panda: pangenome annotations data analysis

Lorenzo Tattini

2025-02-09

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 fasta files A and B, does not align both A vs B and B vs A but instead only aligns A vs B.

TODO

Retrieve the sequences of each block and save them? Probably this is not a wise thing to do. With the nnew output format it is easy to do this on the fly.

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
- choose if mitochondrial features should be used (in case they are attached to the first available haplotype) with `with_mito`

Analysis

```
## settings ---------------------------------------------------------------------

### with mitochondrial sequence in one haplotype?
with_mito="FALSE"
```

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

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

### 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 [[ "${with_mito}" == "TRUE" && \
        -f "${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. Moreover, when we process the phenovar data we can filter out the genes with a systematic name since these names will be brought in by the reference annotations. This reduces the calculations needed from impg and thus results in a smaller impg output.

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

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

```
## Warning: package 'this.path' was built under R version 4.3.3
```

```
library(scriptName)

## settings ---------------------------------------------------------------
```

```r
### fixed settings
dirBase <- dirname(this.dir())
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")
# 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.
```

```r
### read strain-haplotypes ids from file
pathIds <- list.files(path = file.path(dirBase, "ids"), pattern = "ids-ps.txt",
                      full.names = T)
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)
}

### dev
### indS <- vtStrainHaplo[1]
### indS <- "AKH_1a-0"
### loop for phenovar annotations
for (indS in vtStrainHaplo) {
  ### 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("^([^:]*).*$", "\\1", strFeatId)
  ### trim "tRNA_" from feature id
  strFeatIdTrm <- sub(pattern = "tRNA_", replacement = "", x = strFeatIdTrm)
  ### indexes of the rows that do not contain
  ### nuclear gene names with systematic names, e.g. YAL062W
  indSys <- grep(pattern = "^Y[A-P][L,R][0-9]{3}[W,C]",
                 x = strFeatIdTrm, value = F, invert = T)
  ### if strFeatId = dtGff[, Feat_type] set strFeatId = "MN"
  indM <- which(dtGff[, Feat_type] == strFeatIdTrm)
  if (length(indM) != 0) {
    strFeatIdTrm[indM] <- "MN"
  }
  ### paste class and feature id (and the strand)
  strName <- paste0(dtGff[, Feat_type], ":",
                    strFeatIdTrm, "#", dtGff[, Strand_id])
  ### transform the gff into a bed file
  ### and filter out genes with a systematic name
  dtBed <- data.table(chrom = paste0(strIdPref, dtGff[indSys, Chr_id]),
                      chromStart = dtGff[indSys, S_coord],
                      chromEnd = dtGff[indSys, E_coord],
                      name = strName[indSys])
  ### write the bed file
  nameOut <- sub(pattern = "-features.gff$", 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.

We write also the generators, i.e. the “class:feature” (e.g. “gene:YGL259W#+”) that generated a line in the output “pan-features.txt”. A generator may generate multiple lines in “pan-features.txt”, e.g. when we analyse genes and pseudogenes a generator may produce a “gene” line and a “pseudogene” line.

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

options(scipen = 999)
options(stringsAsFactors = F)
rm(list = ls())
library(data.table)
library(this.path)
library(scriptName)
library(GenomicRanges)
```

```
## Loading required package: stats4

## Loading required package: BiocGenerics

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which.max, which.min

## Loading required package: S4Vectors

##
## Attaching package: 'S4Vectors'

## The following objects are masked from 'package:data.table':
##
```

```
##     first, second
## The following object is masked from 'package:utils':
##
##     findMatches
## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname
## Loading required package: IRanges
##
## Attaching package: 'IRanges'
## The following object is masked from 'package:data.table':
##
##     shift
## Loading required package: GenomeInfoDb
## Warning: package 'GenomeInfoDb' was built under R version 4.3.3
```

```
library(tictoc)
```

```
##
## Attaching package: 'tictoc'
## The following object is masked from 'package:GenomicRanges':
##
##     shift
## The following object is masked from 'package:IRanges':
##
##     shift
## The following object is masked from 'package:data.table':
##
##     shift
```

```
## 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(this.dir())
```

```r
### dev dirBase <- "/Users/Lorenzo/dev/panda"
dirAnnoBed <- file.path(dirBase, "anno", "bed")
dirOut <- file.path(dirBase, "png")
unlink(dirOut, recursive = T)
dir.create(dirOut, showWarnings = F)
pathGen <- file.path(dirOut, "generators.txt")
cat("class:feature", "\n", file = pathGen)
idRef <- "SGDref"
hdImpg <- c("Query_id", "Query_start", "Query_end",
            "Target_id", "Target_start", "Target_end",
            "Target_clsfeat", "N_score", "Query_alndir", "Target_alndir")

### 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) {
  cat("[", myName, "] ",
      "User error.\n",
      sep = "")
  ### stop prints "Error: " by default
  stop("multiple paf files found.")
} else if (length(pathAlnPaf) == 0) {
  cat("[", myName, "] ",
      "User error.\n",
      sep = "")
  ### stop prints "Error: " by default
  stop("no paf file found.")
}

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

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

