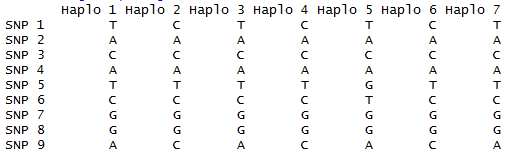
**Guidelines to HaploBlocker 1.4.3**

**HaploBlocker** is an R-package to compute a haplotype block library according to our paper “HaploBlocker: Creation of subgroup specific haplotype blocks and libraries”. The publication is currently available on *biorvix* (https://www.biorxiv.org/content/early/2018/06/19/339788) and submitted to *Genetics*. In the following we will give some short guidelines on how to use the package and what parameters to change according to what one is interested in.

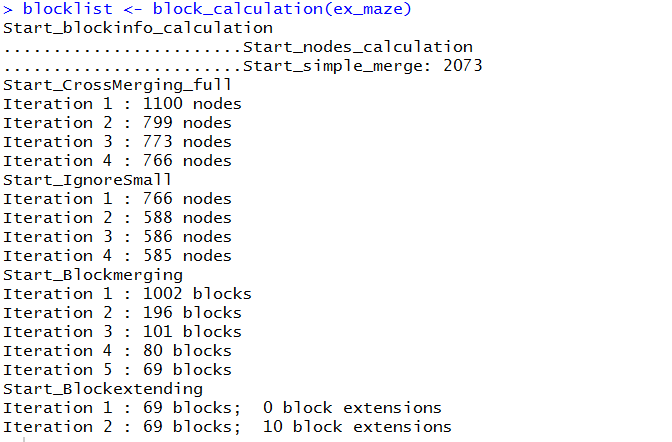
**Table of Content**

1. **General**
2. **Installation**
3. **Parameters of the main function**
4. **Example Inputs**
5. **Outputs**
6. **Data Availability**
7. **Functions for later analysis**
8. **General**

The main function of the package is **block\_calculation** and the only mandatory input of the function is a dataset (parameter: **dhm**) containing haplotypes – input can be up to 256 different characters/numeric/integer but for ideal performance use as little as possible (0L,1L):

:

When running, the user is receives updates regarding the currently performed step of the algorithm. Here we used the test dataset **ex\_maze** which is included in the package:



1. **Installation**

HaploBlocker requires R 3.0+ (and the included graphics and stats package) as well as the R-package RandomFieldsUtils (version 0.4.0), note that this is a newer version than what is available on CRAN and is available at <https://github.com/tpook92/HaploBlocker>.

To install the package we recommend the usage of the R function install.packages (under windows set type=”source”, repo = NULL) or use the .zip version. For Windows the installation of Rtools is required. Some machines additionally require devtools.

1. **Parameters of the main function**

In the following we will discuss the parameters to change the structure of the output according to the step they are occurring in the algorithm. For example inputs we refer to the section afterwards. As default settings are chosen to work for most dataset but still perform fast we recommend the usage of our adaptive mode (**adaptive\_mode**=TRUE) when wanting to create a haplotype library without majorly modifying parameters and no computationally intensive datasets.

**Step 0 – Prefilters:**

Parameters: prefilter, maf, equal\_remove

Before the actual algorithm is performed one can remove non-informative SNPs of the dataset. Those are SNP with a low frequency (minimum minor-allele-frequency) with the parameter **maf** and all SNPs which are the same as the previous one with the parameter **equal\_remove**. On default setting no filtering is done and it has to be activated via the parameter **prefilter**.

**Step 1 – Cluster-building:**

Parameters: window\_sequence, window\_size, merging\_error, max\_groups, bp\_map, window\_anchor\_gens, blockinfo\_mode, at\_least\_one, multi\_window\_mode, blockinfo\_mode\_na, na\_snp\_weight, na\_seq\_weight, actual\_snp\_weight

On default settings the windows of the dataset are of equal length (**window\_size**) and in every window the same number of errors (**merging\_error**) is allowed. Those can be changed flexible. If one wants a different structure one can manually enter all windows of the dataset via the parameter **window\_sequence**. To include the position in base pairs one has to enter the position of each SNP via the parameter **bp\_map**.

