panda: pangenome annotations data analysis

Lorenzo Tattini

2025-01-29

Warning Messages

With impg 0.2.0 it is normal to get a warning message stating:

`thread 'main' panicked at src/main.rs:205:56:`

`Target name not found in index`

This is related to the behaviour of **`wfmash`** who, given to fastas A and B, does not align both A vs B and B vs A but instead only aligns A vs B.

TODO

Dove si mettono le sequenze? Altro file? Yes.

Input Files and Formats

The input files are: 1) an alignment, 2) the annotations of the contigs aligned, 3) a properly (priority) sorted list of ids.

The alignment file is in paf format. The delimiter (see https://github.com/pangenome/PanSN-spec) is "#". The annotations are in gff format. The file are named using the "-" separator. E.g. in the alignment we may have in line 1 and column 1: "SGDref#0#chrI". The corresponding annotations file is named "SGDref-0-features.gff" and it reports all the annotations for all the corresponding contigs.

Quick Run

Before running the markdown remember to:

- check the input data folder of the annotation files (`dir_anno`)
- check all chunks with `eval = FALSE`
- make a copy of the input alignment in paf format ("/. . . /aln/graph-name.paf")
- make the input ("/. . . /ids/ids-ps.txt") file with the ids (e.g. "SGDref-0", "AIF-1", "AIF-2") of the strains used to build the graph induced from the alignment. Mitochondrial features are attached to the first available haplotype.

Analysis

```
### the gff files repository
dir_anno="${HOME}/data/nano-assemblies-pansn-2024/annotations"

### folders
dir_base=$(dirname "${PWD}")
dir_out="${dir_base}/anno/gff"
if [[ -d "${dir_out}" ]]; then rm -rf "${dir_out}"; fi
```

```
mkdir -p "${dir_out}"

### the vector with the strain-haplotypes ids
declare -a gff_ids
file_ids="${dir_base}/ids/ids-ps.txt"
while IFS="\n" read -r one_line; do
    gff_ids+=("${one_line}")
done < "${file_ids}"

for strainhaplo_id in "${gff_ids[@]}"; do
  path_gff=$(find "${dir_anno}" -name "${strainhaplo_id}*gff")
  name_gff=$(basename "${path_gff}")
  strain_id=$(basename "${path_gff}" | cut -f 1 -d "-") # e.g. ADE
  mito_name="${strain_id}-mt-features.gff"
  path_mito_gff=$(find "${dir_anno}" -name "${mito_name}")
  gff_type=$(basename "${path_gff}" | cut -f 2 -d "-") # e.g. 0 or 1
  if [[ -f "${path_mito_gff}" ]] && [[ "${asse_type}" != "2" ]]; then
    cat <(grep -v "^#" "${path_gff}") <(grep -v "^#" "${path_mito_gff}") \
    > "${dir_out}/${name_gff}"
  else
    grep -v "^#" "${path_gff}" > "${dir_out}/${name_gff}"
  fi
done
```

