CiBreed
Center for Integrated Breeding Research

HAPLOBLOCKER

Torsten Pook, Martin Schlather

5. MAI 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 has been published in *Genetics* (https://www.genetics.org/content/212/4/1045). In the following, we will give some short guidelines on how to use the package and introduce input parameters to change the structure of the resulting haplotype library.

2 Installation

HaploBlocker requires R 3.0+ (and the included graphics and stats package) as well as the R-package RandomFieldsUtils. The newest version of HaploBlocker is using RandomFieldsUtils version 0.5.9+. This is highest than the version currently available on CRAN but can be downloaded at https://github.com/tpook92/HaploBlocker. HaploBlocker can directly be installed via the function install_github from the package devtools:

Devtools::install_github(tpook92/HaploBlocker", subdir="pkg")

For manual installation of the package, use the R usage 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.

3 Citation

There is currently no published version of our manuscript in a peer-reviewed journal. This will hopefully change soon. For so long we suggest using following the citations for the preprint on *biorvix* and the R-packages HaploBlocker:

```
@article{Pook.2019,
 author = {Pook, Torsten and Schlather, Martin and {de los Campos}, Gustavo and Mayer, Manfred and Schoen, Chris Carolin and Simianer, Henner},
 year = {2019},
 title = {HaploBlocker: Creation of subgroup specific haplotype blocks and libraries},
journal = {Genetics}
}

@misc{Pook.2019b,
 author = {Pook, Torsten and Schlather, Martin},
 year = {2019},
 title = {HaploBlocker: An R package for the Creation of Haplotype Libraries for DHs and Highly Inbreed Lines},
 url = {https://github.com/tpook92/HaploBlocker}
}
```

4 General

The main function of HaploBlocker is *block_calculation()* and the only mandatory input of the function is a dataset (parameter: **dhm**) containing haplotypes or a path to a vcf/ped-file to import – inputs can contain up to 256 different characters/numeric/integer values. For maximum internal efficiency use just two variants:

```
> ex_maze[1:10,1:8]
       Haplo 1 Haplo 2 Haplo 3 Haplo 4 Haplo 5 Haplo 6 Haplo 7 Haplo 8
SNP 1        T       C       T       C       T       C       T       C
SNP 2        A       A       A       A       A       A       A       A
SNP 3        C       C       C       C       C       C       C       C
SNP 4        A       A       A       A       A       A       A       A
SNP 5        T       T       T       T       G       T       T       T
SNP 6        C       C       C       C       T       C       C       C
SNP 7        G       G       G       G       G       G       G       G
SNP 8        G       G       G       G       G       G       G       G
SNP 9        A       C       A       C       A       C       A       C
SNP 10       C       C       C       C       C       C       C       C
```