Since the manual input can be tiring we offer an additional possibility to generate a **window\_sequence**. The parameter **max\_groups** can be used to set the window boundaries to make sure there is a maximum number of variants in each window (Next window starts whenever the previous block would have more variants than **max\_groups**).

When the physical position of each gene is known one can use these boundaries to create the **window\_sequence** via the parameter **window\_anchor\_gens** (currently only non-overlapping gens!).

To minimize the number of groups in each window one can use the parameter **blockinfo\_mode** (on default the groups are formed according to the most common haplotypes in the window). **At\_least\_one** is an utility parameter to make sure that in each window at least one SNP has to be the same (only relevant for **window\_size ≤** **merging\_error**)

To use multiple window clusters in the fitting procedure set **multi\_window\_mode** to TRUE. Now you can use vectors as input for **window\_size**, **merging\_error** and **min\_share** or use a list of multiple window sequences as an input for **window\_sequence**.

In case the dataset contains missing values, those on default will be modelled as another allelic variant (“9”) in the analysis. To count differences between NAs and allelic variants with different weighting activate **block\_mode\_na**. The difference between NA and an allelic variant is counted as **na\_snp\_weight** merging errors whereas different allelic variants are counted as **actual\_snp\_weight** merging errors. In case a marker contains only one allelic variant and NAs differences are counted as **na\_seq\_weight** merging errors. It has to be noted there that this mode is significantly more time consuming and still open to some change. Since the required input of our method is still haplotypes NA are usually lost in the phasing process.

**Step 2 – Cluster-merging**

Parameters: node\_min, gap

The only two changeable parameter in Step 2 are the minimum size of the nodes in the window cluster (**node\_min**) and the minimum number of windows between two removed nodes for a segment to be included in the cluster (**gap**).

**Step 3 – Block-identification:**

Parameters: min\_share, subgroups, consider\_nodes, consider\_edge, min\_per\_subgroup, consider\_multi, multi\_min, node\_min, edge\_min

To not consider nodes or edges in the identification step one can set the **consider\_nodes**, **consider\_edge** to FALSE. Both those changes are not recommended (setting one to FALSE will decrease computation time). To additionally screen for blocks based on haplotypes in two adjacent edges use **consider\_multi** (only recommened for small dataset & use **multi\_window\_mode** first). To change the minimum number of haplotypes per block one can use **edge\_min** (Blocks by Edge), **node\_min** (Blocks by node) and **multi\_min** (multiple edges).

To change the minimum proportion of a block transitioning in the same node needed to extend the block one can use the parameter **min\_share**. By this one can control the average length of each block and the similarity between haplotypes from a block. A higher value leads to shorter blocks and thereby leads to a higher similarity between the haplotypes in a block and a higher number of blocks overall. Additional the number of overlapping blocks is heavily reduced.

To form blocks not only for the whole dataset but also for subgroups one has to use the parameter **subgroups** and set the minimum number haplotypes of each subgroup to be in each block (**min\_per\_subgroup**). A change in this parameter leads to blocks which are in all subgroups of the dataset and therefore can lead to low coverages. A change here is only recommended when one is explicitly interested in the overlapping regions in multiple subgroups.

**Step 4 – Block-filtering:**

Parameters: min\_majorblock, min\_majorblock\_steps, min\_similarity, save\_allblock, consider\_all, merge\_closeblock, max\_diff\_i, max\_diff\_l, off\_lines, weighting\_length, weighting\_size

The main filtering process is done by identifying the number of positions in which each block is the major block of the dataset. This number can be changed via **min\_majorblock** and should be used to find a balance between the number of blocks and the coverage of the block library. To obtain a haplotype library with a specific coverage we recommend the use of the parameter **target\_coverage** to initialize an automatic fitting procedure to determine a good choice for **min\_majorblock**. To control the number of iterations done to fit **min\_majorblock** in **target\_coverage** use **max\_iteration** with **min\_step\_size** controlling the minimal difference in **min\_majorblock** per step and **target\_stop** providing a maximum difference to the target.