Here we transform the annotation files in gff format to bed files (without header, otherwise impg crashes). The format of the annotations file for the reference genome are slightly different compared to the phenovar data. So we have to process them separately.

```
## header ----------------------------------------------------------------------

options(scipen = 999)
options(stringsAsFactors = F)
rm(list = ls())
library(data.table)
library(here)
```

```
## here() starts at /Users/Lorenzo/dev/panda/mrk
```

```
library(scriptName)

## settings --------------------------------------------------------------------

### fixed settings
dirBase <- dirname(here())
dirAnnoGff <- file.path(dirBase, "anno", "gff")
dirOut <- file.path(dirBase, "anno", "bed")
unlink(dirOut, recursive = T)
dir.create(dirOut, showWarnings = F)
idRef <- "SGDref"
hdGff <- c("Chr_id", "Strain_id", "Feat_type", "S_coord",
           "E_coord", "S_val", "Strand_id", "Frame_id", "Attribute_str")
vtClassSrt <- c("gene",
                "pseudogene",
                "X_element",
                "X_element_partial",
```

```
                "X_element_combinatorial_repeat",
                "X_region",
                "Y_prime_element",
                "Y_region",
                "recombination_enhancer")
# vtClassSrt <- c("gene",
#                 "pseudogene",
#                 "intron",
#                 "five_prime_UTR_intron",
#                 "noncoding_exon",
#                 "plus_1_translational_frameshift",
#                 "blocked_reading_frame",
#                 "external_transcribed_spacer_region",
#                 "internal_transcribed_spacer_region",
#                 "ARS",
#                 "ARS_consensus_sequence",
#                 "TY1",
#                 "TY1/TY2_soloLTR",
#                 "TY1_truncated",
#                 "TY2",
#                 "TY3_soloLTR",
#                 "TY4_soloLTR",
#                 "TSU4_soloLTR",
#                 "TY4_truncated",
#                 "TY5",
#                 "TY5_soloLTR",
#                 "LTR_retrotransposon",
#                 "long_terminal_repeat",
#                 "W_region",
#                 "Z1_region",
#                 "Z2_region",
#                 "centromere",
#                 "centromere_DNA_Element_I",
#                 "centromere_DNA_Element_II",
#                 "centromere_DNA_Element_III",
#                 "matrix_attachment_site",
#                 "X_element",
#                 "X_element_partial",
#                 "X_element_combinatorial_repeat",
#                 "X_region",
#                 "Y_prime_element",
#                 "Y_region",
#                 "recombination_enhancer",
#                 "silent_mating_type_cassette_array",
#                 "mating_type_region",
#                 "tRNA",
#                 "pseudogenic_transcript",
#                 "ncRNA",
#                 "ncRNA_gene",
#                 "non_transcribed_region",
#                 "rRNA",
#                 "rRNA_gene",
#                 "snRNA",
```

```
#                  "snRNA_gene",
#                  "snoRNA",
#                  "snoRNA_gene",
#                  "intein_encoding_region")
### just in case there's some redundancy
vtClassSrt <- unique(vtClassSrt)

## clmnt ----------------------------------------------------------------------

### script name
myName <- current_filename()
cat("[", myName, "] ",
    "Transforming the gff to bed. ",
    "\n", sep = "")
```

```
## [] Transforming the gff to bed.
```

```
### read strain-haplotypes ids from file
pathIds <- file.path(dirBase, "ids", "ids-ps.txt")
vtStrainHaplo <- as.character(fread(file = pathIds, header = F)[[1]])
vtRef <- grep(idRef, vtStrainHaplo, value = T)
