CiBreed
Center for Integrated Breeding Research

HAPLOBLOCKER

Torsten Pook, Martin Schlather

19. MÄRZ 2019

UNIVERSITY OF GOETTINGEN
Animal Breeding and Genetics Group

Table of Contents

1 Preface

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.

2 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):

	Haplo 1	Haplo 2	Haplo 3	Haplo 4	Haplo 5	Haplo 6	Haplo 7
SNP 1	T	C	T	C	T	C	T
SNP 2	A	A	A	A	A	A	A
SNP 3	C	C	C	C	C	C	C
SNP 4	A	A	A	A	A	A	A
SNP 5	T	T	T	T	G	T	T
SNP 6	C	C	C	C	T	C	C
SNP 7	G	G	G	G	G	G	G
SNP 8	G	G	G	G	G	G	G
SNP 9	A	C	A	C	A	C	A

:

Figure 1: Excerpt of the dataset ex_maze

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:

```
> blocklist <- block_calculation(ex_maze)
Start_blockinfo_calculation
.....................Start_nodes_calculation
.....................Start_simple_merge: 2073
Start_CrossMerging_full
Iteration 1 : 1100 nodes
Iteration 2 : 799 nodes
Iteration 3 : 773 nodes
Iteration 4 : 766 nodes
Start_IgnoreSmall
Iteration 1 : 766 nodes
Iteration 2 : 588 nodes
Iteration 3 : 586 nodes
Iteration 4 : 585 nodes
Start_Blockmerging
Iteration 1 : 745 blocks
Iteration 2 : 436 blocks
Iteration 3 : 112 blocks
Iteration 4 : 86 blocks
Iteration 5 : 70 blocks
Start_Blockextending
Iteration 1 : 70 blocks;  0 block extensions
Iteration 2 : 70 blocks;  9 block extensions
```

Figure 2: Example usage of the function block_calculation in Haploblocker

3 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. For installation of the package we recommend the usage of the R function *install.packages* (under windows set type="source", repo=NULL). Usage was tested on Linux and Windows. The usage on Mac OS is currently not recommended.

For Windows the installation of Rtools is required. Some machines additionally require devtools.

4 Citation

There is currently peer-review version of the paper. This will hopefully change soon. For so long we suggest using following to citations for the preprint on biorvix and the R-packages HaploBlocker:

```
@article{Pook.2018,
 author = {Pook, Torsten and Schlather, Martin and {de los Campos}, Gustavo
and Schoen, Chris Carolin and Simianer, Henner},
 year = {2018},
 title = {HaploBlocker: Creation of subgroup specific haplotype blocks and
libraries},
 pages = {339788},
 journal = {bioRxiv}
}

@misc{Pook.2018b,
 author = {Pook, Torsten and Schlather, Martin},
 year = {2018},
 title = {HaploBlocker: An R package for the Creation of Haplotype
Libraries for DHs and Highly Inbreed Lines},
 url = {https://github.com/tpook92/HaploBlocker}
}
```

5 Parameters of the main function (block_calculation)

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.

5.1 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**.

5.2 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.

5.3 Step 2 – cluster-merging

Parameters: node_min, gap, min_reduction_cross, min_reduction_neglet, early_remove, node_min_early

In the cluster-merging-step the number of haplotypes per node can be controlled via **node_min**. To avoid short segments between removed nodes all haplotypes in less than **gap** windows are also removed from the cluster.

To reduce computation time in the SG,SM and NN,SG,SM, SG cycles one can use **min_reduction_cross** and **min_reduction_neglet** to stop the cycles when there are less than that many changes to the window cluster. In case there is a high number of nodes with few haplotypes in it, one can consider removing them before the SG, SM cycle via **early_remove** and **node_min_early**.

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**).

5.4 Step 3 – block-identification

Parameters: min_share, subgroups, consider_nodes, consider_edge, min_per_subgroup, consider_multi, multi_min, node_min, edge_min, double_share

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.

To consider both the long and the short segments in the block-identification (cf. extended-block-identification) set **double_share** to minimum share of the haplotypes that need to transition in the same longer segment.

5.5 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, target_coverage, target_stop

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**.