```
## [] Making the pangenome.
```

```r
### 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")

## impg run -------------------------------------------------------------------
```

```r
### message
cat("[", myName, "] ",
    "Running impg. ",
    "\n", sep = "")
```

```
## [] Running impg.
```

```r
### 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)

  ### run "$ impg -p file.paf -b file.bed"  which is
  ### (50x faster than going line-by-line)
  ### with direct load of the output

  ### string for bash (impg-0.2.0)
  strBashImpg <- paste0("impg -I -p ", pathAlnPaf, " -b ", pathAnnoBed)
  ### string for bash (impg-0.2.3)
  ### strBashImpg <- paste0("impg query -I -p ", pathAlnPaf, " -b ", pathAnnoBed)

  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
  print(format(object.size(lsImpg), units = "auto"))
  indL <- indL + 1
}
```

```
## [1] "12.7 Mb"
## [1] "13 Mb"
## [1] "13.6 Mb"
## [1] "13.9 Mb"
## [1] "14.2 Mb"
## [1] "14.5 Mb"
## [1] "14.7 Mb"
## [1] "14.9 Mb"
## [1] "15 Mb"
## [1] "15.1 Mb"
```

```r
cat("[", myName, "] ",
    "Running impg: done. ",
    "\n", sep = "")
```

```
## [] Running impg: done.
```

```r
### get the redundant data-table
dtImpgAll <- rbindlist(lsImpg)
colnames(dtImpgAll) <- hdImpg
```

```r
### add supplementary columns
dtImpgAll[, c("Query_start", "Query_end") := lapply(.SD, as.numeric),
          .SDcols = c("Query_start", "Query_end")]
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]

### garbage collection
rm(dtImpgAll)
invisible(gc())

### sorting ascending or descending is not equivalent since the
### overlap is done in two rounds and
### is strand-aware (if two features have overlapping
### coordinates but one is in the + strand and the other is in
### the - strand they do not overlap), thus
### we sort in increasing (ascending) order to avoid spurious
### overlaps that may occur when multiple classes are analysed
### (a spurious overlap will produce a big block that cannot be easily
### decomposed, except for the different classes);
### the YFL066C locus (where we have the Y' region TEL06L) is a good
### example of a spurious overlap
setorder(dtImpgAllSzFlt, Target_len)

### transfer the strand data from Target_clsfeat to Query_id and Target_id,
### so that when we calculate the blocks these will be strand-aware
strStrand <- SplitSubCol(dtImpgAllSzFlt[, Target_clsfeat], 2, "#")
dtImpgAllSzFlt[, Query_id := paste0(Query_id, "#", strStrand)]
dtImpgAllSzFlt[, Target_id := paste0(Target_id, "#", strStrand)]