### remove the reference
vtStrainHaplo <- grep(idRef, vtStrainHaplo, value = T, invert = T)

### loop for reference annotations
for (indR in vtRef) {
  ### read the gff
  pathAnnoGff <- list.files(path = dirAnnoGff, pattern = indR,
                            full.names = T, recursive = T)
  dtGff <- fread(file = pathAnnoGff, sep = "\t", header = F, verbose = F)
  colnames(dtGff) <- hdGff
  ### id
  strIdPref <- paste0(sub(pattern = "-", replacement = "#", x = indR), "#")
  ### check start and end coordinates: sometimes start = end and impg breaks
  dtGff <- dtGff[S_coord < E_coord]
  ### filter features
  dtGff <- dtGff[Feat_type %in% vtClassSrt, ]
  ### make the feature id column
  strFeatId <- sub("^.*Name=([^;]*).*$", "\\1", dtGff$Attribute_str)
  ### trim after the first ":", e.g. Name=TY3_soloLTR:chrI:183676-184015:-
  strFeatIdTrm <- sub("^([^:]*).*$", "\\1", strFeatId)
  ### trim trailing redundancy, e.g. _intron in YAL003W_intron
  strFeatIdTrm <- sub("^([^_]*).*$", "\\1", strFeatId)
  ### if strFeatId = dtGff[, Feat_type] set strFeatId = "MN"
  indM <- which(dtGff[, Feat_type] == strFeatIdTrm)
  if (length(indM) != 0) {
    strFeatIdTrm[indM] <- "MN"
  }
  ### paste class and feature id (and the strand)
  strName <- paste0(dtGff[, Feat_type], ":",
                    strFeatIdTrm, "#", dtGff[, Strand_id])
  ### transform the gff into a bed file
  dtBed <- data.table(chrom = paste0(strIdPref, dtGff[, Chr_id]),
                      chromStart = dtGff[, S_coord],
```

```
                      chromEnd = dtGff[, E_coord],
                      name = strName)
  ### write the bed file
  nameOut <- sub(pattern = "-features.gff$", replacement = ".bed",
                 x = basename(pathAnnoGff))
  pathOutBed <- file.path(dirOut, nameOut)
  fwrite(file = pathOutBed, x = dtBed, sep = "\t",
         quote = F, row.names = F, col.names = F)
}

### loop for phenovar annotations
for (indS in vtStrainHaplo) {
  ### dev
  # indS <- vtStrainHaplo[1] ### dev
  # indS <- "AKH_1a-0" ### dev

  ### read the gff
  pathAnnoGff <- list.files(path = dirAnnoGff, pattern = indS,
                            full.names = T, recursive = T)
  dtGff <- fread(file = pathAnnoGff, sep = "\t", header = F, verbose = F)
  colnames(dtGff) <- hdGff
  ### id
  strIdPref <- paste0(sub(pattern = "-", replacement = "#", x = indS), "#")
  ### check start and end coordinates: sometimes start = end and impg breaks
  dtGff <- dtGff[S_coord < E_coord]
  ### filter features
  dtGff <- dtGff[Feat_type %in% vtClassSrt, ]
  ### make the feature id column
  strFeatId <- sub("^.*Name=([^;]*).*$", "\\1", dtGff$Attribute_str)
  ### trim after the first ":", e.g. Name=TY3_soloLTR:chrI:183676-184015:-
  strFeatIdTrm <- sub("^([^:]*).*$", "\\1", strFeatId)
  ### trim "tRNA_" from feature id
  strFeatIdTrm <- sub(pattern = "tRNA_", replacement = "", x = strFeatIdTrm)
  ### if strFeatId = dtGff[, Feat_type] set strFeatId = "MN"
  indM <- which(dtGff[, Feat_type] == strFeatIdTrm)
  if (length(indM) != 0) {
    strFeatIdTrm[indM] <- "MN"
  }
  ### paste class and feature id (and the strand)
  strName <- paste0(dtGff[, Feat_type], ":",
                    strFeatIdTrm, "#", dtGff[, Strand_id])
  ### transform the gff into a bed file
  dtBed <- data.table(chrom = paste0(strIdPref, dtGff[, Chr_id]),
                      chromStart = dtGff[, S_coord],
                      chromEnd = dtGff[, E_coord],
                      name = strName)
  ### write the bed file
  nameOut <- sub(pattern = "-features.gff$", replacement = ".bed",
                 x = basename(pathAnnoGff))
  pathOutBed <- file.path(dirOut, nameOut)
  fwrite(file = pathOutBed, x = dtBed, sep = "\t",
         quote = F, row.names = F, col.names = F)
}
```

