above a given threshold *and*
- their 2-points distance is below a given threshold.

The two conditions must be met together. Once groups are computed using this command, you can use a group using the `groupget` command.

2.3.1 `mrkallget`

Get the list of all the numerical ids of all the markers currently known in the current data-set.

Synopsis: The `mrkallget` command is invoked as either one of :

- `mrkallget` *Options*
- `mrkallget`

Description: returns the list of all the numerical Ids of all the markers known to the system. This can be typicallly used as an argument to `mrkselset`.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns : The list of all the numerical Ids of all the markers known.

Example :

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> dsload Data/rh.cg
{2 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> mrkallget
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29\
 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 5\
5 56 57 58 59
# to select all markers
CG> mrkselset [mrkallget]
```

See also:

- `mrkselset` (2.3.13)
- `mrkselget` (2.3.12)

2.3.2 `mrkdist2p`

Displays the two-points distance matrix using the specified mapping function (h/k).

Synopsis: The `mrkdist2p` command is invoked as either one of :

- `mrkdist2p` *Options*
- `mrkdist2p` *Unit*

Description: `mrkdist2p` prints the two-points distance estimation using either Haldane (h) or Kosambi (k) mapping function. For radiated hybrid data, Ray distances are always used, independently of the argument (which must nevertheless be given).

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *Unit*: may either `h` (Haldane) or `k` (Kosambi).

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> group 0.1 9

Linkage Groups :
---------------:
LOD threshold=9.00
Distance threshold=10.00:

 Group ID : Marker ID List ...
        1 : 17 19 20 18
        2 : 14
        3 : 11 13 12 16 15
        4 : 10
        5 : 2 4 3 7 6 5 9 8
        6 : 1
6
CG> mrkselset [groupget 3]

CG> mrkdist2p k

Print two points distance matrices of the loci selection :
----------------------------------------------------------:

Data Set Number  1 :
                  MS11  MS13  MS12  MS16  MS15
                  ------------------------------
           MS11 |------  7.8  10.3  22.5  16.2
           MS13 |  7.8 ------  0.0  14.5  15.5
           MS12 | 10.3   0.0 ------  8.3   3.5
           MS16 | 22.5  14.5   8.3 ------  4.9
           MS15 | 16.2  15.5   3.5   4.9 ------
```

See also:

- `mrklod2p` (2.3.7)
- `mrkfr2p` (2.3.3)

2.3.3 `mrkfr2p`

Displays the two-points recombination (or breakage) ratio matrix of the current loci selection.

Synopsis : The `mrkfr2p` command is invoked either as:

- `mrkfr2p` *Options*
- `mrkfr2p`

Description : `mrkfr2p` prints the two-points recombination (or breakage) ratio for all pairs of loci in the current list of selected loci.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> group 0.1 9

Linkage Groups :
---------------:
LOD threshold=9.00
Distance threshold=10.00:

 Group ID : Marker ID List ...
        1 : 17 19 20 18
        2 : 14
        3 : 11 13 12 16 15
        4 : 10
        5 : 2 4 3 7 6 5 9 8
        6 : 1
6
CG> mrkselset [groupget 3]

CG> mrkfr2p

Print two points recombination fractions  matrices of the loci selection :
--------------------------------------------------------------------------:

        MS11  MS13  MS12  MS16  MS15
       -----------------------------
 MS11 |------  0.1   0.1   0.2   0.2
 MS13 |  0.1 ------  0.0   0.1   0.1
 MS12 |  0.1   0.0 ------  0.1   0.0
 MS16 |  0.2   0.1   0.1 ------  0.0
 MS15 |  0.2   0.1   0.0   0.0 ------
```

See also:

- `mrklod2p` (2.3.7)
- `mrkdist2p` (2.3.2)

2.3.4 `mrkid`

Returns the numerical Id of the marker of the given name.

Synopsis: The `mrkid` command is invoked either as:

- `mrkid` *Options*
- `mrkid` *name*
- `mrkids` *names* or list of names

Description: the command returns the unique numerical id of the marker whose name is given as an argument. This numerical id is used in several other commands. The variant `mrkids` can take as argument several names or a list of names and returns a list of numerical Ids.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *name*: the name of the marker as it appears in the dataset it has been loaded from.

Returns: the numerical id of the marker.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> mrkid MS9
9
CG> mrkids MS9 MS10
9 10
CG> mrkmerge [mrkid MS9] [mrkid MS10]
Markers 9 and 10 merged in 9.

#Markers 9 and 10 merged in 9.
```

See also:

- `mrkinfo` (2.3.5)

2.3.5 `mrkinfo`

Displays basic information on all known markers.

Synopsis: The `mrkinfo` command is invoked either as:

- `mrkinfo` *Options*
- `mrkinfo`

Description: For each marker known to the system, the command displays its numerical Id, its name, the numerical Id of all the data-sets in which the marker is defined and also if the marker has been merged (see `mrkmerge`, section 2.3.9) with another marker: when the marker i has been merged with the marker j the result of the merging being stored in i, then the information on i mentions j.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: nothing

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> dsload Data/rh.cg
{2 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> dsmergor 1 2
{3 merged by order 59 326}
CG> mrkinfo

Markers :
--------:
Num                Names : Sets         Merges
  1                  MS1 :   1   2   3
  2                  MS2 :   1   2   3
  3                  MS3 :   1   2   3
  4                  MS4 :   1   2   3
  5                  MS5 :   1   2   3
  6                  MS6 :   1   2   3
  7                  MS7 :   1   2   3
  8                  MS8 :   1   2   3
  9                  MS9 :   1   2   3
 10                 MS10 :   1       3
 11                 MS11 :   1       3
 12                 MS12 :   1       3
 13                 MS13 :   1   2   3
 14                 MS14 :   1       3
 15                 MS15 :   1   2   3
 16                 MS16 :   1       3
 17                 MS17 :   1   2   3
 18                 MS18 :   1       3
 19                 MS19 :   1   2   3
 20                 MS20 :   1   2   3
 21                   G1 :       2   3
 22                   G2 :       2   3
 23                   G3 :       2   3
 24                   G4 :       2   3
 25                    G5 :       2   3
 26                    G6 :       2   3
 27                    G7 :       2   3
 28                    G8 :       2   3
 29                    G9 :       2   3
 30                   G10 :       2   3
 31                   G11 :       2   3
 32                   G12 :       2   3
 33                   G13 :       2   3
 34                   G14 :       2   3
 35                   G15 :       2   3
 36                   G16 :       2   3
 37                   G17 :       2   3
 38                   G18 :       2   3
 39                   G19 :       2   3
 40                   G20 :       2   3
 41                   G21 :       2   3
 42                   G22 :       2   3
 43                   G23 :       2   3
 44                   G24 :       2   3
 45                   G25 :       2   3
 46                   G26 :       2   3
 47                   G27 :       2   3
 48                   G28 :       2   3
 49                   G29 :       2   3
 50                   G30 :       2   3
 51                   G31 :       2   3
 52                   G32 :       2   3
 53                   G33 :       2   3
 54                   G34 :       2   3
 55                   G35 :       2   3
 56                   G36 :       2   3
 57                   G37 :       2   3
 58                   G39 :       2   3
 59                   G40 :       2   3
```

See also :

- `mrkmerge` (2.3.9)
- `dsinfo` (2.2.5)

2.3.6 `mrklast`

Returns the numerical Id of the last marker known.

Synopsis : The `mrklast` command is invoked either as:

- `mrklast` *Options*
- `mrklast`

Description: returns the numerical Id of the last marker loaded i.e., the largest marker numerical Id currently used. This can be useful to process all markers in a Tcl macro.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: the numerical Id of the last marker known.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> dsload Data/rh.cg
{2 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> mrklast
59
```

See also:

- `mrkallget` (2.3.1)

2.3.7 `mrklod2p`

Displays the two-points LOD matrix for all currently active markers.

Synopsis : The `mrklod2p` command is invoked either as:

- `mrklod2p` *Options*
- `mrklod2p`

Description: `mrklod2p` prints the two-points LOD for all pairs of loci in the list of currently selected loci.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Example:

```
# we first load a data set
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
# and select some markers in it
CG> mrkselset {11 12 13 14 15 16 17 18 19 20}

# then ask for the 2pt LOD matrix
CG> mrklod2p

             11    12    13    14    15    16    17    18    19    20
           MS11  MS12  MS13  MS14  MS15  MS16  MS17  MS18  MS19  MS20
```

```
            ------------------------------------------------------------
      MS11 |------ 10.9  11.9   3.6   3.6   5.5   0.1   0.4   0.6   0.5
      MS12 | 10.9 ------ 18.7   3.6  26.5  25.9   2.1   4.6   4.2   4.3
      MS13 | 11.9  18.7 ------  6.0   4.7   9.7   0.9   0.7   0.6   0.7
      MS14 |  3.6   3.6   6.0 ------  0.0   3.6   1.5   0.0   0.5   0.7
      MS15 |  3.6  26.5   4.7   0.0 ------ 26.6   3.8   6.9   6.9   6.2
      MS16 |  5.5  25.9   9.7   3.6  26.6 ------  6.3  10.3   8.2   7.7
      MS17 |  0.1   2.1   0.9   1.5   3.8   6.3 ------ 16.0  14.1  11.3
      MS18 |  0.4   4.6   0.7   0.0   6.9  10.3  16.0 ------ 27.1  22.9
      MS19 |  0.6   4.2   0.6   0.5   6.9   8.2  14.1  27.1 ------ 34.8
      MS20 |  0.5   4.3   0.7   0.7   6.2   7.7  11.3  22.9  34.8 ------
```

See also:

- `mrkselset` (2.3.13)
- `mrkmerge` (2.3.9)

2.3.8 `mrkdouble`

Analyzes possible equivalent haplotypes (pairs of markers whose genotyping are “compatible” and could therefore be merged) in the current markers selection.

Synopsis : The `mrkdouble` command is invoked either as:

- `mrkdouble` *Options*
- `mrkdouble`

Description: `mrkdouble` analyzes, for each pair of marker, if the two markers have compatible genotyping (i.e. if the genotyping of the two markers can be assumed to be the same w/o any inconsistency). When two markers are “compatible”, their two-points LOD are presented to decide to merge (or not) the two markers (compatible markers with null LOD are simply markers that are compatible because they do not share any information and should therefore not be merged).

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Example:

```
# we first load a data set
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
# and select some markers in it
CG> mrkselset {11 12 13 14 15 16 17 18 19 20}

# this identifies pairs of markers that can be merged
CG> mrkdouble
```

```
Possible double markers:

               MS11 = MS14           [3.6]
               MS12 = MS13           [18.7]
               MS12 = MS14           [3.6]
               MS13 = MS14           [6.0]
               MS14 = MS16           [3.6]
               MS14 = MS17           [1.5]
               MS17 = MS18           [16.0]
```

See also:

- `mrkmerge` (2.3.9)
- `mrkselset` (2.3.13)
- `mrkmerge` (2.3.7)

2.3.9 `mrkmerge`

Merges two markers in one.

Synopsis : The `mrkmerge` command is invoked either as:

- `mrkmerge` *Options*
- `mrkmerge` *mrkid1 mrkid2*

Description : The command is used to merge two markers whose observed genotypes on all individuals are compatible with each other (closely related markers for which no recombination has been observed). Such “double markers” are identified automatically using the command `mrklod2p`.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *mrkid1 mrkid2*: the observations on marker number *mrkid1* and marker *mrkid2* are merged in the marker *mrkid1* and the marker *mrkid2* is removed from the set of selected markers.

Returns : Nothing

Example :

```
# we first load a data set
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
# and ask for markers that can be merged
CG> mrkdouble
```

```
Possible double markers:

                MS9 = MS10            [7.2]
                MS9 = MS14            [3.6]
               MS10 = MS11            [3.6]
               MS11 = MS14            [3.6]
               MS12 = MS13            [18.7]
               MS12 = MS14            [3.6]
               MS13 = MS14            [6.0]
               MS14 = MS16            [3.6]
               MS14 = MS17            [1.5]
               MS17 = MS18            [16.0]


# then merge markers 9 and 10 in 9.
CG> mrkmerge [mrkid MS9] [mrkid MS10]
Markers 9 and 10 merged in 9.
```

See also :

- `mrklod2p` (2.3.7)

2.3.10 `mrkmerget`

Retrieves the sets of markers that have been merged together.

Synopsis : The `mrkmerget` command is invoked either as:

- `mrkmerget` *Options*

Description: The command is used to retrieve the sets of markers that have been merged together.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns : a list of list of markers id. The first marker of a sublist represent the others.

Example :

```
# we first load a data set
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
# and ask for markers that can be merged
CG> mrkdouble

Possible double markers:

                MS9 = MS10            [7.2]
                MS9 = MS14            [3.6]
```

```
               MS10 = MS11            [3.6]
               MS11 = MS14            [3.6]
               MS12 = MS13            [18.7]
               MS12 = MS14            [3.6]
               MS13 = MS14            [6.0]
               MS14 = MS16            [3.6]
               MS14 = MS17            [1.5]
               MS17 = MS18            [16.0]


# then merge markers 9 and 10 in 9.
CG> mrkmerge [mrkid MS9] [mrkid MS10]
Markers 9 and 10 merged in 9.

#...
CG> mrkmerge [mrkid MS12] [mrkid MS13]
Markers 12 and 13 merged in 12.

CG> mrkmerge [mrkid MS12] [mrkid MS14]
Markers 12 and 14 merged in 12.

#retrive the information
CG> mrkmerget
{9 10 } {12 13 14 }
```

See also:

- `mrkmerge` (2.3.9)

2.3.11 `mrkname`

Displays the name of the marker number *mrkid*.

Synopsis : The CarthaGene `mrkname` command is invoked as either one of :

- `mrkname` *Options*
- `mrkname` *mrkid*
- `mrknames` *mrkids* or a list of *mrkid*

Description : The commande `mrkname` displays the name of the marker whose number is *mrkid*. The `mrkids` commands takes as arguments several Ids or a list of Ids and return a list of marker names.

Arguments :

- *Options*: `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *mrkid*: the numerical id of the marker whose name will be displayed.

Returns : The name of the marker.

Example :

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> mrkname 1
MS1
CG> mrknames 1 2 4 5
MS1 MS2 MS4 MS5
```

See also :

- `mrkid` (2.3.4)
- `mrklast` (2.3.6)
- `mrkinfo` (2.3.5)
- `mrkselget` (2.3.12)

2.3.12 `mrkselget`

Returns the list of the numerical Id of all active markers.

Synopsis: The `mrkselget` command is invoked either as:

- `mrkselget` *Options*
- `mrkselget`

Description: the command returns the list of the numerical Ids of the active markers (see `mrkselset`).

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: the list of the numerical Id of all the active markers.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> group 0.3 3

Linkage Groups :
---------------:
LOD threshold=3.00
Distance threshold=30.00:

 Group ID : Marker ID List ...
        1 : 52
        2 : 42 48 47 45 46 44 43
        3 : 41
```

```
         4 : 38
         5 : 37 39
         6 : 36 40
         7 : 35
         8 : 25
         9 : 22
        10 : 19 32 31 28 30 27 26 20 21
        11 : 13 14
        12 : 11 12
        13 : 10 15 16 18 17
        14 : 7
        15 : 6 51 8 53 50 9
        16 : 4 34 33 24 23 5
        17 : 3 29
        18 : 2 49
        19 : 1
19
CG> mrkselset [groupget 10]

CG> mrkselget
19 32 31 28 30 27 26 20 21
CG> mrknames [mrkselget]
G5 G18 G17 G14 G16 G13 G12 G6 G7
```

See also:

- `mrkselset` (2.3.13)
- `mrkallget` (2.3.1)

2.3.13 `mrkselset`

Changes the set and the default order of currently active markers.

Synopsis: The `mrkselset` command is invoked either as:

- `mrkselset` *Options*
- `mrkselset` *List of markers id*

Description: the command sets the list of currently active markers to the list of markers received as argument. The order of the markers in the list is used as the default order by some commands (such as `sem`).

If the set of the markers chosen differs from the previous list of active markers (available through `mrkselget`), then the current heap of maps becomes obsolete and is therefore emptied and reset.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *List of markers id*: a list of all the numerical ids of all the markers that will be active.

Returns: nothing.

Example :

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> group 0.3 3

Linkage Groups :
---------------:
LOD threshold=3.00
Distance threshold=30.00:

 Group ID : Marker ID List ...
        1 : 52
        2 : 42 48 47 45 46 44 43
        3 : 41
        4 : 38
        5 : 37 39
        6 : 36 40
        7 : 35
        8 : 25
        9 : 22
       10 : 19 32 31 28 30 27 26 20 21
       11 : 13 14
       12 : 11 12
       13 : 10 15 16 18 17
       14 : 7
       15 : 6 51 8 53 50 9
       16 : 4 34 33 24 23 5
       17 : 3 29
       18 : 2 49
       19 : 1
19
CG> mrkselset [groupget 10]

CG> sem

Map -1 : log10-likelihood =  -161.87
-------:
 Set : Marker List ...
   1 : G5 G18 G17 G14 G16 G13 G12 G6 G7
```

See also:

- `mrkselget` (2.3.12)
- `mrkallget` (2.3.1)
- `groupget` (2.3.18)

2.3.14 `mrkadd`

Inserts the marker of the given numerical Id at the end of the current marker selection.

Synopsis: The `mrkadd` command is invoked as:

- `mrkadd` *name*

Description: the command inserts the marker with the numerical id given as argument after the last marker in the current marker selection. En error is reported if the marker already appears in the list of selected markers.

Arguments:

- *name*: the numerical Id of the marker.

Returns: Nothing.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> group 0.3 3

Linkage Groups :
---------------:
LOD threshold=3.00
Distance threshold=30.00:

 Group ID : Marker ID List ...
        1 : 52
        2 : 42 48 47 45 46 44 43
        3 : 41
        4 : 38
        5 : 37 39
        6 : 36 40
        7 : 35
        8 : 25
        9 : 22
       10 : 19 32 31 28 30 27 26 20 21
       11 : 13 14
       12 : 11 12
       13 : 10 15 16 18 17
       14 : 7
       15 : 6 51 8 53 50 9
       16 : 4 34 33 24 23 5
       17 : 3 29
       18 : 2 49
       19 : 1
19
CG> mrkselset [groupget 10]

CG> mrkselget
19 32 31 28 30 27 26 20 21
CG> mrkadd 7

CG> mrkselget
19 32 31 28 30 27 26 20 21 7
```

See also:

- `mrkid` (2.3.4)
- `mrkdel` (2.3.16)
- `mrkselget` (2.3.12)
- `mrkselset` (2.3.13)

2.3.15 `mrksub`

Removes the marker of the given numerical Id from the current marker selection.

