Vignette for the ‘rearrvisr’ package

Doro Lindtke

2019-10-06

Contents

This vignette describes the R package rearrvisr, which can be used to detect, classify, and visualize genome rearrangements along a focal genome. The package requires two input files: one that gives the positions of orthologous markers in an extant genome (the focal genome), and one that specifies the organization of orthologous markers in an ancestral genome reconstruction or a second extant genome (the compared genome).

This vignette provides a tutorial on how to use the package functions and interpret their output by applying them step by step to a *Drosophila* example data set. The vignette also describes the main steps of the implemented algorithm on a simple toy example, and outlines the methods used to create the *Drosophila* example data set from 12 publicly available genome assemblies. See ?rearrvisr for a brief overview of the functions of the package. A detailed description of the rearrvisr algorithm can be found in the Supplementary information of the associated manuscript.

The *Drosophila* example data set used in this vignette (MEL_markers, SIM_markers, YAK_markers, and MSSYE_PQTREE_HEUR) was generated from 12 publicly available genome assemblies downloaded from Ensemble Release 91 (http://dec2017.archive.ensembl.org; *D. melanogaster*) or Ensemble Metazoa Release 37 (http://oct2017-metazoa.ensembl.org; other genomes), using steps and software as described below. An additional example data set (TOY24_focalgenome and TOY24_compgenome) was created for illustrative purposes. A short description of these example data can be accessed with the commands ?MEL_markers, ?SIM_markers, ?YAK_markers, ?MSSYE_PQTREE_HEUR, ?TOY24_focalgenome, and ?TOY24_compgenome.

1 Input files

Two input files are required: a representation of the focal genome in focalgenome, for which rearrangements will be identified, and a representation of the compared genome in compgenome, which serves as reference for the arrangement of markers within genome segments. Deviations in the order of focal markers from the order of reference markers indicate rearrangements. *Genome segments* are chromosomes, scaffolds, or contiguous sets of genetic markers. Genomes do not need to be assembled at chromosome-level, however the detection of rearrangements is inherently limited for assemblies that are highly fragmented. *Markers* denote orthologous genomic regions that are unique within each genome, such as genes or synteny blocks. Orthologous markers require the same ID in both input files, and need to be ordered by their absolute or relative position within their genome segments. Further details are below.

1.1 Focal genome

The focal genome representation in focalgenome is an extant genome in a standard linear (one-dimensional) genome map format that associates markers to genome segments and gives their map positions (e.g., in base pairs) within them. Genome segments of the focal genome are chromosomes or scaffolds and are referred to as *focal genome segments* or *focal segments*.

The focal genome map needs to be a data frame with the mandatory columns *marker* (orthologous marker ID; can be NA for markers that have no ortholog), *scaff* (name of the focal segment where the marker is located), *start* (absolute start position of the marker on its focal segment, for example in base pairs), *end* (absolute end position of the marker), and *strand* (the reading direction of the marker, either "+" or "-"). Additional columns are ignored and may store custom information, such as marker names. Markers need to be ordered by their map position within each focal segment, for example by running the orderGenomeMap() function. More information on the focalgenome file format is available from the ?checkInfile command.

```
## Example for the focal genome map of Drosophila melanogaster
## (marker start and end positions are midpoints +/- 1 base pair)

head(MEL_markers)
#>   marker scaff  start    end strand            name
#> 1   1631     4   3194   3196      + MEL_FBpp0312297
#> 2    207     4  36917  36919      - MEL_FBpp0300616
#> 3    563     4  51751  51753      - MEL_FBpp0088245
#> 4  10611     4  66757  66759      - MEL_FBpp0088242
#> 5   2582     4  87156  87158      + MEL_FBpp0088228
#> 6    474     4 121621 121623      - MEL_FBpp0088240
```

To load a text file with a focal genome map into `R`, commands like the following can be used:

```
## Example for reading a focal genome map from file

markersfile <- "~/path/to/focalgenome.txt"

focalgenome <- read.table(file = markersfile, as.is = TRUE,
                          col.names = c("marker", "scaff",
                                        "start", "end", "strand"))

## assure that column 'scaff' is a character vector
focalgenome$scaff <- as.character(focalgenome$scaff)
```

Important: The column *scaff* must be a character vector, even when all focal segment names are numeric. This is assured by the above `as.character(focalgenome$scaff)` command.

1.1.1 The `orderGenomeMap()` function

The focal genome map in `focalgenome` needs to be ordered so that markers appear according to their map position within each focal segment. Focal segments can be in arbitrary order. The function `orderGenomeMap()` orders markers within each focal segment by their map position (i.e., the midpoint between positions given by the columns *start* and *end* in `focalgenome`). Focal segments can be ordered by user-defined names, in alphabetical order, or by their size. See `?orderGenomeMap` for details.

```
## Example for ordering the focal genome map of D. melanogaster

## make random order (for illustration only)
myorder <- sample(1:nrow(MEL_markers))
MEL_markers_unordered <- MEL_markers[myorder, ]
head(MEL_markers_unordered)
#>       marker scaff    start      end strand            name
#> 13661  19089     X 20413977 20413979      - MEL_FBpp0077020
#> 4976    5887    2R 22070483 22070485      - MEL_FBpp0071716
#> 2446   15473    2L 20650491 20650493      + MEL_FBpp0080917
#> 7716    4921    3L 19001797 19001799      - MEL_FBpp0074807
#> 6282    2763    3L  6691306  6691308      - MEL_FBpp0076668
#> 12761  12212     X 11851947 11851949      - MEL_FBpp0073441

## order genome map by size of focal segments and marker position
MEL_markers_reordered <- orderGenomeMap(MEL_markers_unordered,
                                        ordnames = "all",
                                        sortby = "size")
head(MEL_markers_reordered)
#>      marker scaff   start     end strand            name
#> 8229    912    3R 1653327 1653329      + MEL_FBpp0352251
#> 8230     NA    3R 2969170 2969172      - MEL_FBpp0401450
#> 8231   9484    3R 3338383 3338385      + MEL_FBpp0289444
#> 8232   1832    3R 3535404 3535406      - MEL_FBpp0112608
#> 8233  14129    3R 3624573 3624575      + MEL_FBpp0291024
#> 8234  13818    3R 3738461 3738463      + MEL_FBpp0112618
```

1.1.2 The `checkInfile()` function

To verify that the format of the focal genome map is correct, the `checkInfile()` function can optionally be run. This function is also called internally from other functions of the package where `focalgenome` is used as input. The function will return an error message when a problem has been detected, or nothing otherwise. See `?checkInfile` for more information.

```
## Example for checking the focal genome map of D. melanogaster

checkInfile(MEL_markers, "focalgenome", checkorder = TRUE)
```

Note: If the *marker* column was assigned the class `numeric` rather than the required class `integer`, run for example the command `focalgenome <- type.convert(focalgenome, as.is = TRUE)` to fix the resulting error.

1.2 Compared genome

The compared genome representation in `compgenome` can be an ancestral genome reconstruction or an extant genome, and is organized into genome segments called *CARs* (Contiguous Ancestral Regions; following Ma *et al.*, 2006; Chauve and Tannier, 2008). For simplicity, compared genome segments are referred to as CARs throughout, but can equally be any contiguous sets of genetic markers of an extant genome. Each CAR is represented by a *PQ-tree* (described below; Booth and Lueker, 1976; Chauve and Tannier, 2008), which specifies the relative positions of markers within genome segments, incorporating ambiguity in marker order (i.e., all alternative orders).

The *PQ-tree format* joins all *PQ-trees* (i.e., CARs) of the compared genome representation in a single data frame `compgenome` (details below). For simplicity, the joined *PQ-trees* in the `compgenome` data frame are referred to as *PQ-structure* throughout, while individual *PQ-trees* are referred to as CARs or *PQ-trees*.

An ancestral genome representation in form of *PQ-trees*, as output by the software ANGES (Chauve and Tannier, 2008; Jones *et al.*, 2012), can be converted into a *PQ-structure* with the `convertPQtree()` function. A one-dimensional genome map, for example of an extant genome, can be converted into a two-dimensional *PQ-structure* with the `genome2PQtree()` function.

1.2.1 *PQ-trees*, *P-nodes*, and *Q-nodes*

A *PQ-tree* is a combinatorial structure consisting of *leaves*, *P-nodes*, and *Q-nodes* (Booth and Lueker, 1976; Chauve and Tannier, 2008; Figure 1). This structure allows the representation of ambiguity in marker order in an ancestral genome reconstruction (i.e., encoding all possible arrangements of markers). Each marker is represented by a *leaf*, which is connected to the root of the *PQ-tree* (i.e., a CAR) through a hierarchical sequence of internal nodes (*P-nodes* or *Q-nodes*). The type of internal nodes and the root specify the order of their children (i.e., internal nodes or leaves at the subsequent hierarchy level): they are in arbitrary order for *P-nodes*, and in fixed order (including their reversal) for *Q-nodes*. The Figure 1 illustrates two alternative marker orders (among many) that can be encoded by a single *PQ-tree*.

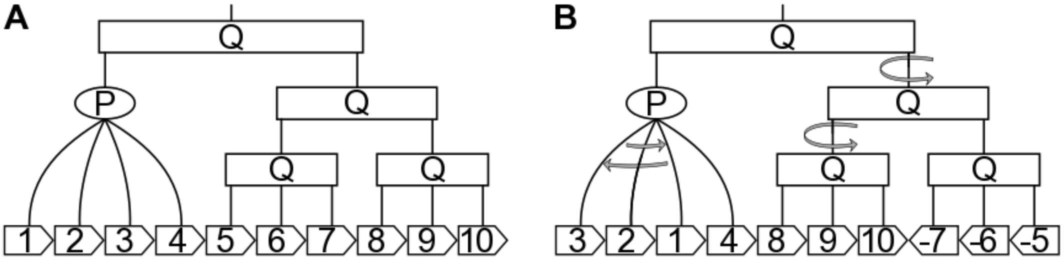

Figure 1: Example for two equivalent PQ-trees.

Figure 1: *P-nodes* are illustrated with ovals and *Q-nodes* with rectangles. Leaves (i.e., markers) are illustrated with pentagons and include their marker ID. Children of *P-nodes* are in arbitrary order, and children of *Q-nodes* are in fixed order (including their reversal). The marker orders `1 2 3 4 5 6 7 8 9 10` in **A** and `3 2 1 4 8 9 10 -7 -6 -5` in **B** can be encoded by the same *PQ-tree*, as branches that origin from *P-nodes* can be transposed, and those that origin from *Q-nodes* can be reversed in their order (i.e., turning the *Q-node* around), as indicated by the gray arrows.

1.2.2 *PQ-structure*

To facilitate the handling of *PQ-trees* in `R`, their hierarchical organization is encoded as a data frame `compgenome` with markers in rows and a description of the structure of *PQ-trees* in columns (the *PQ-tree format*, or *PQ-structure*). The Figure 2 illustrates graphically the tabular *PQ-structure* of the example data `TOY24_compgenome`. All branches that origin at a node are labeled with an integer ID (i.e., the *node element ID*) according to their position at that node, starting at `1`. When considered jointly through all hierarchy levels, these IDs provide the relative position of each marker within their *PQ-tree* (e.g., compare the branch labels in the Figure 2 with the entries in the rows of `TOY24_compgenome` for each marker below). The full compared genome can be represented by a single *PQ-tree* with a *P-node* at its root (i.e., CARs are positioned in arbitrary order to each other), which is encoded by integer IDs at the CAR level.

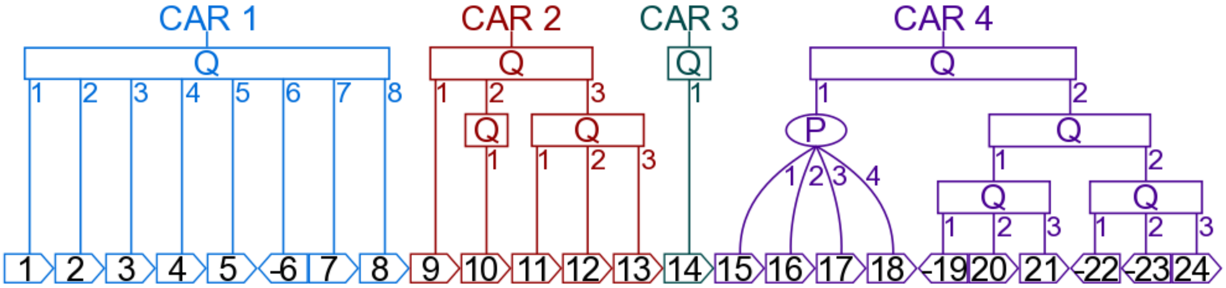

Figure 2: Example PQ-structure.