We invoke impg through a system command to compute the features pangenome. Then we remove the redundancy generated by impg by checking the overlap of the regions computed by impg. To do this, every annotation interval is used to generate a block of homology. Each block of homology is decomposed in sub-blocks which are defined by a feature class. The reduced (i.e. nonredundant) sub-block is then transformed (i.e. formatted) and appended to output.

```
## header ----------------------------------------------------------------------

options(scipen = 999)
options(stringsAsFactors = F)
rm(list = ls())
library(data.table)
library(here)
library(scriptName)
library(GenomicRanges)
library(tictoc)

## function(s) -----------------------------------------------------------------

SplitSubCol <- function(x, n, s) {
  y <- sapply(strsplit(x, split = s, fixed = T), "[[", n)
  return(y)
}
### It splits a character column of a data-table and collects one sub-column.
###
### Arguments:
### (1) x: a character column of a data-table
### (2) n: the number of sub-column to print in output
### (3) s: the string to apply the split
###
### Returns:
### (1) y: a character vector

## settings --------------------------------------------------------------------

### fixed settings
dirBase <- dirname(here())
dirAnnoBed <- file.path(dirBase, "anno", "bed")
dirOut <- file.path(dirBase, "png")
unlink(dirOut, recursive = T)
dir.create(dirOut, showWarnings = F)
idRef <- "SGDref"
hdImpg <- c("Query_id", "Query_start", "Query_end",
            "Target_id", "Target_start", "Target_end",
            "Target_clsfeat", "N_score", "Query_alndir", "Target_alndir")
# vtClassSrt <- c("gene",
#                 "pseudogene",
#                 "intron",
#                 "five_prime_UTR_intron",
#                 "noncoding_exon",
#                 "plus_1_translational_frameshift",
#                 "blocked_reading_frame",
#                 "external_transcribed_spacer_region",
#                 "internal_transcribed_spacer_region",
```

```
#                   "ARS",
#                   "ARS_consensus_sequence",
#                   "TY1",
#                   "TY1/TY2_soloLTR",
#                   "TY1_truncated",
#                   "TY2",
#                   "TY3_soloLTR",
#                   "TY4_soloLTR",
#                   "TSU4_soloLTR",
#                   "TY4_truncated",
#                   "TY5",
#                   "TY5_soloLTR",
#                   "LTR_retrotransposon",
#                   "long_terminal_repeat",
#                   "W_region",
#                   "Z1_region",
#                   "Z2_region",
#                   "centromere",
#                   "centromere_DNA_Element_I",
#                   "centromere_DNA_Element_II",
#                   "centromere_DNA_Element_III",
#                   "matrix_attachment_site",
#                   "X_element",
#                   "X_element_partial",
#                   "X_element_combinatorial_repeat",
#                   "X_region",
#                   "Y_prime_element",
#                   "Y_region",
#                   "recombination_enhancer",
#                   "silent_mating_type_cassette_array",
#                   "mating_type_region",
#                   "tRNA",
#                   "pseudogenic_transcript",
#                   "ncRNA",
#                   "ncRNA_gene",
#                   "non_transcribed_region",
#                   "rRNA",
#                   "rRNA_gene",
#                   "snRNA",
#                   "snRNA_gene",
#                   "snoRNA",
#                   "snoRNA_gene",
#                   "intein_encoding_region")

### the path to the paf file
dirAlnPaf <- file.path(dirBase, "aln")
pathAlnPaf <- list.files(path = dirAlnPaf, pattern = "paf$",
                         recursive = T, full.names = T)
if (length(pathAlnPaf) > 1) {