Synopsis: The `mrksub` command is invoked as:

- `mrksub` *Id*

Description: the command removes the marker with the numerical Id given as argument from the current marker selection. No error is reported if the marker does not appear in the current list.

Arguments:

- *Id*: the numerical Id of the marker as it appears in the dataset it has been loaded from.

Returns: Nothing.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> group 0.3 3

Linkage Groups :
---------------:
LOD threshold=3.00
Distance threshold=30.00:

 Group ID : Marker ID List ...
        1 : 52
        2 : 42 48 47 45 46 44 43
        3 : 41
        4 : 38
        5 : 37 39
        6 : 36 40
        7 : 35
        8 : 25
        9 : 22
       10 : 19 32 31 28 30 27 26 20 21
       11 : 13 14
       12 : 11 12
       13 : 10 15 16 18 17
       14 : 7
       15 : 6 51 8 53 50 9
       16 : 4 34 33 24 23 5
       17 : 3 29
       18 : 2 49
       19 : 1
19
CG> mrkselset [groupget 10]

CG> mrkselget
19 32 31 28 30 27 26 20 21
CG> mrksub 20

CG> mrkselget
19 32 31 28 30 27 26 21
```

See also:

- `mrkid` (2.3.4)
- `mrkdel` (2.3.16)
- `mrkselget` (2.3.12)
- `mrkselset` (2.3.13)

2.3.16 `mrkdel`

Remove the marker of the given name from the current marker selection.

Synopsis: The `mrkdel` command is invoked as:

- `mrkdel` *name*

Description: the command removes the marker with the name given as argument from the current marker selection. No error is reported if the marker does not appear in the current list.

Arguments:

- *name*: the name of the marker as it appears in the dataset it has been loaded from.

Returns: Nothing.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> group 0.3 3

Linkage Groups :
---------------:
LOD threshold=3.00
Distance threshold=30.00:
 Group ID : Marker ID List ...
        1 : 52
        2 : 42 48 47 45 46 44 43
        3 : 41
        4 : 38
        5 : 37 39
        6 : 36 40
        7 : 35
        8 : 25
        9 : 22
       10 : 19 32 31 28 30 27 26 20 21
       11 : 13 14
       12 : 11 12
       13 : 10 15 16 18 17
       14 : 7
       15 : 6 51 8 53 50 9
       16 : 4 34 33 24 23 5
       17 : 3 29
       18 : 2 49
       19 : 1
19
CG> mrkselset [groupget 10]

CG> mrknames [mrkselget]
G5 G18 G17 G14 G16 G13 G12 G6 G7
CG> mrkdel G6

CG> mrknames [mrkselget]
G5 G18 G17 G14 G16 G13 G12 G7
```

See also:

- `mrkname` (2.3.11)
- `mrksub` (2.3.15)
- `mrkselget` (2.3.12)
- `mrkselset` (2.3.13)

2.3.17 `group`

Computes linkage groups in a data-set.

Synopsis: The `group` command is invoked as either:

- `group` *Options*
- `group` *Dist-Threshold LOD-Threshold*

Description: The `group` command computes cluster of markers, trying to identify linkage groups. The clustering is performed by putting in the same cluster any two markers such that their estimated two-points distance (using Haldane for genetic data) and their two-points LOD is respectively smaller than *Dist-Threshold* and larger than *LOD-Threshold*. This analysis is transitive in the sense that if there is reason to cluster together markers A and B on one side and markers B and C on another, then all threee markers will be put together (whether there is evidence for or against putting A and C together). Parameters should be typically adjusted to get a number of linkage groups consistent with the number of chromosomes in the organism.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *Dist-Threshold*: the maximum two-point distance (Haldane/Ray).
- *LOD-Threshold*: the minimum two-point LOD.

Returns: the number of linkage groups detected.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> group 0.5 3

Linkage Groups :
---------------:
LOD threshold=3.00
Distance threshold=50.00:

 Group ID : Marker ID List ...
        1 : 13 35 14
        2 : 10 18 17 16 15 11 12
        3 : 7
        4 : 6 53 52 8 51 50 9
        5 : 4 34 33 24 23 5 25
        6 : 3 32 29 31 30 28 27 26 20 19 21 22
        7 : 2 49 48 47 46 45 44 43 42 38 37 40 39 36 41
        8 : 1
8
CG>
```

See also:

- `groupget` (2.3.18)

2.3.18 `groupget`

Returns the linkage group from its group number.

Synopsis : The CarthaGene `groupget` command is invoked as either one of :

- `groupget` *Options*
- `groupget` *GrID*

Description : `groupget` returns a linkage group as a list of markers from its group number (or so-called Group ID). The command `group` must have been claled before hand to compute these groups.

Arguments :

- *Options*: `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *GrID*: the number of the group to return.

Returns : The contents of the linkage group as a list of markers.

Example :

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> group 0.3 3

Linkage Groups :
---------------:
LOD threshold=3.00
Distance threshold=30.00:

 Group ID : Marker ID List ...
        1 : 52
        2 : 42 48 47 45 46 44 43
        3 : 41
        4 : 38
        5 : 37 39
        6 : 36 40
        7 : 35
        8 : 25
        9 : 22
       10 : 19 32 31 28 30 27 26 20 21
       11 : 13 14
       12 : 11 12
       13 : 10 15 16 18 17
       14 : 7
       15 : 6 51 8 53 50 9
       16 : 4 34 33 24 23 5
       17 : 3 29
       18 : 2 49
       19 : 1
19
CG> groupget 10
19 32 31 28 30 27 26 20 21
CG> mrkselset [groupget 10]

CG> sem

Map -1 : log10-likelihood =  -161.87
-------:
 Set : Marker List ...
   1 : G5 G18 G17 G14 G16 G13 G12 G6 G7
```

See also :

- `group` (2.3.17)

2.3.19 `groupmerge`

Sets the current marker selection list to the list of all the markers appearing in the groups indicated.

Synopsis : The CarthaGene `groupmerge` command is invoked as:

- `groupmerge` *GrID1 GrID2...*

Description : `groupmerge` sets the current list of selected markers to the union of all the markers that appear in the indicated linkage groups. The groups are indicated by their group number (so-called Group ID). The command `group` must have been called before hand to compute these groups. This command can be useful if you know that two different groups correspond to a single chromosome in practice.

Arguments:

- *GrID**: the number of the groups to merge.

Returns : Nothing.

Example :

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> group 0.5 3

Linkage Groups :
---------------:
LOD threshold=3.00
Distance threshold=50.00:

 Group ID : Marker ID List ...
        1 : 13 35 14
        2 : 10 18 17 16 15 11 12
        3 : 7
        4 : 6 53 52 8 51 50 9
        5 : 4 34 33 24 23 5 25
        6 : 3 32 29 31 30 28 27 26 20 19 21 22
        7 : 2 49 48 47 46 45 44 43 42 38 37 40 39 36 41
        8 : 1
8
CG> groupmerge 3 4 5

CG> mrkselget
7 6 53 52 8 51 50 9 4 34 33 24 23 5 25
```

See also :

- `group` (2.3.17)

2.4 Maps and the Heap

A map in CarTHaGene defines the parameters of a probabilistic model of the genetic/RH data available on the currently active set of markers. In the simplest cases (eg. one genetic population), a map is completely defined by two components: an order of the markers and the recombination probabilities between adjacent markers. As in most existing mapping software (eg. MapMaker), all the probabilistic models used in CarTHaGene assume a complete absence of interference on genetic data. In the case of RH data, recombination probabilities are replaced by breakage probabilities along with a retention probability parameter. The probabilistic model used is the so-called “equal retention model” [LBLC95] which seems to be an ideal compromise between simplicity, efficiency and realism. Given a map, since all the parameters of the probabilistic model are fixed by the map, it is possible to compute the probability of the data given the model often coined as the likelihood of the data. Note that the continuous part of the probabilistic model (probabilities) are always estimated using a maximum-likelihood criteria in CarTHaGene. This criteria ia a rigorous criteria which has several attractive theoretical properties. It is optimised in CarTHaGene using variants of the EM algorithm [DLR77].

The probabilistic parameters used in the model can be converted to so-called distances using traditional mapping functions: Haldane and Kosambi that convert recombination probabilities to Morgans for genetic data and the usual $-\log(1.0-\theta)$ for mapping breakage probabilities to Rays. Note that the probabilistic model used in consistent with Haldane mapping function. The Kosambi mapping function, which is usually the favorite one, has theoretical weaknesses and we advise not to use it.

When several datasets are merged using the `dsmergen` command, since the underlying assumption is that all datasets share the same parameters, a single “consensus” map suffices to define a probabilistic model for all the data. But such maps do not suffice for more complex models, eg. when several datasets are merged using the `dsmergor` command. In this case, the assumption is that all dataset merged using `dsmergor` share a consistent order but each with different specific distances (i.e., probabilities).

In this case, a map is defined by a collection of simple maps (as above) that share a consistent order.

2.4.1 The heap

As long as the set of selected markers is unchanged, all the maps produced and examined by CarTHaGene in all the mapping process are tentatively stored in a central map storage called “the heap”[1]. The heap remembers all the k best (w.r.t. the loglikelihood) maps found since the last time the marker selection was changed. The parameter k is user configurable using the `heapsize` command.

To actually visualize the maps stored in the heap, you can use either a standard or detailed impression of the contents of the heap. This is done using respectively the `heaprint` and `heaprintd` commands. The whole content of the heap can be fetch and stored in a list (for further analysis using the scripting language Tcl) with the `heapget` command.

Actually, each map stored in the heap has a numerical Id and this numerical id can be used to print/fetch the maps stored in the heap. This is done using repectively the `maprint` and `maprintd` commands or the `mapget` command. Because the maps are always stored in the heap in a “standardized” order, a `maprintdr` function is also available to print a map using a reverse marker order. It is also possible to compute the number of obligate chromosome breaks of a map using the `mapocb` command. Finally, if you only want to access the order of the markers used in a map, the command `mapordget` will returns the marker ordering of the map indicated in a list.

The heap main use is to provide an approximation of all the markers ordering around the optimal markers ordering. The heap can be directly examined using the previous `heaprint` commands but a specific command is available to perform the analysis of alternative maps in the heap. The `heaprinto` command dumps all orders in the heap, one by line, from the best to the worst. For each such order, the way the order differs from the order of the original map is made explicit and the difference of loglikelihood with the best map is reported.

---

[1] This is the usual name for the data structure [CLR90] used in CarTHaGene to actually store the map. In practice, a hash map is also used to check if a map aready appears in the heap.

2.4.2 `heapsize`

Resize the Heap.

Synopsis : The CarthaGene `heapsize` command is invoked as either one of :

- `heapsize` *Options*
- `heapsize` *NbMap*

Description : `heapsize` enables to resize the heap. Even if the heap already contains maps, the k best maps are kept where k is the new size. If the size of a full heap is decreased, the worst maps are lost. The size of the heap as a limited simpact on CarthaGène efficiency and sizes up to `1023` should be fine.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *NbMap* : The number of maps the heap must contain. The *NbMap* argument must be a positive non null integer.

Example :

```
# starting with a heap of size 7
CG> heapsize 7

# filling it
CG> dsload Data/bc1.cg
{1 f2 backcross 17 208 /homes/thomas/CartaGene/dev/test/Data/bc1.cg}
CG> sem

Map -1 : log10-likelihood =  -485.24
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS13 MS6 MS11 MS17 MS16 MS8 MS7 MS2 MS3 MS9 MS15 MS12 MS20 MS1\
9 MS1

CG> flips 3 3.0 1
...
CG> heaprint

Map  0 : log10-likelihood =  -454.74
-------:
 Set : Marker List ...
   1 : MS6 MS5 MS4 MS11 MS13 MS16 MS17 MS8 MS7 MS2 MS3 MS9 MS12 MS15 MS19 MS2\
0 MS1

Map  2 : log10-likelihood =  -454.58
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS17 MS16 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS19 MS2\
0 MS1

Map  6 : log10-likelihood =  -454.09
```

```
-------:
 Set : Marker List ...
   1 : MS6 MS5 MS4 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS19 MS2\
0 MS1

Map  1 : log10-likelihood =  -452.77
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS2 MS3 MS9 MS12 MS15 MS20 MS1\
9 MS1

Map  4 : log10-likelihood =  -452.11
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS20 MS1\
9 MS1

Map  5 : log10-likelihood =  -451.97
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS2 MS3 MS9 MS12 MS15 MS19 MS2\
0 MS1

Map  3 : log10-likelihood =  -451.32
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS19 MS2\
0 MS1

# keeping the 3 best one.
CG> heapsize 3

CG> heaprint

Map  0 : log10-likelihood =  -452.11
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS20 MS1\
9 MS1

Map  1 : log10-likelihood =  -451.97
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS2 MS3 MS9 MS12 MS15 MS19 MS2\
0 MS1

Map  2 : log10-likelihood =  -451.32
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS19 MS2\
0 MS1

CG>
```

See also :

- `heaprint` (2.4.4)
- `heaprintd` (2.4.5)

2.4.3 `heapsizeget`

Get the size of the Heap.

Synopsis : The CarthaGene `heapsizeget` command is invoked as either one of :

- `heapsizeget` *Options*

Description : `heapsizeget` enables to get the size of the heap.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: a integer.

Example :

```
CG> heapsizeget
15
# incresing of 5 the size oh the heap
CG> heapsize [expr [heapsizeget] + 5]

CG> heapsizeget
20
CG>
```

See also :

- `heapsize` (2.4.2)

2.4.4 `heaprint`

Display the Heap of maps.

Synopsis : The CarthaGene `heaprint` command is invoked as either one of :

- `heaprint` *Options*
- `heaprint`

Description : `heaprint` provide a output of the heap of maps. The maps are ordered by loglikelihood. The best map is always the last printed. The informations available are the id of a map, the score, and the ordered list of the markers printed by names. If the data set the maps are computed on, is merged by order, relatives orders for each merged data sets are printed for each maps.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Example :

```
# small heap for the example
CG> heapsize 3

CG> dsload Data/bc1.cg
{1 f2 backcross 17 208 /homes/thomas/CartaGene/dev/test/Data/bc1.cg}
CG> sem

Map -1 : log10-likelihood =  -485.24
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS13 MS6 MS11 MS17 MS16 MS8 MS7 MS2 MS3 MS9 MS15 MS12 MS20 M...

CG> flips 3 3.0 1
...
CG> heaprint

Map  0 : log10-likelihood =  -452.11
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS20 M...

Map  1 : log10-likelihood =  -451.97
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS2 MS3 MS9 MS12 MS15 MS19 M...

Map  2 : log10-likelihood =  -451.32
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS19 M...

CG> dsload Data/rh1.cg
{2 haploid RH 13 118 /homes/thomas/CartaGene/dev/test/Data/rh1.cg}
CG> dsmergor 1 2
{3 merged by order 21 326}
CG> sem

Map -1 : log10-likelihood =  -801.56
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS13 MS6 MS11 MS17 MS16 MS8 MS7 MS2 MS3 MS9 MS15 MS12 MS20 M...
   2 : MS4 MS5      MS6                 MS8 MS7     MS3 MS9 MS15             ...

CG> flips 3 3.0 1
...
CG> heaprint

Map  0 : log10-likelihood =  -728.39
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS7 MS8 MS9 MS3 MS2 MS12 MS15 MS19 M...
   2 : MS4 MS5 MS6                     MS7 MS8 MS9 MS3          MS15       ...

Map  2 : log10-likelihood =  -728.19
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS9 MS8 MS7 MS3 MS2 MS12 MS15 MS19 M...
   2 : MS4 MS5 MS6                     MS9 MS8 MS7 MS3          MS15       ...

Map  1 : log10-likelihood =  -727.95
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS7 MS9 MS8 MS3 MS2 MS12 MS15 MS19 M...
   2 : MS4 MS5 MS6                     MS7 MS9 MS8 MS3          MS15       ...
```

See also :

- `heapsize` (2.4.2)
- `heapprintd` (2.4.5)
- `heapprinto` (2.4.16)
- `maprint` (2.4.8)

2.4.5 `heapprintd`

Display the Heap of maps in details.

Synopsis : The CarthaGene `heapprintd` command is invoked as either one of :

- `heapprintd` *Options*
- `heapprintd`

Description : `heapprintd` provide a output of the heap of maps. The maps are ordered by loglikelihood. The best map is always the last printed. The informations available are the id of a map, the score, and the ordered list of the markers printed by names, ids and positions on the map. This last information is usefull for ”merged by order” data sets. Distances and two points informations for consecutives markers are also available. At the end, some statistics are delivered to give the computational cost of the result.

If the data set the maps are computed on, is merged by order, relatives orders for each merged data sets are printed for each maps.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Example :

```
# a very small heap for the example
CG> heapsize 1

CG> dsload Data/bc1.cg
{1 f2 backcross 17 208 /homes/thomas/CartaGene/dev/test/Data/bc1.cg}
CG> sem

Map -1 : log10-likelihood =  -485.24
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS13 MS6 MS11 MS17 MS16 MS8 MS7 MS2 MS3 MS9 MS15 MS12 MS20 M...