To control which block is the major block in each position one can control the weighting between the length and number of haplotypes in each block by using the parameters **weighting\_length** and **weighting\_size**.

To avoid excluding important blocks the minimum number is increased slowly (in **min\_majorblock\_steps** linear steps). The minimum similarity of a haplotype with a block to be included can be set by the parameter **min\_similarity**. By this one can control the minimum similarity between two haplotypes of the same block. Haplotypes not fulfilling **min\_similarity** but being in all node used to identify the block are not removed unless the parameter **save\_allblock** is set to FALSE.

Additionally there are some minor parameters in the filtering process. To not consider haplotypes which are not in the block original one has to set **consider\_all** to FALSE. To allow blocks with similar haplotypes and location to be merged one has to activate **merge\_closeblock** and set the maximum differences between them via **max\_diff\_i** (different haplotypes) and **max\_diff\_l** (differences between both). The minimum number of additional haplotypes a block has to have compared to another block when the sequence of windows is the same can be set via **off\_lines**.

**Step 5 – Block-extending:**   
Parameters: block\_extending, max\_extending\_diff, extending\_ratio, snp\_extending, max\_extending\_diff\_snp, extending\_ratio\_snp

If one does not want the block and SNP extension to be performed set **block\_extending** and/or **snp\_extending** to FALSE. If one wants to use it one can control the maximum number of windows that are different in some haplotypes (**max\_extending\_diff**, **max\_extending\_diff\_snp**) and ratio between windows with and without variation (**extending\_ratio, extending\_ratio\_snp**).

**Step 6 - Off-variant-identification (optional)**

This step is not included in the manuscript as its application is only recommend to obtain an absolute maximum coverage for a dataset. Here, in addition to the window cluster additional blocks are generated based on those positions not included in the block library before.

Parameters: off\_node\_addition, raster, off\_node\_minimum\_blocklength, off\_node\_minimum\_size

This step is only performed when **off\_node\_addition** is set to TRUE. In the actual step each position is screen for section of **off\_node\_minimum\_size** haplotypes with the same sequence in **off\_node\_minimum\_blocklength** windows. Afterward all other steps are peformed again (especially filtering for **min\_majorblock**.

To save computation time not every window is consider, but instead only each raster window (this should still be a value below **off\_node\_minimum\_blocklength**).

**No-Step: Performance parameters – computation time:**

Parameters: recoding, recoding\_notneeded, fast\_compiler, intersect\_func, c\_dhm\_mode

To reduce the computation time there are some additional possible options. Internal computations are faster when a low number of different characters is use. To change the coding to major\_variant “A”, minor\_variant “C” in every SNP set the parameter **recoding** to TRUE. If this is already done you can further use **recoding\_notneeded** to skip this recoding step and still profit from the advantages of the recoding.

All other options are there mostly because of testing purposes and should already be set to the optimal value. **Fast\_compiler** enables the compiler-packages and just-in-time computing. **Intersect\_func** loads in a more efficient variant of base::intersect and **c\_dhm\_mode** uses bit-wise-computing of the dataset.