  stop("Multiple paf files found.")
} else if (length(pathAlnPaf) == 0) {
  stop("No paf file found.")
}
```

```
## clmnt ----------------------------------------------------------------------

### script name
myName <- current_filename()
cat("[", myName, "] ",
    "Making the pangenome ",
    "\n", sep = "")

### read strain-haplotypes ids from file (the order is maintained in the output)
pathIds <- file.path(dirBase, "ids", "ids-ps.txt")
vtStrainHaplo <- as.character(fread(file = pathIds, header = F)[[1]])
vtStrainHaploHash <- gsub(pattern = "-", replacement = "#", x = vtStrainHaplo)
### initialise the output features table
nHaplos <- length(vtStrainHaploHash)
dtPanFeats <- data.table(matrix(character(length = 0), ncol = nHaplos))
setnames(dtPanFeats, vtStrainHaploHash)
### dev
# vtStrainHaploHash <- c("SGDref#0", "AFI#0", "S288C#0",
#                        "XXX-h3", "stoca-zo", "DBVPG6765#0")

### dev
indS <- vtStrainHaplo[1]
indL <- 1
lsImpg <- list()
### for each strain-haplotypes (hyphen-separated)
for (indS in vtStrainHaplo) {

  ### pick the strain-haplotypes bed file (query)
  pathAnnoBed <- list.files(path = dirAnnoBed, pattern = indS, full.names = T)
  ### string for bash
  strBashImpg <- paste0("impg -I -p ", pathAlnPaf, " -b ", pathAnnoBed)
  ### run "$ impg -p file.paf -b file.bed"  which is
  ### (50x faster than going line-by-line)
  ### with direct load of the output
  strOut <- system(strBashImpg, intern = T)
  ### make a single-file data-table
  lsImpgOne <- strsplit(strOut, split = "\t")
  dtImpgOne <- rbindlist(lapply(lsImpgOne,
                                function(x) as.data.table(as.list(x))))
  ### append the single-file data-table to a list
  lsImpg[[indL]] <- dtImpgOne
  indL <- indL + 1
}

### get the redundant data-table
dtImpgAll <- rbindlist(lsImpg)
colnames(dtImpgAll) <- hdImpg
### add supplementary columns
dtImpgAll[, c("Query_start", "Query_end") := lapply(.SD, as.numeric),
          .SDcols = c("Query_start", "Query_end")]
### dev: riattiva la riga qui sotto se ci sono problemi
# dtImpgAll[, Target_cls := SplitSubCol(dtImpgAll[, Target_clsfeat], 1, "+")]
dtImpgAll[, Target_cls := SplitSubCol(Target_clsfeat, 1, ":")]
```

```
dtImpgAll[, c("Target_start", "Target_end") := lapply(.SD, as.numeric),
          .SDcols = c("Target_start", "Target_end")]
dtImpgAll[, Target_len := Target_end - Target_start]

### size filters
dtImpgAllSzFlt <- dtImpgAll[Target_len > 5 & Query_end - Query_start > 5]

### sort by length, we have two options:
### 1) decreasing, so overlapping features appear in the same row, but
###    the presence of a large annotation e.g. a whole chromosome
###    may result in a row reporting any other annotation in the same block
### 2) no problem with large annotations but we get more rows
###    (ma potrebbe essere la scelta più corretta perché poi, che cazzo c'entra
###    se ordiniamo decreasing o increasing!?, se due feature si overlappano
###    lo fanno a prescindere da come le ordiniamo no?!)
### TODO: questo per il momento è decreasing, siamo QUI
setorder(dtImpgAllSzFlt, -Target_len)

### sort the data-table so that in each homology block the first entry is
### (e.g.) a gene
### dtImpgAllSzFlt <- dtImpgAllSzFlt[order(match(dtImpgAllSzFlt[, Target_cls],
###                                              vtClassSrt))]

### transfer the strand data from Target_clsfeat to Query_id and Target_id
strStrand <- SplitSubCol(dtImpgAllSzFlt[, Target_clsfeat], 2, "#")
dtImpgAllSzFlt[, Query_id := paste0(Query_id, "#", strStrand)]
dtImpgAllSzFlt[, Target_id := paste0(Target_id, "#", strStrand)]

nBlocks <- 1
vtUnq <- unique(dtImpgAllSzFlt[, Target_clsfeat])