Figure 1: Excerpt of the dataset ex_maze

When running, the user is receives updates regarding the currently stage of the algorithm:

```
> 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

5 Parameters of the main function (block_calculation)

In the following, we will discuss the parameters for tuning of the structure of the derived haplotype library according to the step they are occurring in the algorithm. For exemplary inputs, we refer to section 6. As the default settings are chosen to work for most dataset but still perform fast we recommend the usage of our adaptive mode (**adaptive_mode**=TRUE) to create a haplotype library without modifying parameters. The interested reader is referred to the manuscript for a more detailed discussion and comparison of the structure of different haplotype libraries obtained under different parameter settings. To identify only non-overlapping blocks set **overlap_remove** to TRUE. Locally highly different haplotypes can be removed from all haplotype blocks but overall coverages should be basically the same (and are the same if **min_similarity** = 1).

5.1 Prefilters

Parameters: prefilter, maf, equal_remove

Before the actual algorithm is executed, one can remove non-informative SNPs of the dataset. Firstly, filtering can be performed according to a minor allele frequency filter (parameter **maf**). Secondly, one can remove all SNPs in perfect LD to the previous one by activating **equal_remove**. On default, no filtering is done and it has to be activated via the parameter **prefilter**.

5.2 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, the windows of the dataset are of equal length (**window_size**) and in every window the same number of errors (**merging_error**) is allowed. In case one wants to use different window sizes and number of errors per region (e.g. to exactly span windows according to the position of genes) one can manual set this up 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 chose the window boundaries to obtain a certain number of variants per window (Next window starts whenever the previous block would have more variants than **max_groups**).

When providing the physical position of wanted window boundaries (e.g. start/end point of genes) provide them in the parameter **window_anchor_gens** and the resulting **window_sequence** is automatically calculated. Note that no overlapping windows are supported!

To minimize the number of groups in each window one can use the parameter **blockinfo_mode** (on default the groups are derived 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. By doing this **window_size**, **merging_error** and **min_share** are able to process vectors as input and/or **window_sequence** can be just as a list of different window sequences. Each element is processed separately. In case no vector/list is provided the input is used for all cases.

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_sep_weight** merging errors. It has to be noted there that this mode is significantly more time consuming and still open to some change. Note that the required input of our tool are haplotypes – so phasing is required before application.

5.3 Cluster-merging

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

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

To reduce computing 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 merges occuring in one cycle of the algorithm. 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**.

5.4 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 as starting blocks 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 computing 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 required to extend the block one can use the parameter **min_share**. By doing 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 therefore leads to a higher similarity between the haplotypes in a block and a higher number of blocks overall. Additionally, the number of overlapping blocks is heavily reduced.

To form blocks not only for the whole dataset but also for subgroups one can 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 overlapping regions of multiple subgroups.

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

5.5 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 performed by identifying the number of cells in which each block is the most relevant 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 performed 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 most relevant block in each cell 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 increasing steps). The minimum similarity of a haplotype with a block to be included can be set via the parameter **min_similarity**. By doing 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 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 controlled via **off_lines**.

5.6 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 Off-variant-identification (optional)

This step is not executed on default 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 entries/cells of the dataset not included in the block library before.

Parameters: off_node_addition, raster, off_node_minimum_blocklength, off_node_minimum_size

Set **off_node_addition** to TRUE to active this step. In this step each cell is screen for a section of **off_node_minimum_size** haplotypes with the same sequence in **off_node_minimum_blocklength** windows. Afterward all other steps are executed again (especially filtering for **min_majorblock**).

To reduce computing time not every window is consider, but instead only each **raster** window (this should be a value smaller than **off_node_minimum_blocklength**).

5.8 Performance parameters – computing time:

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

Internal computations are faster when a low number of different characters is use is the input dataset (**dhm**). 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 the recoding step and still profit from the advantages of the recoding.

To further reduce computing time one can active parallel computing via **parallel_window**. Here the dataset is split into windows containing **parallel_window** markers. On defaults, window do not overlap but in principle can via **window_overlap**. The number of cores used is controlled via **window_cores**.

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** controls if bitwise coding is used internally (don't see a reason why you would not want this).

6 Exemplary 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 (automatically 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	2	Natural number (1,3,4m…)
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 will use base::intersect HaploBlocker::intersect requires that vector in an ascending sequence of numerics
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
Overlap_remove	FALSE	TRUE

7 Output

The output of *block_calculation()* 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 (joint allelic sequence)
7. 2. Frequency of the joint allelic sequence per marker

8. – 12. Internal stuff to save computing time in the algorithm (Only in the output using developer-mode)

To not only generate the haplotype 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

A full dataset containing 80'200 markers for 910 individuals is provided with the publication and is 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. The package itself contains a dataset of the first 9'999 SNPs of 313 KE DH-lines (**ex_maze**).

9 Functions for later analysis

So far, we have only implemented smaller utility 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, 9.2, 9.3 is encouraged 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 9.8 contain function to derive bEHH and iHH scores for the haplotype library. 9.9, 9.10, 9.11 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

This function can be used to generate a graphical representation of a **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 **orientation** to "front", "mid" or "back". We recommend aligning against a location in SNPs. 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 – the default of 500 should be more than 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 displayed with a low color **intensity**.

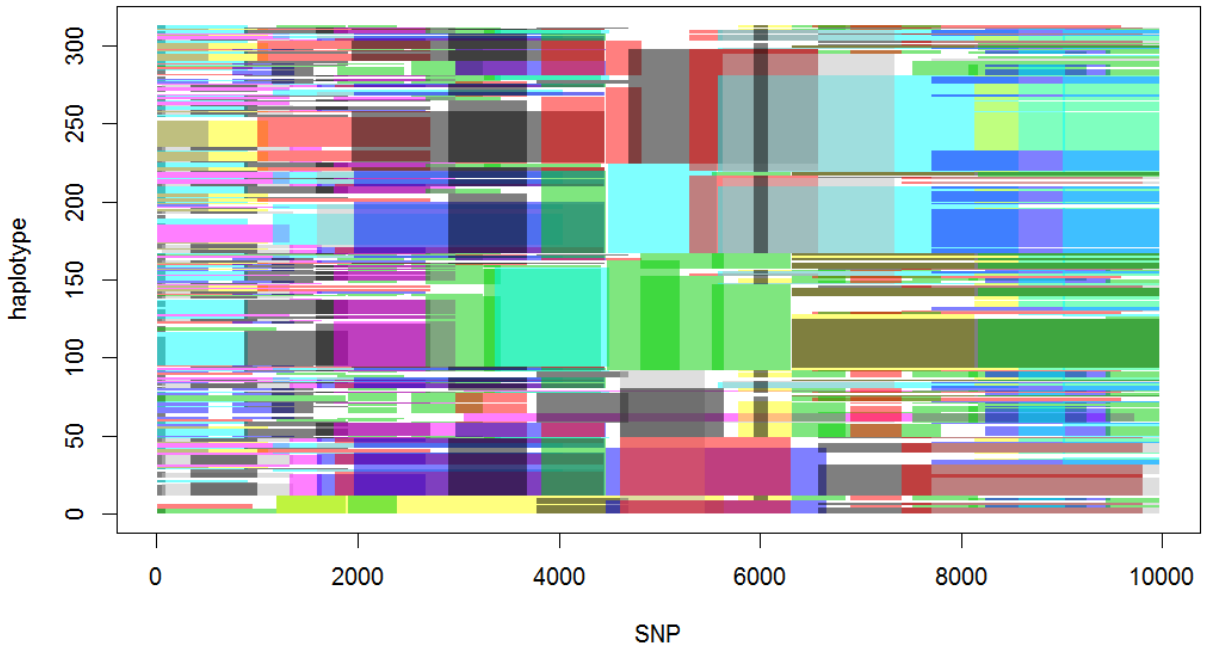

Figure 3: Exemplary output of plot_block for the ex_maze blocklist.

9.2 blocklist_startend()

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

Calculate the start and endpoint of each block. Select the **type** ("window", "snp", "bp") accordingly. Use **first_block** to skip deriving start/endpoints of the first blocks (this is needed internally).