## blocks calculation ---------------------------------------------------------

cat("[", myName, "] ",
    "Blocks calculation. ",
    "\n", sep = "")
```

```
## [] Blocks calculation.
```

```r
nBlocks <- 1
vtUnq <- unique(dtImpgAllSzFlt[, Target_clsfeat])
### dev
# indTarClsFeat <- "Y_prime_element:TEL06L#-"
# 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]
# indTarClsFeat <- vtUnq[1]

### YAL005C overlaps with YAL004W, we do not want these two genes
### to generate a single sub-block, so we make the overlap strand-aware
# indTarClsFeat <- "gene:YAL004W#+"
# indTarClsFeat <- "gene:YAL005C#-"
for (indTarClsFeat in vtUnq) {

  ### 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], ]
  ### dev 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], ]))
  ### dev 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"
# indSb <- "Y_prime_element"
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 string
  strFeats <- paste(gsub(pattern = paste0(indSb, ":"), replacement = "",
                         x = vtClsFeat),
                    collapse = ",")

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

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

  ### e.g. this dtSblockRed:
  ###                 V1               V2
  ###             <char>           <char>
  ###  SGDref#0#chrVII#+ 1084864-1090591
  ###   S288C#0#chrVII#+ 1085116-1090843
  ###
  ### becomes a two-column data-table with columns:
  ###                 SGDref#0                     S288C#0
  ###                   <char>                      <char>
  ### chrVII:1084864-1090591#+    chrVII:1085116-1090843#+

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

  ### transposition and collapsing explained with
  ### another example, this dtSblockRed:
  ###                      V1            V2
  ###                  <char>        <char>
  ###     DBVPG6765#0#chrIV#- 960288-966619
  ###         AFI#0#chrXIII#- 349539-349680
  ###         AFI#0#chrXIII#- 349713-349989
  ###   DBVPG6765#0#chrXIII#- 354903-355179
  ###           AFI#0#chrIV#- 960661-961002
  ###       S288C#0#chrXIII#- 384508-384784
  ###      SGDref#0#chrXIII#- 378732-379008
  ###
  ### must produce this column:
  ### DBVPG6765#0
  ### chrIV:960288-966619#-,chrXIII:354903-355179#-

  ### 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, "#"),
                                   sep = "#")]
  ### collapsing with ";" all the Haplo_id elements of a Haplo_id
  dtSblockRedCo <- dtSblockRed[, paste(Info_str, collapse = ";"),
                                 by = Haplo_id]
  ### transpose, producing a data-table
  dtTra <- transpose(dtSblockRedCo)
  ### set column names and format
  setnames(dtTra, as.character(dtTra[1, ]))
  dtTra <- dtTra[-1]
  ### add class and features columns
  dtTra[, ':='(class = rep(indSb, .N), features = rep(strFeats, .N))]

  ### 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)

  ### append the generator feature to the output
  cat(rep(indTarClsFeat, nrow(dtTra)), sep = "\n",
      append = T, file = pathGen)
  ### dev print(nrow(dtTra))
}
```

```r
  ### 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

  ### stop prints "Error: " by default
  ### dev if (nBlocks == 20) stop("we did 20 iterations!")
}
cat("[", myName, "] ",
    "Blocks calculation left ",
    nrow(dtImpgAllSzFlt),
    " impg lines. ",
    "\n", sep = "")
```

```
## [] Blocks calculation left 0 impg lines.
```

```r
cat("[", myName, "] ",
    "Blocks calculation: done. ",
    "\n", sep = "")
```

```
## [] Blocks calculation: done.
```

```r
### format and write dtPanFeats
leftCols <- c("class", "features")
colOrder <- c(leftCols, setdiff(names(dtPanFeats), leftCols))
setcolorder(dtPanFeats, colOrder)
pathOutPanFeat <- file.path(dirOut, "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"))
```