# we fill the heap with the best map
CG> flips 3 0 0
...
# and look at it
CG> heaprintd

Map  0 : log10-likelihood =  -460.57, log-e-likelihood = -1060.49
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

  1   1  MS4           3.1 cM      3.1 cM      3.0 cM     3.0 %%    21.1
  2   2  MS5           4.0 cM      7.1 cM      3.9 cM     3.9 %%    15.4
  3   4  MS6          28.2 cM     35.3 cM     23.0 cM    21.5 %%     5.1
  4   5 MS11           8.0 cM     43.3 cM      7.4 cM     7.4 %%    11.9
  5   3 MS13          39.8 cM     83.1 cM     30.9 cM    27.5 %%     0.9
  6   6 MS17          22.1 cM    105.2 cM     18.7 cM    17.9 %%     6.3
  7   7 MS16          33.5 cM    138.7 cM     26.7 cM    24.4 %%    10.3
  8   8  MS8           6.3 cM    145.0 cM      5.9 cM     5.9 %%    35.1
  9   9  MS7          17.2 cM    162.2 cM     15.0 cM    14.5 %%    21.1
 10  10  MS2           0.6 cM    162.8 cM      0.6 cM     0.6 %%    47.6
 11  11  MS3          28.9 cM    191.7 cM     23.6 cM    22.0 %%    12.8
 12  12  MS9          20.8 cM    212.5 cM     17.7 cM    17.0 %%    13.6
 13  13 MS15           3.6 cM    216.1 cM      3.5 cM     3.5 %%    26.5
 14  14 MS12          51.8 cM    267.9 cM     38.3 cM    32.2 %%     4.3
 15  15 MS20           2.9 cM    270.7 cM      2.8 cM     2.8 %%    34.8
 16  16 MS19         345.4 cM    616.1 cM    190.0 cM    50.0 %%    -0.0
 17  17  MS1        ----------              ----------
                     616.1 cM                411.0 cM


       17 markers, log10-likelihood =  -460.57
                   log-e-likelihood = -1060.49

        EM calls:
           Set  1 : 67 (60,0)
        CPU Time (secs): 0.04
        Maps within -3.0: 1

# same thing with maps merged by order
CG> dsload Data/rh1.cg
{2 haploid RH 13 118 /homes/thomas/CartaGene/dev/test/Data/rh1.cg}
```

```
CG> dsmergor 1 2
{3 merged by order 21 326}
CG> sem

Map -1 : log10-likelihood =  -801.56
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS13 MS6 MS11 MS17 MS16 MS8 MS7 MS2 MS3 MS9 MS15 MS12 MS20 M...
   2 : MS4 MS5      MS6                MS8 MS7     MS3 MS9 MS15            ...

CG> flips 3 3.0 1
...
# each map is a pair of map
CG> heaprintd

Map  0 : log10-likelihood =  -727.95, log-e-likelihood = -1676.16
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

  1   1  MS4           3.1 cM      3.1 cM      3.0 cM     3.0 %%    21.1
  2   2  MS5           3.8 cM      6.9 cM      3.7 cM     3.7 %%    15.4
  3   4  MS6          26.1 cM     33.0 cM     21.6 cM    20.3 %%     5.1
  4   5 MS11           8.6 cM     41.6 cM      8.0 cM     7.9 %%    11.9
  5   3 MS13          14.3 cM     56.0 cM     12.7 cM    12.5 %%     9.7
  6   7 MS16          19.6 cM     75.6 cM     16.8 cM    16.2 %%     6.3
  7   6 MS17          60.6 cM    136.2 cM     43.6 cM    35.1 %%     0.0
  8   9  MS7           5.7 cM    141.9 cM      5.4 cM     5.4 %%    29.4
  9  12  MS9           1.9 cM    143.8 cM      1.8 cM     1.8 %%    40.8
 10   8  MS8          25.8 cM    169.6 cM     21.4 cM    20.1 %%    15.1
 11  11  MS3           0.6 cM    170.2 cM      0.6 cM     0.6 %%    47.6
 12  10  MS2          62.0 cM    232.2 cM     44.4 cM    35.5 %%     4.3
 13  14 MS12           4.3 cM    236.5 cM      4.1 cM     4.1 %%    26.5
 14  13 MS15          41.3 cM    277.8 cM     31.8 cM    28.1 %%     6.9
 15  16 MS19           2.9 cM    280.7 cM      2.9 cM     2.9 %%    34.8
 16  15 MS20         345.4 cM    626.1 cM    190.0 cM    50.0 %%    -0.0
 17  17  MS1        ----------              ----------
                     626.1 cM                411.8 cM


       17 markers, log10-likelihood =  -447.28
                   log-e-likelihood = -1029.91

Data Set Number  2 :

       Markers         Distance      Cumulative   Theta        2pt
 Pos  Id name                                   (%%age)       LOD

  1   1  MS4            85.0 cR       85.0 cR     57.3 %%       3.6
  2   2  MS5            57.9 cR      143.0 cR     44.0 %%       7.4
  3   4  MS6            33.2 cR      176.2 cR     28.2 %%      12.8
  8   9  MS7            77.0 cR      253.2 cR     53.7 %%       5.4
  9  12  MS9            17.0 cR      270.1 cR     15.6 %%      18.5
 10   8  MS8           233.1 cR      503.3 cR     90.3 %%       0.2
 11  11  MS3           760.1 cR     1263.4 cR    100.0 %%      -0.0
 14  13 MS15           760.1 cR     2023.4 cR    100.0 %%      -0.0
 17  17  MS1            95.5 cR     2118.9 cR     61.5 %%       2.7
 18  20  G37            25.3 cR     2144.2 cR     22.4 %%      15.0
 19  21  G40            21.5 cR     2165.7 cR     19.3 %%      17.2
 20  18  G36            59.8 cR     2225.5 cR     45.0 %%       7.7
 21  19  G39          ---------
                      2225.5 cR


       13 markers, log10-likelihood =  -280.67
                   log-e-likelihood =  -646.26
                   retention proba. =     0.24

        EM calls:
           Set  3 : 876 (0,0)
           Set  1 : 877 (855,0)
           Set  2 : 877 (858,0)
        CPU Time (secs): 1.03
        Maps within -3.0: 1
```

See also :

- `heapsize` (2.4.2)
- `heaprint` (2.4.4)
- `heaprinto` (2.4.16)
- `maprintd` (2.4.9)

2.4.6 heapget

Return a list of maps of the heap.

Synopsis : The CarthaGene `heapget` command is invoked as either one of :

- `heapget` *Options*
- `heapget` *UnitFlag NbMap*

Description : `heapget` to dump maps from the heap to a list. This command is available for programming...

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *UnitFlag* : 'k' for Kosambi, 'h' for Haldane. This field is mandatory, but not used for RH data sets. *NbMap* : The number of best map to get. If set to 0, all the map of the heap are returned. A positive integer is expected.

Returns : A list of lists is returned :

```
{
   {MapId_0 MapScore_0 {DataSetId_0 OrderScore_0_0 MrkName_0_0_0 CumulDist_0_0_0 MrkName_0_0_1 CumulDist_0_0_1 ... MrkName_0_0_n CumulDist_0_0_n}
                       {DataSetId_1 OrderScore_0_1 MrkName_0_1_0 CumulDist_0_1_0 MrkName_0_1_1 CumulDist_0_1_1 ... MrkName_0_1_n CumulDist_0_1_n’}
                       ...
                       {DataSetId_m OrderScore_0_m MrkName_0_m_0 CumulDist_0_m_0 MrkName_0_m_1 CumulDist_0_m_1 ... MrkName_0_m_n CumulDist_0_m_n’’}
   }
   {MapId_1 MapScore_1 {DataSetId_0 OrderScore_1_0 MrkName_1_0_0 CumulDist_1_0_0 MrkName_1_0_1 CumulDist_1_0_1 ... MrkName_1_0_n CumulDist_1_0_n}
                       {DataSetId_1 OrderScore_1_1 MrkName_1_1_0 CumulDist_1_1_0 MrkName_1_1_1 CumulDist_1_1_1 ... MrkName_1_1_n CumulDist_1_1_n’}
                       ...
                       {DataSetId_m OrderScore_1_m MrkName_1_m_0 CumulDist_1_m_0 MrkName_1_m_1 CumulDist_1_m_1 ... MrkName_1_m_n CumulDist_1_m_n’’}
   }
   ...
   {MapId_l MapScore_l {DataSetId_0 OrderScore_l_0 MrkName_l_0_0 CumulDist_l_0_0 MrkName_l_0_1 CumulDist_l_0_1 ... MrkName_l_0_n CumulDist_l_0_n}
                       {DataSetId_1 OrderScore_l_1 MrkName_l_1_0 CumulDist_l_1_0 MrkName_l_1_1 CumulDist_l_1_1 ... MrkName_l_1_n CumulDist_l_1_n’}
                       ...
                       {DataSetId_m OrderScore_l_m MrkName_l_m_0 CumulDist_l_m_0 MrkName_l_m_1 CumulDist_l_m_1 ... MrkName_l_m_n CumulDist_l_m_n’’}
   }
}
```

Where l is NbMap, the number of map you get. Where m is the number of ”merged by order” data sets. Where n is the number or markers of the map known by the data set.

Example :

```
# starting with a heap of size 7
CG> heapsize 7

# filling it
CG> dsload Data/bc1.cg
{1 f2 backcross 17 208 /homes/thomas/CartaGene/dev/test/Data/bc1.cg}
CG> sem

Map -1 : log10-likelihood =  -485.24
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS13 MS6 MS11 MS17 MS16 MS8 MS7 MS2 MS3 MS9 MS15 MS12 MS20 M...

CG> flips 3 3.0 1
...
CG> heaprint

Map  0 : log10-likelihood =  -454.74
-------:
 Set : Marker List ...
   1 : MS6 MS5 MS4 MS11 MS13 MS16 MS17 MS8 MS7 MS2 MS3 MS9 MS12 MS15 MS19 M...

Map  2 : log10-likelihood =  -454.58
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS17 MS16 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS19 M...

Map  6 : log10-likelihood =  -454.09
-------:
 Set : Marker List ...
   1 : MS6 MS5 MS4 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS19 M...

Map  1 : log10-likelihood =  -452.77
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS2 MS3 MS9 MS12 MS15 MS20 M...

Map  4 : log10-likelihood =  -452.11
-------:
 Set : Marker List ...
```

```
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS20 M...

Map  5 : log10-likelihood =  -451.97
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS2 MS3 MS9 MS12 MS15 MS19 M...

Map  3 : log10-likelihood =  -451.32
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS19 M...

# getting the 3 best one.
CG> heapget k 3
{3 -451.32 {1 -451.32 MS4 0.0 MS5 3.0 MS6 6.7 MS11 28.2 MS13 36.2 MS16 48.9 M\
S17 64.7 MS8 105.1 MS7 111.0 MS3 125.2 MS2 125.8 MS9 149.4 MS12 164.7 MS15 16\
8.9 MS19 200.8 MS20 203.6 MS1 393.6}} {5 -451.97 {1 -451.97 MS4 0.0 MS5 3.0 M\
S6 6.7 MS11 28.2 MS13 36.2 MS16 48.9 MS17 64.7 MS8 105.1 MS7 111.0 MS2 125.7 \
MS3 126.3 MS9 149.8 MS12 165.1 MS15 169.2 MS19 200.9 MS20 203.7 MS1 393.8}} {\
4 -452.11 {1 -452.11 MS4 0.0 MS5 3.0 MS6 6.7 MS11 28.2 MS13 36.2 MS16 48.9 MS\
17 64.7 MS8 105.1 MS7 111.0 MS3 125.2 MS2 125.8 MS9 149.4 MS12 164.7 MS15 168\
.9 MS20 202.8 MS19 205.6 MS1 395.6}}
CG>
```

See also :

- `heaprint` (2.4.4)
- `heaprintd` (2.4.5)
- `mapget` (2.4.13)

2.4.7 `heapordget`

Return a list of all the marker orders appearing in the heap.

Synopsis: The CarthaGene `heapordget` command is invoked as:

- `heapordget`

Description: `heapordget` returns the list of all the markers orders used in all the maps appearing in the heap. Markers are denoted by their numerical Id and the set of orders returned as a list of lists of numerical Ids.

Arguments: None.

Returns: A list of lists of numerical Ids of markers is returned. Each elementary list in the global list corresponds to the order of the markers used in a map in the heap.

Example :

```
# starting with a heap of size 3
CG> heapsize 3

# filling it
CG> dsload Data/bc1.cg
{1 f2 backcross 17 208 /homes/thomas/CartaGene/dev/test/Data/bc1.cg}
CG> sem

Map -1 : log10-likelihood =  -485.24
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS13 MS6 MS11 MS17 MS16 MS8 MS7 MS2 MS3 MS9 MS15 MS12 MS20 M...

CG> flips 3 3.0 1
...
CG> heaprint

Map  0 : log10-likelihood =  -452.11
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS20 M...

Map  1 : log10-likelihood =  -451.97
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS2 MS3 MS9 MS12 MS15 MS19 M...

Map  2 : log10-likelihood =  -451.32
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS6 MS11 MS13 MS16 MS17 MS8 MS7 MS3 MS2 MS9 MS12 MS15 MS19 M...

# getting the corresponding orders
CG> heapordget
{1 2 4 5 3 7 6 8 9 11 10 12 14 13 16 15 17} {1 2 4 5 3 7 6 8 9 10 11 12 14 13\
 16 15 17} {1 2 4 5 3 7 6 8 9 11 10 12 14 13 15 16 17}
CG>
```

See also :

- `heaprint` (2.4.4)
- `heaprintd` (2.4.5)
- `mapget` (2.4.6)

2.4.8 `maprint`

Concisely describes the map whose numerical Id in the heap is given.

Synopsis: The `maprint` command is invoked either:

- `maprint` *Options*
- `maprint` *MapId*

Description: The `maprint` command prints a concise description of the map whose numerical id has been given as argument. The $\log_{10}$-likelihood of the map is printed followed by the order of all the markers (by name), and for each data-set.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *MapId*: the numerical id of the map in the heap.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> buildfw  3 3 {} 0
...
CG> maprint 0

Map  0 : log10-likelihood =  -248.48
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS4 MS5 MS7 MS8 MS9 MS11 MS12 MS15 MS16 MS18
```

See also:

- `maprintd` (2.4.9)
- `heaprint` (2.4.4)

2.4.9 `maprintd`

Precisely describes the map whose numerical Id in the heap is given.

Synopsis: The `maprintd` command is invoked either:

- `maprintd` *Options*
- `maprintd` *MapId*

Description: The `maprintd` command prints an extensive description of the map whose numerical id has been given as argument. For each data-set for which parameters have been estimated (merged using `mergor`), the map is described by successively giving, line per line:

1. the position of the marker in the overall map for this data set
2. the numerical Id of the marker
3. the name of the marker (truncated to a mxaimum number of characters)

4. the Haldane distance to the next marker (if any). Rays are used for RH data.
5. The cumulative distance of the marker from the beginning of the map (Haldane/Ray).
6. For genetic data, the Kosambi distance from the next marker (if any)
7. the estimated recombination ratio/breakage with the next marker
8. the two-points LOD with the next marker

Then the length of the map, the number of markers and the $\log_{10}$ and natural logarithm of the likelihood of the map are displayed.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *MapId*: the numerical id of the map in the heap.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> buildfw  3 3 {} 0
...
CG> maprintd 0

Map  0 : log10-likelihood =  -248.48, log-e-likelihood =  -572.16
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance    Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi     (%%age)      LOD

  1   1  MS1          21.0 cM     21.0 cM     17.9 cM     17.1 %%     15.7
  2   2  MS2           6.2 cM     27.2 cM      5.8 cM      5.8 %%     33.8
  3   4  MS4           3.4 cM     30.5 cM      3.3 cM      3.3 %%     21.1
  4   5  MS5           5.5 cM     36.0 cM      5.2 cM      5.2 %%     16.3
  5   7  MS7           6.6 cM     42.6 cM      6.2 cM      6.2 %%     35.1
  6   8  MS8           2.2 cM     44.9 cM      2.2 cM      2.2 %%     40.8
  7   9  MS9           8.9 cM     53.7 cM      8.2 cM      8.1 %%     11.2
  8  11 MS11           8.9 cM     62.6 cM      8.2 cM      8.1 %%     10.9
  9  12 MS12           4.1 cM     66.7 cM      4.0 cM      4.0 %%     26.5
 10  15 MS15           4.9 cM     71.6 cM      4.7 cM      4.6 %%     26.6
 11  16 MS16          22.7 cM     94.4 cM     19.2 cM     18.3 %%     10.3
 12  18 MS18        ----------              ----------
                      94.4 cM                 84.8 cM


       12 markers, log10-likelihood =  -248.48
                   log-e-likelihood =  -572.16

CG>
```

See also:

- `maprint` (2.4.8)
- `heaprintd` (2.4.5)

2.4.10 `maprintdr`

Precisely describes the map whose numerical Id in the heap is given using a reverse order.

Synopsis: The `maprintdr` command is invoked either:

- `maprintdr` *Options*
- `maprintdr` *MapId*

Description: The `maprintdr` command prints an extensive description of the map whose numerical id has been given as argument using a reverse marker ordering. The purpose of using a reverse marker ordering is to perform an easier map comparison when two maps in the heap have order which are similar up to a map reversal. Otherwise, the commands performs as `maprintd`.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *MapId*: the numerical id of the map in the heap.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> buildfw  3 3 {} 0
...
CG> maprintd 0

Map  0 : log10-likelihood =  -248.48, log-e-likelihood =  -572.16
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

  1   1  MS1          21.0 cM     21.0 cM     17.9 cM    17.1 %%    15.7
  2   2  MS2           6.2 cM     27.2 cM      5.8 cM     5.8 %%    33.8
  3   4  MS4           3.4 cM     30.5 cM      3.3 cM     3.3 %%    21.1
  4   5  MS5           5.5 cM     36.0 cM      5.2 cM     5.2 %%    16.3
  5   7  MS7           6.6 cM     42.6 cM      6.2 cM     6.2 %%    35.1
  6   8  MS8           2.2 cM     44.9 cM      2.2 cM     2.2 %%    40.8
  7   9  MS9           8.9 cM     53.7 cM      8.2 cM     8.1 %%    11.2
  8  11 MS11           8.9 cM     62.6 cM      8.2 cM     8.1 %%    10.9
  9  12 MS12           4.1 cM     66.7 cM      4.0 cM     4.0 %%    26.5
 10  15 MS15           4.9 cM     71.6 cM      4.7 cM     4.6 %%    26.6
 11  16 MS16          22.7 cM     94.4 cM     19.2 cM    18.3 %%    10.3
 12  18 MS18        ----------              ----------
                      94.4 cM                 84.8 cM


       12 markers, log10-likelihood =  -248.48
                   log-e-likelihood =  -572.16

CG> maprintdr 0

Map  0 : log10-likelihood =  -248.48, log-e-likelihood =  -572.16
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

  1  18 MS18          22.7 cM     22.7 cM     19.2 cM    18.3 %%    10.3
  2  16 MS16           4.9 cM     27.6 cM      4.7 cM     4.6 %%    26.6
  3  15 MS15           4.1 cM     31.7 cM      4.0 cM     4.0 %%    26.5
  4  12 MS12           8.9 cM     40.6 cM      8.2 cM     8.1 %%    10.9
  5  11 MS11           8.9 cM     49.5 cM      8.2 cM     8.1 %%    11.2
  6   9  MS9           2.2 cM     51.7 cM      2.2 cM     2.2 %%    40.8
  7   8  MS8           6.6 cM     58.3 cM      6.2 cM     6.2 %%    35.1
  8   7  MS7           5.5 cM     63.8 cM      5.2 cM     5.2 %%    16.3
  9   5  MS5           3.4 cM     67.2 cM      3.3 cM     3.3 %%    21.1
 10   4  MS4           6.2 cM     73.4 cM      5.8 cM     5.8 %%    33.8
 11   2  MS2          21.0 cM     94.4 cM     17.9 cM    17.1 %%    15.7
 12   1  MS1        ----------              ----------
                      94.4 cM                 84.8 cM


       12 markers, log10-likelihood =  -248.48
                   log-e-likelihood =  -572.16

CG>
```

See also:

- `maprint` (2.4.8)
- `heaprintd` (2.4.5)

2.4.11 `bestprint`

Concisely describes the best map in the heap.