5.6 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**).

5.7 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**).

5.8 No-Step: performance parameters – computation time:

Parameters: recoding, recoding_notneeded, fast_compiler, intersect_func, c_dhm_mode, parallel_window, window_overlap, window_cores

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.

To further increase computation time one can active parallel computing via **parallel_window**. Here the dataset is split into windows containings **parallel_window** markers. On defaults window do not overlap but can via **window_overlap**. The number of cores to be used is controlled via **window_cores**.

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.

6 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_sequence[1:5,]`
		`Start-SNP End-SNP Length Min-Same Start-BP End-BP`
		`[1,] 1 20 20 19 0 0`
		`[2,] 21 40 20 19 0 0`
		`[3,] 41 60 20 19 0 0`
		`[4,] 61 80 20 19 0 0`
		`[5,] 81 100 20 19 0 0`

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	`base_pair` `[1] 48208 49308 49527 50846 51053 52778`

window_anchor_gens	NULL	`BP-start BP-end`
		`GRMZM2G354611 66347 68582`
		`GRMZM2G100965 169103 170546`
		`GRMZM5G833275 176866 177244`
		`GRMZM2G100979 183185 183884`
		`GRMZM2G310569 311374 318659`
		`GRMZM2G341658 563756 566860`
		`GRMZM2G039325 567358 580418`
		`GRMZM2G415022 706080 707213`
		`AC195959.2_FG004 709107 712584`
		`GRMZM2G307823 725998 729956`

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	TRUE	FALSE (base::intersect), usage of HaploBlocker::intersect requires that vector in an ascending sequence of numbers

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
parallel_window	Inf	Natural number – bigger than the biggest blocks one wants to identify
window_overlap	0	Natural number – nothing bigger than the size of the largest block is needed
window_cores	1	Natural number (2,3,4,…)
double_share	1	Value between 0 and 1 (nothing below 0.5 is recommended)
min_reduction_cross	-Inf	Non-negative-number (0,1,2,3,…)
min_reduction_neglet	-Inf	Non-negative-number (0,1,2,3,…)
early_remove	FALSE	TRUE
node_min_early	NULL	Natural number (1,2,3,…) – e.g. node_min / edge_min

7 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)
7. 2. Frequency of the major variant
8. – 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.

8 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**).

9 Functions for later analysis

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.

Usage of function 9.1-9.3 is recommended to get a general feeling about the structure of the haplotype library. Both 9.4 and 9.5 can be used the generated block datasets for later analysis. 9.6 and 9.7 contain function to derive bEHH and iHH scores for a haplotype library. 9.8-9.10 are old utility functions to get a generate feeling about the structure of the haplotype library.

9.1 plot_block

Parameter: 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**.