The data frame `compgenome` contains the mandatory columns *marker* (orthologous marker ID), *orientation* (the reading direction of the marker, either `"+"` or `"-"`), and *car* (ID of the CAR where the marker is located). Subsequent columns describe the internal structure of each CAR, where two columns are needed for each subordinate hierarchy level: the first column of each set gives the node type (either `"P"` or `"Q"`), and the second column the integer ID of the children of the node (i.e., the node element ID). `NA`s are permitted from the second subordinate hierarchy level onwards and are required to fill otherwise empty cells that arise when the preceding hierarchy level encodes leaves. Markers need to be ordered by their node element IDs within the *PQ-structure*. More information on the *PQ-tree format* is available from the `?checkInfile` command.

```
## Example for a compared genome representation in PQ-tree format
## ('TOY24_compgenome'; see Figure for a graphical representation)

TOY24_compgenome
#>    marker orientation car type1 elem1 type2 elem2 type3 elem3
#> 1       1           +   1     Q     1  <NA>    NA  <NA>    NA
#> 2       2           +   1     Q     2  <NA>    NA  <NA>    NA
#> 3       3           +   1     Q     3  <NA>    NA  <NA>    NA
#> 4       4           +   1     Q     4  <NA>    NA  <NA>    NA
#> 5       5           +   1     Q     5  <NA>    NA  <NA>    NA
#> 6       6           -   1     Q     6  <NA>    NA  <NA>    NA
#> 7       7           +   1     Q     7  <NA>    NA  <NA>    NA
#> 8       8           +   1     Q     8  <NA>    NA  <NA>    NA
#> 9       9           +   2     Q     1  <NA>    NA  <NA>    NA
#> 10     10           +   2     Q     2     Q     1  <NA>    NA
#> 11     11           +   2     Q     3     Q     1  <NA>    NA
#> 12     12           +   2     Q     3     Q     2  <NA>    NA
#> 13     13           +   2     Q     3     Q     3  <NA>    NA
#> 14     14           +   3     Q     1  <NA>    NA  <NA>    NA
#> 15     15           +   4     Q     1     P     1  <NA>    NA
#> 16     16           +   4     Q     1     P     2  <NA>    NA
#> 17     17           +   4     Q     1     P     3  <NA>    NA
#> 18     18           +   4     Q     1     P     4  <NA>    NA
#> 19     19           -   4     Q     2     Q     1     Q     1
#> 20     20           +   4     Q     2     Q     1     Q     2
#> 21     21           +   4     Q     2     Q     1     Q     3
#> 22     22           -   4     Q     2     Q     2     Q     1
#> 23     23           -   4     Q     2     Q     2     Q     2
#> 24     24           +   4     Q     2     Q     2     Q     3
```

To load a text file with a compared genome representation into `R`, commands like the following can be used:

```
## Example for reading a compared genome representation from file

pqtreefile <- "~/path/to/compgenome.txt"

## (A) format is whitespace separated with missing entries
##     and no column names
ncol <- max(count.fields(file = pqtreefile, sep = " "))
compgenome <- read.table(file = pqtreefile, fill = TRUE,
                         col.names = c("marker", "orientation", "car",
                         paste0(rep(c("type", "elem"),
                                    times = (ncol - 3) / 2),
                                rep(1:((ncol - 3) / 2),
                                    each = 2))),
                         as.is = TRUE, na.strings = "")

## (B) format has NA for missing entries and has column names
compgenome <- read.table(file = pqtreefile, header = TRUE,
                         as.is = TRUE)
```

1.2.3 The convertPQtree() function

The ancestral genome reconstruction that is generated by the software ANGES (Chauve and Tannier, 2008; Jones *et al.*, 2012) is encoded as a text file with the first row giving the name of the ancestor (preceded by `>`), followed by two rows for each CAR. The first row of each set gives the CAR ID (preceded by `#CAR`), and the second row provides the *PQ-tree* structure of that CAR. The children of each node in the *PQ-tree* are enclosed by `_P` and `P_` markups for *P-nodes*, or by `_Q` and `Q_` markups for *Q-nodes*. Markers (i.e., orthologs) that belong to a particular node are located between its corresponding markups. Markers with reversed orientation are preceded by a `-` sign. The opening (`_P` or `_Q`) and closing (`P_` or `Q_`) markups can be nested to allow the representation of the hierarchical structure of the *PQ-tree*. Such linearly encoded *PQ-trees* can be converted into a *PQ-structure* with the `convertPQtree()` function. See `?convertPQtree` for more information.

Important: The `convertPQtree()` function was designed to work with the ANGES `*PQTREE*` output file (e.g., the `MSSYE_PQTREE_HEUR` example file). It was not designed to work with the ancillary `*PQRTREE*` or the `*DOUBLED*` output files.

```
## Example for a linearly encoded PQ-tree for #CAR7 of the
## Drosophila ancestral genome 'MSSYE' computed with the
## software ANGES (Jones et al., 2012)
MSSYE_PQTREE_HEUR[which(MSSYE_PQTREE_HEUR[ ,1] == "#CAR7") + 1, ]
#> [1] "_Q 11698 11765 -15944 15826 _Q 15827 Q_ Q_ "

## Convert linearly encoded PQ-trees into PQ-structure
MSSYE_compgenome <- convertPQtree(MSSYE_PQTREE_HEUR)
#>
#> converting data for MSSYE
#> ... processed 20 CARs ...

## Show converted #CAR7
MSSYE_compgenome[MSSYE_compgenome$car == 7, ]
#>      marker orientation car type1 elem1 type2 elem2 type3 elem3
#> 8953  11698           +   7     Q     1  <NA>    NA  <NA>    NA
#> 8954  11765           +   7     Q     2  <NA>    NA  <NA>    NA
#> 8955  15944           -   7     Q     3  <NA>    NA  <NA>    NA
#> 8956  15826           +   7     Q     4  <NA>    NA  <NA>    NA
#> 8957  15827           +   7     Q     5     Q     1  <NA>    NA
```

Note: For the following examples it is assumed that the object `MSSYE_compgenome` has been created with the above command.

To load a text file with an ancestral genome reconstruction from the software ANGES into `R` and to convert it into a *PQ-structure*, commands like the following can be used:

```
## Example for reading an ancestral genome reconstruction
## from the software ANGES from file and converting it into
## a PQ-structure
```

```
pqtreefile <- "~/path/to/PQTREE"

## read file with the genome reconstruction
rawtree <- read.table(file = pqtreefile, sep = ",",
                      comment.char = "", as.is = TRUE)

## convert into PQ-structure
compgenome <- convertPQtree(rawtree)
```

1.2.4 The `genome2PQtree()` function

Alternatively to a genome reconstruction in *PQ-tree format*, a one-dimensional genome map can be converted into a two-dimensional *PQ-structure*. This can be, for example, a genome map of an extant genome, or a genome map of an unambiguous genome reconstruction. The compared genome map has to be in the same format as the focal genome map (described above), and needs to be ordered by the map position of markers within each compared genome segment, for example by running the `orderGenomeMap()` function. An unambiguously ordered genome representation can be seen as a subclass of a *PQ-structure*, where each genome segment is encoded by a single *Q-node* that only contains leaves as children. Accordingly, the *PQ-structure* of a converted genome map has exactly five columns (i.e., *marker*, *orientation*, *car*, and two columns for node type and node element). The `genome2PQtree()` function performs such a conversion. See `?genome2PQtree` for details.

Important: Compared genome segments need to be contiguous sets of genetic markers. Genome segments that are (potentially) overlapping, such as minor scaffolds or contigs that were not assembled into chromosomes and might in fact be part of assembled chromosomes or enclosed in other scaffolds, need to be excluded.

```
## Example for converting the genome map of D. simulans into
## a PQ-structure

## Exclude potentially overlapping minor scaffolds from genome map
SIM_markers_chr <- SIM_markers[is.element(SIM_markers$scaff,
                                          c("2L", "2R", "3L", "3R", "4", "X")), ]

## Order genome map by specified chromosomes
SIM_markers_chr <- orderGenomeMap(SIM_markers_chr,
                                  c("2L", "2R", "3L", "3R", "4", "X"))

## Show original genome
head(SIM_markers_chr)
#>    marker scaff start   end strand            name
#> 68   1133    2L 19900 19902      - SIM_FBpp0221398
#> 69   6934    2L 26387 26389      - SIM_FBpp0221397
#> 70    109    2L 37177 37179      - SIM_FBpp0221396
#> 71   1470    2L 68885 68887      + SIM_FBpp0221399
#> 72   5211    2L 74739 74741      + SIM_FBpp0221400
#> 73   8187    2L 85887 85889      - SIM_FBpp0221395

## Convert genome map into PQ-structure
SIM_compgenome <- genome2PQtree(SIM_markers_chr)

## Show converted genome
head(SIM_compgenome)
#>   marker orientation car type1 elem1
#> 1   1133           -   1     Q     1
#> 2   6934           -   1     Q     2
#> 3    109           -   1     Q     3
#> 4   1470           +   1     Q     4
#> 5   5211           +   1     Q     5
#> 6   8187           -   1     Q     6

## Print a translation between chromosome names and CAR IDs
head(data.frame(chr = unique(SIM_markers_chr$scaff),
                car = 1:length(unique(SIM_markers_chr$scaff)),
                stringsAsFactors = FALSE))
#>   chr car
#> 1  2L   1
#> 2  2R   2
#> 3  3L   3
#> 4  3R   4
#> 5   4   5
#> 6   X   6
```

Note: CAR IDs will be assigned according to the order of compared genome segments in the genome map. Markers that are `NA` in the genome map will be excluded from the *PQ-structure*.

1.2.5 The `checkInfile()` function

To verify that the format of the compared genome representation is correct, the `checkInfile()` function can optionally be run. This function is also called internally from other functions of the package where `compgenome` is used as input. The function will return an error message when a problem has been detected, or nothing otherwise. See `?checkInfile` for more information.

```
## Example for checking the compared genome representation of the
## Drosophila ancestral genome 'MSSYE' that was converted into
## 'MSSYE_compgenome' above

checkInfile(MSSYE_compgenome, "compgenome", checkorder = TRUE)
```

Note: If the *marker* column was assigned the class `numeric` rather than the required class `integer`, run for example the command `compgenome <- type.convert(compgenome, as.is = TRUE)` to fix the resulting error.

2 Identifying rearrangements

The `computeRearrs()` function detects and classifies rearrangements with the `rearrvisr` algorithm. The `summarizeBlocks()` function summarizes the output of the `computeRearrs()` function for each synteny block.

2.1 The `computeRearrs()` function

The `computeRearrs()` function requires as input files an ordered map of the focal genome (`focalgenome`), for which rearrangements will be identified, and an ordered representation of the compared genome (`compgenome`), which serves as reference for the arrangement of markers within genome segments. In addition, it needs to be specified whether markers in the ancestral genome reconstruction contain information about their orientation (`doubled`). If orientation information is not available (i.e., `doubled = FALSE`), all values in the *orientation* column of `compgenome` should be `"+"`. See `?computeRearrs` for more information on the input and output of the function. Further details are also provided in the walk-through example of the `rearrvisr` algorithm below.

```
## Example for identifying rearrangements in the D. melanogaster
## genome that occurred after divergence from the Drosophila
## ancestor 'MSSYE'

## identify rearrangements (may run a few seconds)
SYNT_MEL_MSSYE <- computeRearrs(MEL_markers, MSSYE_compgenome, doubled = TRUE)

## show names of the matrices in the output
names(SYNT_MEL_MSSYE)
#>  [1] "TLBS"     "TLWS"     "TLWC"     "IV"       "TLBSbS"   "TLBSbE"  
#>  [7] "TLWSbS"   "TLWSbE"   "TLWCbS"   "TLWCbE"   "IVbS"     "IVbE"    
#> [13] "nodeori"  "blockori" "blockid"  "premask"  "subnode"
```

Note: For the following examples it is assumed that the object `SYNT_MEL_MSSYE` has been created with the above command.