Synopsis: The `bestprint` command is invoked as:

- `bestprint`

Description: The `bestprint` command prints a concise description of the best map in the heap. The $\log_{10}$-likelihood of the map is printed followed by the order of all the markers (by name), and for each data-set.

Arguments: None.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> sem

Map -1 : log10-likelihood =  -297.78
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS3 MS4 MS5 MS6 MS7 MS8 MS9 MS10 MS11 MS12 MS13 MS14 MS15 MS...

CG> flips 6 0 1
...
CG> bestprint

Map  9 : log10-likelihood =  -297.78
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS3 MS4 MS5 MS6 MS7 MS8 MS9 MS10 MS11 MS13 MS12 MS14 MS15 MS...
```

See also:

- `bestprintd` (2.4.12)
- `heaprint` (2.4.4)

2.4.12 `bestprintd`

Precisely describes the best map in the heap.

Synopsis: The `bestprintd` command is invoked as:

- `bestprintd`

Description: The `bestprintd` command prints an extensive description of the best map in the heap. For each data-set for which parameters have been estimated (merged using `mergor`), the map is described by successively giving, line per line:

1. the position of the marker in the overall map for this data set
2. the numerical Id of the marker
3. the name of the marker (truncated to a mxaimum number of characters)

4. the Haldane distance to the next marker (if any). Rays are used for RH data.
5. The cumulative distance of the marker from the beginning of the map (Haldane/Ray).
6. For genetic data, the Kosambi distance from the next marker (if any)
7. the estimated recombination ratio/breakage with the next marker
8. the two-points LOD with the next marker

Then the length of the map, the number of markers and the $\log_{10}$ and natural logarithm of the likelihood of the map are displayed.

Arguments: none.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> sem

Map -1 : log10-likelihood =  -297.78
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS3 MS4 MS5 MS6 MS7 MS8 MS9 MS10 MS11 MS12 MS13 MS14 MS15 MS...

CG> flips 6 0 1
...
CG> bestprintd

Map  9 : log10-likelihood =  -297.78, log-e-likelihood =  -685.66
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

  1   1  MS1          19.5 cM     19.5 cM     16.8 cM    16.2 %%    15.7
  2   2  MS2           0.6 cM     20.1 cM      0.6 cM     0.6 %%    47.6
  3   3  MS3           5.8 cM     25.9 cM      5.5 cM     5.5 %%    35.1
  4   4  MS4           3.7 cM     29.6 cM      3.6 cM     3.5 %%    21.1
  5   5  MS5           3.7 cM     33.3 cM      3.6 cM     3.6 %%    15.4
  6   6  MS6           1.6 cM     34.9 cM      1.6 cM     1.6 %%    17.3
  7   7  MS7           6.7 cM     41.6 cM      6.3 cM     6.3 %%    35.1
  8   8  MS8           2.2 cM     43.8 cM      2.2 cM     2.2 %%    40.8
  9   9  MS9           3.0 cM     46.8 cM      2.9 cM     2.9 %%     7.2
 10  10 MS10           5.4 cM     52.2 cM      5.1 cM     5.1 %%     3.6
 11  11 MS11           8.6 cM     60.7 cM      7.9 cM     7.9 %%    11.9
 12  13 MS13           0.0 cM     60.7 cM      0.0 cM     0.0 %%    18.7
 13  12 MS12           0.0 cM     60.7 cM      0.0 cM     0.0 %%     3.6
 14  14 MS14           5.8 cM     66.6 cM      5.5 cM     5.5 %%     0.0
 15  15 MS15           4.6 cM     71.1 cM      4.4 cM     4.4 %%    26.6
 16  16 MS16          26.1 cM     97.2 cM     21.6 cM    20.3 %%     6.3
 17  17 MS17           0.0 cM     97.2 cM      0.0 cM     0.0 %%    16.0
 18  18 MS18           4.9 cM    102.1 cM      4.7 cM     4.6 %%    27.1
 19  19 MS19           3.0 cM    105.1 cM      2.9 cM     2.9 %%    34.8
 20  20 MS20        ----------              ----------
                     105.1 cM                95.0 cM


       20 markers, log10-likelihood =  -297.78
                   log-e-likelihood =  -685.66
```

See also:

- `bestprint` (2.4.11)
- `maprint` (2.4.8)
- `maprint` (2.4.9)
- `maprint` (2.4.10)
- `heaprint` (2.4.4)

2.4.13 `mapget`

Returns a list describing the map with a given numerical Id in the heap.

Synopsis: The `mapget` command is invoked either as:

- `mapget` *Options*
- `mapget` *Unit MapId*

Description: The `mapget` command returns a list that describes the map whose numerical id in the heap is *MapId*. The list contains successively:

- the numerical Id of the map in the heap (*MapId*)
- the map's overall maximum loglikelihood (in $\log_{10}$ units)
- a sequence of sublist, one for each current data-set for which parameters are estimated (merged by `mergor`). These sublists contain successively:
  - the data set numerical Id,
  - the map loglikelihood in this data-set (in $\log_{10}$ units)
  - the name of each loci separated by their distances in the requested *Unit* (either Haldane (h) or Kosambi (k) for genetic data, Rays are always used for radiated hybrid data)

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *Unit*: the distance function used to compute distances from recombination/breakage ratios. It is either Haldane (h) or Kosambi (k) for genetic data. Rays are always used for radiated hybrid data.
- *MapId*: the numerical Id of the map in the heap.

Returns: a list describing the map. See above.

Example:

```
CG> dsload Data/bc1.cg
{1 f2 backcross 17 208 /homes/thomas/CartaGene/dev/test/Data/bc1.cg}
CG> dsload Data/rh1.cg
{2 haploid RH 13 118 /homes/thomas/CartaGene/dev/test/Data/rh1.cg}
CG> dsmergor 1 2
{3 merged by order 21 326}
CG> sem

Map -1 : log10-likelihood =  -801.56
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS13 MS6 MS11 MS17 MS16 MS8 MS7 MS2 MS3 MS9 MS15 MS12 MS20 M...
   2 : MS4 MS5      MS6                MS8 MS7     MS3 MS9 MS15            ...

CG> mapget k 0
0 -801.56 {1 -485.24 MS4 0.0 MS5 3.3 MS13 38.8 MS6 64.2 MS11 86.4 MS17 137.9 \
MS16 159.5 MS8 186.5 MS7 192.4 MS2 207.4 MS3 208.0 MS9 231.5 MS15 249.3 MS12 \
252.8 MS20 291.1 MS19 293.8 MS1 483.9} {2 -316.32 MS4 0.0 MS5 84.4 MS6 138.5 \
MS8 229.0 MS7 307.2 MS3 493.5 MS9 706.1 MS15 862.8 MS1 1622.9 G36 1726.5 G39 \
1786.2 G37 1845.9 G40 1871.3}
```

See also:

- `mapordget` (2.4.14)
- `maprint` (2.4.8)

2.4.14 `mapordget`

Returns a list describing the order of the markers in the map whose numerical Id in the heap is given.

Synopsis: The `mapordget` command is invoked either as:

- `mapordget` *Options*
- `mapordget` *MapId*

Description: The `mapordget` command returns a list that gives the order of the numerical Ids of the markers in the map whose numerical Id in the heap in given as argument.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *MapId*: the numerical Id of the map in the heap.

Returns: a list giving the order of the markers numerical Ids.

Example:

```
CG> dsload Data/bc1.cg
{1 f2 backcross 17 208 /homes/thomas/CartaGene/dev/test/Data/bc1.cg}
CG> dsload Data/rh1.cg
{2 haploid RH 13 118 /homes/thomas/CartaGene/dev/test/Data/rh1.cg}
CG> dsmergor 1 2
{3 merged by order 21 326}
CG> sem

Map -1 : log10-likelihood =  -801.56
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS13 MS6 MS11 MS17 MS16 MS8 MS7 MS2 MS3 MS9 MS15 MS12 MS20 M...
   2 : MS4 MS5      MS6                MS8 MS7     MS3 MS9 MS15            ...

CG> mapordget 0
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
```

See also:

- `mapget` (2.4.13)
- `maprint` (2.4.8)

2.4.15 `mapocb`

Returns the number of Obligate Chromosome Breaks for the map whose numerical Id in the heap is given.

Synopsis: The `mapocb` command is invoked either:

- `mapocb` *Options*
- `mapocb` *MapId*

Description: The `mapocb` command computes the number of Obligate Chromosome Breaks for the map whose numerical Id in the heap is given as argument. For each data-set, for each individual/hybrid, for each pair of consecutive markers in the map, there is an obligate chromosome break if we have the corresponding typing data “0 1” or “1 0” (e.g. 0=Absent, 1=Here). `mapocb` returns the sum of all obligate chromosome breaks.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *MapId*: the numerical id of the map in the heap.

Returns: number of Obligate Chromosome Breaks.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> sem

Map -1 : log10-likelihood = -1090.04
-------:
 Set : Marker List ...
   1 : MS1 MS13 MS15 MS17 MS19 MS2 MS20 MS3 MS4 MS5 MS6 MS7 MS8 MS9 G1 G2 G...

CG> mapocb 0
1025
CG>
```

See also:

- `maprint` (2.4.8)

2.4.16 `heaprinto`

Displays the contents of the heap.

Synopsis : The CarthaGene `heaprinto` command is invoked as either one of:

- `heaprinto` *Options*
- `heaprinto` *NbMrk Blank Comp*

Description : `heaprinto` compares each map contained in the heap to the best current map. The position of each locus on the best map is printed at its new position on the current map. If the *NbMrk* argument is equal to 0, all the positions are printed. Otherwise, only the differences are printed. The *NbMrk* argument allows to visualize *NbMrk* consecutive differences or more with square brackets. If *Blank* argument is set to 1, the positions between the extremities of an interval are replaced by ’-’. If *Comp* is set to 1, only the extremities of intervals are printed consecutively. At the end of each line, the contribution to the delta lod is printed for each data set ”merged by order”.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *NbMrk* : The minimum number of markers needed to mention an area of change. The *NbMrk* argument must be a positive integer.
- *Blank* : The flag to choose either to print blanks (1) or to print all the positions inside a interval (0). The *Blank* argument must be set to 0 or 1.
- *Comp* : The flag to compress intervals (1) or not (0). The *Blank* argument must be set to 0 or 1.

Example :

```
CG> dsload Data/bc1.cg
{1 f2 backcross 17 208 /homes/thomas/CartaGene/dev/test/Data/bc1.cg}
CG> dsload Data/rh1.cg
{2 haploid RH 13 118 /homes/thomas/CartaGene/dev/test/Data/rh1.cg}
CG> dsmergor 1 2
{3 merged by order 21 326}
CG> sem

Map -1 : log10-likelihood =  -801.56
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS13 MS6 MS11 MS17 MS16 MS8 MS7 MS2 MS3 MS9 MS15 MS12 MS20 M...
   2 : MS4 MS5      MS6                 MS8 MS7     MS3 MS9 MS15            ...

CG> flips 5 3.0 1
...
# To print all the positions.
CG> heaprinto 0 0 0
Loci Id  ..........
                  :                    1  1           1  1  1  1  2  2  1  ...
                  :  2  4  5  3  7  6  0  1  1  9  8  2  4  3  6  0  1  8  ...
Loci Pos ..........  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  ...
Map Id : log10    :  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  ...
     1 :  -693.62 :  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 1...
    12 :     0.00 =  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 18 15 16 1...
     3 :     0.00 =  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 18 1...
     4 :     0.00 =  0  1  2  3  4  5  6  7  8  9 10 11 12 13 15 14 16 17 1...
     5 :     0.00 =  0  1  2  3  4  5  6  7  8  9 10 11 12 13 15 16 17 14 1...
    13 :     0.00 =  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 1...
     7 :     0.00 =  0  1  2  3  4  5  6  7  8  9 10 11 12 13 15 14 18 16 1...
     8 :     0.00 =  0  1  2  3  4  5  6  7  8  9 10 11 12 13 15 14 16 18 1...
     9 :     0.00 =  0  1  2  3  4  5  6  7  8  9 10 11 12 13 15 16 14 18 1...
    10 :     0.00 =  0  1  2  3  4  5  6  7  8  9 10 11 12 13 15 16 14 17 1...
    11 :     0.00 =  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 18 16 1...
    14 :     0.19 =  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 18 20 19 1...
     6 :     0.77 =  1  0  2  3  4  5  6  7  8  9 10 11 12 13 14 15 18 16 1...
     2 :     0.77 =  1  0  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 1...
     0 :     0.77 =  1  0  2  3  4  5  6  7  8  9 10 11 12 13 14 18 15 16 1...

# To see consecutive differences of length greater than 3.
CG> heaprinto 3 0 0
Loci Id  ..........
                  :                    1  1           1  1  1  1  2  2  1  ...
                  :  2  4  5  3  7  6  0  1  1  9  8  2  4  3  6  0  1  8  ...
Loci Pos ..........  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  ...
Map Id : log10    :  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  ...
     1 :  -693.62 :  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 1...
    12 :     0.00 =                                              18[15 16 1...
     3 :     0.00 =                                                    18 1...
     4 :     0.00 =                                           15 14        ...
     5 :     0.00 =                                          [15 16 17]14  ...
    13 :     0.00 =                                                       1...
     7 :     0.00 =                                           15 14 18 16 1...
     8 :     0.00 =                                           15 14    18 1...
     9 :     0.00 =                                           15 16 14 18 1...
    10 :     0.00 =                                           15 16 14     ...
    11 :     0.00 =                                                 18 16 1...
    14 :     0.19 =                                              18 20 19[1...
     6 :     0.77 =  1  0                                           18 16 1...
     2 :     0.77 =  1  0                                                  ...
     0 :     0.77 =  1  0                                        18[15 16 1...

# To see the ends of the consecutive differences only.
CG> heaprinto 3 1 0
Loci Id  ..........
                  :                    1  1           1  1  1  1  2  2  1  ...
                  :  2  4  5  3  7  6  0  1  1  9  8  2  4  3  6  0  1  8  ...
Loci Pos ..........  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  ...
Map Id : log10    :  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  ...
     1 :  -693.62 :  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 1...
    12 :     0.00 =                                              18[15 -- 1...
     3 :     0.00 =                                                    18 1...
     4 :     0.00 =                                           15 14        ...
     5 :     0.00 =                                          [15 -- 17]14  ...
    13 :     0.00 =                                                       1...
     7 :     0.00 =                                           15 14 18 16 1...
     8 :     0.00 =                                           15 14    18 1...
     9 :     0.00 =                                           15 16 14 18 1...
    10 :     0.00 =                                           15 16 14     ...
    11 :     0.00 =                                                 18 16 1...
    14 :     0.19 =                                              18 20 19[1...
     6 :     0.77 =  1  0                                           18 16 1...
     2 :     0.77 =  1  0                                                  ...
     0 :     0.77 =  1  0                                        18[15 -- 1...