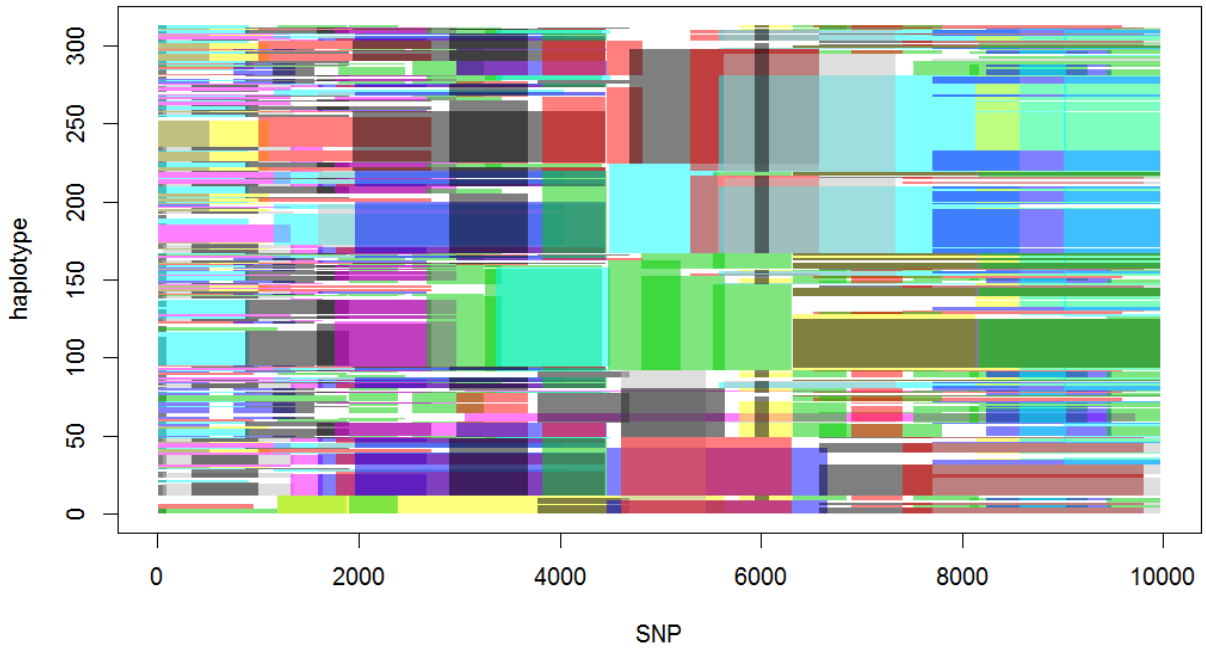

Figure 3: Exemplary output of plot_block for the ex_maze blocklist

9.2 blocklist_startend

Parameters: blocklist, type="snp", first_block=1

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

```
> blocklist_startend(blocklist)
      [,1] [,2]
 [1,]    1   89
 [2,]    1  955
 [3,]    1 1192
 [4,]    7  521
 [5,]    7  903
 [6,]    7 1325
 [7,]    7 1112
 [8,]   54 1365
 [9,]  340 1003
[10,]  749 1367
[11,]  869 1902
[12,] 1000 2723
[13,] 1149 1556
[14,] 1155 4041
[15,] 1194 1883
```

Figure 4: Exemplary output of blocklist_startend for the ex_maze blocklist

9.3 coverage_test

Parameters: blocklist, indi=NULL, type="snp", max=1

Calculate with positions of the dataset are covered by any block. Set **max** to a value above 1 to display how many blocks are presented in each cell. For big datasets usage of **type** set to "window" is recommended to save computation time.

```
> t <- coverage_test(blocklist)
> mean(t)
[1] 0.9097332
```

9.4 block_matrix_construction

Parameter: blocklist

Calculate a block-dataset according to the block library

```
> new_data <- block_matrix_construction(blocklist)
> rownames(new_data) <- paste0("block", 1:length(blocklist))
> colnames(new_data) <- paste0("DH_KE", 1:ncol(ex_maze))
> new_data[1:8,1:8]
       DH_KE1 DH_KE2 DH_KE3 DH_KE4 DH_KE5 DH_KE6 DH_KE7 DH_KE8
block1      1      0      1      0      0      0      1      0
block2      0      0      0      0      0      0      0      0
block3      0      0      0      0      0      0      0      0
block4      0      1      0      1      0      1      0      1
block5      0      0      1      0      1      0      0      0
block6      0      0      0      0      0      0      0      0
block7      0      1      0      1      0      1      0      0
block8      1      0      0      0      0      0      0      0
```

Figure 5: Exemplary output of block_matrix_construction for the ex_maze blocklist

9.5 block_windowdataset

Parameter: 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.

```
> new_data <- block_windowdataset(blocklist)
> rownames(new_data) <- paste0("block", 1:nrow(new_data))
> colnames(new_data) <- paste0("DH_KE", 1:ncol(ex_maze))
> new_data[1:8,1:8]
       DH_KE1 DH_KE2 DH_KE3 DH_KE4 DH_KE5 DH_KE6 DH_KE7 DH_KE8
block1      1      0      1      0      0      0      1      0
block2      0      0      0      0      0      0      0      0
block3      1      0      1      0      1      0      1      0
block4      0      0      0      0      0      0      0      0
block5      0      1      0      1      0      1      0      1
block6      0      0      0      0      0      0      0      0
block7      1      0      1      0      1      0      1      0
block8      0      0      0      0      0      0      0      0
```

Figure 6: Exemplary output of block_matrix_construction for the ex_maze blocklist

9.6 block_ehh

Parameters: 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.