1. **Example inputs**

|  |  |  |
| --- | --- | --- |
| Parameter-name | Default | Other option: |
| prefilter | FALSE | TRUE |
| maf | 0.00 | Value between 0 and 0.5 |
| equal\_remove | FALSE | TRUE |
| window\_sequence | NULL (automatic generated) |  |
| window\_size | 20 | Natural number (1,2,3,…) |
| merging\_error | 1 | Natural number (1,2,3,…) - lower than window\_size! |
| max\_groups | 0 | to active: Natural number >= 2 |
| bp\_map | NULL |  |
| window\_anchor\_gens | NULL |  |
| blockinfo\_mode | 0 | 1 to minimize groups per window |
| at\_least\_one | TRUE | FALSE |
| multi\_window\_mode | FALSE | TRUE (use e.g. window\_size=c(5,10,20,50)) |
| blockinfo\_mode\_na | FALSE | TRUE (adjust merging\_error ! ) |
| na\_snp\_weight | 2 | Numeric value >0 |
| na\_seq\_weight | 0 | Numeric value > 0 |
| actual\_snp\_weight | 5 | Numeric value > 0 |
| gap | 10 | Natural number (1,2,3,…) |
| min\_share | 0.975 | Value between 0.5 and 1 (highly recommend to not use small values! |
| node\_min | 5 | Natural number (1,2,3,…) |
| edge\_min | 5 | Natural number (1,2,3,…) |
| multi\_min | 5 | Natural number (1,2,3,…) |
| consider\_nodes | TRUE | FALSE |
| consider\_edge | TRUE | FALSE |
| consider\_multi | FALSE | TRUE |
| subgroups | NULL (automatic generated) | List(1:500, 1:200, 1:300)  Subpopulation 1 in first 200 colums  Subpopulation 2 in last 300 colums |
| min\_per\_subgroup | 0 | Natural number (1,2,3,…)  Only when one is explicitly interested in the overlap between both populations! |
| min\_majorblock | 5’000 | Non-negativ-number (0,1,2,…) |
| min\_majorblock\_steps | 4 | Non-negativ-number (0,1,2,…) |
| min\_similarity | 0.99 | Value between 0 and 1 (highly recommend to not use values below 0.9! |
| save\_allblock | TRUE | FALSE |
| consider\_all | TRUE | FALSE |
| merge\_closeblock | FALSE | TRUE |
| max\_diff\_i | 1 | Non-negative-number (0,2,3,…) |
| max\_diff\_l | 1 | Non-negative-number (0,2,3,…) |
| off\_lines | 5 | Natural number (1,2,3,…) |
| Weighting\_length | 1 | Numeric value (<0 not recommended) |
| Weighting\_size | 1 | Numeric value (<0 not recommended) |
| block\_extending | TRUE | FALSE |
| snp\_extending | TRUE | FALSE |
| max\_extending\_diff | 1 | Non-negative-number (0,2,3,…) |
| max\_extending\_diff\_snp | 0 | Non-negative-number (1,2,3,…) |
| extending\_ratio | 20 | Natural number (1,2,3,…) Avoid low values |
| extending\_ratio\_snp | Inf | Naturual number (1,2,3,…) Only change for long windows and high number of haplotypes in blocks |
| off\_node\_addition | FALSE | TRUE |
| raster | 5 | Natural number (1,2,3,…) |
| recoding | FALSE | TRUE |
| recoding\_notneeded | FALSE | TRUE |
| fast\_compiler | TRUE | FALSE |
| intersect\_func | 2 (efficient C-version) | 0 (base::intersect), 1 ( semi-efficient R-version) |
| c\_dhm\_mode | TRUE | FALSE |
| big\_output | FALSE | TRUE |
| target\_coverage | NULL | Value between 0 and 1 |
| max\_iteration | 10 | Natural number (1,2,3,…) |
| min\_step\_size | 25 | Natural number (1,2,3,…) |
| target\_stop | 0.001 | Value between 0 and 1 (recommend close to 0) |
| multi\_window\_mode | FALSE | TRUE;  Can be actived by using a vector for window\_size; merging\_error or min\_share |
| adaptive\_mode | FALSE | TRUE;  Sets window\_size = c(5,10,20,50) and  Target\_coverage = 0.9 |
| developer\_mode | FALSE | TRUE |

1. **Output**

The Output of the function is a list containing a block in each element. For each block the following information are stored:

1. Sequence of nodes in the window cluster
2. Start of the block (in windows, SNPs and bp)
3. End of the block (in windows, SNPs and bp)
4. Sequence of the group in each window
5. Number of haplotypes in the block
6. List of Haplotypes in the block
7. 1. Sequence of alleles in the block (major variant)
8. 2. Frequency of the major variant
9. – 12. Internal stuff to save computation time in the algorithm (Only in the output using developer-mode)

To not only get the haplotype block library but additionally the window-dataset, the window-cluster and general information on each window get the parameter **big\_output** to TRUE.