# To make the line shorter.
CG> heaprinto 3 1 1
Loci Id  ..........
                  :                    1  1           1  1  1  1  2  2  1  ...
                  :  2  4  5  3  7  6  0  1  1  9  8  2  4  3  6  0  1  8  ...
Loci Pos ..........  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  ...
Map Id : log10    :  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  ...
     1 :  -693.62 :  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 1...
    12 :     0.00 = 18[15 17]  (     0.00 +     0.00 )
     3 :     0.00 = 18 17  (     0.00 +     0.00 )
     4 :     0.00 = 15 14  (     0.00 +     0.00 )
     5 :     0.00 =[15 17] 14  (     0.00 +     0.00 )
    13 :     0.00 = 19 18  (     0.00 +     0.00 )
     7 :     0.00 = 15 14 18 16 17  (     0.00 +     0.00 )
     8 :     0.00 = 15 14 18 17  (     0.00 +     0.00 )
     9 :     0.00 = 15 16 14 18 17  (     0.00 +     0.00 )
    10 :     0.00 = 15 16 14  (     0.00 +     0.00 )
    11 :     0.00 = 18 16 17  (     0.00 +     0.00 )
    14 :     0.19 = 18 20 19[17 15] (     0.00 +     0.19 )
     6 :     0.77 =  1  0 18 16 17  (     1.38 +    -0.62 )
     2 :     0.77 =  1  0  (     1.38 +    -0.62 )
     0 :     0.77 =  1  0 18[15 17]  (     1.38 +    -0.62 )
```

See also :

- `heaprint` (2.4.4)
- `heaprintd` (2.4.5)

2.5 Building maps from scratch

In CARTHAGENE everything starts by building a first initial map from the current set of selected markers. This first map will be stored in the heap and will be used ad the indispensable "seed" for further map improving commands. There are several methods to produce sucha first map and they can be simply classified as methods that aim at providing *comprehensive* maps and methods that aim at providing *framework* maps.

- Comprehensive maps are maps that will contain all the markers mentionned in the set of currently active markers. The maps sought are typically maps whose loglikelihood is maximum in this case.
- Framework maps are maps that may contain a subset of the markers mentionned in the set of currently active markers. The aim is to select markers that can be reliably ordered (i.e., which can be ordered in such a way that all alternative orders are less likely by several orders of magnitudes).

All the initial mapping commands of CARTHAGENE are simple heuristics commands that do not spend a large amount of time trying to build an ideal map. The consequence of this, which is true for all existring software using such "quick" mapping techniques, is that the maps produced by CARTHAGENE may be still be improved. For this, use the "improving" commands (section 2.6).

2.5.1 Comprehensive mapping

The first simplest command to produce a map is the `sem` command. SEM is a shortcut for "Single EM": it will produce a map using the current order used to specify the list of active markers (specified using the `mrkselset` command). The map parameters are automatically optimized by the EM algorithm (hence the name of the command). Naturally, unless you have some good idea of the markers ordering, this command has no reason to produce a likely map.

The next level of sophistication is provided by the `nicemapl`, `nicemapd`, `mfmapl` and `mfmapd` commands that exploit 2-points measures such as 2-points LOD and 2-points distances. A strong limitation of such commands is that often, in multiple population mapping, some 2-points measures may be undefined. this is the case eg. when a marker is typed in one population and not in another. In such case, it is likely that such commands will produce poor maps. The heuristics used to build maps is the so-called "nearest neighbor" heuristics. It consists in adding as a new marker the marker which is the closer (resp. more strongly linked) to the previous marker. The heuristics is called repeatedly starting from all possible markers and the best map produced (w.r.t. loglikelihood) is reported. There is another kind of heuristic based on the powerful Keld Helsgaun's LKH software. LKH implements the Lin-Kernighan heuristic for solving Traveling Salesman Problems. See commands `lkh`, `lkhn`, `lkhl`, `lkhd`, `lkhocb` and `lkhocbn`. It is also possible to use another TSP solver by using the `cg2tsp` command.

The next level of sophistication is provided by the `build` function, inspired by similar facilities in CRIMAP [Gre88]. This command starts by examining all pairs of markers in the current list of active markers and keeps the pair that has a maximum likelihood. Then a new marker is selected: it is the markers which is the most strogly linked with markers already in the map. It is tentatively inserted in all possible positions and the best position is kept. Because this process is still far from always yielding good maps, it is typically performed on several distinct maps in parallel (the maps are the k best maps found at each step instead of just the best one). The parameter k is specified as a numerical argument to the build command. This command is largely superseded by the framework mapping command `buildfw` which can also perform comprehensive mapping.

2.5.2 `sem`

Computes the maximum likelihood of the current active markers order.

Synopsis: The `sem` command is invoked either as:

- `sem` *Options*
- `sem`

Description: The `sem` command (for “Single EM”) performs the maximum likelihood estimation of all the needed recombination /breakage/retention ratio parameters for the current data-set using the current marker ordering (available using the `mrkselget` command). The map obtained is submitted for the heap and is therefore memorized if it is close enough to the optimum map.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: nothing. The map is submitted to the heap.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> mrkselget
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
CG> sem

Map -1 : log10-likelihood =  -297.78
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS3 MS4 MS5 MS6 MS7 MS8 MS9 MS10 MS11 MS12 MS13 MS14 MS15 MS16\
 MS17 MS18 MS19 MS20

CG>
```

See also:

- `mrkselset` (2.3.13)

2.5.3 `cgtolerance`

Fixes the convergence threshold used in EM, the optimization algorithm used for multipoint recombination/breakage/retention estimation.

Synopsis: The `tolerance` command is invoked as either:

- `cgtolerance` *Options*
- `cgtolerance` *Fine_Threshold Raw_Threshold*

Description: The `tolerance` command sets the convergence threshold used in EM, the optimization algorithm used for multipoint recombination (genetic data), breakage and retention estimation (RH data). The EM algorithm is an iterative loglikelihood algorithm that stops iterating when the loglikelihood increases become too small, below a given threshold.

The first argument is used for final estimation. All the maps presented to the user are converged up to this threshold, initially set to 0.001. The second argument is used for raw estimation inside ordering commands. The loglikelihood is first maximized using this raw threshold (initially set to 0.1) and if it is still far from the loglikelihood of the best know map, then further iteration are abandoned. Otherwise, the iteration are continued until a final level of convergence is reached.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *Fine Threshold*: the final EM convergence threshold.
- *Raw Threshold*: the raw EM convergence threshold.

Example:

```
CG> dsload Data/rh1.cg
{1 haploid RH 13 118 /homes/thomas/CartaGene/dev/test/Data/rh1.cg}
CG> sem

Map -1 : log10-likelihood =  -286.91
-------:
 Set : Marker List ...
   1 : G36 MS5 MS6 MS7 MS9 MS8 MS1 G39 MS3 G37 MS15 G40 MS4

CG> dsload Data/rh1.cg
{2 haploid RH 13 118 /homes/thomas/CartaGene/dev/test/Data/rh1.cg}
CG> cgtolerance 0.00001 0.01

CG> sem

Map -1 : log10-likelihood =  -286.90
-------:
 Set : Marker List ...
   2 : G36 MS5 MS6 MS7 MS9 MS8 MS1 G39 MS3 G37 MS15 G40 MS4
```

2.5.4 `nicemapl`

Computes an intuitively nice marker ordering using 2-points LOD.

Synopsis: The `nicemapl` command is invoked as either one of :

- `nicemapl` *Options*
- `nicemapl`

Description: The `nicemapl` command orders the current list of active markers using a simple `Nearest Neighbor` heuristics. Starting from each marker successively, it builds a map by always selecting the marker with the strongest LOD as the next marker. The best map according to multipoint maximum likelihood is inserted in the heap.

Warning: since 2-points estimations are typically unavailable for some pairs of markers when data-sets are merged, this command will return very poor maps on such merged data. This command should be used preferably to `nicemapd` when some data-sets have been merged with separate parameter estimation (using `mergor`).

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> nicemapl

Map -1 : log10-likelihood =  -350.28
-------:
 Set : Marker List ...
   1 : MS14 MS13 MS11 MS10 MS5 MS6 MS1 MS2 MS3 MS4 MS7 MS8 MS9 MS12 MS15 MS16\
 MS17 MS18 MS19 MS20
```

See also:

- `sem` (2.5.2)
- `nicemapd` (2.5.5)
- `mfmapl` (2.5.6)
- `mfmapd` (2.5.7)
- `build` (2.5.15)
- `buildfw` (2.5.17)

2.5.5 `nicemapd`

Computes an intuitively nice marker ordering using 2-points distances.

Synopsis: The `nicemapd` command is invoked as either one of :

- `nicemapd` *Options*
- `nicemapd`

Description: The `nicemapd` command orders the current list of active markers using a simple `Nearest Neighbor` heuristics. Starting from each marker successively, it builds a map by always selecting the nearest marker. The best map according to multipoint maximum likelihood is inserted in the heap.

Warning: since 2-points estimations are typically unavailable for some pairs of markers when data-sets are merged, this command will return very poor maps on such merged data. This command should especially not be used when two data-sets have been merged with separate parameters estimation (using `mergor`).

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> nicemapd

Map -1 : log10-likelihood =  -331.49
-------:
 Set : Marker List ...
   1 : MS13 MS14 MS11 MS10 MS9 MS8 MS7 MS6 MS5 MS4 MS3 MS2 MS1 MS12 MS15 MS16\
 MS18 MS17 MS19 MS20
```

See also:

- `nicemapl` (2.5.4)
- `mfmapd` (2.5.7)
- `mfmapl` (2.5.6)
- `sem` (2.5.2)

2.5.6 mfmapl

Computes an intuitively nice marker ordering using 2-points LOD.

Synopsis: The `mfmapl` command is invoked as either one of :

- `mfmapl` *Options*
- `mfmapl`

Description: The `mfmapl` command builds gradually a map by inserting the best (whose LOD criteria are minima) available pairs of markers among the current list of active markers. This “Greedy” heuristic applies a multi-fragment (mf) strategy: several order fragments of markers which are close together are constructed and then linked together.

Warning: since 2-points estimations are typically unavailable for some pairs of markers when data-sets are merged, this command will return very poor maps on such merged data. This command should especially not be used when two data-sets have been merged with separate parameters estimation (using `mergor`).

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> mfmapl

Map -1 : log10-likelihood =  -354.66
-------:
 Set : Marker List ...
   1 : MS10 MS5 MS6 MS1 MS2 MS3 MS4 MS7 MS8 MS9 MS12 MS15 MS16 MS13 MS11 MS14\
 MS17 MS18 MS19 MS20
```

See also:

- `mfmapd` (2.5.7)
- `nicemapl` (2.5.4)
- `nicemapd` (2.5.5)
- `sem` (2.5.2)

2.5.7 `mfmapd`

Computes an intuitively nice marker ordering using 2-points distances.

Synopsis: The `mfmapd` command is invoked as either one of :

- `mfmapd` *Options*
- `mfmapd`

Description: The `mfmapd` command builds gradually a map by inserting the best (whose distances are minima) available pairs of markers among the current list of active markers. This “Greedy” heuristic applies a multi-fragment (mf) strategy: several order fragments of markers which are close together are constructed and then linked together.

Warning: since 2-points estimations are typically unavailable for some pairs of markers when data-sets are merged, this command will return very poor maps on such merged data. This command should especially not be used when two data-sets have been merged with separate parameters estimation (using `mergor`).

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> mfmapd

Map -1 : log10-likelihood =  -327.30
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS3 MS4 MS5 MS7 MS6 MS8 MS11 MS10 MS9 MS14 MS12 MS13 MS16 MS15\
 MS17 MS18 MS19 MS20
```

See also:

- `mfmapl` (2.5.6)
- `nicemapd` (2.5.5)
- `nicemapl` (2.5.4)
- `sem` (2.5.2)

2.5.8 `lkh`

Computes an intuitively nice marker ordering using 2-points loglikelihoods.

Synopsis: The `lkh` command is invoked as either one of :

- `lkh` *Options*
- `lkh` *NbRun CollectMaps*

Description: The `lkh` command converts the current marker selection and associated genetic / Radiated Hybrid data into a Traveling Salesman Problem which is solved using Keld Helsgaun's LKH software. LKH is based on the Lin-Kernighan heuristic. It is possible to collect every tour found by LKH into the CarthaGene heap, in order to get a map with a possible better multipoint loglikelihood.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *NbRun* : Repeats *NbRun* times the LK heuristic.
- *CollectMaps* : Possible values are -1, 0, 2, 3, 4 and 5. If *CollectMaps* ¿= 0 then every tour found by LKH is inserted into the CarthaGene heap. Moreover, a positive value sets the backtrack move type of LKH.

Returns: nothing.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> lkh 1 -1
[-822.89]
Best map with log10-likelihood = -822.89
TSP: optimum= 801.055000 lowerbound= 801.035800 gap= 0.002397% totaltime= 0.08

Map -1 : log10-likelihood =  -822.89
-------:
 Set : Marker List ...
   1 : MS1 G39 MS3 MS2 G37 G40 G36 MS4 G3 G4 MS5 G1 G2 MS6 MS7 MS9 MS8 G21 G2\
6 G22 G25 G23 G27 G24 G28 G34 G33 G30 G31 G29 G32 G35 MS13 MS20 G10 G9 MS19 G\
19 MS17 G20 G11 G8 G7 G12 G13 G16 G6 G5 G17 G18 G14 G15 MS15

CG>
```

See also:

- `lkhn` (2.5.9)
- `lkhl` (2.5.10)
- `lkhd` (2.5.11)
- `cg2tsp` (2.5.14)

2.5.9 `lkhn`

Computes an intuitively nice marker ordering using normalized 2-points loglikelihoods.

Synopsis: The `lkhn` command is invoked as either one of :

- `lkhn` *Options*
- `lkhn` *NbRun CollectMaps*

Description: The `lkhn` command converts the current marker selection and associated genetic / Radiated Hybrid data into a Traveling Salesman Problem which is solved using Keld Helsgaun's LKH software. LKH is based on the Lin-Kernighan heuristic. It is possible to collect every tour found by LKH into the CarthaGene heap, in order to get a map with a possible better multipoint loglikelihood.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *NbRun* : Repeats *NbRun* times the LK heuristic.
- *CollectMaps* : Possible values are -1, 0, 2, 3, 4 and 5. If *CollectMaps* ¿= 0 then every tour found by LKH is inserted into the CarthaGene heap. Moreover, a positive value sets the backtrack move type of LKH.

Returns: nothing.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> lkhn 1 -1
[-823.30]
Best map with log10-likelihood = -823.30
TSP: optimum= 822.235000 lowerbound= 822.222200 gap= 0.001557% totaltime= 0.06

Map -1 : log10-likelihood =  -823.30
-------:
 Set : Marker List ...
   1 : MS1 G39 MS3 MS2 G37 G40 G36 MS4 G3 G4 MS5 G1 G2 MS6 MS7 MS9 MS8 G21 G2\
6 G22 G25 G23 G27 G24 G28 G34 G33 G30 G31 G29 G32 G35 MS13 MS20 MS17 G20 G19 \
MS19 G9 G10 G11 G8 G7 G12 G13 G16 G6 G5 G17 G18 G14 MS15 G15

CG>
```

See also:

- `lkh` (2.5.8)
- `lkhl` (2.5.10)
- `lkhd` (2.5.11)
- `cg2tsp` (2.5.14)

2.5.10 `lkhl`

Computes an intuitively nice marker ordering using 2-points LOD criteria.

Synopsis: The `lkhl` command is invoked as either one of :

- `lkhl` *Options*
- `lkhl` *NbRun CollectMaps*

Description: The `lkhl` command converts the current marker selection and associated genetic / Radiated Hybrid data into a Traveling Salesman Problem which is solved using Keld Helsgaun's LKH software. LKH is based on the Lin-Kernighan heuristic. It is possible to collect every tour found by LKH into the CarthaGene heap, in order to get a map with a possible better multipoint loglikelihood.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *NbRun* : Repeats *NbRun* times the LK heuristic.
- *CollectMaps* : Possible values are -1, 0, 2, 3, 4 and 5. If *CollectMaps* ¿= 0 then every tour found by LKH is inserted into the CarthaGene heap. Moreover, a positive value sets the backtrack move type of LKH.

Returns: nothing.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> lkhl 1 -1
[-822.78]
Best map with log10-likelihood = -822.78
TSP: optimum= -762.461000 lowerbound= -762.471000 gap= -0.001312% totaltime= \
0.06

Map -1 : log10-likelihood =  -822.78
-------:
 Set : Marker List ...
   1 : MS1 G39 MS3 MS2 G37 G40 G36 MS4 G3 G4 MS5 G1 G2 MS6 MS7 MS9 MS8 G21 G2\
6 G22 G25 G23 G27 G24 G28 G34 G33 G30 G29 G31 G32 MS13 G35 MS20 G10 G9 MS19 G\
19 MS17 G20 G11 G8 G7 G12 G13 G16 G6 G5 G17 G18 G14 G15 MS15

CG>
```

See also:

- `lkh` (2.5.8)
- `lkhn` (2.5.9)
- `lkhd` (2.5.11)
- `cg2tsp` (2.5.14)

2.5.11 `lkhd`

Computes an intuitively nice marker ordering using 2-points distances.

Synopsis: The `lkhd` command is invoked as either one of :

- `lkhd` *Options*
- `lkhd` *NbRun CollectMaps*

Description: The `lkhd` command converts the current marker selection and associated genetic / Radiated Hybrid data into a Traveling Salesman Problem which is solved using Keld Helsgaun's LKH software. LKH is based on the Lin-Kernighan heuristic. It is possible to collect every tour found by LKH into the CarthaGene heap, in order to get a map with a possible better multipoint loglikelihood.

Warning: since 2-points estimations are typically unavailable for some pairs of markers when data-sets are merged, this command will return very poor maps on such merged data. This command should especially not be used when two data-sets have been merged with separate parameters estimation (using `mergor`).

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *NbRun* : Repeats *NbRun* times the LK heuristic.
- *CollectMaps* : Possible values are -1, 0, 2, 3, 4 and 5. If *CollectMaps* ¿= 0 then every tour found by LKH is inserted into the CarthaGene heap. Moreover, a positive value sets the backtrack move type of LKH.

Returns: nothing.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> lkhd 1 -1
[-833.04]
Best map with log10-likelihood = -833.04
TSP: optimum= 16.678000 lowerbound= 16.676800 gap= 0.007196% totaltime= 0.04

Map -1 : log10-likelihood =  -833.04
-------:
 Set : Marker List ...
   1 : MS1 G39 MS3 MS2 G37 G40 G36 MS4 G1 MS5 G4 G3 G2 MS6 MS7 MS9 MS8 G21 G2\
6 G22 G25 G23 G27 G24 G28 G34 G33 G30 G29 G31 G32 G35 MS13 G5 G17 G14 MS15 G1\
5 G18 G16 G6 G13 G12 G7 G8 G11 G20 MS17 G19 MS19 G9 G10 MS20

CG>
```

See also:

- `lkh` (2.5.8)
- `lkhn` (2.5.9)
- `lkhl` (2.5.10)
- `cg2tsp` (2.5.14)

2.5.12 `lkhocb`

Computes an intuitively nice marker ordering using 2-points Obligate Chromosome Breaks criterion.

Synopsis: The `lkhocb` command is invoked as either one of :

- `lkhocb` *Options*
- `lkhocb` *NbRun CollectMaps*

Description: The `lkhocb` command converts the current marker selection and associated genetic / Radiated Hybrid data into a Traveling Salesman Problem which is solved using Keld Helsgaun's LKH software. LKH is based on the Lin-Kernighan heuristic. It is possible to collect every tour found by LKH into the CarthaGene heap, in order to get a map with a possible better multipoint loglikelihood.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *NbRun* : Repeats *NbRun* times the LK heuristic.
- *CollectMaps* : Possible values are -1, 0, 2, 3, 4 and 5. If *CollectMaps* ¿= 0 then every tour found by LKH is inserted into the CarthaGene heap. Moreover, a positive value sets the backtrack move type of LKH.

