R codes of MCnebula

Contents

1 File: annotate_child_nebula.R

```
#' @title annotate_child_nebulae
#' @description Visualization of structure, PPCP and statistic data in child-nebula.
#' @param nebula_name Character. The name of child-nebula.
#' @param compound_class_list A list, generaged by `collate_ppcp`, Default: .MCn.nebula_class
#' @param write_output Logic. Write output to a directory, Default: T
#' @param output Character, Default: paste0(.MCn.output, "/", .MCn.results)
#' @param layout Character, 'igraph' layout, Default: 'fr'
#' @param height A number, Default: 'auto'
#' @param width A number, Default: 'auto'
#' @param plot_nodes_id Logic, Default: T
#' @param plot_structure Logic, Default: T
#' @param plot_ppcp Logic, Default: T
#' @param ratio_df A data.frame, Default: NA
#' @param merge_image Logic, Default: T
#' @param return_plot Logic, Default: F
#' @param nodes_mark A is.data.frame, Default: NA
```

```r
#' @param global.node.size A number, Default: 0.6
#' @param theme_args A list, Default: NA
#' @param ... ...
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[dplyr]{filter}}, \code{\link[dplyr]{select}}, \code{\link[dplyr]{rename}}, \code{\link[
#'  \code{\link[data.table]{rbindlist}}
#' @rdname annotate_child_nebulae
#' @export
#' @importFrom dplyr filter select rename mutate
#' @importFrom data.table rbindlist
annotate_child_nebulae <-
  function(
           nebula_name,
           compound_class_list = .MCn.nebula_class,
           nebula_index = .MCn.nebula_index,
           write_output = T,
           output = paste0(.MCn.output, "/", .MCn.results),
           layout = "fr",
           height = "auto",
           width = "auto",
           plot_nodes_id = T,
           plot_structure = T,
           plot_ppcp = T,
           ratio_df = NA,
           merge_image = T,
           return_plot = F,
           nodes_mark = NA,
           global.node.size = 0.6,
           theme_args = NA,
           ...
           ){
    cat("[INFO] MCnebula run: annotate_child_nebulae\n")
    ## ----------------------------------------------------------------------
    ## all nodes in graph
```

```
nodes <- dplyr::filter(nebula_index, name == nebula_name)$".id"
## get top compound class (nodes_color data)
## as well as, collate metadata
metadata <- lapply(compound_class_list, head, n = 1) %>%
  data.table::rbindlist(idcol = T) %>% # as data.frame
  dplyr::filter(.id %in% nodes) %>% # filter via nodes
  dplyr::select(.id, name) %>%
  dplyr::rename(vis_class = name)
## ------------------------------------------------------------------------
## mark nodes in color
if(is.data.frame(nodes_mark)){
  ## the secound col as mark col
  colnames(nodes_mark) <- c(".id", "mark")
  ## merge with metadata
  metadata <- merge(metadata, nodes_mark, by = ".id", all.x = T) %>%
    dplyr::mutate(vis_class = ifelse(is.na(mark), "Others", mark))
}
## ------------------------------------------------------------------------
## push environment name into parent.env,
## let some data could be catch in sub-environment via 'get' function
assign("envir_meta", environment(), envir = parent.env(environment()))
## ------------------------------------------------------------------------
## gather data for annotation (nebula_name, hierarchy)
hierarchy <- head(dplyr::filter(nebula_index, name == nebula_name), n = 1)
anno = c(nebula_index = nebula_name, hierarchy = hierarchy$hierarchy)
## set a environment to store layout data
envir_layout <- new.env()
## set to remove nodes or not (set to 0, remove)
if(plot_ppcp | plot_structure){
  remove_nodes <- T
}else{
  remove_nodes <- F
}
## plot origin network (child network, with legend)
p <- grid_child_nebula(.MCn.child_graph_list[[nebula_name]],
                       anno = anno,
                       class = metadata,
                       print_into = F,
                       layout = layout,
                       ## save layout data in this environment
                       save_layout_df = envir_layout,
```

```
                        ## remove origin nodes
                        remove_nodes = remove_nodes,
                        theme_args = theme_args,
                        ...)
## ------------------------------------------------------------------------
## whether plot pie diagram
if(is.data.frame(ratio_df)){
  plot_ratio <- T
}else{
  plot_ratio <- F
}
## ------------------------------------------------------------------------
## tmp dir
tmp_dir <- paste0(output, "/", "tmp")
if(!file.exists(tmp_dir)){
  dir.create(tmp_dir)
}
## add annotation ----------------------------------------------------------
## nodes id
if(plot_nodes_id  & !plot_ppcp){
  p <- p + geom_node_text(aes(label = name), size = 1)
}
## add annotation ----------------------------------------------------------
## plot 2D structure, require ChemmineOB and ChemmineR
with_structure <- 0
if(requireNamespace("ChemmineOB", quietly = T)){