1. **Data Availability**

Our R-package currently included an exemplary dataset containing the first 9’999 markers and 313 individuals of chromosome 1 in maize. A full dataset containing 80’200 markers for 910 individuals is provided with the publication and is also included in our GitHub repository (https://github.com/tpook92/HaploBlocker). All results presented in the publication are limited to chromosome 1. Datasets for other chromosomes will be made available with publication of other project partners and hopefully then included in the package. In the package itself a dataset of the first 9’999 SNPs of 313 KE DH-lines is included (**ex\_maze**).

1. **Functions for later analysis**

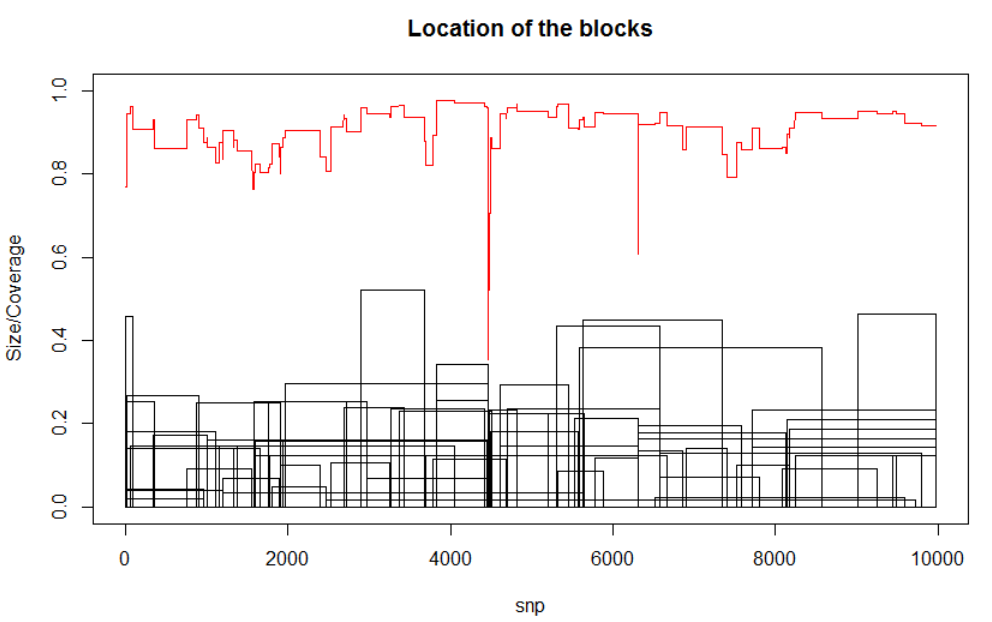
Since only limited later analysis was done so far we have only programmed some smaller functions to assess the relevant parts of the output and generate basic plots to get an overview of the structure of the blocks.

We are always happy for feedback on additional wishes for possible outputs or other options to include in our algorithm.

***block\_plot(blocklist, indi=NULL, type=”snp”, bw=1)***

Plot of each blocks position (x-axis length, y-axis number of haplotypes in block). The red lines indicated the coverage of the full block library per position.

Parameter **indi** is automatically calculated – for big datasets computation time can be saved by setting the parameter as it coded the number of haplotypes in the sample. Type can be set to “window”, “snp” or “bp” depending on the wanted scaling. The usage of “window” is not recommended when using multiple window sizes. To smooth the coverage function change the bandwidth in the smoothing (default: no smoothing) with the parameter **bw**.



***block\_ehh(blocklist, data=NULL, marker, plot=FALSE, position1=NULL, standardization=3, group=NULL, return\_ehh=TRUE)***

Function to derive bEHH scores for a given **blocklist**. In case no blocklist is provided a SNP-dataset can be provided in **data**. bEHH is computed the marker given in **marker**. To plot the bEHH curve set **plot** to TRUE. On default, distance between markers is assumed to be equidistant. Position of markers can be given in **position1**.