Returns: nothing.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> lkhocb 1 -1
[-822.75]
Best map with log10-likelihood = -822.75
TSP: optimum= 611.000000 lowerbound= 610.872000 gap= 0.020954% totaltime= 0.07

Map -1 : log10-likelihood =  -822.75
-------:
 Set : Marker List ...
   1 : MS1 G39 MS3 MS2 G37 G40 G36 MS4 G3 G4 MS5 G1 G2 MS6 MS7 MS9 MS8 G21 G2\
6 G22 G25 G23 G27 G24 G28 G34 G33 G30 G31 G29 G32 MS13 G35 MS20 G10 G9 MS19 G\
19 MS17 G20 G11 G8 G7 G12 G13 G16 G6 G5 G17 G18 G14 MS15 G15

CG>
```

See also:

- `lkhocbn` (2.5.13)
- `lkh` (2.5.8)
- `cg2tsp` (2.5.14)

2.5.13 `lkhocbn`

Computes an intuitively nice marker ordering using normalized 2-points Obligate Chromosome Breaks criterion.

Synopsis: The `lkhocbn` command is invoked as either one of :

- `lkhocbn` *Options*
- `lkhocbn` *NbRun CollectMaps*

Description: The `lkhocbn` command converts the current marker selection and associated genetic / Radiated Hybrid data into a Traveling Salesman Problem which is solved using Keld Helsgaun's LKH software. LKH is based on the Lin-Kernighan heuristic. It is possible to collect every tour found by LKH into the CarthaGene heap, in order to get a map with a possible better multipoint loglikelihood.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *NbRun* : Repeats *NbRun* times the LK heuristic.
- *CollectMaps* : Possible values are -1, 0, 2, 3, 4 and 5. If *CollectMaps* ¿= 0 then every tour found by LKH is inserted into the CarthaGene heap. Moreover, a positive value sets the backtrack move type of LKH.

Returns: nothing.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> lkhocbn 1 -1
[-823.85]
Best map with log10-likelihood = -823.85
TSP: optimum= 627.818000 lowerbound= 627.582000 gap= 0.037605% totaltime= 0.07

Map -1 : log10-likelihood =  -823.85
-------:
 Set : Marker List ...
   1 : MS17 G20 G19 MS19 G9 G10 G11 G8 G7 G12 G13 G16 G6 G5 G17 G18 G14 G15 M\
S15 G35 MS13 G32 G29 G31 G30 G33 G34 G28 G24 G27 G23 G25 G22 G26 G21 MS8 MS9 \
MS7 MS6 G2 G1 MS5 G4 G3 MS4 G36 G40 G37 MS2 MS3 G39 MS1 MS20

CG>
```

See also:

- `lkhocb` (2.5.12)
- `lkh` (2.5.8)
- `cg2tsp` (2.5.14)

2.5.14 `cg2tsp`

Convert current marker selection to TSP.

Synopsis: The `cg2tsp` command is invoked either as:

- `cg2tsp` *Options*
- `cg2tsp` *FileName*

Description: The `cg2tsp` command converts the current marker selection and associated genetic / radiated hybrid data into different traveling salesman problems, one for each available criterion. Each file name is a concatenation of a prefix and the *FileName* parameter. The prefix can be: an empty string for 2-points loglikelihoods, *norm* for normalized 2-points loglikelihoods, *dist* for 2-points distances, *lod* for 2-points LOD criteria, *min* for minimum 2-points loglikelihoods, *max* for maximum 2-points loglikelihoods. Resulting files are in the TSPLIB file format.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *FileName*: the suffix name of the TSP files.

Example:

```
...
CG> dsload Data/rh.cg
CG> cg2tsp rh.tsp
...
```

See also:

-
- `lkh` (2.5.8)
-
- `lkhn` (2.5.9)
-
- `lkhl` (2.5.10)
-
- `lkhd` (2.5.11)

2.5.15 `build`

This command is obsolete. It is largely superseded by the `buildfw` command which may also build comprehensive maps. It heuristically builds a relatively good comprehensive map using an incremental insertion procedure.

Synopsis: The CarthaGene `build` command is invoked as either one of :

- `build` *Options*
- `build` *NbMap*

Description : starting from the most tightly connected (maximum 2 points LOD) pair of markers, the loci whose mean LOD with already inserted markers is maximum is inserted at its best position. Actually, all best *NbMap* insertion points are kept. The process is repeated for all markers.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *NbMap*: the number of candidate maps kept at each iteration.

Returns: nothing but all maps explored are candidate for the heap.

Example :

```
# we first load a data set
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
# we perform group detection (output omitted)
CG> group 0.3 3
...
# we select group 10
CG> mrkselset [groupget 10]

# we use the command build (20 parallel maps)
CG> build 20

Build(20) : |||||||||

Map 14 : log10-likelihood =  -141.81
-------:
 Set : Marker List ...
   1 : G7 G12 G13 G16 G6 G5 G17 G18 G14

# and check the quality of the final map
CG> flips 9 0 0

Single Flip(window size : 9, threshold : 0.00).


Map -1 : log10-likelihood =  -141.81
-------:
 Set : Marker List ...
   1 : G7 G12 G13 G16 G6 G5 G17 G18 G14


 2 2 2 3 2 1 3 3 2  log10
 1 6 7 0 0 9 1 2 8   -141.81
[8 7 6 5 - 3 2 1 0]      0.00


# it should be Ok.
```

See also:

- `buildfw` (2.5.17)
- `nicemapl` (2.5.4)
- `nicemapd` (2.5.5)

- `mfmapl` (2.5.6)
- `mfmapd` (2.5.7)

2.5.16 Framework mapping

There is only one command in CARTHAGENE to perform initial framework mapping but it is quite powerful. It is the `buildfw` command and it may also perform comprehensive mapping. The command automatically select a subset of the current set of active markers and orders them. The aim is to build a map such that no alternative order of the selected markers has a loglikelihood within a given threshold of the loglikelihood of the map.

The procedure used to tackle this difficult problem is heuristics and is inspired from similar functionalities in RHMAP [LBLC95] and CRIMAP [Gre88]. It uses an incremental insertion method, trying to insert available markers in a current map. For a given marker that could be inserted, it tries to insert the marker at all possible position of the map. The difference in loglikelihood between the best insertion and the second best insertion is used to qualify the marker. The marker inserted in practice is the marker that maximizes this difference and such that this difference is larger than a first threshold (called the "Adding Threshold"). The marker is inserted at its optimal position. If there are orders whose difference in loglikelihood with this best position is less than a second threshold (called the "Keep Threshold"), they are also kept as possible new starting points for the next marker to be inserted. This threshold must naturally be equal to or larger than the adding threshold.

The build process may start from an empty map or from an existing order (in this case, it must contain at least 3 markers). When the starting map is empty, all triplets are considered as candidates and only the best triplet according to the difference in loglikelihood between the best order and the second best order is used. Otherwise, the initial order specified is used directly.

After a map is built, using this process, some postprocessing can be applied optionally: all remaining non insertable markers are inserted one by one, in all possible positions in the final map and then removed. For each such marker, the command reports how the loglikelihood changes wrt to the best insertion point. The best position is marked with a "+". The difference in loglikelihood between the best position and each position are printed out if this difference is less than a threshold.

It is important to note that whatever the threshold you specify, this procedure may easily be fooled and the final map produced will not really be true framework map (i.e., a map such that all alternative orders have a difference in loglikelihood with the initial map larger than the "adding threshold"). This is not a weakness of CARTHAGENE but a weakness of all simple heuristics procedure. To be more confident about the quality of your maps, it is strongly advised to apply improving commands to it and analyse the heap to check that no close alternative order appears in it.

2.5.17 `buildfw`

Tries to heuristically build a good framework map, *i.e.*, a map such that all alternatives orders have a loglikelihood not within a given threshold of the framework map. The command may also be used to build comprehensive maps (including all markers) by adjusting thresholds.

Synopsis : The CarthaGene `buildfw` command is invoked as either one of :

- `buildfw` *Options*
- `buildfw` *keepThres AddThres MrkList MrkTest*

Description : The procedure is an incremental insertion procedure. It starts either by selecting a triplet of loci or by using the list of loci *MrkList*. The difference in log-likelihood of the two best maps you can build with three given loci is used as criterion. The greater the difference the better the triplet.

Then, each available locus is tentatively inserted at each possible position. If there is no locus that provides a difference in log-likelihood larger than the *AddThres* threshold, the algorithm stops. If there are many orders for which the difference of log-likelihood is larger than the *KeepThres* Threshold, they are kept for the next step. The *MrkTest* flag indicates if post-processing is applied. If it is set to 0 no post-processing occurs. If it is set to 1, each remaining locus is tentatively inserted at each position in the previously built framework map and the difference in log-likelihood between each possible insertion and the best one (denoted by a '+') is reported. If it set to 2, no framework is built in the previous stage, instead the *MrkList* list of loci is used and the same post-processing as above applies. If the *MrkTest* equals 0 or 1, the framework's markers become the list of currently active markers.

When *MrkTest* flag is different from 0, further information on the remaining markers that have not been integrated in the map is provided in a table. Each remaining marker is represented on a line. The information is displayed in 6 group of columns.

```
BuildFW, remaining loci test :

      |                  |
      |                  |     Lod2pt         Dist2pt
      | 1 2 3 4 5 6 7 8  |  Left<-M->Right Left<-M->Right | 0->N    N->M | Weight Nb:W<AM  Id  | Name
    --|------------------|--------------------------------|--------------|---------------------|-------
M1702 |              + 0 |   2.16   23.97   115.1   2.5   |  680.8 106.7 |     0     2     9   | M1702
```

From left to right, we get:

- the first group of columns displays the name of the marker that has been tentatively inserted (M1702 above)
- the second group of columns displays the result of inserting this marker in the framework map. The best possible insertion position is reported by a "+". When alternative insertion positions exists with a small difference in log-likelihood (less than *KeepThres*), they are also reported by the nearest integer that represents this difference (eg "`0`" above corresponds to a difference in log-likelihood between in $[0, 0.5[$).
- the third group of columns indicates the relations of the inserted markers with its two neighbors in the best insertion position. The two-points LOD and the two-points distances (Haldane or centiRay depending on the dataset type) with the two flanking markers are displayed (when available).

  When merged datasets are used, the two-points distances reported are computed only on informative data-sets and averaged among the informative data-sets. When RH data is used, and because different irradiation level (or mixed RH/genetic data) may have been used, distances are not averaged: only the first dataset merged is used.
- the fourth group of columns aims at positioning the remaining markers wrt. the framework map. It first (`0->N`) indicates the multi-point distance from the left origin of the map to the left flanking marker estimated on the original framework map (before marker insertion). If the marker best insertion is at the extreme left of the map, a `-` is displayed. Then (`N->M`), an estimation of the distance from this left flanking marker to the marker inserted is reported. This distance is simply computed by rescaling the multi-point distance between the two flanking markers using the ratio between the two-points distances from the inserted marker to the left and right flanking markers reported in the previous group of columns. If either the left or right flanking markers are undefined, the two-point distance alone is reported. If there is no available information on either of the flanking markers, `NA>` or `NA<` is printed (indicating the marker that generates the indetermination).
- the 5^{th} group of columns successively reports:

  - `Weight`: the sum of the difference of log-likelihood with the best map for all maps within *KeepThres* of the best map. A large weight indicates a marker whose position is not well defined.
  - `Nb:W<AM`: the number of insertion position where the difference in log-likelihood exceeds *KeepThres*.
  - `Id`: the marker numerical Id.
- the last column gives the name of the inserted marker again (same as first column).

Arguments :

- *Options* : `-u` to obtain the synopsis of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *KeepThres*: the minimum difference in log-likelihood between the best insertion point and the second best insertion point required for the map to be considered in the future.
- *AddThres*: the minimum difference in log-likelihood between the best insertion point and the second best insertion point required for a locus to be insertable. This threshold is also used to filter out the differences in log-likelihood reported by the postprocessing option (only differences lower than the threshold will be reported).
- *MrkList*: an order of markers to start from (may be empty or caontain at least 3 markers).
- *MrkTest*: a flag to select either framework mapping only (0), framework mapping followed by a postprocessing (1) or just the post-processing applied to (2)*MrkList*.

Returns : nothing but the set of active markers is automatically set to the framework map built if the *MrkTest* equal 0 or 1, not 2.

Example :

```
# we first load a data set
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
# we perform linkage group detection (output omitted)
CG> group 0.3 3
...
# we select the group  10
CG> mrkselset [groupget 10]

# we build a framework map for this group
CG> buildfw 3 3 {} 1

BuildFW, Adding  Threshold = 3.00, Saving Threshold = 3.00.

>>> Delta = 6.89 :

Map  0 : log10-likelihood =   -66.75
-------:
 Set : Marker List ...
   1 : G7 G12 G16

>>> Delta = 6.30 , Id = 32,  Locus = G18 :
```

```
Map  0 : log10-likelihood =   -80.72
-------:
 Set : Marker List ...
   1 : G7 G12 G16 G18

>>> Delta = 7.04 , Id = 28,  Locus = G14 :

Map  0 : log10-likelihood =   -93.15
-------:
 Set : Marker List ...
   1 : G7 G12 G16 G18 G14

BuildFW, remaining loci test :
       |            |
       | 2 2 3 3 2  |     Lod2pt         Dist2pt
       | 1 6 0 2 8  |  Left<-M->Right Left<-M->Right | 0->N    N->M | Weigh...
     --|------------|--------------------------------|--------------|------...
    G5 |       3 1 + |  13.70     -      29.6    -    |   91.9  29.6 |     4...
   G17 |       2 + 0 |  18.05   15.66    21.1  25.3   |   72.0   9.0 |     2...
   G13 |    + 3      |  20.41   22.03    14.6  10.6   |   30.4  12.7 |     3...
    G6 |    + 0      |  16.23   20.68    25.7  14.6   |   30.4  14.0 |     0...

# and check its reliability (any map at less than 3 LOD ?)
CG> flips 4 3 0

Single Flip(window size : 4, threshold : 3.00).


Map -1 : log10-likelihood =   -93.15
-------:
 Set : Marker List ...
   1 : G7 G12 G16 G18 G14


 2 2 3 3 2  log10
 1 6 0 2 8    -93.15


# no.
```

See also:

- `build` (2.5.15)
- `heaprinto` (2.4.16)
- `cgrobustness` (2.5.18)

2.5.18 `cgrobustness`

Sets the reliability testing threshold for framework maps.

Synopsis: The `cgrobustness` command is invoked as either:

- `cgrobustness` *Options*
- `cgrobustness` *Threshold*

Description: The `cgrobustness` command sets the threshold used to verify if the current best known map is robust or not. A map M is not robust if there exists another map M' such that $L(M') > L(M) + \delta$, where $L(M)$ is the multipoint log-likelihood of M. A common value for the threshold δ is -3.0. When robustness checking is activated, each time a new map is inserted into the heap, the system verifies if the current best map is still robust. If not then it will stop any running search procedure because it has found a proof that the best known map is not robust enough. This is typically used when checking the reliability of a framework map using a cpu-intensive command such as `flips` or `greedy` in order to stop the search process as soon as the unreliability is proven.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *Threshold*: minimum log likelihood distance to the current best map.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> buildfw 3 3 {} 1

BuildFW, Adding  Threshold = 3.00, Saving Threshold = 3.00.

>>> Delta = 10.84 :

Map  0 : log10-likelihood =  -114.56
-------:
 Set : Marker List ...
   1 : MS4 MS9 MS12

>>> Delta = 9.73 , Id = 1,  Locus = MS1 :

Map  0 : log10-likelihood =  -155.67
-------:
 Set : Marker List ...
   1 : MS1 MS4 MS9 MS12

>>> Delta = 10.78 , Id = 7,  Locus = MS7 :

Map  0 : log10-likelihood =  -166.66
-------:
 Set : Marker List ...
   1 : MS1 MS4 MS7 MS9 MS12

>>> Delta = 13.30 , Id = 2,  Locus = MS2 :

Map  0 : log10-likelihood =  -178.48
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS4 MS7 MS9 MS12

>>> Delta = 7.08 , Id = 16,  Locus = MS16 :

Map  0 : log10-likelihood =  -203.02
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS4 MS7 MS9 MS12 MS16

>>> Delta = 11.51 , Id = 11,  Locus = MS11 :

Map  0 : log10-likelihood =  -207.54
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS4 MS7 MS9 MS11 MS12 MS16

>>> Delta = 10.59 , Id = 18,  Locus = MS18 :

Map  0 : log10-likelihood =  -232.48
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS4 MS7 MS9 MS11 MS12 MS16 MS18

>>> Delta = 8.70 , Id = 15,  Locus = MS15 :

Map  0 : log10-likelihood =  -237.17
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS4 MS7 MS9 MS11 MS12 MS15 MS16 MS18

>>> Delta = 7.08 , Id = 5,  Locus = MS5 :

Map  0 : log10-likelihood =  -239.87
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS4 MS5 MS7 MS9 MS11 MS12 MS15 MS16 MS18

>>> Delta = 4.08 , Id = 8,  Locus = MS8 :

Map  0 : log10-likelihood =  -248.48
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS4 MS5 MS7 MS8 MS9 MS11 MS12 MS15 MS16 MS18

BuildFW, remaining loci test :
       |                          |
       |               1 1 1 1 1  |     Lod2pt         Dist2pt
       | 1 2 4 5 7 8 9 1 2 5 6 8  |  Left<-M->Right Left<-M->Right | 0->N  ...
     --|--------------------------|--------------------------------|-------...
   MS3 |  2 +                     |  47.62   35.05     0.6   5.7    |   21.0...
   MS6 |        + 3               |  15.41   17.32     4.5   1.6    |   30.5...
  MS10 |          2 0 + 0 3 3 2   |   7.22    3.61     0.0   0.0    |   44.9...
  MS13 |                + 0       |  11.91   18.66     8.4   0.0    |   53.7...
  MS14 |            2 1 0 0 0 + 1 |   3.61    0.00     0.0 345.4    |   71.6...
  MS17 |                      0 + |  15.95     -       0.0    -     |   94.4...
  MS19 |                      2 + |  27.14     -       2.8    -     |   94.4...
  MS20 |                      3 + |  22.88     -       5.0    -     |   94.4...

CG> cgrobustness -5

CG> greedy 1 0 1 20 0

Map -1 : log10-likelihood =  -248.48
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS4 MS5 MS7 MS8 MS9 MS11 MS12 MS15 MS16 MS18
Run number 0
Current best map is not robust!
Aborted!