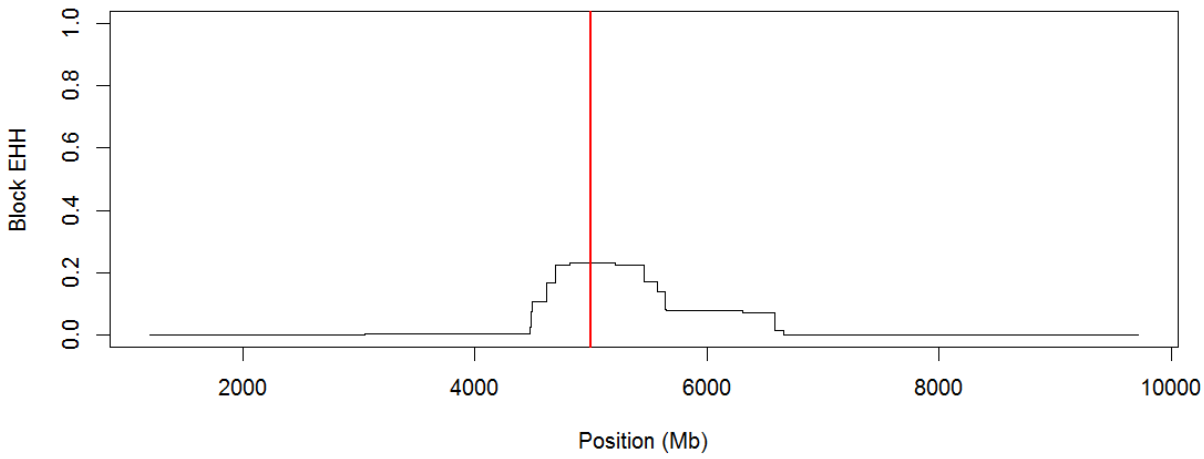

Figure 7: Exemplary output of block_ehh (marker=5000, plot=TRUE).

9.7 block_ihh

Parameters: blocklist, data=NULL, plot=FALSE, position1=NULL, standardization=3, group=NULL

Compute iHH scores for the whole genome – same use as *block_ehh.*

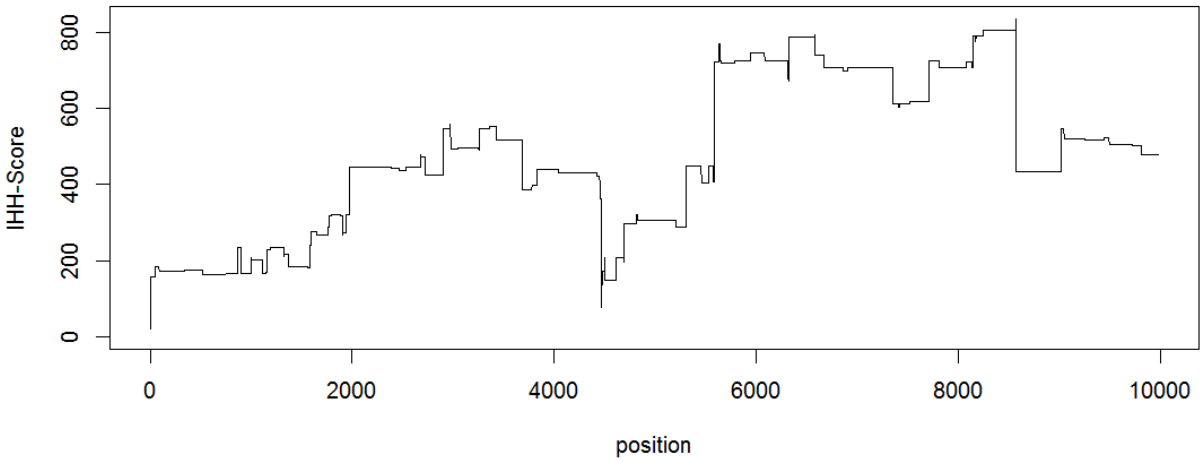

Figure 8: Exemplary output of block_ihh (plot=TRUE).