A basal step of the `rearrvisr` algorithm is the alignment of the compared genome representation in `compgenome` to the sorted genome map of the focal species in `focalgenome`. This can be reconstructed with the following command:

```
## Example for reconstructing the alignment between the genome of
## the Drosophila ancestor 'MSSYE' and the D. melanogaster genome

## make alignment
MEL_MSSYE<-merge(MEL_markers, MSSYE_compgenome, by = "marker",
                 sort = FALSE)

## marker IDs should be identical to the ones in SYNT_MEL_MSSYE
identical(as.character(MEL_MSSYE$marker),
          rownames(SYNT_MEL_MSSYE$TLBS))
#> [1] TRUE
```

Note: For the following examples it is assumed that the alignment `MEL_MSSYE` has been created with the above command.

Next steps: *The data returned by the* `computeRearrs()` *function can be filtered for rearrangements of a particular size with the* `filterRearrs()` *function, visualized with the* `genomeImagePlot()` *function, or summarized and visualized with the* `summarizeBlocks()` *and* `genomeRearrPlot()` *functions. Breakpoint coordinates for rearrangements can be extrated with the* `getBreakpoints()` *function.*

Output: Each matrix in the output stores data on detected rearrangements and additional information. Markers are in rows, and the row names of each matrix correspond to the IDs in the *marker* column of `focalgenome` and `compgenome`.

The matrices `$TLBS`, `$TLWS`, `$TLWC`, and `$IV` contain rearrangements of four different classes, and are briefly described below. Further details and information on the other matrices are provided in the function documentation.

Rearrangements are classified as translocations between compared genome segments (i.e., CARs), analogous to interchromosomal rearrangements (`TLBS` and `TLWS`), and as translocations or inversions within compared genome segments, analogous to intrachromosomal rearrangements (`TLWC` and `IV`). `$TLBS` stores TransLocations between CARs Between focal Segments; `$TLWS` stores TransLocations between CARs Within focal Segments; `$TLWC` stores TransLocations Within CARs within focal segments; `$IV` stores InVersions within CARs within focal segments. See the Figure 3 for examples of these four classes of rearrangements.

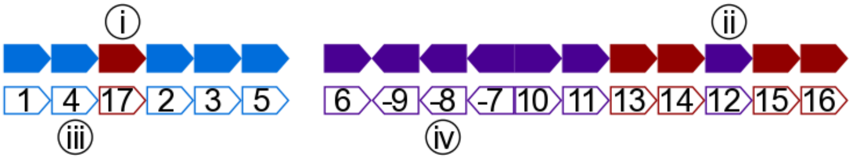

Figure 3: Four classes of rearrangements.

Figure 3: Markers located on two focal genome segments (left and right) are represented by pentagons. Markers are arranged from left to right by their position in the focal genome. The top row shows markers colored according to their CAR of origin, and integers in the bottom row indicate the order of markers within the compared genome. **(i)**, a TransLocation between CARs Between focal Segments (`TLBS`; i.e., marker `17` of the red CAR is located in the middle of the left focal segment, while the remaining markers of the red CAR are on the right focal segment; note that in this example, the translocated marker `17` also belongs to class ii). **(ii)**, a TransLocation between CARs Within focal Segments (`TLWS`; i.e., marker `12` of the purple CAR is located in the middle of the red CAR within the right focal segment). **(iii)**, a TransLocation Within CARs within focal segments (`TLWC`; i.e., marker `4` has changed position within the blue CAR within the left focal segment). **(iv)**, InVersion within CARs within focal segments (`IV`; i.e., markers `7 8 9` are inverted to `-9 -8 -7` within the purple CAR).

Each rearrangement is represented by a separate column. Except for `TLBS`, which are identified across focal segments, columns for individual focal segments are joined across rows to save space (i.e., for `TLWS`, `TLWC`, and `IV`, which are identified within focal segments). To preserve the tabular format, these matrices are filled by zeros for focal segments with a non-maximal number of rearrangements, if necessary. If no rearrangements were detected for a certain class, the matrix has zero columns.

Markers that are part of a rearrangement have a tag value of `>0` within their respective column. Tagged markers within a column are not necessarily consecutive, for example, when a rearrangement is split into several parts through an insertion of a different CAR, or when a rearrangement has an upstream and a downstream component. Larger tag values (up to `1`) correspond to a higher confidence that the marker has been rearranged.[1] This arises from the fact that translocations can only be identified relative to each other. For example, if the marker order in the compared genome is `1 2 3 4 5`, and that in the focal genome is `1 3 4 2 5`, marker `2` might have been translocated to the right, or markers `3 4` might have been translocated to the left. In such a case, the markers that are more parsimonious to have been translocated relative to the alternative markers receive tag values of `1 - remWgt`, while alternative markers receive tag values of `remWgt` (`remWgt` can be set by the user). If such a distinction cannot be made, all involved markers are tagged with `0.5`. Markers that are part of inversions are always tagged with `1`. Full details are given in the description of the `rearrvisr` algorithm in the Supplementary information of the associated manuscript.

```
## Example for extracting inversions that have been identified on
## chromosome 2L of D. melanogaster when compared to the genome of
## the Drosophila ancestor 'MSSYE'

## extract marker IDs for chromosome 2L from alignment
```

[1] Note that these values are not probabilities, rather labels to tag markers that have arrangements in the focal genome that are in conflict to their arrangement in the compared genome.

```r
MEL_2L_markers <- MEL_MSSYE$marker[MEL_MSSYE$scaff == "2L"]

## get position of markers on chromosome 2L in SYNT
tmp <- is.element(rownames(SYNT_MEL_MSSYE$IV), MEL_2L_markers)

## show inverted markers on chromosome 2L
SYNT_MEL_MSSYE$IV[rowSums(SYNT_MEL_MSSYE$IV) != 0 & tmp, ,
                  drop = FALSE]
#>       [,1] [,2] [,3] [,4] [,5]
#> 13315    1    0    0    0    0
#> 11698    0    1    0    0    0
#> 11765    0    1    0    0    0
#> 7605     0    0    1    0    0

## look at inverted marker 13315 in its genomic context
mpos<-which(MEL_MSSYE$marker == 13315)
MEL_MSSYE[(mpos - 4):(mpos + 4), c(1, 2, 5, 8, 7, 10)]
#>      marker scaff strand car orientation elem1
#> 1633   8691    2L      -   3           +   105
#> 1634   6481    2L      -   3           +   104
#> 1635    279    2L      -   3           +   103
#> 1636  14325    2L      +   3           -   102
#> 1637  13315    2L      -   3           -   110
#> 1638   3420    2L      -   3           +   101
#> 1639  14363    2L      -   3           +   100
#> 1640  13300    2L      +   3           -    99
#> 1641  12424    2L      -   3           +    98
## this shows that the genomic region where marker 13315 is located
## has been aligned in reverse direction: strand and orientation show
## inverted directionality, and the direction of elements in elem1 is
## descending; within this context, marker 13315 has identical
## directionality, which indicates an inversion (in this case, a
## single-marker inversion)
```

2.2 The summarizeBlocks() function

The `summarizeBlocks()` function summarizes the output of the `computeRearrs()` function for each synteny block in a more compact way. The function requires as input files the output of the `computeRearrs()` function (the list `SYNT`), and the data frames `focalgenome` and `compgenome`, which have been used to generate `SYNT`. In addition, the IDs of the focal genome segments that should be summarized need to be specified with the character vector `ordfocal`. See `?summarizeBlocks` for more information on the input and output of the function.

```r
## Example for summarizing rearrangements in the D. melanogaster
## genome that occurred after divergence from the Drosophila
## ancestor 'MSSYE'

## summarize rearrangements for each synteny block
## (may run a few seconds)
BLOCKS_MEL_MSSYE <- summarizeBlocks(SYNT_MEL_MSSYE,
                                    MEL_markers, MSSYE_compgenome,
                                    c("2L", "2R", "3L", "3R", "X"))
```

```
## show names of the lists in the output
names(BLOCKS_MEL_MSSYE)
#> [1] "2L" "2R" "3L" "3R" "X"

## show names of matrices within lists
names(BLOCKS_MEL_MSSYE[[1]])
#> [1] "blocks" "TLBS"   "TLWS"   "TLWC"   "IV"     "IVsm"
```

Note: For the following examples it is assumed that the object `BLOCKS_MEL_MSSYE` has been created with the above command.

Next steps: *The data returned by the* `summarizeBlocks()` *function can be visualized with the* `genomeRearrPlot()` *function.*

Output: The function output is a list of lists for each focal genome segment in `ordfocal`. Each of these lists summarizes the alignment between `focalgenome` and `compgenome`, and the rearrangements in `SYNT`.

Each list per focal genome segment contains the data frame `$blocks`, which contains information on the alignment and structure of each *PQ-tree*, and five matrices `$TLBS`, `$TLWS`, `$TLWC`, `$IV`, and `$IVsm` that store detected rearrangements. Each synteny block is represented by a row. Note that separate blocks are also generated when the hierarchical structure of the underlying *PQ-tree* changes, therefore not all independent rows are caused by a rearrangement. Further note that if the list `SYNT` has been filtered with the `filterRearrs()` function, only the detected rearrangements will be affected, while the information contained in `$blocks` will remain unchanged.

```
## print $blocks data frame for chromosome 3R of D. melanogaster
BLOCKS_MEL_MSSYE$`3R`$blocks
#>    start  end markerS markerE car type1 elemS1 elemE1 node1 nodeori1
#> 1      1  323    3157    1747   1     Q   2372   2050     0       -1
#> 2    324  775    1597   15589   1     Q    893   1344     0       -1
#> 3    776  776    1542    1542   1     Q   1346   1346     0       -1
#> 4    777  777    9155    9155   1     Q   1345   1345     0       -1
#> 5    778 1124    7278   17099   1     Q   1347   1693     0       -1
#> 6   1125 1125    9874    9874   1     Q   1695   1695     0       -1
#> 7   1126 1126   10236   10236   1     Q   1694   1694     0       -1
#> 8   1127 1250   10399   14959   1     Q   1696   1819     0       -1
#> 9   1251 1251     524     524   1     Q   1821   1821     0       -1
#> 10  1252 1252   12688   12688   1     Q   1820   1820     0       -1
#> 11  1253 1480    6868    6456   1     Q   1822   2049     0       -1
#> 12  1481 1885    1526    4203   1     Q    892    488     0       -1
#> 13  1886 1886    6780    6780   1     Q    486    486     0       -1
#> 14  1887 1887    3757    3757   1     Q    487    487     0       -1
#> 15  1888 2242   14838   13047   1     Q    485    131     0       -1
#> 16  2243 2244   18844   18865   1     Q    129    130     0       -1
#> 17  2245 2372   18849    4653   1     Q    128      1     0       -1
#>    subnode1 blockid1 blockori1 premask1
#> 1         0       13        -1        0
#> 2         0        6         1        0
#> 3         0      7.1        -1        0
#> 4         0      7.2        -1        0
#> 5         0        8         1        0
#> 6         0      9.1        -1        0
#> 7         0      9.2        -1        0
#> 8         0       10         1        0
#> 9         0     11.1        -1        0
#> 10        0     11.2        -1        0
#> 11        0       12         1        0
#> 12        0        5        -1        0
#> 13        0      4.1         1        0
#> 14        0      4.2         1        0
#> 15        0        3        -1        0
#> 16        0        2         1        0
#> 17        0        1        -1        0
## this shows that chromosome 3R is composed of 17 synteny blocks;
## details on the first four blocks are given below
```

In the $blocks data frame, the columns *start* and *end* give the start and end positions of the synteny block in SYNT (positions start at 1 separately for each focal genome segment), *markerS* and *markerE* give the marker IDs of the first and last marker per block, and *car* gives the ID of the aligned CAR.

```
## Details on the boundary between the first and second block on
## chromosome 3R of D. melanogaster between markers 1747 and 1597
## (node elements 2050 and 893; positions 323 and 324)

MEL_MSSYE[MEL_MSSYE$scaff == "3R", c(1, 2, 5, 8, 7, 10)][313:334, ]
#>      marker scaff strand car orientation elem1
#> 5887  11871    3R      +   1           -  2060
#> 5888   2352    3R      +   1           -  2059
#> 5889   9266    3R      -   1           +  2058
#> 5890  14789    3R      -   1           +  2057
#> 5891   7221    3R      +   1           -  2056
#> 5892   5094    3R      -   1           +  2055
#> 5893   3121    3R      +   1           -  2054
#> 5894   9210    3R      -   1           +  2053
#> 5895  10369    3R      +   1           -  2052
#> 5896  11781    3R      -   1           +  2051
#> 5897   1747    3R      +   1           -  2050
#> 5898   1597    3R      +   1           +   893
#> 5899   4925    3R      +   1           +   894
#> 5900   5520    3R      +   1           +   895
#> 5901   5715    3R      +   1           +   896
#> 5902   3263    3R      +   1           +   897
#> 5903  10490    3R      -   1           -   898
#> 5904  13175    3R      +   1           +   899
#> 5905  14690    3R      -   1           -   900
#> 5906   4995    3R      +   1           +   901
#> 5907   2771    3R      +   1           +   902
#> 5908   6894    3R      -   1           -   903
## this shows that the alignment changes from descending
## (i.e., inverted) to ascending (i.e., standard) direction
## at the block boundary
```

Additional columns in the $blocks data frame provide more information on the alignment and structure of each CAR for all hierarchy levels of the underlying *PQ-tree* (the first hierarchy level has columns with the suffix 1, the second 2, and so on). See the documentation of the summarizeBlocks() and the computeRearrs() function for a full description of the columns in the $blocks data frame.