CG> heapsize 3

CG> heaprint

Map  0 : log10-likelihood =  -255.57
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS5 MS4 MS7 MS8 MS9 MS11 MS12 MS15 MS16 MS18

Map  1 : log10-likelihood =  -252.56
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS4 MS5 MS7 MS9 MS8 MS11 MS12 MS15 MS16 MS18

Map  2 : log10-likelihood =  -248.48
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS4 MS5 MS7 MS8 MS9 MS11 MS12 MS15 MS16 MS18

CG>
```

See also:

- `cgnotrobust` (2.5.19)

2.5.19 `cgnotrobust`

Disable reliability test of framework maps.

Synopsis: The `cgnotrobust` command is invoked as either:

- `cgnotrobust` *Options*
- `cgnotrobust`

Description: The `cgnotrobust` command disables the reliability test of framework maps. It is equivalent to call `cgrobustness` with *Threshold* equal to 1e100.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

See also:

- `cgrobustness` (2.5.18)

2.6 Improving and validating existing maps

Map improving and validating commands all try to find a map whose loglikelihood is improved over an initial map. For all such commands, the initial map used is the best map in the heap. Beyond a possible map improvement, all these commands may affect the contents of the heap: all maps considered by these commands are candidate for insertion.

The first category of commands are simple heuristics command that slightly perturbate the initial map in a systematic manner:

- The famous `flips` command applies all possible permutations in a sliding window of fixed size on the initial map. The command reports all maps found whose loglikelihood lies within a given threshold of the loglikelihood of the original map. If a new improved map is found, the command may be automatically iterated. It can also be used to explore all possible orders when a small number of markers is selected by choosing a window size equal to the number of markers.
- the `polish` command removes one marker of the initial map and tries to insert it in all possible intervals. The variation in loglikelihood is reported for all markers and all positions in a matrix.

These two commands are simple and systematic. CARTHAGENE offers much more powerful commands that use a class of discrete optimization techniques called “stochastic optimization” or more precisely “local search methods”. This class of methods is usually acknowledged as the most efficient class of technique for hard discrete optimisation problems such as the genetic marker ordering[2]

These optimisation methods all follow a similar principle: one or several current orders are stochastically “perturbated” and depending on the result of the pertubation and a random choice, the perturbated maps may replace the original maps or not. Three such methods are integrated in CARTHAGENE: simulated annealing [LA84], taboo search [Glo89, Glo90] and genetic algorithm [Gol89]. The three methods use the same perturbation mechanism: a subsection of the map is chosen and flipped [3]. Simulated annealing and the genetic algorithm may also use more complex perturbation techniques.

These commands described next can take a long time to improve or not the current solution. The system does not provide a prediction of the time cost of this algorithms. Nevertheless, these methods can be interrupted online, without loss of information (see section 2.6.9).

2.6.1 Simulated annealing

The `annealing` command simulates the annealing of a metal using an analogy between the energy of the system and the criteria minimized (the opposite of the loglikelihood). Given an initial marker ordering and a current parameter called the “temperature” (noted T), the map is perturbated and the difference δ in loglikelihood is computed:

- if the loglikelihood is improved ($\delta \geq 0$), the perturbation is accepted and the perturbated map becomes the new map.
- else, the perturbated is accepted only with probability $e^{\frac{\delta}{T}}$. This simulated the Boltzman distribution used in statistical physics.

Starting from an initial temperature T_i, this process is repeated a given number of times (noted N). then the temperature is cooled by a constant ratio $\alpha < 1$. This process is repeated until a final temperature T_f is reached. Naturally, the slower the cooling and the larger the number of trial at constant temperature, the slower the method but also the more powerful it is. Basically, we strongly advise users to play with the parameters T_i, T_f, α, N on their data. CARTHAGENE includes an auto-adjusting methods for T_i which makes the parameter of limited importance. The parameter α should be taken close to 1 and a value between

---

[2] Which is closely related to the so-called travelling salesman problem, an NP-hard problem.

[3] This is called a 2-OPT move

0.9 and 0.995 is typical. The parameter N should be of the same order as n^2 where n is the number of markers selected.

The parameters should be adjusted by trial and error. When the simulated annealing is executed, the software reports its behavior. If a new improved map is found, a '+" is printed. If such an event occurs not too long before the end of the annealing, the process should be ideally repeated with a slower/longer cooling schedule.

2.6.2 Taboo search

The `greedy` command uses a taboo search technique. Starting from an initial map, all the possible maps that can be obtained from the initial map by flipping a subsection of the map are built and their maximum loglikelihood computed. The best map among all these maps is chosen as the new map unless this move is "taboo" (in CarTHaGene a move is taboo is it has been used "recently" where the definition of recently varies stochastically during search). Actually, if the move is taboo and if the best map is better than the best known map, this best taboo map becomes the new map despite its taboo property. This process is repeated a number of times (precomputed by CarTHaGene and equal to twice the number of markers) but extra iterations can be specified.

The whole process can be repeated a given number of times, starting each time from a different starting point. As the search proceeds, the user is informed of the status: if a new better solution is found, its loglikelihood is reported. If no better solution is found a "-" is printed. If the search detets it is stuck, always moving in the same region, a "*" is printed and the search automatically restarted from a new starting point. If many such "*" are printed, this is usually a good indication that the best solution has been found and is, repeatedly, found again and again.

The parameters of the search are given by the number of time the serach process must be run, how many extra iterations are allowed and the bounds of variation for the definition of what a taboo move is. These bounds are defined as percentages and should therefore be between 1 and 99. Typical values are 5 and 30.

2.6.3 Genetic algorithm

The `algogen` command uses a genetic algorithm i.e., an optimizing algorithm based on the simulation of a reproduction/mutation/selection process. Maps are considered as individuals in a population of fixed size. They are bred together using a specific "mixing" process, some of them are mutated by flipping a random subsection and selected (maps which the highest loglikelihood have higher probabilities of participating in the next generation). The process is repeated over and over for a given number of generation

The command takes a number of parameters: number of generations, size of the population (a 0 here means the heap is used as the initial population), the selection type, the probability of breeding, of mutation and an extra parameter specifying that selection strength increases as generations go or not.

2.6.4 `flips`

Applies all possible permutations in a sliding window on the current best map and reports likelihood variations.

Synopsis: The `flips` command is invoked either as:

- `flips` *Options*
- `flips` *Size LOD-Threshold Iterative*

Description: The `flips` command applies all possible permutations in a sliding window of a given *Size* on the current best available map (the best map in the heap). If the map obtained has a loglikelihood within *LOD-Threshold* of the loglikelihood of the original map, then the permutation that lead to this map is displayed. If a new improved map is found, the filps command can be automatically iterated using a value of `1` for the last argument.

Note that this command can be also used to perform exhaustive search of all possible order when a small (< 10) number of markers is active by simply specifying the number of active markers as the sliding window size.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *Size* : the *size* of the sliding window (cannot exceed 9),
- *LOD-Threshold*: the maximum difference of loglikelihood with the best map to report the permutation,
- *Iterative*: a boolean that indicates whether the flips command should be iterated as long as a new improved map has been found.

Example :

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> group 0.1 7

Linkage Groups :
---------------:
LOD threshold=7.00
Distance threshold=10.00:

 Group ID : Marker ID List ...
        1 : 17 19 20 18
        2 : 14
        3 : 11 13 12 16 15
        4 : 2 4 3 7 6 5 9 10 8
        5 : 1
5
CG> mrkselset [groupget 4]

# run this heuristic
CG> nicemapd

Map -1 : log10-likelihood =  -135.41
-------:
 Set : Marker List ...
   1 : MS2 MS3 MS4 MS5 MS6 MS7 MS8 MS9 MS10

# the reliability of the result with a large flip
CG> flips 9 1 1

Repeated Flip(window size : 9, threshold : 1.00).
```

```
Map -1 : log10-likelihood =  -135.41
-------:
 Set : Marker List ...
   1 : MS2 MS3 MS4 MS5 MS6 MS7 MS8 MS9 MS10


                 1  log10
 2 3 4 5 6 7 8 9 0   -135.41
[- - - - - - - 8 7]     -0.00
[7 8 6 5 - 3 2 1 0]     -0.00
[8 7 6 5 - 3 2 1 0]      0.00


# the map was not optimal
```

See also:

- `polish` (2.6.5)
- `annealing` (2.6.6)
- `greedy` (2.6.7)

2.6.5 `polish`

Checks map reliability by swapping pairs of markers.

Synopsis: The `polish` command is invoked either as:

- `polish` *Options*
- `polish`

Description: For all the markers in the current best known map in the heap, the `polish` command displaces each marker in all possible intervals and reports the difference in log-likelihood in a matrix. Each line and column corresponds to a marker and displays the difference in loglikelihood of the maps obtained by displacing the marker of the line to the direct right of the marker in the column. A negative number indicates a better map has been found.

All the maps considered are candidates to be inserted in the heap adn therefore any improvement obtained here is automatically memorized.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> group 0.1 7

Linkage Groups :
---------------:
LOD threshold=7.00
Distance threshold=10.00:

 Group ID : Marker ID List ...
        1 : 17 19 20 18
        2 : 14
        3 : 11 13 12 16 15
        4 : 2 4 3 7 6 5 9 10 8
        5 : 1
5
CG> mrkselset [groupget 4]

CG> nicemapl

Map -1 : log10-likelihood =  -154.08
-------:
 Set : Marker List ...
   1 : MS2 MS3 MS4 MS7 MS8 MS9 MS6 MS5 MS10

CG> polish

Local map analysis:

         MS2   MS3   MS4   MS7   MS8   MS9   MS6   MS5  MS10
      ------------------------------------------------------
  MS2 |-----   1.2  22.3  43.7  58.2  38.4  35.9  29.9  27.9
  MS3 |  1.2 -----  21.7  45.4  62.3  39.8  37.4  30.8  30.4
  MS4 |  9.0  21.7 -----  23.1  38.1  19.5  14.2  10.5  12.5
  MS7 | 21.0  44.3  23.1 -----  15.4   0.2   2.9   5.9  11.0
  MS8 | 30.2  62.9  42.3  15.4 -----   0.1  16.2  11.0  12.2
  MS9 | 26.6  57.9  40.1  15.9   0.1 -----  13.9   8.9   8.9
  MS6 |  7.1  24.4  10.6  -4.3  -1.8  13.9 -----   2.7   1.5
  MS5 |  7.8  24.6   2.8  -4.3   4.2  16.5   2.7 -----   1.4
 MS10 |  4.3  19.9   4.5   3.1  -0.9  -1.7  -1.7   1.4 -----
      ------------------------------------------------------
```

See also:

- `flips` (2.6.4)
- `annealing` (2.6.6)
- `greedy` (2.6.7)

2.6.6 `annealing`

Tries to optimize the maximum multipoint loglikelihood using a dedicated simulated annealing stochastic optimization algorithm.

Synopsis: The `annealing` command is invoked either as:

- `annealing` *Options*
- `annealing` *LPlateau InitTemp FinalTemp Cooling*

Description: The `annealing` command tries to improve the loglikelihood of the best known order for the current list of active loci (the best map in the heap) using a dedicated simulated annealing search algorithm. The annealing simulation starts from the temperature *InitTemp* and repeats until the temperature *FinalTemp* has been reached using a geometric cooling schedule with a cooling parameter *Cooling* and constant temperature plateaus of length *LPlateau*. The initial temperature is automatically reajusted if it is too cold/hot.

During the search, the algorithm gives some feedback by printing the current temperature and a character "`+`" if an imporved map has been found.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *LPlateau*: length of constant temperature plateaus in the colling schedule.
- *InitTemp FinalTemp*: starting/ending temperature for the simulated annealing.
- *Colling*: cooling schedule parameter. This parameter should be between 0 and 1 and is typically taken to be close to 1. The closest to 1 the longer but also the better the algorithm.

Returns: nothing. All the map explored by the algorithm are candidate for the heap.

Example:

```
# we first load a dataset
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
# perform linkage group detection (output omitted)
CG> group 0.3 3
...
# select group number 10
CG> mrkselset [groupget 10]

# put a (stupid) map in the heap
CG> sem

Map -1 : log10-likelihood =  -161.87
-------:
 Set : Marker List ...
   1 : G5 G18 G17 G14 G16 G13 G12 G6 G7

# use a (fast) annealing command
CG> annealing 100 100 0.1 0.9

Map -1 : log10-likelihood =  -161.87
-------:
 Set : Marker List ...
   1 : G5 G18 G17 G14 G16 G13 G12 G6 G7
```

```
100.00? :  +++
90.00? :
81.00? :  +
72.90? :
65.61? :
59.05? :  +
53.14? :
47.83? :
43.05? :
38.74? :
34.87? :
31.38? :
28.24? :
25.42? :
22.88? :
20.59? :
18.53? :
16.68? :
15.01? :
13.51? :
12.16? :
10.94? :
9.85? :
8.86? :
7.98? :
7.18? :
6.46? :
5.81? :
5.23? :
4.71? :
4.24? :
3.82? :
3.43? :
3.09? :
2.78? :  +
2.50? :
2.25? :
2.03? :
1.82? :
1.64? :
1.48? :
1.33? :
1.20? :
1.08? :
0.97? :
0.87? :
0.79? :
0.71? :
0.64? :
0.57? :
0.52? :
0.46? :


# we check the map with a large flip
CG> flips 9 0 0
```

```
Single Flip(window size : 9, threshold : 0.00).


Map -1 : log10-likelihood =  -141.81
-------:
 Set : Marker List ...
   1 : G7 G12 G13 G16 G6 G5 G17 G18 G14


 2 2 2 3 2 1 3 3 2  log10
 1 6 7 0 0 9 1 2 8   -141.81
[8 7 6 5 - 3 2 1 0]      0.00


# the map found was indeed optimal
```

See also:

- `greedy` (2.6.7)
- `flips` (2.6.4)
- `polish` (2.6.5)

2.6.7 `greedy`

Tries to optimize the maximum multipoint loglikelihood using a dedicated taboo search algorithm.

Synopsis: The `greedy` command is invoked either as:

- `greedy` *Options*
- `greedy` *NbLoop Fuel TabooMin TabooMax Anytime*

Description: The `greedy` command tries to improve the loglikelihood of the best known order for the current list of active loci (the best map in the heap) using a dedicated taboo search algorithm. This iterative algorithm will automatically compute a minimum number of iterations but additional iterations can be requested by setting the parameter *Fuel* to perform additional iterations. The *TabooMin TabooMax* indicates the minimum and maximum length of the taboo list which varies stochastically during search. These two parameters are %ages (between 0 and 100). Finally, the search is repeated *NbLoop* times if requested. The Anytime ratio controls the tradeoff between search speed and solution quality. It may vary from 0 to 100.

During the search, the algorithm gives some feedback by printing either:

- the "`-`" character: iterations have been performed without improving the best known map.
- the "`*`" character: the algorithm has detected it is exploring orders it has already considered before on several occasions and decides to start a new search from a new order.
- "`[`number]": a new improved map has been found.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *NbLoop*: the number of times the taboo search is repeated.
- *Fuel*: the extra number of iterations requested. By default, an heurustically determined number of iterations is computed. If the Fuel is positive, an extra number of iterations are performed. If the Fuel is negative then the number of iterations performed will be *exactly* equal to the absolute value of the Fuel.
- *TabooMin TabooMax*: the minimum and maximum size of the taboo list in %age (between 0 and 100).
- *Anytime*: the %age (between 0 and 100) of the neighborhood explored at each move. A value 100% means all the possible swaps (i.e. reversing the order of a consecutive sublist of markers) are examined. A smaller strictly positive value will speed up the search, improving solution quality faster at the beginning of the search. But solution quality may be altered at the end. A typical value may be 25%. A value of 0 corresponds to the original Carthagene release 0.5 taboo search algorithm.

Returns: nothing. All the map explored by the algorithm are candidate for the heap.

Example:

```
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
CG> group 0.3 3
...
CG> mrkselset [groupget 10]

CG> sem

Map -1 : log10-likelihood =  -161.87
-------:
 Set : Marker List ...
   1 : G5 G18 G17 G14 G16 G13 G12 G6 G7

CG> greedy 1 0 5 25 0

Map -1 : log10-likelihood =  -161.87
-------:
 Set : Marker List ...
   1 : G5 G18 G17 G14 G16 G13 G12 G6 G7
Run number 0

[-153.81]
[-148.65]
[-145.59]
[-145.08]-
[-143.70]
[-141.81]------------****---********---*-

CG> flips 9 0 0

Single Flip(window size : 9, threshold : 0.00).
```

```
Map -1 : log10-likelihood =  -141.81
-------:
 Set : Marker List ...
   1 : G7 G12 G13 G16 G6 G5 G17 G18 G14


 2 2 2 3 2 1 3 3 2  log10
 1 6 7 0 0 9 1 2 8   -141.81
```

See also:

- `annealing` (2.6.6)
- `algogen` (2.6.8)
- `flips` (2.6.4)
- `polish` (2.6.5)

2.6.8 `algogen`

Tries to optimize the maximum multipoint loglikelihood using a dedicated genetic algorithm.

Synopsis : The CarthaGene `algogen` command is invoked as either one of :

- `lelacommande` *Options*
- `lelacommande` *NbGen NbMap SelType ProbaCross ProbaMut EvolFitn*

Description : The genetic algorithm tries to optimize the loglikelihood of orders using a simulation of mating, mutation and selection by fitness. The initial population is defined by the best maps stored in the heap or from generated maps.

As the algorithm runs, “+” are printed on a generation such that a map is found that improves over the best map known.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- The *Nbgen* argument correspond to the number of generations simulated.
- The *NbMap* argument correspond to the size of the population. If *NbMap* is set to 0, the command uses the heap to initialize the first generation population.
- The *SelType* argument specifies the type of selection used (0 for the roulette wheel, 1 for the stochastic remainder without replacement).
- The *ProbaCross* argument correspond to the probability of crossing-over. A typical value is 60 %.
- The *ProbaMut* argument correspond to the probability of mutation. A typical value is 5 to 10 %.
- The *EvolFitn* is a flag to set the evolutive fitness (0 or 1) which mean that selection becomes harder and harder with generations.

Returns : nothing, all maps explored are candidate for being stored in the heap.

Example :

```
# we load a dataset
CG> dsload Data/rh.cg
{1 haploid RH 53 118 /homes/thomas/CartaGene/dev/test/Data/rh.cg}
# perform linkage group detection (output omitted)
CG> group 0.3 3
...
# we select the group 10
CG> mrkselset [groupget 10]

# put one (stupid) map in the heap using the ’sem’ command
CG> sem

Map -1 : log10-likelihood =  -161.87
-------:
 Set : Marker List ...
   1 : G5 G18 G17 G14 G16 G13 G12 G6 G7

# launch the ’algogen’ command
CG> algogen 10 8 1 0.6 0.1 1
+
-
-
-
-
-
-
-
-
-

# check the reliability of the result with a large flip
CG> flips 9 0 0

Single Flip(window size : 9, threshold : 0.00).