9.8 block_plot

Parameters: 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**.

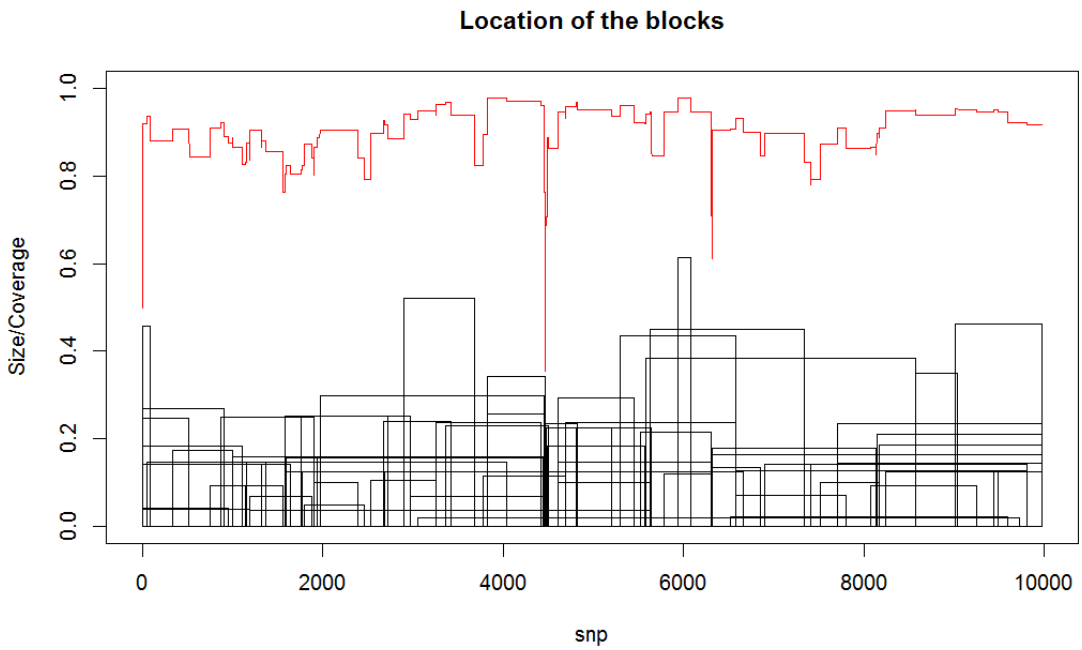

Figure 9: Exemplary output of block_plot for the ex_maze blocklist

9.9 blocklist_plot

Parameters: 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.