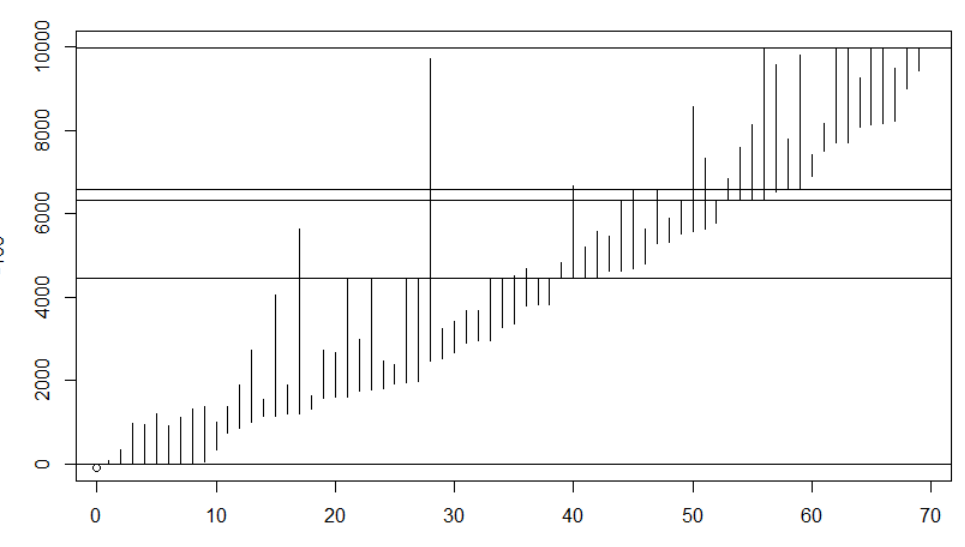
Change of **standardization** is not recommended. To compute bEHH scores for different subgroups use **group**. To instead of bEHH return iHH set **return\_ehh** to FALSE.

***block\_ihh(blocklist, data=NULL, plot=FALSE, position1=NULL, standardization=3, group=NULL, return\_ehh=TRUE)***

Compute iHH scores for the whole genome – same use as *block\_ehh().*

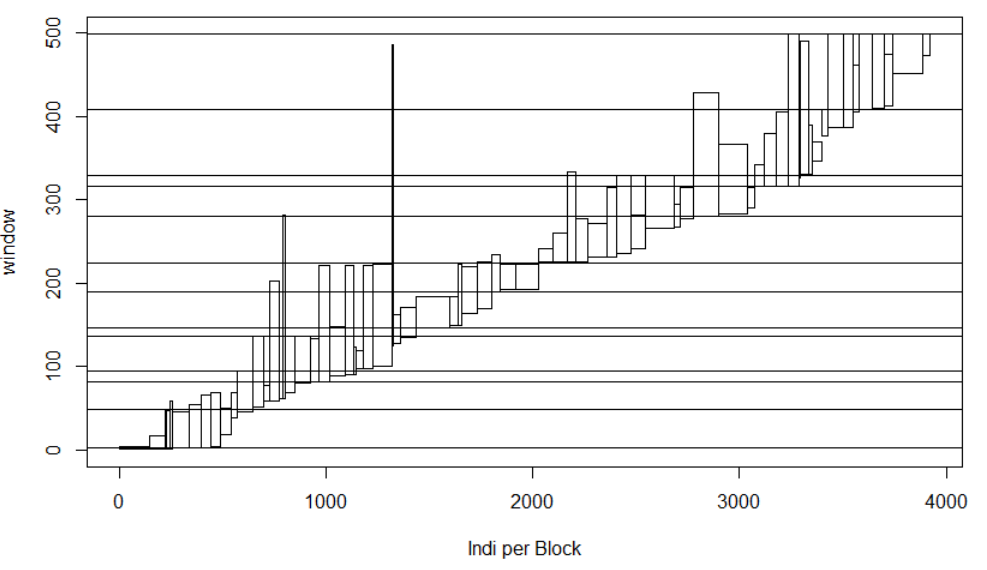
***blocklist\_plot(blocklist, cutoff2=5, bound\_weighted=TRUE, type=”snp”)***

Location of the blocks is according to the y-axis. Additionally recombination hotspots are indicated by horizontal lines. **Cutoff2** (for the example 3 was used) is the minimum number of blocks to end to mark a position as a hotspot and **bound\_weighted** scales blocks according to the size of the block. We not only count the position itself but adjacent markers via a kernel regression method.