### dev
# indTarClsFeat <- "gene:YAR014C#-"
# indTarClsFeat <- vtUnq[1]
# indTarClsFeat <- "gene:YBL037W#+"
# indTarClsFeat <- "X_element:TEL01L#-"
# indTarClsFeat <- "gene:YBR140C#-"
# indTarClsFeat <- "ARS:ARS102#+"
# indTarClsFeat <- "gene:YCR012W#+"
### quello con molti haplo-id doppioni, indSb <- "TY1/TY2_soloLTR"
# indTarClsFeat <- "TY1:MN#-"
# indTarClsFeat <- "gene:YGR296W#+"
# which(vtUnq == "gene:YGR296W#+")
# indTarClsFeat <- "gene:S288C_G0022800#+"
# which(vtUnq == "gene:S288C_G0022800#+")
# which(vtUnq == "ARS+ARS102#+")
# which(vtUnq == "X_element+TEL01L#+")
# indTarClsFeat <- unique(dtImpgAllSzFlt[, Target_clsfeat])[3]
for (indTarClsFeat in vtUnq) {

  ## block calculation ----------------------------------------------------------

  ### dev
```

```
print(nBlocks)

### get the target class-features that will generate the block
dtTarClsFeat <- dtImpgAllSzFlt[indTarClsFeat, on = "Target_clsfeat"]
### check if the corresponding interval is still in dtImpgAllSzFlt
### (if it was ovelapping with a former indTarClsFeat
### it has already been removed, but dtTarClsFeat will still have a line
### e.g. the following):
### Query_id Query_start Query_end Target_id Target_start Target_end
###   <char>       <num>     <num>    <char>        <num>      <num>
###     <NA>          NA        NA      <NA>           NA         NA
### Target_clsfeat N_score Query_alndir Target_alndir Target_cls Target_len
###          <char>  <char>       <char>        <char>     <char>      <num>
###    buciodeculo    <NA>         <NA>          <NA>       <NA>         NA

if (any(is.na(dtTarClsFeat[, Query_id]))) {
  next
}

### first round of overlap (on query coordinates)
setkey(dtTarClsFeat, Query_id, Query_start, Query_end)
dtIndOverOne <- foverlaps(dtImpgAllSzFlt, dtTarClsFeat,
                          nomatch = NULL, type = "any", which = T)
dtBlock <- dtImpgAllSzFlt[dtIndOverOne[, xid], ]
print(format(object.size(dtBlock), units = "auto"))

### second round of overlap (on target coordinates)
setkey(dtBlock, Target_id, Target_start, Target_end)
dtIndOverTwo <- foverlaps(dtImpgAllSzFlt, dtBlock,
                          nomatch = NULL, type = "any", which = T)
dtBlock <- rbindlist(list(dtBlock, dtImpgAllSzFlt[dtIndOverTwo[, xid], ]))
print(format(object.size(dtBlock), units = "auto"))

### block classes
allClassesInBlock <- unique(dtBlock[, Target_cls])

### dev
# cat(indTarClsFeat, "\t",
#     nBlocks, "\t", length(allClassesInBlock), "\n",
#     file = "~/Desktop/check-nsb.txt", append = T)

## sub-blocks reduction (class-by-class, strand-aware) ----------------------

### dev
# indSb <- "gene"
# indSb <- "TY1/TY2_soloLTR"
for (indSb in allClassesInBlock) {
  ### reduce (same as bedtools merge) the rows for each class
  ### on the basis of the query coordinates
  grSblock <- GRanges(seqnames = dtBlock[Target_cls == indSb, Query_id],
                      ranges = IRanges(start = dtBlock[Target_cls == indSb,
                                                       Query_start],
                                       end = dtBlock[Target_cls == indSb,
```

```
                                                    Query_end]),
                    mcols = dtBlock[Target_cls == indSb, Target_clsfeat])
grSblockRed <- reduce(grSblock)

### vector with the features names and class name
### cleaned of the "#+" or "#-" at the end of the string
### e.g. "gene:S288C_G0022800" and "gene:YGR296W"
vtClsFeat <- sub(pattern = "#[\\+\\-]$",
                 replacement = "",
                 x = unique(grSblock$mcols))
### features character
vtFeats <- paste(gsub(pattern = paste0(indSb, ":"), replacement = "",
                      x = vtClsFeat),
                 collapse = ",")
### formatted character, e.g. "gene:S288C_G0022800,YGR296W"
strClsFeatFrm <- paste0(indSb, ":", vtFeats)

dtSblockRed <- data.table(as.character(grSblockRed@seqnames),
                          as.character(grSblockRed@ranges),
                          rep(strClsFeatFrm, length(grSblockRed)))