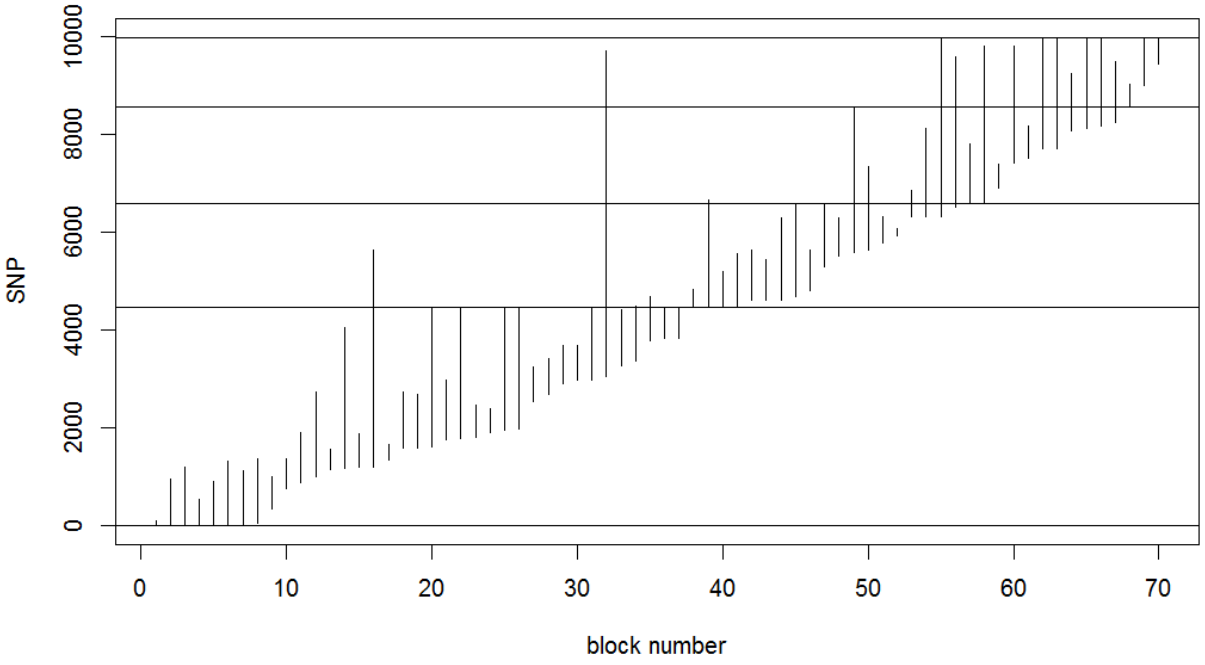

Figure 10: Exemplary output of blocklist_plot for the ex_maze blocklist (cutoff2=3)

9.10 blocklist_plot_xsize

Parameters: 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.

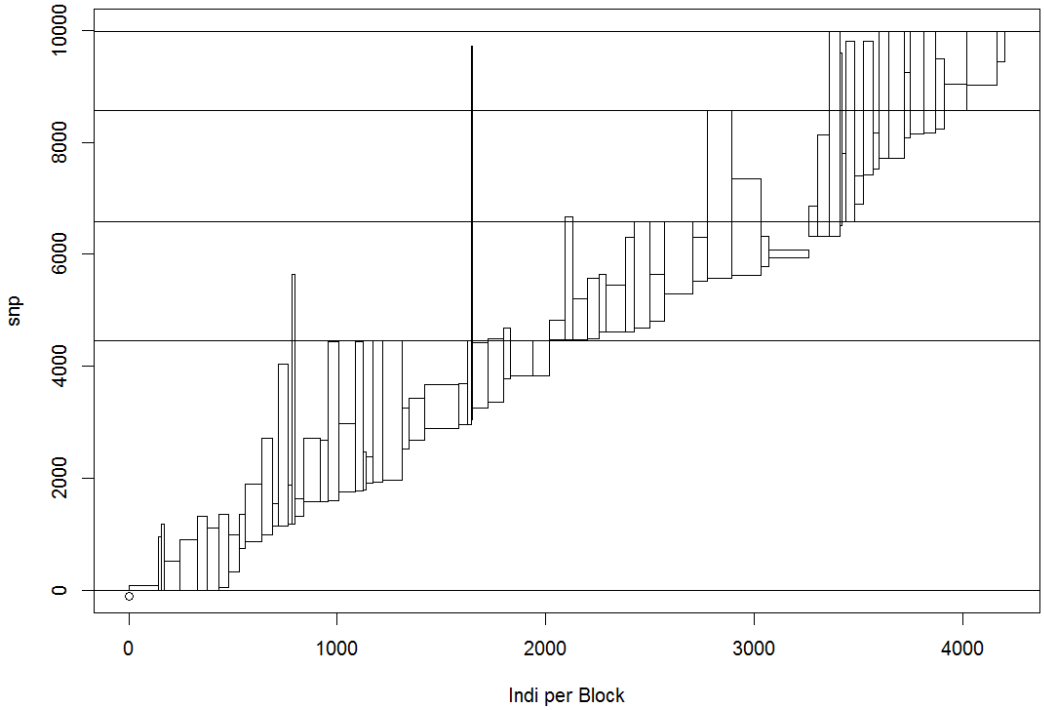

Figure 11: Exemplary output of blocklist_plot_xsize for the ex_maze blocklist (cutoff2=3)

10 Acknowledgements

This package was developed in the context of the BMBF project "MAZE – Accessing the genomic and functional diversity of maize to improve quantitative traits" (Grant ID 031B0195).

Additional thanks goes to the Research Training Group 1644 "Scaling Problems in Statistics" for financing travelling and Manfred Mayer for proofreading of this manuscript.