```
> blocklist_startend(blocklist)
         start  end
block 1      1   89
block 2      1  955
block 3      1 1192
block 4      7  521
block 5      7  903
block 6      7 1325
block 7      7 1112
block 8     54 1365
block 9    340 1003
block 10   749 1367
block 11   869 1902
block 12  1000 2723
block 13  1149 1556
block 14  1155 4041
block 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 which cells 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 setting **type** to "window" is recommended to reduce computing 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)
> new_data[1:8,1:8]
       haplo1 haplo2 haplo3 haplo4 haplo5 haplo6 haplo7 haplo8
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, non_haploblocker=FALSE

Generate a window-based block dataset. Blocks span over the same windows of markers for better comparability to other block based approaches. Overall, windows are much shorter than HaploBlocker blocks. Set **consider_nonblock** to TRUE allow haplotypes in no haplotype block in HaploBlocker to be in a block. Set **return_dataset** to TRUE to instead of a dataset coding presence/absence allow for more variants in each window. The minimum length each window has to have can be controlled via **min_length_window**.

To ignore the haplotype blocks derived and just compute all variants set **non_haploblocker** to TRUE. In case **consider_nonblock** or **non_haploblocker** are used, the original dataset has to be provided via **data**.

```
> new_data <- block_windowdataset(blocklist)
> new_data[1:8,1:8]
                   haplo1 haplo2 haplo3 haplo4 haplo5 haplo6 haplo7 haplo8
window:1-6variant1      1      0      1      0      0      0      1      0
window:1-6variant2      0      0      0      0      0      0      0      0
window:7-53variant1     1      0      1      0      1      0      1      0
window:7-53variant2     0      0      0      0      0      0      0      0
window:7-53variant3     0      1      0      1      0      1      0      1
window:7-53variant4     0      0      0      0      0      0      0      0
window:54-89variant1    1      0      1      0      1      0      1      0
window:54-89variant2    0      0      0      0      0      0      0      0
```