There is only one hierarchy level for chromosome 3R of *D. melanogaster* in the example above. The column *nodeori1* stores the alignment direction of the first level node in the *PQ-tree* of the aligned CAR, with 1 indicating ascending (i.e., standard), and -1 descending (i.e., inverted) alignment (given that the node is a *Q-node* and the alignment direction could be determined). Only one CAR (CAR 1) has been aligned to chromosome 3R of *D. melanogaster* in descending direction. The column *blockori1* stores the orientation of each synteny block, again with 1 indicating ascending (i.e., standard), and -1 descending (i.e., inverted) orientation (given that the bock orientation could be determined). In the above example, the first block has descending and the second block ascending direction, which is also reflected by the first and last node element IDs of each block, which are given in the columns *elemS1* and *elemE1*.

Finally, the column *blockid1* stores the ID of each synteny block within its node. For *Q-nodes*, block IDs reflect the order of synteny blocks in `compgenome`. In the above example, the first block on chromosome 3R of *D. melanogaster* is the last block on CAR 1 of the *MSSYE* ancestor. Block IDs with .1 or .2 suffixes (in arbitrary order) indicate blocks that were subject to an additional subdivision step. In the above example, the third and fourth blocks were initially joined to a descending block with node elements 1346 and 1345. Block subdivisions are used to assign a more parsimonious TLWC instead of an IV to a block.

```
## Details on the boundary between the third and fourth block on
## chromosome 3R of D. melanogaster between node elements 1346
## and 1345 (positions 776 and 777)

MEL_MSSYE[MEL_MSSYE$scaff == "3R", c(1, 2, 5, 8, 7, 10)][772:781, ]
#>      marker scaff strand car orientation elem1
#> 6346   5359    3R      +   1           + 1341
#> 6347   2932    3R      -   1           - 1342
#> 6348   5539    3R      +   1           + 1343
#> 6349  15589    3R      +   1           + 1344
#> 6350   1542    3R      +   1           + 1346
#> 6351   9155    3R      -   1           - 1345
#> 6352   7278    3R      -   1           - 1347
#> 6353   6603    3R      -   1           - 1348
#> 6354   7843    3R      +   1           + 1349
#> 6355  13365    3R      -   1           - 1350
## this shows that the larger-scale alignment around the two node
## elements is ascending (i.e., from 1341 to 1350), while elements
## 1346 and 1345 form a descending block. As their strand and
## orientation show identical directionality, it is more
## parsimonious to assign a translocation instead of an inversion,
## which would require two additional single-marker inversions to
## achieve the required inverted directionality. Accordingly, the
## initial block of two elements is split into two single-element
## blocks
```

The matrices `$TLBS`, `$TLWS`, `$TLWC`, `$IV`, and `$IVsm` summarize the four classes of rearrangements in SYNT for the synteny blocks characterized in `$blocks` (tag values within `$TLBS`, `$TLWS`, and `$TLWC` are the same as in SYNT, see above). Inversions that are part of a multi-marker inversion are stored in `$IV`, and blocks part of such inversions are tagged with 1 (with separate columns for different inversions). Single-marker inversions (i.e., markers with switched orientation) are stored in `$IVsm`. The positions of such single-marker inversions within their block are indicated by integers >0 (with separate columns for different single-marker inversions).

```
## print $IV matrix for chromosome 3R of D. melanogaster
BLOCKS_MEL_MSSYE$`3R`$IV
#>       [,1] [,2]
#>  [1,]    0    0
#>  [2,]    1    0
#>  [3,]    1    0
#>  [4,]    1    0
#>  [5,]    1    0
#>  [6,]    1    0
#>  [7,]    1    0
#>  [8,]    1    0
#>  [9,]    1    0
#> [10,]    1    0
#> [11,]    1    0
#> [12,]    0    0
#> [13,]    0    0
#> [14,]    0    0
#> [15,]    0    0
#> [16,]    0    1
#> [17,]    0    0
## this shows that two multi-marker inversions have been detected;
## the first one spans blocks 2:11, and the second one involves
## block 16
```

```
## print $IVsm matrix for chromosome 3R of D. melanogaster
BLOCKS_MEL_MSSYE$`3R`$IVsm
#>       [,1] [,2]
#>  [1,]    0    0
#>  [2,]  419    0
#>  [3,]    0    0
#>  [4,]    0    0
#>  [5,]    0    0
#>  [6,]    0    0
#>  [7,]    0    0
#>  [8,]    0    0
#>  [9,]    0    0
#> [10,]    0    0
#> [11,]    0    0
#> [12,]    0    0
#> [13,]    0    0
#> [14,]    0    0
#> [15,]    0    0
#> [16,]    0    0
#> [17,]    0   18
## this shows that two single-marker inversions have been
## detected, the first is on position 419 within block 2, the
## second on position 18 within block 17

## look at the first single-marker inversion, which is located
## at position 743 (start position of block 2 + position of the
## inversion)
MEL_MSSYE[MEL_MSSYE$scaff == "3R", c(1, 2, 5, 8, 7, 10)][738:746, ]
#>      marker scaff strand car orientation elem1
#> 6312   5819    3R      +   1           +  1307
#> 6313   6560    3R      -   1           -  1308
#> 6314   5087    3R      +   1           +  1309
#> 6315   3137    3R      -   1           -  1310
#> 6316   3588    3R      +   1           -  1311
#> 6317   3714    3R      +   1           +  1312
#> 6318  13397    3R      +   1           +  1313
#> 6319   4303    3R      -   1           -  1314
#> 6320   5603    3R      -   1           -  1315
## this shows that marker 3588 is located in an ascending block,
## but has inverted directionality between strand and orientation
```

```
## print $TLWC matrix for chromosome 3R of D. melanogaster
BLOCKS_MEL_MSSYE$`3R`$TLWC
#>       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
#>  [1,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
#>  [2,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
#>  [3,]  0.5  0.0  0.0  0.0  0.0  0.0  0.0  0.0
#>  [4,]  0.0  0.5  0.0  0.0  0.0  0.0  0.0  0.0
#>  [5,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
#>  [6,]  0.0  0.0  0.5  0.0  0.0  0.0  0.0  0.0
#>  [7,]  0.0  0.0  0.0  0.5  0.0  0.0  0.0  0.0
#>  [8,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
#>  [9,]  0.0  0.0  0.0  0.0  0.5  0.0  0.0  0.0
#> [10,]  0.0  0.0  0.0  0.0  0.0  0.5  0.0  0.0
#> [11,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
#> [12,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
#> [13,]  0.0  0.0  0.0  0.0  0.0  0.0  0.5  0.0
#> [14,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.5
#> [15,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
#> [16,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
#> [17,]  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
## this shows a total of eight translocations, each involving a
## single block. In all cases, blocks are tagged with 0.5,
## indicating that it could not be distinguished which part of
## a translocation would be more parsimonious to have changed
## position relative to the alternative part (e.g., for the
## translocation between the third and fourth block on
## chromosome 3R, described above, it cannot be distinguished
## whether node element 1346 has moved one block up, or element
## 1345 one block down)
```

3 Visualizing rearrangements

3.1 The `genomeImagePlot()` function

The `genomeImagePlot()` function visualizes rearrangements that were detected with the `computeRearrs()` function along the focal genome. Markers that are part of different classes of rearrangements are tagged with different colors at their map position, allowing the visual examination of the extent, number, and location of rearrangements. As an example, the Figure 4 visualizes rearrangements in the *D. melanogaster* genome that occurred after divergence from the *Drosophila* ancestor *MSSYE*.

The function requires as input files the output of the `computeRearrs()` function (i.e., the list `SYNT`), the data frame `focalgenome`, and the IDs of the focal genome segments that should be plotted (the character vector `ordfocal`). The list `SYNT` may have optionally been filtered with the `filterRearrs()` function.

The graphic can directly be output as PDF with the argument `makepdf = TRUE` (which can be much faster than plotting to screen). By default, the function automatically determines the optimal width and height of the plot (this can be turned off by setting both `makepdf = FALSE` and `newdev = FALSE`). Various function arguments allow to adjust the appearance of the graphic, including font sizes, axes labels, and plot margins. See `?genomeImagePlot` for more information and examples on the usage of the function.

Note: The automatic determination of the optimal width and height of the graphic is not possible when using the RStudio graphical device (`RStudioGD`) as default, as it does not accept width and height as arguments. In this case, the plot is send to an alternative device. (Only relevant when `makepdf = FALSE`.)

```
## Example for determining and changing the default graphics device

## determine current default graphics device
getOption("device")
## see list of available graphics devices
?Devices
## for example, set new default to X11
options(device = "X11")
```

```
## Example for visualizing rearrangements in the D. melanogaster
## genome that occurred after divergence from the Drosophila
## ancestor 'MSSYE'. The large maroon sector reveals a derived
## inversion on chromosome 3R, in line with previous work
## (Ranz et al., 2007)
genomeImagePlot(SYNT_MEL_MSSYE, MEL_markers,
                c("2L", "2R", "3L", "3R", "X"),
                main = "D. melanogaster - MSSYE")
```

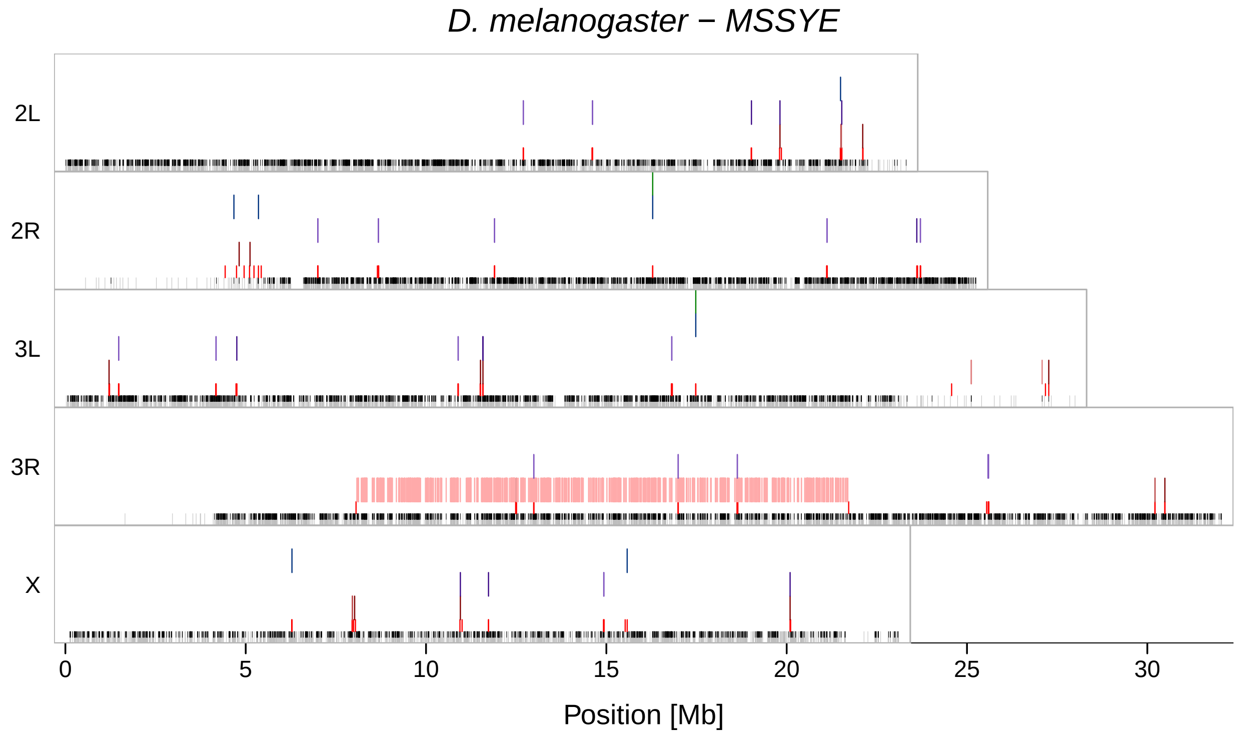

Figure 4: Graphical output of the genomeImagePlot() function

Output: On the y-axis, each focal genome segment that was included in `ordfocal` is plotted from top to bottom. Marker positions are on the x-axis. Vertical lines (ticks) indicate the positions of markers, rearrangement breakpoints, and different classes of rearrangements within six rows per focal segment.