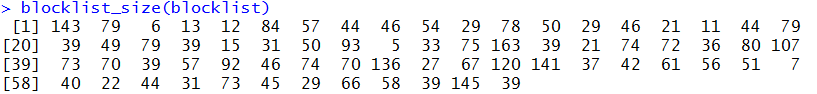
***blocklist\_plot\_xsize(blocklist, cutoff2=5, bound\_weighted=TRUE, type=”snp”)***

Plot of the blocks with width according to the number of haplotypes in the respective block. Location according to the y-axis. Additionally recombination hotspots are indicated by horizontal lines. **Cutoff2** is the minimum number of blocks to end to mark a position as a hotspot and **bound\_weighted** scales blocks according to the size of the block.



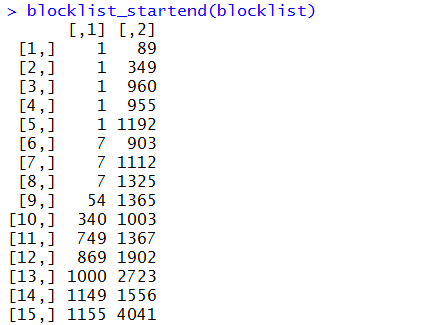
***blocklist\_size(blocklist, intersect\_func=intersect,first\_block=1)***

Calculate the number of haplotypes in each block – other parameters are only needed for internal use in the block\_calculation function



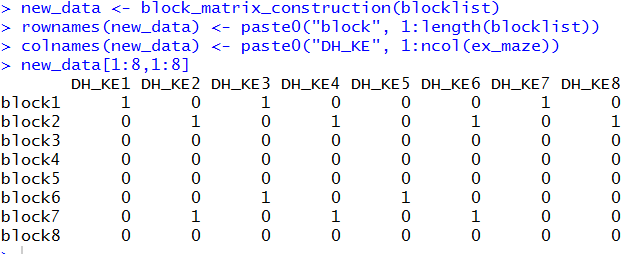
***blocklist\_startend(blocklist, type=”snp”, first\_block=1)***

Calculate the start and end point of each block. Select the **type** (“window”, “snp”, “bp”) accordingly.



***block\_matrix\_construction(blocklist, indi=NULL)***

Calculate a block-dataset according to the block library



**Block\_windowdataset(blocklist=NULL, data=NULL, consider\_nonblock=FALSE, return\_dataset=FALSE)**

Generate a window based block dataset. Blocks span over the same window for better comparability to other block based approaches. Overall windows are much shorter than HaploBlocker blocks. Set **consider\_nonblock** to TRUE to haplotypes in no haplotype block in HaploBlocker to be in blocks. Set **return\_dataset** to TRUE to instead of a dataset coding presence/absence allow for more variants in each window.

**plot\_block(blocklist, type=”snp”, orientation=”snp”, include=TRUE, indi=NULL, min\_to\_plot=5, intensity=0.5, add\_sort=TRUE, max\_step=500, snp\_ori=NULL, export\_order=FALSE, import\_order=FALSE)**

Generate a graphical representation of the blocklist. Use type to select scaling of the x-axis (“bp”, “snp”, “window”). To sort haplotypes use the parameter **orientation** – To align against blocks in set it to “front”, “mid” or “back”. We recommend to align against a location in SNP. On default the middle of the dataset is used but can be manually set using **snp\_ori**. For ordering haplotypes only the adjacent **max\_step** blocks are considered – 500 should be enough for all applications. Instead of using our sorting algorithm one can import the order of haplotypes using **import\_order** (or export using **export\_order**=TRUE).

Only those blocks are displayed with at least **min\_to\_plot** haplotypes in it. To show overlap blocks are shown with a low color **intensity**.