Figure 6: Exemplary output of block_windowdataset for the ex_maze blocklist

```
> new_data <- block_windowdataset(blocklist, data= ex_maze, non_haploblocker = TRUE)
Start_blockinfo_calculation
.....................
> new_data[1:8,1:8]
                    haplo1 haplo2 haplo3 haplo4 haplo5 haplo6 haplo7 haplo8
window:1-6variant1       1      0      1      0      0      0      1      0
window:1-6variant2       0      1      0      1      0      1      0      1
window:1-6variant3       0      0      0      0      1      0      0      0
window:1-6variant4       0      0      0      0      0      0      0      0
window:1-6variant5       0      0      0      0      0      0      0      0
window:1-6variant6       0      0      0      0      0      0      0      0
window:1-6variant7       0      0      0      0      0      0      0      0
window:7-53variant1      1      0      1      0      0      0      1      0
```

Figure 7: Exemplary output of block_windowdataset for the ex_maze blocklist considering all variants.

9.6 Overlap_removal()

Parameters: blocklist=NULL, data=NULL

This function make sure that there are no overlapping segments between haplotype blocks. The input data can be generated via activating **big_output** in *block_calculation()*. Alternatively this mode can be activated by setting **overlap_remove** to TRUE in *block_calculation()*.

```
> blocklist <- block_calculation(ex_maze, overlap_remove = TRUE)
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
Start_Overlap_removal:
Before: 70 Blocks, 90.64 % Coverage, 73.38 % Overlapping segments.
After: 121 Blocks, 88.15 % Coverage, 0 % Overlapping segments.
```

9.7 block_ehh()

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

This function can be used 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 are assumed to be equidistant. Position of markers can be provided via **position1**.

A change of **standardization** is not recommended for external use. To compute bEHH scores for different subgroups use **group**. To instead of bEHH return iHH set **return_ehh** to FALSE.

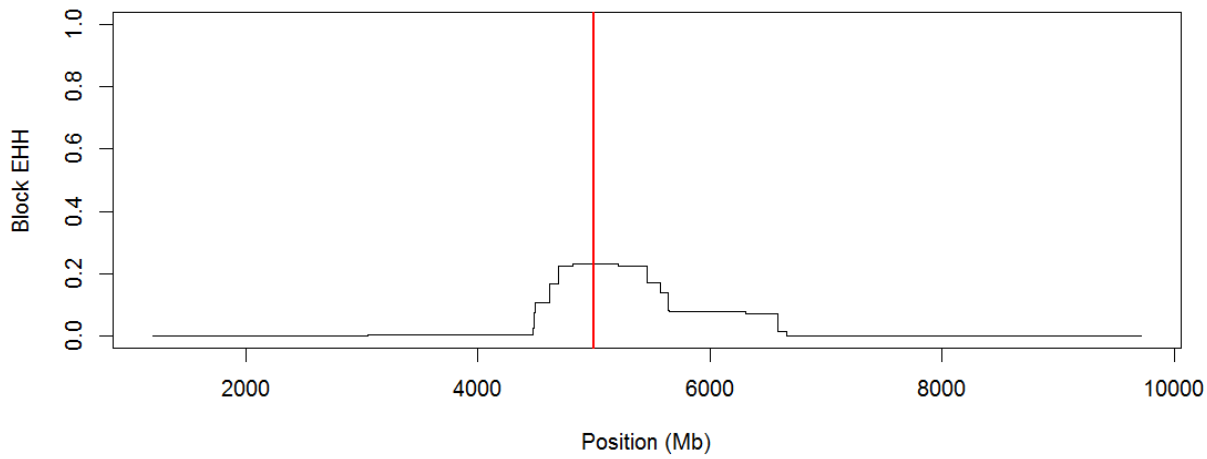

Figure 8: Exemplary output of block_ehh (marker=5000, plot=TRUE).

9.8 block_ihh()

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

This function can be used to compute iHH scores for the whole genome – input parameters work in the same way as *block_ehh* (9.6).