The bottom row shows the positions of all markers contained in `focalgenome` as gray ticks, and the positions of markers present in `SYNT` as black ticks in the upper half of the bottom row. The second row from the bottom gives the positions of rearrangement breakpoints by red ticks. Ticks are drawn at the midpoints between the positions of the two markers present in `SYNT` that are adjacent to the breakpoints. Only markers with black ticks can receive colored ticks in the remaining rows, which highlight markers that are part of different classes of rearrangements. Maroon indicates markers that are part of inversions within CARs within focal genome segments (`IV`); purple indicates markers that are part of translocations within CARs within focal genome segments (`TLWC`); blue indicates markers that are part of translocations between CARs within focal genome segments (`TLWS`); green indicates markers that are part of translocations between CARs between focal genome segments (`TLBS`). See the description of the `rearrvisr` algorithm in the Supplementary information of the associated manuscript or `?computeRearrs` for more information on these classes of rearrangements.

Unless the argument `remThld` is set to a value smaller than that of `remWgt` in the `computeRearrs()` function, only markers that are less parsimonious to have changed position relative to alternative markers are highlighted. Lighter tints denote markers that are part of large rearrangements, while darker shades denote markers that are part of small rearrangements. To distinguish between individual large rearrangements versus multiple short adjacent rearrangements, it may be helpful to take the position of breakpoints into account.

3.2 The `genomeRearrPlot()` function

The `genomeRearrPlot()` function visualizes the output of the `summarizeBlocks()` function along the focal genome. The produced plot may be used to reconstruct how the compared genome segments were aligned to focal genome segments, to notice the structure of the aligned *PQ-trees*, and to comprehend why particular synteny blocks were tagged as rearranged by the `computeRearrs()` function. As an example, the Figure 5 visualizes rearrangements in the *D. melanogaster* genome that occurred after divergence from the *Drosophila* ancestor *MSSYE*.

The function requires as input files the output of the `summarizeBlocks()` function (i.e., the list `BLOCKS`), the data frame `compgenome`, and the IDs of the focal genome segments that should be plotted (the character vector `ordfocal`).

The graphic can directly be output as PDF with the argument `makepdf = TRUE` (which can be much faster than plotting to screen). By default, the function automatically determines the optimal width and height of the plot (this can be turned off by setting both `makepdf = FALSE` and `newdev = FALSE`). Various function arguments allow to adjust the appearance of the graphic, including font sizes, axes labels, and plot margins. See `?genomeRearrPlot` for more information and examples on the usage of the function.

Note: The automatic determination of the optimal width and height of the graphic is not possible when using the RStudio graphical device (`RStudioGD`) as default, as it does not accept width and height as arguments. In this case, the plot is send to an alternative device. (Only relevant when `makepdf = FALSE`.) For an example on how to determine and change the default graphics device, see the section on the `genomeImagePlot()` function above.

```r
## Example for visualizing rearrangements in the D. melanogaster
## genome that occurred after divergence from the Drosophila
## ancestor 'MSSYE' and that were summarized with the
## summarizeBlocks() function. Note that CAR 1 joins chromosomes
## 3L and 3R of D. melanogaster
genomeRearrPlot(BLOCKS_MEL_MSSYE, MSSYE_compgenome,
                c("2L", "2R", "3L", "3R", "X"),
                main = "D. melanogaster - MSSYE",
                blockwidth = 1.15, y0pad = 3)
```

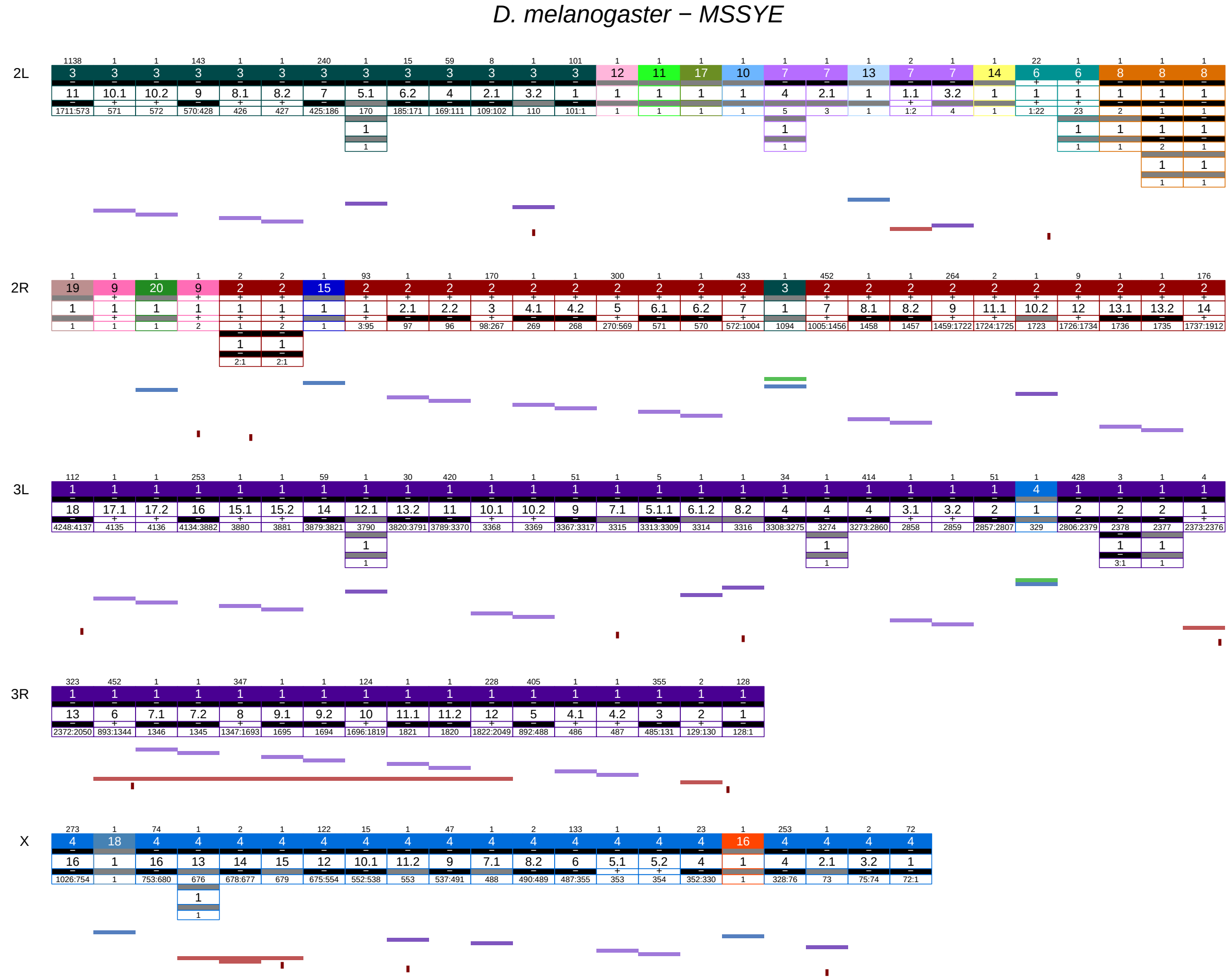

Figure 5: Graphical output of the genomeRearrPlot() function

Note: The function argument `y0pad` sets the amount of additional space between the bottom plot margin and the bottom plot area. Setting `y0pad` too small may result in some rearrangements for the bottom-most focal segment to be invisible because they will be outside the bottom plot area. This can be avoided by setting `y0pad` sufficiently large.

Output: On the y-axis, each focal genome segment that was included in `ordfocal` is plotted from top to bottom. Each synteny block is represented by a column (note that separate blocks are also generated when the hierarchical structure of the underlying *PQ-tree* changes, therefore not all column boundaries are caused by a rearrangement).

The top part of each focal segment visualizes the data contained in the `$blocks` data frame in `BLOCKS` (i.e., information on the structure of each *PQ-tree* and its alignment to the focal genome). The top row gives the number of markers within each synteny block, calculated from the *start* and *end* columns in the `$blocks` data frame. The second row gives the IDs of the CARs, corresponding to entries in the *car* column in the `$blocks` data frame (different colors distinguish CARs).

For each hierarchy level, up to four additional rows (depending on the values in the argument `plotelem`) show (*i*) the alignment orientation of the *PQ-tree* node to the focal genome (white rectangles and + indicating ascending, and black rectangles and `-` indicating descending alignment), (*ii*) the block IDs, (*iii*) the block orientation within its node (white rectangles and `+` indicating ascending, and black rectangles and `-` indicating descending orientation), and (*iv*) the range of element IDs for each block within its node and for its level of hierarchy. These four rows correspond to entries in the (*i*) *nodeori**, (*ii*) *blockid**, (*iii*) *blockori**, and (*iv*) *elemS** – *elemE** columns in the `$blocks` data frame. See the `summarizeBlocks()` function above for a description of the `$blocks` data frame (using chromosome 3R of *D. melanogaster* as an example), and `?genomeRearrPlot` for further details.

The bottom part of each focal segment visualizes different classes of rearrangements in the matrices `$TLBS`, `$TLWS`, `$TLWC`, `$IV`, and `$IVsm` in `BLOCKS`. See the description of the `rearrvisr` algorithm in the Supplementary information of the associated manuscript or the `?computeRearrs` documentation for more information on these classes of rearrangements. In contrast to the `genomeImagePlot()` function, the `genomeRearrPlot()` function allows to visualize rearrangements that are nested, and to distinguish between individual large rearrangements versus multiple short adjacent rearrangements.

Horizontal lines that are at identical height denote the same rearrangement (potentially disrupted by inserted CARs). Green indicates blocks that are part of translocations between CARs between focal genome segments (`TLBS`); blue indicates blocks that are part of translocations between CARs within focal genome segments (`TLWS`); purple indicates blocks that are part of translocations within CARs within focal genome segments (`TLWC`); maroon indicates blocks that are part of inversions within CARs within focal genome segments (`IV`). Single-marker inversions (`IVsm`) are indicated by a short vertical line at their position within their block, while multi-marker inversions are indicated by a horizontal line spanning the contributing block(s).

Lighter coloration denotes smaller weights for rearrangement tags in the respective matrices in `BLOCKS`. Unless the argument `remThld` is set to a value smaller than that of `remWgt` in the `computeRearrs()` function, only lines for blocks that are less parsimonious to have changed position relative to alternative blocks are plotted.

4 A brief walk through the main steps of the `rearrvisr` algorithm

This section describes the main steps of the `rearrvisr` algorithm on a simple data set. A detailed description can be found in the Supplementary information of the associated manuscript.