Map -1 : log10-likelihood =  -141.81
-------:
 Set : Marker List ...
   1 : G7 G12 G13 G16 G6 G5 G17 G18 G14


 2 2 2 3 2 1 3 3 2  log10
 1 6 7 0 0 9 1 2 8   -141.81
[8 7 6 5 - 3 2 1 0]      0.00


# the map was optimal
```

See also:

- `greedy` (2.6.7)

- `annealing` (2.6.6)
- `flips` (2.6.4)
- `polish` (2.6.5)

2.6.9 Interrupting

The ctrl-c sequence will enable the user to register a request of interruption of the current command. The demand will be treated next time a map is computed. The heap of already computed maps remains consistent.

This functionality allow to abort a command taking to much time. The best maps met so far will be kept into the heap and also you can keep on running commands.

2.7 System commands

There are several commands in CARTHAGENE that deal with several system aspects. They can affect the verbosity of CARTHAGENE, gives some basic statistics on CARTHAGENE behavior or restart the software to forget all data loaded. A more detailed description of each command is given below.

2.7.1 `cgout`

Saves the output of CarthaGène to a file. This is different from `cgsave` that allows to save the current state of CarthaGène (in order to later restore it). `cgout` is intended merely to build log file.

Synopsis: The `cgout` command is invoked either as:

- `cgout` *Options*
- `cgout` *FilePath*

Description: The `cgout` command enables to save the output of the session in a file specified by the *FilePath* argument. If the file already exist, the output is appened to the end of the file. If a output file was already specified, the old log file is closed and the output goes to the new one.This process is terminated by using a empty string or by ending CarthaGene.

Only the inputs and outputs of the correct commands of Carthagene will be saved. You may want to launch the shell with the `tee` command (under Unix) to keep a complete log (`tclsh | tee logfile`).

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *FilePath*: the path to a valid text file existing or not.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> sem

Map -1 : log10-likelihood =  -297.78
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS3 MS4 MS5 MS6 MS7 MS8 MS9 MS10 MS11 MS12 MS13 MS14 MS15 MS...

CG> exec rm -f "carthagene.log"
...
CG> cgout "carthagene.log"

CG> heaprintd
...
CG> cgout ""

CG> exec cat "carthagene.log"

CG_log> heaprintd

Map  0 : log10-likelihood =  -297.78, log-e-likelihood =  -685.66
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos   Id name        Haldane     Haldane     Kosambi    (%%age)      LOD

  1   1  MS1          19.5 cM     19.5 cM     16.8 cM    16.2 %%    15.7
  2   2  MS2           0.6 cM     20.1 cM      0.6 cM     0.6 %%    47.6
  3   3  MS3           5.8 cM     25.9 cM      5.5 cM     5.5 %%    35.1
  4   4  MS4           3.7 cM     29.6 cM      3.6 cM     3.5 %%    21.1
  5   5  MS5           3.7 cM     33.3 cM      3.6 cM     3.6 %%    15.4
  6   6  MS6           1.6 cM     34.9 cM      1.6 cM     1.6 %%    17.3
  7   7  MS7           6.7 cM     41.6 cM      6.3 cM     6.3 %%    35.1
  8   8  MS8           2.2 cM     43.8 cM      2.2 cM     2.2 %%    40.8
  9   9  MS9           3.0 cM     46.8 cM      2.9 cM     2.9 %%     7.2
 10  10 MS10           5.4 cM     52.2 cM      5.1 cM     5.1 %%     3.6
 11  11 MS11           8.6 cM     60.7 cM      7.9 cM     7.9 %%    10.9
 12  12 MS12           0.0 cM     60.7 cM      0.0 cM     0.0 %%    18.7
 13  13 MS13           0.0 cM     60.7 cM      0.0 cM     0.0 %%     6.0
 14  14 MS14           5.8 cM     66.6 cM      5.5 cM     5.5 %%     0.0
 15  15 MS15           4.6 cM     71.1 cM      4.4 cM     4.4 %%    26.6
 16  16 MS16          26.1 cM     97.2 cM     21.6 cM    20.3 %%     6.3
 17  17 MS17           0.0 cM     97.2 cM      0.0 cM     0.0 %%    16.0
 18  18 MS18           4.9 cM    102.1 cM      4.7 cM     4.6 %%    27.1
 19  19 MS19           3.0 cM    105.1 cM      2.9 cM     2.9 %%    34.8
 20  20 MS20         ----------              ----------
                     105.1 cM                 95.0 cM


       20 markers, log10-likelihood =  -297.78
                   log-e-likelihood =  -685.66

        EM calls:
           Set  1 : 2 (1,0)
        CPU Time (secs): 0.01
        Maps within -3.0: 1

CG_log> cgout
CG>
```

2.7.2 `cgversion`

Returns the release version of the software.

Synopsis: The `cgversion` command is invoked as either one of :

- `cgversion` *Options*
- `cgversion`

Description: returns the release version of the software.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: nothing.

Example:

```
CG> cgversion

   CarthaGene version 0.999-LKH, Copyright (c) 1997-2004 (INRA).

   CarthaGene comes with ABSOLUTELY NO WARRANTY.
   CarthaGene is free software. You are welcome to redistribute it,
   under certain conditions. See the License file for information.

Type ’help’ for help.


CG>
```

2.7.3 `cglicense`

Obsolete. Initially used to unlock the software using a software key.

Synopsis: The `cglicense` command is invoked either as:

- `cglicense` *Options*
- `cglicense` *key*

Description: Initially, the use of the software Carthagene was limited to simple backcross data sets until it ws “unlocked” using this command with a free software key. This is no longer true since release 0.99.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *key*: a free software key.

Returns: nothing

Example:

```
CG> cglicense FGTZMBTBBCTGWT
```

2.7.4 `cgstat`

Displays some benchmarking statistics on CarthaGene and the Heap.

Synopsis: The `cgstat` command is invoked either as:

- `cgstat` *Options*
- `cgstat` *LOD-Threshold*

Description: The command `cgstat` displays some basic statistics on the software and the contents of the heap. It reports the total CPU-time consumed, the number of calls to the EM (Expectation/Maximization) algorithm and the number of maps in the heap within *LOD-Threshold* of the best known map in the heap.

For internal use only: if the data-set loaded is identified as a randomly generated test data-set, further statistics are reported: difference in log-likelihood of the best know map with the log-likelihood of the order used to generate the data, at what cpu-time stamp this map was found and if it is identical to the order used to generate the data.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *LOD-Threshold*: maximum difference in loglikelihood with the best known map.

Returns: a list of these different statistics (some may be unapplicable when the data set used is not a benchmarking random data-set).

- a boolean that indicates whether the best map found is the correct (model) map.
- the total cpu-time used
- the difference of loglikelihood between the best maporder and the “true” order.
- the cpu-time stamp of the best known map.
- the number of maps in the heap whose loglikelihood is within *LOD-Threshold* of the best map.
- the number of EM calls overall.

Example:

```
# we load a dataset
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
#compute groups
CG> group 0.1 7
...
#and select group 4
CG> mrkselset [groupget 4]
...
# compute a first map
CG> nicemapd
...
# check the quality of this order
CG> flips 2 0 0
...
#check some statistics and ask for the number of maps within 1 of the optimum
CG> cgstat 1

Map  8 : log10-likelihood =  -135.41, log-e-likelihood =  -311.80
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

  1   2  MS2           0.6 cM      0.6 cM      0.6 cM     0.6 %%    47.6
  2   3  MS3           5.8 cM      6.4 cM      5.5 cM     5.5 %%    35.1
  3   4  MS4           3.6 cM     10.1 cM      3.5 cM     3.5 %%    21.1
  4   5  MS5           3.7 cM     13.8 cM      3.6 cM     3.6 %%    15.4
  5   6  MS6           1.6 cM     15.4 cM      1.6 cM     1.6 %%    17.3
  6   7  MS7           6.7 cM     22.1 cM      6.3 cM     6.3 %%    35.1
  7   8  MS8           2.5 cM     24.6 cM      2.5 cM     2.5 %%     7.4
  8  10 MS10           0.0 cM     24.6 cM      0.0 cM     0.0 %%     7.2
  9   9  MS9        ----------              ----------
                     24.6 cM                  23.6 cM


        9 markers, log10-likelihood =  -135.41
                   log-e-likelihood =  -311.80
Total CPU Time (secondes): 0.01
Number of calls to EM: 28
# of maps within -136.410765: 1
Current best LogLike is -135.410765 and worst is -159.395426
Reliability Threshold = 1e+100
Tolerance Threshold1 = 0.1
Tolerance Threshold2 = 0.001
{1 0.010000 1 0.000000 0.010000 1 28}
```

See also:

- `cgrestart` (2.7.5)
- `heaprint` (2.4.4)

2.7.5 `cgrestart`

Reinitializes CarthaGene.

Synopsis: The `cgrestart` command is invoked either as:

- `cgrestart` *Options*
- `cgrestart`

Description: The `cgrestart` command completely reinitialises the software. All loaded data-sets, maps. . . are erased from memory and a new session can be started.

Arguments :

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.

Returns: nothing

Example:

```
# we load a dataset
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
# change our mind and zap everything
CG> cgrestart

# no dataset is loaded now
CG> dsinfo

Data Sets :
----------:
ID        Data Type    markers individuals             filename constraints...

CG>
```

See also:

- `cgstat` (2.7.4)

2.7.6 `quietset`

Sets the conciseness of the output of several commands.

Synopsis: The `quietset` command is invoked either as:

- `quietset` *Options*
- `quietset` *level*

Description:

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *level*: the level of quietness (0 or 1).

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> sem

Map -1 : log10-likelihood =  -297.78
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS3 MS4 MS5 MS6 MS7 MS8 MS9 MS10 MS11 MS12 MS13 MS14 MS15 MS...

CG> quietset 1

CG> sem
```

See also:

- `verbset` (2.7.7)

2.7.7 `verbset`

Sets the verbosity of the output of several commands.

Synopsis: The `verbset` command is invoked either as:

- `verbset` *Options*
- `verbset` *level*

Description:

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *level*: the level of verbosity (0, 1 or 2).

Returns: nothing.

Example:

```
CG> dsload Data/bc.cg
{1 f2 backcross 20 208 /homes/thomas/CartaGene/dev/test/Data/bc.cg}
CG> sem

Map -1 : log10-likelihood =  -297.78
-------:
 Set : Marker List ...
   1 : MS1 MS2 MS3 MS4 MS5 MS6 MS7 MS8 MS9 MS10 MS11 MS12 MS13 MS14 MS15 MS...

CG> verbset 2

CG> sem

Map -1 : log10-likelihood =  -297.78, log-e-likelihood =  -685.66
-------:

Data Set Number  1 :

      Markers        Distance    Cumulative  Distance   Theta       2pt
Pos  Id name         Haldane     Haldane     Kosambi    (%%age)      LOD

  1   1  MS1          19.5 cM     19.5 cM     16.8 cM    16.2 %%    15.7
  2   2  MS2           0.6 cM     20.1 cM      0.6 cM     0.6 %%    47.6
  3   3  MS3           5.8 cM     25.9 cM      5.5 cM     5.5 %%    35.1
  4   4  MS4           3.7 cM     29.6 cM      3.6 cM     3.5 %%    21.1
  5   5  MS5           3.7 cM     33.3 cM      3.6 cM     3.6 %%    15.4
  6   6  MS6           1.6 cM     34.9 cM      1.6 cM     1.6 %%    17.3
  7   7  MS7           6.7 cM     41.6 cM      6.3 cM     6.3 %%    35.1
  8   8  MS8           2.2 cM     43.8 cM      2.2 cM     2.2 %%    40.8
  9   9  MS9           3.0 cM     46.8 cM      2.9 cM     2.9 %%     7.2
 10  10 MS10           5.4 cM     52.2 cM      5.1 cM     5.1 %%     3.6
 11  11 MS11           8.6 cM     60.7 cM      7.9 cM     7.9 %%    10.9
 12  12 MS12           0.0 cM     60.7 cM      0.0 cM     0.0 %%    18.7
 13  13 MS13           0.0 cM     60.7 cM      0.0 cM     0.0 %%     6.0
 14  14 MS14           5.8 cM     66.6 cM      5.5 cM     5.5 %%     0.0
 15  15 MS15           4.6 cM     71.1 cM      4.4 cM     4.4 %%    26.6
 16  16 MS16          26.1 cM     97.2 cM     21.6 cM    20.3 %%     6.3
 17  17 MS17           0.0 cM     97.2 cM      0.0 cM     0.0 %%    16.0
 18  18 MS18           4.9 cM    102.1 cM      4.7 cM     4.6 %%    27.1
 19  19 MS19           3.0 cM    105.1 cM      2.9 cM     2.9 %%    34.8
 20  20 MS20        ----------              ----------
                     105.1 cM                 95.0 cM


       20 markers, log10-likelihood =  -297.78
                   log-e-likelihood =  -685.66
```

See also:

- `quietset` (2.7.6)

2.7.8 `cgsave`

Saves the state of the current session in a file that can be later used to restore the current state (data sets loaded, markers merged, heap state). This is different from `cgout` which allows to save the output of the commands issued as they are executed.

Synopsis: The `cgsave` command is invoked either as:

- `cgsave` *Options*
- `cgsave` *FilePath*

Description: The `cgsave` command enable to save a CarthaGene session state to a Tcl script. The state of the session is defined by all the data sets including merged datasets, the loci selected, the merged loci, and all the map stored in the heap. By running the tcl script all the entities mentioned before will be regenerated. To restore a session, type the command `source` followed by the name of the file.

Be careful, only the path to the the data sets file are kept by the script file. They may be relatives, and your script will depend on the working location. Nevertheless, the script is editable.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *FilePath*: the path of the file to which you want CarthaGene to save the session.

Example:

```
...
CG> cgsave secur0.tcl
The file secure0.tcl already exist, replacing it? ([y] or n):y
Your session has been successfully saved.
To restore it, type : source secure0.tcl
CG> cgrestart
CG> source secur0.tcl
...
```

See also:

-
- `cgrestart` (2.7.5)
-
- `cgout` (2.7.1)

2.7.9 `cgexport`

Synopsis: The `cgexport` command is invoked either as:

- `cgexport` *Options*
- `cgexport` *fileName mapName chromosomeName*

Description: The `cgexport` command saves the current best map of the heap to a file. If the chosen file already exist, the command will (after asking for authorizationj) append the current map to the end of the existing file. Two output formats are simultaneously created: a MapMaker format is used to output maps in the file whose *fileName* is given while an XML format file (designed to interact with existing BioMercator and MCQTL software, see http://www.genoplante.com) is generated in a file with the *fileName* extended with `.xml`.

Arguments:

- *Options* : `-u` to obtain the synopsys of the normal use, `-h` to print a one line description, `-H` to print a short help.
- *fileName*: the path of the file to which you want to save the map.
- *mapName*: any identifier you want to associate with the map produced.
- *chromosomeName*: any identifier you want to associate with the chromosome mapped.

Example:

```
CG> dsload Data/bc1.cg
{1 f2 backcross 17 208 /homes/thomas/CartaGene/dev/test/Data/bc1.cg}
CG> sem

Map -1 : log10-likelihood =  -485.24
-------:
 Set : Marker List ...
   1 : MS4 MS5 MS13 MS6 MS11 MS17 MS16 MS8 MS7 MS2 MS3 MS9 MS15 MS12 MS20 MS1\
9 MS1

CG> cgexport StupidMap MyMap MyChromosome
export successful!

CG>

CG>
```

See also:

-
- `cgrestart` (2.7.8)
-
- `cgout` (2.7.1)

Bibliography

[BDCP00] Amir Ben-Dor, Benny Chor, and Dan Pelleg. RHO – radiation hybrid ordering. *Genome Research*, 10:365–378, 2000.

[CLR90] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to algorithms*. MIT Press, 1990. ISBN : 0-262-03141-8.

[DLR77] A. P. Dempster, N. M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *J. R. Statistic. Soc. Ser. B*, 39:1–38, 1977.

[Glo89] F. Glover. Tabu search – part I. *ORSA Journal on Computing*, 1(3):190–206, Summer 1989.

[Glo90] F. Glover. Tabu search – part II. *ORSA Journal on Computing*, 2(1):4–31, Winter 1990.

[Gol89] D. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, 1989.

[Gre88] P. Green. Rapid construction of multilocus genetic linkage maps. i. maximum likelihood estimation. draft manuscript, 1988.

[Hel00] K. Helsgaun. An effective implementation of the lin-kernighan traveling sal esman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000. http://www.dat.ruc.dk/~keld/research/LKH.

[LA84] P. Van Laarhoven and E. Aarts. *Simulated Annealing : Theory and Applications*. D.Reidel Publishing Company, 1984.

[LBLC95] Kathryn L. Lunetta, Michael Boehnke, Kenneth Lange, and David R. Cox. Experimental design and error detection for polyploid radiation hybrid mapping. *Genome Research*, 5:151–163, 1995.

[LGA$^+$87] E.S. Lander, P. Green, J. Abrahamson, A. Barlow, M. J. Daly, S. E. Lincoln, and L. Newburg. MAPMAKER: An interactive computer package for constructing primary genetic linkage maps of experimental and natural populations. *Genomics*, 1:174–181, 1987.

[LK73] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman pr oblem. *Oper. Res.*, 21:498–516, 1973.

[SCBM01] T. Schiex, P. Chabrier, M. Bouchez, and D. Milan. Boosting em for radiation hybrid and genetic mapping. In B. Moret, editor, *Proc. of the first Workshop on Algorithms in Bioinformatics*, volume 2149 of *LNCS*, pages 41–51. Springer Verlag, 2001.

Index