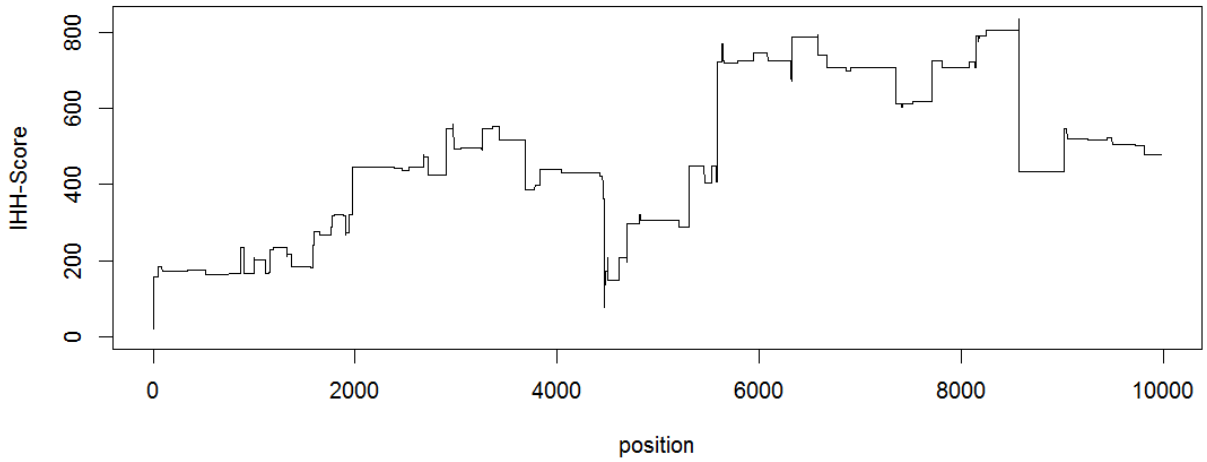

Figure 9: Exemplary output of block_ihh (plot=TRUE).

9.9 block_plot()

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

This function can be used to generate a 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 region.

The parameter **indi** is automatically calculated. **Type** can be set to "window", "snp" or "bp" depending on the desired scaling. The use of "window" is not recommended when using multiple window sizes. For smoothing of the coverage curve a Nadaraya-Watson estimator with bandwidth **bw** can be used (default: no smoothing).

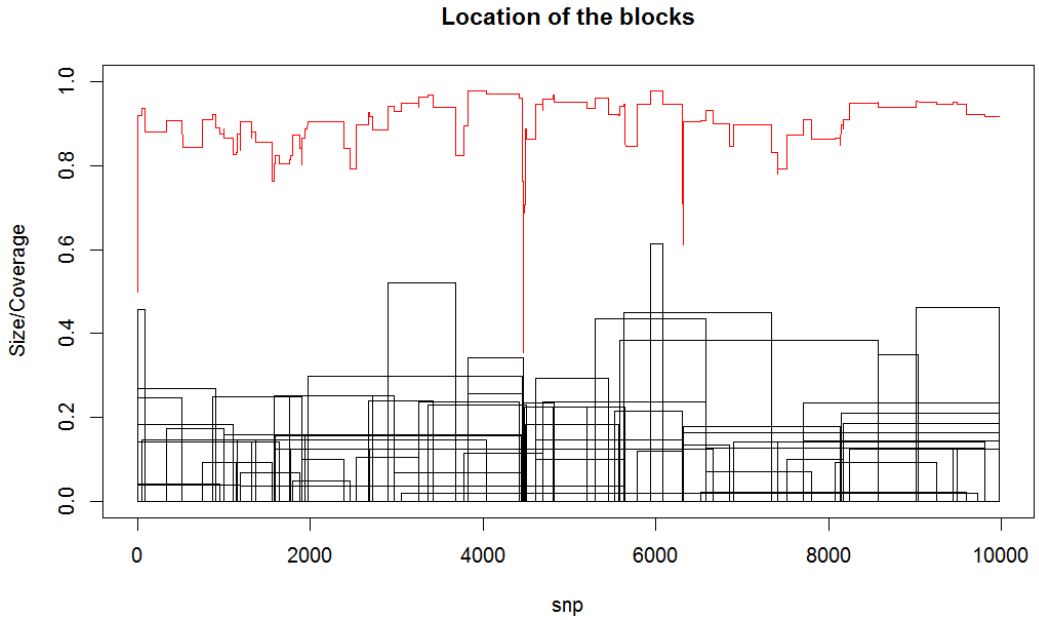

Figure 10: Exemplary output of block_plot for the ex_maze blocklist

9.10 blocklist_plot()

Parameters: blocklist, cutoff2=5, bound_weighted=TRUE, type="snp"

The location of the blocks is given 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 that are required in that region to mark a position as a hotspot and **bound_weighted** scales blocks according to the size of the block. We do not only count the position itself but adjacent markers via a kernel regression method.