4.1 Example data

The example data `TOY24_focalgenome` and `TOY24_compgenome` were created for illustrative purposes and are included in the package. They can be regenerated with the following commands:

```
## Create a simple example data set

## focalgenome
TOY24_focalgenome <- data.frame(
  marker = as.integer(c(1,7,2,6:4,8:10,3,13:11,14,17:15,18,21,20,22:24,19)),
  scaff = as.character(rep(c(1,2,3), times = c(7,11,6))),
  start = as.integer(c(seq(10^6, by = 10^6, length.out = 7),
                       seq(10^6, by = 10^6, length.out = 11),
                       seq(10^6, by = 10^6, length.out = 6))),
  end = as.integer(c(seq(10^6+2, by = 10^6, length.out = 7),
                     seq(10^6+2, by = 10^6, length.out = 11),
                     seq(10^6+2, by = 10^6, length.out = 6))),
  strand = rep(rep(c("+", "-"), 4),
               times = c(4,2,4,3,8,1,1,1)),
  stringsAsFactors = FALSE)
```

```
## compgenome
TOY24_rawtree <- matrix(
  c(">TOY",
    "#CAR1",
    "_Q 1 2 3 4 5 -6 7 8 Q_",
    "#CAR2",
    "_Q 9 _Q 10 Q_ _Q 11 12 13 Q_ Q_",
    "#CAR3",
    "_Q 14 Q_",
    "#CAR4",
    "_Q _P 15 16 17 18 P_ _Q _Q -19 20 21 Q_ _Q -22 -23 24 Q_ Q_ Q_"),
  nrow = 9)
TOY24_compgenome <- convertPQtree(TOY24_rawtree)
```

4.2 Overview

Rearrangements are identified by aligning the compared genome representation in `TOY24_compgenome` (i.e., the *PQ-structure*, which is the data frame joining all *PQ-trees*, or CARs, of the compared genome) to the sorted genome map of the focal species in `TOY24_focalgenome` (Figure 6A and B; where 'sorted' means that the sequence of markers within each focal genome segment is ordered by map position). Only the set of markers common to both genomes is considered. Markers (orthologous genes or synteny blocks) are called *doubled* if orientation information is available for the focal and the compared genome, for example when a genome reconstruction was obtained with the software ANGES (Jones *et al.*, 2012) applying the option `markers_doubled 1` (Ouangraoua *et al.*, 2011).

In practice, the alignment of the markers in `TOY24_compgenome` to their orthologs in `TOY24_focalgenome` is achieved by ordering the rows in `TOY24_compgenome` so that the sequence of markers is identical to the sequence of markers in `TOY24_focalgenome`.

```
## PQ-structure in 'TOY24_compgenome' ordered so that the
## sequence of markers corresponds to the sequence of markers
## in the focal genome in 'TOY24_focalgenome' (i.e.,
## 'TOY24_compgenome' is "aligned" to 'TOY24_focalgenome'; see
## Figure A and B for a graphical representation of the
## alignment)

TOY24_compgenome[match(TOY24_focalgenome$marker,
                       TOY24_compgenome$marker), ]
#>    marker orientation car type1 elem1 type2 elem2 type3 elem3
#> 1       1           +   1     Q     1  <NA>    NA  <NA>    NA
#> 7       7           +   1     Q     7  <NA>    NA  <NA>    NA
#> 2       2           +   1     Q     2  <NA>    NA  <NA>    NA
#> 6       6           -   1     Q     6  <NA>    NA  <NA>    NA
#> 5       5           +   1     Q     5  <NA>    NA  <NA>    NA
#> 4       4           +   1     Q     4  <NA>    NA  <NA>    NA
#> 8       8           +   1     Q     8  <NA>    NA  <NA>    NA
#> 9       9           +   2     Q     1  <NA>    NA  <NA>    NA
#> 10     10           +   2     Q     2     Q     1  <NA>    NA
#> 3       3           +   1     Q     3  <NA>    NA  <NA>    NA
#> 13     13           +   2     Q     3     Q     3  <NA>    NA
#> 12     12           +   2     Q     3     Q     2  <NA>    NA
#> 11     11           +   2     Q     3     Q     1  <NA>    NA
#> 14     14           +   3     Q     1  <NA>    NA  <NA>    NA
#> 17     17           +   4     Q     1     P     3  <NA>     NA
#> 16     16           +   4     Q     1     P     2  <NA>     NA
#> 15     15           +   4     Q     1     P     1  <NA>     NA
#> 18     18           +   4     Q     1     P     4  <NA>     NA
#> 21     21           +   4     Q     2     Q     1     Q      3
#> 20     20           +   4     Q     2     Q     1     Q      2
#> 22     22           -   4     Q     2     Q     2     Q      1
#> 23     23           -   4     Q     2     Q     2     Q      2
#> 24     24           +   4     Q     2     Q     2     Q      3
#> 19     19           -   4     Q     2     Q     1     Q      1
```

The alignment potentially disrupts the original sequence of *PQ-structure* elements (i.e., node element IDs) at one or more hierarchy levels (i.e., in columns $car*, *elem1$, $elem2*, or*elem3$). Such disruptions of the contiguity of aligned node element IDs indicate rearrangements. Each hierarchy level and each node is checked separately for rearrangements, starting at the CAR level and then descending through the hierarchy levels. The definition of a disrupted sequence of node element IDs differs among CARs, *P-nodes*, and *Q-nodes*. Briefly, for CARs and *P-nodes*, where children are in arbitrary order, the sequence of node element IDs does not need to be ordered, but the presence of *scattered elements* indicates a rearrangement (e.g., element `2` within the sequence `2 2 1 2 2 2` is *scattered* and indicates that the coherence of markers descending from child `2` is disrupted by markers descending from child `1`). For *Q-nodes*, the sequence of node element IDs needs to be ordered in either ascending or descending direction, any deviation from this (e.g., `3 1 2`) indicates a rearrangement (a translocation or an inversion).

If markers are doubled, their *directionality* in the focal genome relative to the compared genome is taken into account for the detection and classification of rearrangements within *Q-nodes*. The directionality can be identical (then the strand in both genomes is either `"+"` or `"-"`) or inverted (then the strand is `"+"` in one genome but is `"-"` in the other). Note that if the hierarchically lowest node of a given marker in the corresponding *PQ-tree* does not contain any other branches or markers (a *single-marker* node), the directionality is irrelevant, as nodes can be reversed. The directionality is also irrelevant for the detection of rearrangements at the CAR level or for *P-nodes*.

While markers part of an inversion can be unambiguously delimitated, translocations between sets of markers are only defined relative to each other (e.g., within the sequence `2 2 1 2 2 2`, the first two elements `2` may have been translocated from original positions two and three to new positions one and two, or element `1` may have been translocated from original position one to its new position three). In many cases it cannot be distinguished which component of a translocation (i.e., which set of elements) was causal to a detected rearrangement. This uncertainty is addressed by assigning different tag values to alternative translocation components based on the principle of parsimony, where larger values indicate elements that are more parsimonious to have been translocated.

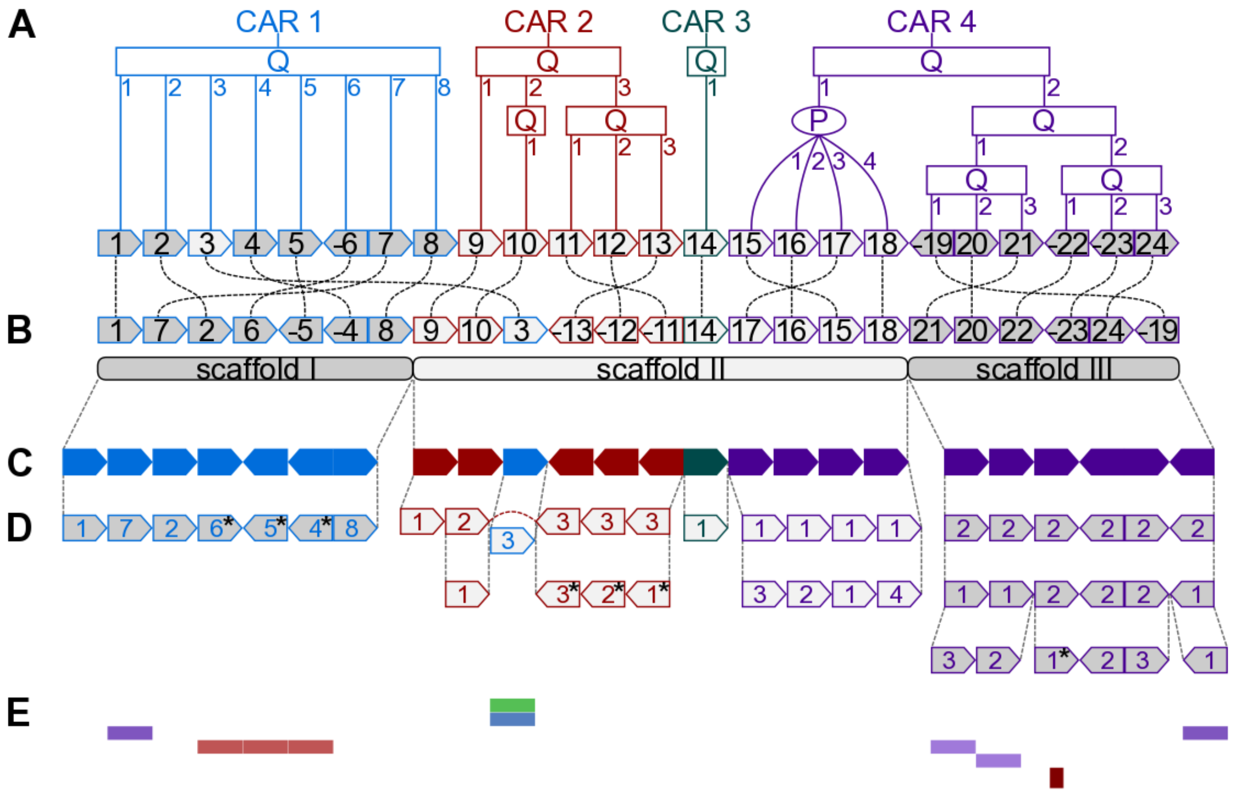

Figure 6: Rearrangement identification steps.

Figure 6: Alignment of the compared genome representation (i.e., *PQ-structure*) in TOY24_compgenome to the focal genome map in TOY24_focalgenome, and hierarchical subdivision of the aligned *PQ-structure* for rearrangement identification. **A**, graphical representation of the *PQ-structure* in TOY24_compgenome in its original order. *PQ-trees* of four CARs are shown in colors. *P-nodes* are illustrated with ovals and *Q-nodes* with rectangles. Marker IDs are given by black integers within pentagons at the leaves of the *PQ-trees*. All branches are labeled with a colored integer ID (i.e., the node element ID) according to their position at their node of origin, starting at 1 for each node and hierarchy level. **B**, graphical representation of the focal genome map in TOY24_focalgenome, ordered by map position. Markers are shown as pentagons, filled with different grays according to their focal segment (scaffolds I – III) of origin. Black dashed lines show the alignment between the *PQ-structure* and the focal genome map. **C**, subdivision of the aligned *PQ-structure* by focal segments at CAR level (i.e., hierarchy level 1). **D**, subdivision of the aligned *PQ-structure* by focal segments at node levels (i.e., hierarchy levels 2 – 4). Hierarchy levels are shown in rows. The colored integers within pentagons are node element IDs of the aligned *PQ-trees*. Asterisks indicate markers that have inverted directionality in the focal genome relative to the compared genome. Subdivisions (based on identical elements at higher hierarchy levels) are indicated by small vertical gaps between markers and connected to the preceding hierarchy by gray dashed lines. For the red CAR (CAR 2), the red dashed line indicates that node elements 1 2 3 3 3 are joined across the inserted marker from the blue CAR (CAR 1) for rearrangement detection at the node level. **E**, tags for rearranged markers. The green bar indicates a translocation between CARs between focal genome segments (TLBS); the blue bar indicates a translocation between CARs within focal genome segments (TLWS); purple bars indicate markers that are part of translocations within CARs within focal genome segments (TLWC); maroon bars indicate markers that are part of inversions within CARs within focal genome segments (IV). A single-marker inversion (IVsm) is indicated by a short vertical bar, while multi-marker inversions are indicated by horizontal bars. Lighter coloration for purple bars for markers 21 and 20 on scaffold III indicate half-weight tag values, as it cannot be distinguished which of the two is more parsimonious to have been translocated. Bars are only shown for tag values greater than remWgt.