## transformation of the reduced sub-block --------------------------------

### e.g. this dtSblockRed:
###                 V1              V2                          V3
###             <char>          <char>                      <char>
###  SGDref#0#chrVII#+ 1084864-1090591 gene:S288C_G0022800,YGR296W
###   S288C#0#chrVII#+ 1085116-1090843 gene:S288C_G0022800,YGR296W
###
### becomes a two-column data-table with column 1:
###                                                 SGDref#0
###                                                   <char>
### chrVII:1084864-1090591#+#gene:S288C_G0022800,YGR296W
###
### and column 2:
###                                                  S288C#0
###                                                   <char>
### chrVII:1085116-1090843#+#gene:S288C_G0022800,YGR296W

### dev
# vtHaplo <- sub("#chr.*", "", dtSblockRed[, V1])
# tbHaplo <- table(vtHaplo)
# if (length(which(tbHaplo > 1)) != 0) {
#   stop("found it!")
# }

### transposition and collapsing explained with
### another example, this dtSblockRed:
###                     V1            V2                V3
###                 <char>        <char>            <char>
###     DBVPG6765#0#chrIV#- 960288-966619 TY1/TY2_soloLTR:MN
###         AFI#0#chrXIII#- 349539-349680 TY1/TY2_soloLTR:MN
###         AFI#0#chrXIII#- 349713-349989 TY1/TY2_soloLTR:MN
```

```
    ###   DBVPG6765#0#chrXIII#- 354903-355179 TY1/TY2_soloLTR:MN
    ###           AFI#0#chrIV#- 960661-961002 TY1/TY2_soloLTR:MN
    ###       S288C#0#chrXIII#- 384508-384784 TY1/TY2_soloLTR:MN
    ###      SGDref#0#chrXIII#- 378732-379008 TY1/TY2_soloLTR:MN
    ###
    ### must produce this column:
    ### DBVPG6765#0
    ### chrIV:960288-966619#-#TY1/TY2_soloLTR:MN,\go-to-next-line
    ### chrXIII:354903-355179#-#TY1/TY2_soloLTR:MN

    ### formatting dtSblockRed
    dtSblockRed[, Haplo_id := sub("#chr.*", "", dtSblockRed[, V1])]
    vtCoord <- paste(SplitSubCol(x = dtSblockRed[, V1], n = 3, s = "#"),
                     dtSblockRed[, V2],
                     sep = ":")
    dtSblockRed[, Info_str :=  paste(vtCoord,
                                     SplitSubCol(dtSblockRed[, V1], 4, "#"),
                                     dtSblockRed[, V3],
                                     sep = "#")]

    ### retrieve and store the sequences with samtools

    ### collapsing with ";" all the Haplo_id elements of a Haplo_id
    dtSblockRedCo <- dtSblockRed[, paste(Info_str, collapse = ";"),
                                 by = Haplo_id]

    ### transpose
    dtTra <- transpose(dtSblockRedCo)
    ### set column names and format
    setnames(dtTra, as.character(dtTra[1, ]))
    dtTra <- dtTra[-1]

    ### add the missing columns: not needed
    ### since rbindlist makes it by default using fill = TRUE
    # newCols <- setdiff(vtStrainHaploHash, colnames(dtTra))
    # dtTra[, (newCols) := lapply(newCols, function(x) NA)]
    # setcolorder(dtTra, vtStrainHaploHash)

    ### append with rbindlist matching (default operation) column names
    dtPanFeats <- rbindlist(list(dtPanFeats, dtTra), fill = T)
  }

  ### delete dtIndOverOne and dtIndOverTwo rows in dtImpgAllSzFlt,
  ### the (any(is.na(dtTarClsFeat[, Query_id]))) will avoid
  ### the loop to break
  indOut <- unique(c(dtIndOverOne[, xid], dtIndOverTwo[, xid]))
  dtImpgAllSzFlt <- dtImpgAllSzFlt[-indOut]

  nBlocks <- nBlocks + 1
}

### write dtPanFeats
pathOutPanFeat <- file.path(dirOut, "pan-features.txt")
```

```
# fwrite(dtPanFeats, file = pathOutPanFeat, append = F, quote = F, sep = "\t",
#        col.names = T, na = "MA", nThread = 8, buffMB = 1024)
#>Error in fwrite(dtPanFeats, file = pathOutPanFeat, append = F, quote = F,  :
#> Bad address: '/home/ltattini/prog/graphs/panda/run-11/png/pan-features.txt'
write.table(x = dtPanFeats, file = pathOutPanFeat, append = F, quote = F,
            sep = "\t", col.names = T, row.names = F, na = "MA")
save(dtPanFeats, file = file.path(dirOut, "pan-features.RData"))
### write dtPanSeqs
```