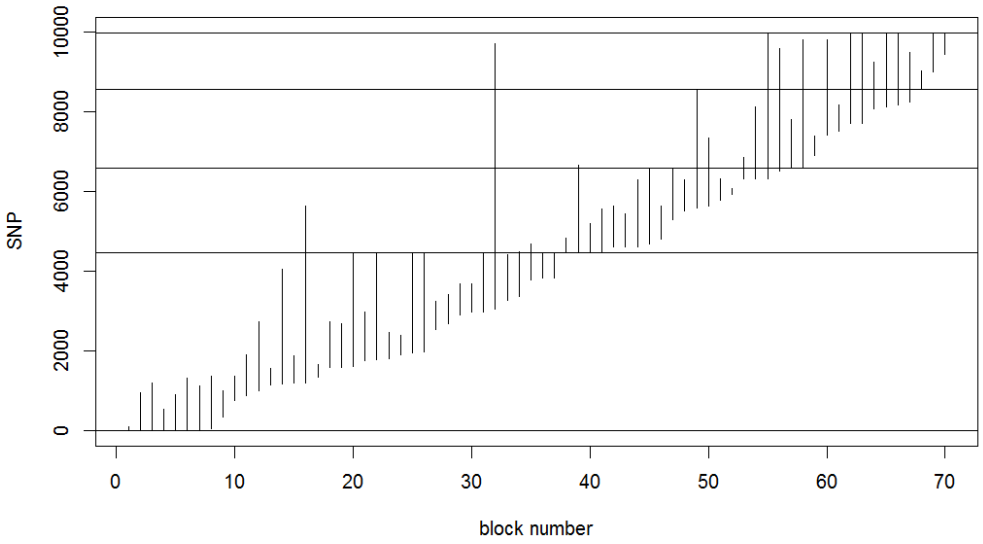

Figure 11: Exemplary output of blocklist_plot for the ex_maze blocklist (cutoff2=3)

9.11 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.

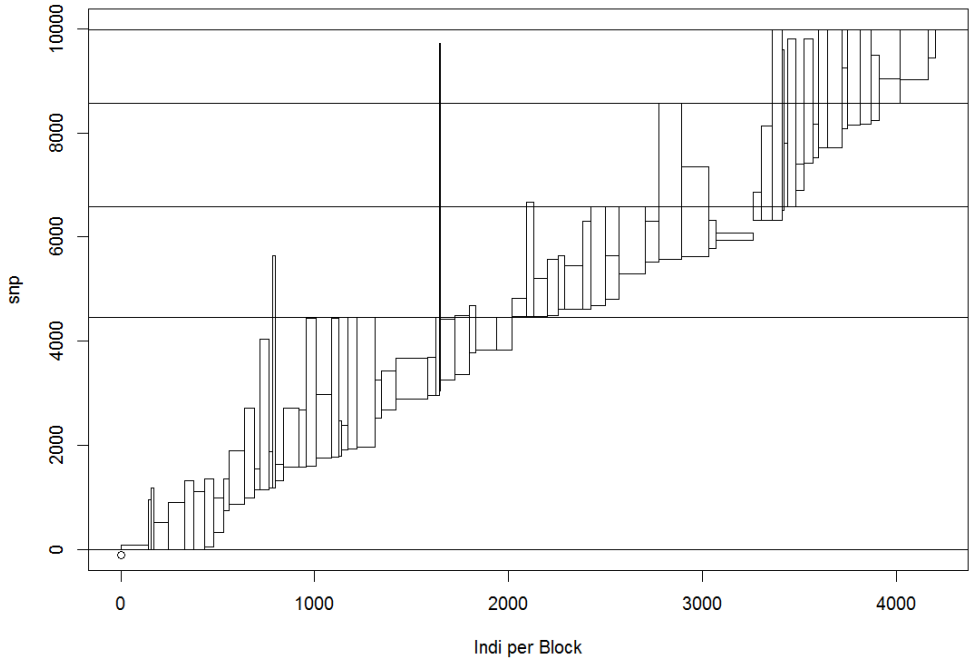

Figure 12: Exemplary output of blocklist_plot_xsize for the ex_maze blocklist (cutoff2=3)

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.