4.3 Tranlocations between CARs between focal segments (TLBS)

A CAR is identified as translocated if it was aligned to more than one focal genome segment in a way that is incompatible with a potentially incomplete focal genome assembly. Tests are performed by considering all focal genome segments jointly. The algorithm takes into account that for the focal genome, not all scaffolds may have been placed within a chromosome. Likewise, CARs are only assumed to be contiguous sets of genetic markers, but do not need to represent full (ancestral) chromosomes. For example, CAR 4 on scaffolds II and III in Figure 6C is not identified as TLBS, as the CAR can join the ends of the two scaffolds. By contrast, the fragments of CAR 1 were aligned to scaffolds I and II in a way that is incompatible with a potentially incomplete focal genome assembly (i.e., the smaller fragment is located in the middle of scaffold II), indicating a TLBS. (Note that for the case where a large fragment of a CAR is located in the middle of a scaffold, and a small fragment of the same CAR fully spans a small scaffold, a TLBS will not be assigned, as the small fragment might in fact be placed in a gap between contigs on the large genome segment.)

```
## Reconstruct the alignment of TOY24_compgenome to
## TOY24_focalgenome

TOY24_aligned <- merge(TOY24_focalgenome, TOY24_compgenome,
  by = "marker", sort = FALSE)
```

```
## Show alignment of CARs in TOY24_compgenome to
## scaffolds in TOY24_focalgenome

TOY24_aligned[,c(1,2,7)]
#>    marker scaff car
#> 1       1     1   1
#> 2       7     1   1
#> 3       2     1   1
#> 4       6     1   1
#> 5       5     1   1
#> 6       4     1   1
#> 7       8     1   1
#> 8       9     2   2
#> 9      10     2   2
#> 10      3     2   1
#> 11     13     2   2
#> 12     12     2   2
#> 13     11     2   2
#> 14     14     2   3
#> 15     17     2   4
#> 16     16     2   4
#> 17     15     2   4
#> 18     18     2   4
#> 19     21     3   4
#> 20     20     3   4
#> 21     22     3   4
#> 22     23     3   4
#> 23     24     3   4
#> 24     19     3   4
```

```
## Identify rearrangements in TOY24_focalgenome
## relative to TOY24_compgenome

SYNT <- computeRearrs(TOY24_focalgenome, TOY24_compgenome,
  doubled = TRUE)
```

```
## TLBS identified in TOY24_focalgenome relative to
## TOY24_compgenome

SYNT$TLBS
#>    [,1]
#> 1     0
#> 7     0
#> 2     0
#> 6     0
#> 5     0
#> 4     0
#> 8     0
#> 9     0
#> 10    0
#> 3     1
#> 13    0
#> 12    0
#> 11    0
#> 14    0
#> 17    0
#> 16    0
#> 15    0
#> 18    0
#> 21    0
#> 20    0
#> 22    0
#> 23    0
#> 24    0
#> 19    0
```

This shows that marker `3` was identified as `TLBS` (i.e., it has a tag value of `1`), as the marker is part of CAR 1 that was aligned to both scaffolds I and II (see Figure 6C). As the smaller fragment of CAR 1 is located in the middle of scaffold II, it cannot join with the other fragment of CAR 1 on scaffold I and thus must be involved in a rearrangement. By contrast, markers part of CAR 4 are not identified as `TLBS` (i.e., tag values are `0`), as the CAR can join the ends of scaffolds II and III.

4.4 Tranlocations between CARs within focal segments (`TLWS`)

A CAR is identified as translocated if its contiguity within a focal genome segment is disrupted when aligned to the focal genome. Tests are performed separately for each focal genome segment. Again, CARs are only assumed to be contiguous sets of genetic markers, but do not need to represent full (ancestral) chromosomes. For example, CAR 3 and CAR 4 on scaffold II in Figure 6C are not identified as `TLWS`, as the CARs may be part of a larger contiguous block that was split into smaller CARs for technical reasons (i.e., analogous to small contigs that could not be joined to larger contigs in a genome assembly). By contrast, the contiguity of CAR 2 on scaffold II is disruped by a fragment of CAR 1, indicating a `TLWS`.

```
## TLWS identified in TOY24_focalgenome relative to
## TOY24_compgenome

SYNT$TLWS
#>    [,1]
#> 1     0
#> 7     0
#> 2     0
#> 6     0
#> 5     0
#> 4     0
#> 8     0
#> 9     0
#> 10    0
#> 3     1
#> 13    0
#> 12    0
#> 11    0
#> 14    0
#> 17    0
#> 16    0
#> 15    0
#> 18    0
#> 21    0
#> 20    0
#> 22    0
#> 23    0
#> 24    0
#> 19    0
```

This shows that marker `3` was identified as `TLWS` (i.e., it has a tag value of `1`), as it is part of CAR 1 that disrupts the contiguity of CAR 2 on scaffold II (see Figure 6C).

4.5 Tranlocations (`TLWC`) or inversions (`IV`) within CARs

A CAR is identified as translocated or inverted if its contiguity within itself is disrupted when aligned to a focal genome segment. Tests are performed separately for each focal genome segment. Starting at the hierarchically highest and then descending to the lowest nodes, tests are performed for each node separately and differ between *P-nodes* and *Q-nodes*. For example, in Figure 6D, only CAR 1 with a single node was aligned to scaffold I, thus only one test needs to be performed.

```
## Print hierarchically highest node elements for CAR 1 on
## scaffold I

TOY24_aligned$elem1[TOY24_aligned$scaff == "1" &
    TOY24_aligned$car == 1]
#> [1] 1 7 2 6 5 4 8
```

This shows the order of elements in column *$elem1* is not consistent with a *Q-node*, indicating rearrangements (see Figure 6A, B, and D).

```
## Translocations identified scaffold I

SYNT$TLWC[TOY24_aligned$scaff == "1", , drop = FALSE]
#>   [,1] [,2] [,3]
#> 1 0.00 0.00    0
#> 7 0.95 0.00    0
#> 2 0.00 0.05    0
#> 6 0.00 0.05    0
#> 5 0.00 0.05    0
#> 4 0.00 0.05    0
#> 8 0.00 0.00    0
```

This shows that one `TLWC` was identified that involves marker `7` (tagged with `0.95`, i.e., `1 - remWgt`) and the block of markers `2`, `6`, `5`, and `4` (tagged with `0.05`, i.e., `remWgt`). The larger tag value for marker `7` indicates that this element is more parsimonious to have been translocated (i.e., it is more parsimonious that marker `7` has moved to the left than that markers `2`, `4`, `5`, and `6` have moved to the right; see Figure 6A and B). The third column only contains zeros and is used for padding columns of other scaffolds.

```
## Inversions identified on scaffold I

SYNT$IV[TOY24_aligned$scaff == "1", , drop = FALSE]
#>   [,1]
#> 1    0
#> 7    0
#> 2    0
#> 6    1
#> 5    1
#> 4    1
#> 8    0
```

This shows that one `IV` was identified that involved markers `6`, `5`, and `4` (see Figure 6A and B).

For scaffold II in Figure 6D, both CAR 2 and the fragment of CAR 4 need to be tested for rearrangements. CAR 2 has three nodes and CAR 4 has one node aligned to scaffold II, thus at least four tests need to be performed.

```
## Print hierarchically highest node elements for CAR 2 on
## scaffold II

TOY24_aligned$elem1[TOY24_aligned$scaff == "2" &
    TOY24_aligned$car == 2]
#> [1] 1 2 3 3 3
```

This shows that the order of elements in column *$elem1* is consistent with a *Q-node* (see Figure 6A, B, and D).

```
## Print node elements from second branch of the hierarchically
## highest node (i.e., the first node at the second-highest
## hierarchy level)

TOY24_aligned$elem2[TOY24_aligned$scaff == "2" &
    TOY24_aligned$car == 2 & TOY24_aligned$elem1 == 2]
#> [1] 1
```

This shows that only one marker descends from the node, thus it cannot be rearranged.

```
## Print node elements from third branch of the hierarchically
## highest node (i.e., the second node at the second-highest
## hierarchy level)

TOY24_aligned$elem2[TOY24_aligned$scaff == "2" &
    TOY24_aligned$car == 2 & TOY24_aligned$elem1 == 3]
#> [1] 3 2 1
```

This shows that the order of elements in column *$elem2* is consistent with a *Q-node*, as the order can be reversed.

```
## Print hierarchically highest node elements for the fragment
## of CAR 4 on scaffold II

TOY24_aligned$elem1[TOY24_aligned$scaff == "2" &
    TOY24_aligned$car == 4]
#> [1] 1 1 1 1
```

This shows that all markers are located on the same branch and thus cannot be rearranged.

```
## Print node elements from first branch of the hierarchically
## highest node (i.e., the node at the second-highest hierarchy
## level that was aligned to scaffold II)

TOY24_aligned$elem2[TOY24_aligned$scaff == "2" &
    TOY24_aligned$car == 4 & TOY24_aligned$elem1 == 1]
#> [1] 3 2 1 4
```

This shows that the order of elements in column *$elem2* is consistent with a *P-node*, where the order is arbitrary.

```
## Rearrangements identified on scaffold II

SYNT$TLWC[TOY24_aligned$scaff == "2", , drop = FALSE]
#>    [,1] [,2] [,3]
#> 9     0    0    0
#> 10    0    0    0
#> 3     0    0    0
#> 13    0    0    0
#> 12    0    0    0
#> 11    0    0    0
#> 14    0    0    0
#> 17    0    0    0
#> 16    0    0    0
#> 15    0    0    0
#> 18    0    0    0

SYNT$IV[TOY24_aligned$scaff == "2", , drop = FALSE]
#>    [,1]
#> 9     0
#> 10    0
#> 3     0
#> 13    0
#> 12    0
#> 11    0
#> 14    0
#> 17    0
#> 16    0
#> 15    0
#> 18    0
```

This shows that no `TLWC` and no `IV` was identified. Although markers `-13`, `-12`, and `-11` on scaffold II appear to be inverted relative to markers `11`, `12`, and `13` on CAR 2 (see Figure 6A and B), they are all on the same *Q-node*, where the order can be reversed. Similarly, markers 17, 16, and 15 on scaffold II appear to be rearranged relative to markers 15, 16, and 17 on CAR 4, but as they are on a *P-node*, their order is arbitrary.

For scaffold III in Figure 6D, only the fragment of one CAR was aligned and needs to be tested. The fragment of CAR 4 on scaffold III has three nodes, thus at least three tests need to be performed.

```
## Print hierarchically highest node elements for CAR 4 on
## scaffold III

TOY24_aligned$elem1[TOY24_aligned$scaff == "3" &
    TOY24_aligned$car == 4]
#> [1] 2 2 2 2 2 2
```

This shows that all markers are located on the same branch and thus cannot be rearranged.

```
## Print node elements from second branch of the hierarchically
## highest node (i.e., the node at the second-highest hierarchy
## level that was aligned to scaffold III)

TOY24_aligned$elem2[TOY24_aligned$scaff == "3" &
    TOY24_aligned$car == 4 & TOY24_aligned$elem1 == 2]
#> [1] 1 1 2 2 2 1
```

This shows that the order of elements in column *$elem2* is not consistent with a *Q-node*, indicating rearrangements (see Figure 6A, B, and D). Specifically, the contiguity of elements of the *Q-node* on branch 1 is disrupted by elements of the *Q-node* on branch 2. In this specific example, the last element (i.e., marker -19 in Figure 6B) will be identified as rearranged. As the marker is part of a node that still needs to be tested for rearrangements at the next hierarchy level, the computeRearrs() function will by default perform tests separately for the two node fragments (i.e., for markers 21 and 20, and for marker -19) at the following hierarchy levels. This behavior is controlled by the splitnodes argument in the computeRearrs() function.

```
## Create subnode IDs for the two node fragments
## (for illustration of the splitnodes = TRUE setting)

TOY24_aligned$subnode <- numeric(nrow(TOY24_aligned))
TOY24_aligned$subnode[TOY24_aligned$marker == 19] <- 1
```

```
## Print node elements from first branch of the hierarchically
## second-highest node (i.e., the first node at the third-highest
## hierarchy level) for the first subnode

TOY24_aligned$elem3[TOY24_aligned$scaff == "3" &
    TOY24_aligned$car == 4 & TOY24_aligned$elem1 == 2 &
    TOY24_aligned$elem2 == 1 & TOY24_aligned$subnode == 0]
#> [1] 3 2
```

This shows that exactly two branches descend from the first subnode of the *Q-node*, requiring special treatment for the detection and classification of rearrangements. In this specific example, one can (i) reverse the order of elements, not requiring a rearrangement, or (ii) not reversing the order, requiring a rearrangement. However when testing the next hierarchy level for rearrangements (in this particular case the test is for single-marker inversions, see below), two inversions are required for case (i), as a reversal of the order of elements implicates a reversal of their directionality, which is not realized (i.e., markers 20 and 21 are both on the "+" strand in either genome; see Figure 6A and B). For case (ii), no additional reversals are required. Thus, the order of the *Q-node* is not reversed, and a translocation is assigned.

```
## Print node elements from first branch of the hierarchically
## second-highest node (i.e., the first node at the third-highest
## hierarchy level) for the second subnode

TOY24_aligned$elem3[TOY24_aligned$scaff == "3" &
    TOY24_aligned$car == 4 & TOY24_aligned$elem1 == 2 &
    TOY24_aligned$elem2 == 1 & TOY24_aligned$subnode == 1]
#> [1] 1
```

This shows that only one marker descends from the second subnode, thus it cannot be rearranged in addition to the already identified rearrangement at the preceding hierarchy level.

```
## Print node elements from second branch of the hierarchically
## second-highest node (i.e., the second node at the third-highest
## hierarchy level)

TOY24_aligned$elem3[TOY24_aligned$scaff == "3" &
    TOY24_aligned$car == 4 & TOY24_aligned$elem1 == 2 &
    TOY24_aligned$elem2 == 2]
#> [1] 1 2 3
```

This shows that the order of elements in column *$elem3* is consistent with a *Q-node*.

```
## Translocations identified on scaffold III

SYNT$TLWC[TOY24_aligned$scaff == "3", , drop = FALSE]
#>    [,1] [,2] [,3]
#> 21    0  0.5  0.0
#> 20    0  0.0  0.5
#> 22    0  0.0  0.0
#> 23    0  0.0  0.0
#> 24    0  0.0  0.0
#> 19    1  0.0  0.0
```

This shows that two `TLWC` were identified. The first is for marker `19` (tagged with `1`). The second involves markers `21` and `20` (both tagged with `0.5`). Having half-weight tags for either marker indicates that it cannot be distinguished which of the two elements is more parsimonious to have been translocated (i.e., whether it is more parsimonious that marker `21` has moved to the left or marker `20` has moved to the right; see Figure 6A and B).

```
## Inversions identified on scaffold III

SYNT$IV[TOY24_aligned$scaff == "3", , drop = FALSE]
#>    [,1]
#> 21    0
#> 20    0
#> 22    1
#> 23    0
#> 24    0
#> 19    0
```

This shows that one `IV` was identified that involved marker `22` (i.e., *single-marker inversion*; see Figure 6A and B). Single-marker inversions signify markers whose directionality deviates from their expected directionality, given the final alignment direction of all *Q-nodes* traversed from the root the marker, and the number of (nested) inversions that were assigned to the marker. All traversed *Q-nodes* for marker `22` were aligned in ascending direction and no inversion was assigned, thus the strand of marker `22` is expected to be identical in both genomes (i.e., both strands are expected to be "+" or both to be "-"). As this is not the case, a single-marker inversion was assigned.

4.6 Graphical output

```
## Visualize the identified rearrangements with the
## genomeImagePlot() function

genomeImagePlot(SYNT, TOY24_focalgenome,
                c("1", "2", "3"), main = "TOY24",
                yaxlab = c("I", "II", "III"))
```

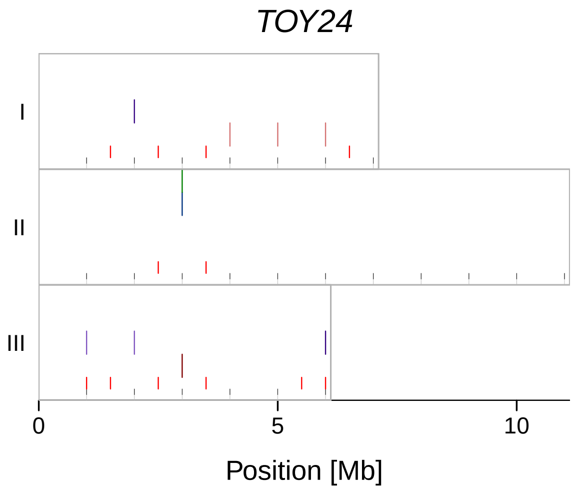

Figure 7: Graphical output of the genomeImagePlot() function

Figure 7: The three focal genome segments are shown as separate panels. The x-axis gives the position of markers on their genome segments. Markers are indicated with short black ticks in the bottom row of each panel. Rearrangement breakpoints are indicated with red ticks and are drawn at the midpoints between the markers adjacent to the breakpoints. Markers part of different classes of rearrangements are highlighted in the upper four rows of each panel by colored ticks. From top to bottom: `TLBS` (green), `TLWS` (blue), `TLWC` (purple), and `IV` (maroon). Ticks are only shown for tag values greater than `remWgt`.

```
## Visualize the identified rearrangements with the
## genomeRearrPlot() function

## compute blocks
BLOCKS <- summarizeBlocks(SYNT, TOY24_focalgenome,
                          TOY24_compgenome, c("1","2","3"))

## specify colors
carcol <- c("#006DDB","#920000","#004949","#490092")
textcol <- rep("white", length(carcol))

## make plot
genomeRearrPlot(BLOCKS, TOY24_compgenome, c("1","2","3"),
                main = "TOY24", yaxlab = c("I", "II", "III"),
                blockwidth = 1.15, yOpad = 3,
                carColors = carcol, carTextColors = textcol,
                sortColsBySize = FALSE)
```

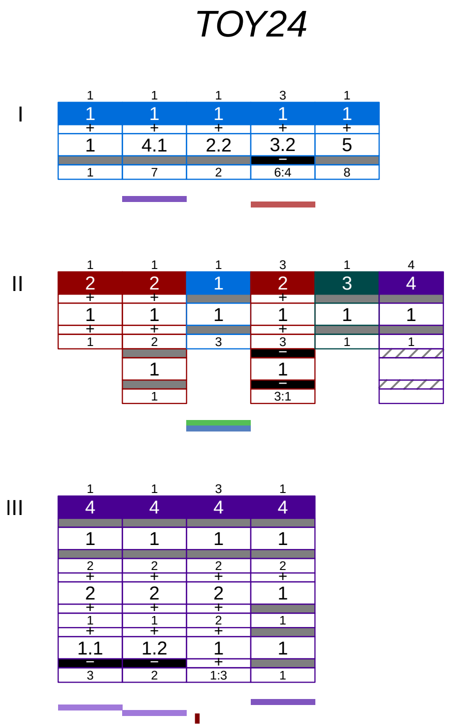

Figure 8: Graphical output of the genomeRearrPlot() function

Figure 8: The three focal genome segments are shown as separate panels. Each synteny block is represented by a column along the x-axis. The top part of each panel shows information on the structure of each *PQ-tree* and its alignment to the focal genome, and the bottom part highlights blocks part of different classes of rearrangements by colored bars. Green, `TLBS`. Blue, `TLWS`. Purple, `TLWC`. Maroon, `IV` (with a horizontal bar for the multi-marker inversion, and a short vertical bar for the single-marker inversion). Lighter coloration for purple bars for scaffold III indicate half-weight tag values. Bars are only shown for tag values greater than `remWgt`. See `?genomeRearrPlot` for further details on the top part of the panels.

5 *Drosophila* data set

5.1 Data preparation

Peptide sequences and annotation information (`pep.all.fa` and `gff3` files) for genes from 12 *Drosophila* species were downloaded on Dec 23 2017 from Ensemble Release 91 (http://dec2017.archive.ensembl.org; *D. melanogaster*) or Ensemble Metazoa Release 37 (http://oct2017-metazoa.ensembl.org; *D. ananassae*, *D. erecta*, *D. grimshawi*, *D. mojavensis*, *D. persimilis*, *D. pseudoobscura*, *D. sechellia*, *D. simulans*, *D. virilis*, *D. willistoni*, and *D. yakuba*). Sequences that were shorter than 50 amino acids or contained premature stop codons were excluded. 20,803 orthologous groups were identified with OMA standalone v2.2.0 (Altenhoff *et al.*, 2015), using as guidance tree the phylogeny published in Drosophila 12 Genomes Consortium (2007), and default settings otherwise. A single representative splicing variant per gene was identified by OMA, and alternative splicing variants were excluded from subsequent analyses. The retained number of genes per species were 15,052, 14,998, 14,969, 13,818, 14,581, 16,856, 15,845, 16,409, 15,355, 14,477, 15,490, and 16,039 for *D. ananassae*, *D. erecta*, *D. grimshawi*, *D. melanogaster*, *D. mojavensis*, *D. persimilis*, *D. pseudoobscura*, *D. sechellia*, *D. simulans*, *D. virilis*, *D. willistoni*, and *D. yakuba*, respectively.

Sequences of 4,792 OMA orthologous groups that only included one-to-one orthologous genes present in all 12 species were extracted and individually aligned with MAFFT v7.407 (Katoh *et al.*, 2002; Katoh and Standley, 2013), using the iterative refinement method incorporating local pairwise alignment information (`--localpair --maxiterate 1000` settings). Alignments with not more than 20% missing data (4,308 orthologous groups) were concatenated, and a phylogenetic tree was computed with RAxML v8.2.12 (Stamatakis, 2014). The best protein substitution model (JTT with empirical base frequencies) was determined by the program (`-f a -# autoMRE -m PROTGAMMAAUTO --auto-prot=ml` settings). The resulting tree was rooted at the branch that best balanced the subtree lengths.

5.2 Ancestral genome reconstruction

Genome maps were prepared for all retained genes (i.e., excluding low quality sequences and non-representative splicing variants) based on gene position information extracted from the `gff3` files. Gene start and end positions were calculated as the average of CDS midpoints -/+ 1 base pair to avoid the occurrence of overlapping gene positions, which are not supported by the genome reconstruction software ANGES v1.01 (Jones *et al.*, 2012). A few remaining overlaps between gene positions were resolved manually. The ANGES `marker` input file was generated using the genome maps of all 12 *Drosophila* species and 20,803 OMA orthologous groups (from the OMA `OrthologousGroups.txt` file) with the R script `oma2anges.R` (available in the `exec` directory of the `rearrvisr` package). The ANGES `tree` input file was based on the best rooted RAxML tree computed above (including branch lengths). The ancestral genome of the *melanogaster* subgroup *'MSSYE'* (Drosophila 12 Genomes Consortium, 2007; i.e., separating *D. melanogaster*, *D. simulans*, *D. sechellia*, *D. yakuba*, and *D. erecta* from the remainder of the *Drosophila* species) was reconstructed using the ANGES master pipeline (`anges_CAR.py`) and options `markers_doubled 1` (infer ancestral marker orientation), `markers_unique 2` (no duplicated markers), `markers_universal 1` (no missing markers in ingroup), `c1p_telomeres 0` (no telomeres), and `c1p_heuristic 1` (using a greedy heuristic).

A total of 27,242 *Ancestral Contiguous Sets* (ACS; Jones *et al.*, 2012) were identified by ANGES, of which 26,594 ACS were organized into 20 CARs (648, or 2.4%, of ACS were discarded by the program). These CARs comprised a total of 8,973 ancestral markers, and 99.2% of them were grouped into four major CARs (i.e., the 20 CARs included 4,250, 1,914, 1,711, 1,026, 28, 23, 5, 3, 2, and 11 x 1 ancestral markers).

6 References

Altenhoff,A.M. *et al.* (2015) The OMA orthology database in 2015: function predictions, better plant support, synteny view and other improvements. *Nucleic Acids Research*, **43**, D240–D249.

Booth,K.S. and Lueker,G.S. (1976) Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, **13**, 335–379.

Chauve,C. and Tannier,E. (2008) A methodological framework for the reconstruction of contiguous regions of ancestral genomes and its application to mammalian genomes. *PLoS Computational Biology*, **4**, e1000234.

Drosophila 12 Genomes Consortium (2007) Evolution of genes and genomes on the *Drosophila* phylogeny. *Nature*, **450**, 203–218.

Jones,B.R. *et al.* (2012) ANGES: reconstructing ANcestral GEnomeS maps. *Bioinformatics*, **28**, 2388–2390.

Katoh,K. and Standley,D.M. (2013) MAFFT multiple sequence alignment software version 7: improvements in performance and usability. *Molecular Biology and Evolution*, **30**, 772–780.

Katoh,K. *et al.* (2002) MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research*, **30**, 3059–3066.

Ma,J. *et al.* (2006) Reconstructing contiguous regions of an ancestral genome. *Genome Research*, **16**, 1557–1565.

Ouangraoua,A. *et al.* (2011) Reconstructing the architecture of the ancestral amniote genome. *Bioinformatics*, **27**, 2664–2671.

Ranz,J.M. *et al.* (2007) Principles of genome evolution in the *Drosophila melanogaster* species group. *PLOS Biology*, **5**, e152.

Stamatakis,A. (2014) RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. *Bioinformatics*, **30**, 1312–1313.