  ## structure
  tmp_stru <- paste0(tmp_dir, "/", "structure")
  if(!file.exists(tmp_stru)){
    dir.create(tmp_stru)
  }
  if(plot_structure){
    with_structure <- 1
    batch_mode_structure(metadata = metadata, tmp_stru = tmp_stru)
  }
}
## add annotation ----------------------------------------------------------
## re draw nodes with or without ppcp bar
tmp_ppcp <- paste0(tmp_dir, "/", "ppcp")
if(!file.exists(tmp_ppcp)){
  dir.create(tmp_ppcp)
```

```
}
if(plot_ppcp | plot_structure | plot_ratio ){
  batch_mode_nodes(
                   metadata = metadata,
                   tmp_ppcp = tmp_ppcp,
                   with_structure = with_structure,
                   plot_ppcp = plot_ppcp,
                   plot_ratio = plot_ratio,
                   ratio_df = ratio_df,
                   ...)
}
## ------------------------------------------------------------------------
## merge image
if(merge_image){
  if(requireNamespace("ggimage", quietly = T) &
     requireNamespace("gridExtra", quietly = T)){
    ## remove legend of size
    p <- p + guides(size = "none")
    merge_image(p, envir_layout$layout_n, tmp_ppcp,
                global.node.size = global.node.size)
  }
}
## ------------------------------------------------------------------------
## write_output ## estimate width
if(write_output){
  if(height == "auto" | width == "auto"){
    ## estimate width upon legend number of 'fill'
    n = length(unique(metadata$vis_class))
    height = 8
    width = ifelse(n <= 17, 9, ## 'class' less than 17
                   ifelse(n <= 34, 12.5,
                          ifelse(n <= 51, 15, 18)))
  }
  ## output
  ggsave(p, file = paste0(output, "/", nebula_name, "_graph.svg"),
         width = width, height = height)
}
## ------------------------------------------------------------------------
rm("envir_meta", envir = parent.env(environment()))
## ------------------------------------------------------------------------
cat("[INFO] MCnebula Job Done: annotate_child_nebulae\n")
```

```
    if(return_plot){
      return(p)
    }
  }
gather_subview <-
  function(
           subview,
           x,
           y,
           width,
           height,
           p = get("p", envir = get("envir_meta"))
           ){
    p <- p + ggimage::geom_subview(x = x, y = y, width = width, height = height,
                           subview = subview)
    assign("p", p, envir = get("envir_meta"))
    return("Done")
    ##
  }
merge_image <-
  function(
           p, ## ggplot2 object
           layout_n,
           tmp_ppcp,
           global.node.size = 1,
           ...
           ){
    ## ------------------------------------------------------------------------
    ## check svg image
    df <- dplyr::select(layout_n, x, y, name, tanimotoSimilarity) %>%
      dplyr::mutate(nodes_path = paste0(tmp_ppcp, "/", name, ".svg"),
                    check_nodes = file.exists(nodes_path)) %>%
      dplyr::filter(check_nodes == T)
    cat("## read_cairo_svg:", nrow(df), "(number)\n")
    ## read svg image
    subview_list <- pbapply::pblapply(df$name, base_read_cairo,
                                      path = tmp_ppcp,
                                      ...)
    ## ------------------------------------------------------------------------
    ## calculate width and height for subview, according to attributes of tanimotoSimilarity
    df <- dplyr::mutate(df,
```

```
                        width = ifelse(is.na(tanimotoSimilarity) == T, 1,
                                       1 + tanimotoSimilarity) * global.node.size,
                        height = width)
    ## ------------------------------------------------------------------------
    ## as subview
    cat("## Advance visualization: gather_subview\n")
    pbapply::pbmapply(gather_subview, ## function
                      subview_list,
                      df$x,
                      df$y,
                      df$width,
                      df$height
    )
  }
```

2 File: batch_mode_nodes.R

```
batch_mode_nodes <-
  function(
           metadata,
           tmp_ppcp,
           with_structure = 0,
           plot_ppcp = plot_ppcp,
           plot_ratio = F,
           ratio_df = NA,
           palette = .MCn.palette,
           palette_stat = .MCn.palette_stat,
           annotate_ppcp.class.id = F,
           ...
           ){
    ## remove exist files
    lapply(list.files(tmp_ppcp, full.names = T), file.remove)
    ## ------------------------------------------------------------------------
    ## nodes color setting, which parallel to the nodes color in plot of visualize_child_nebula functio
    meta_color <- dplyr::select(metadata, vis_class) %>%
      dplyr::distinct() %>%
      dplyr::arrange(vis_class)
    if(is.vector(attr(palette, "name"))){
      ## filter palette
      palette <- palette[which(names(palette) %in% meta_color$vis_class)]
      ## sort according to the order of 'vis_class'
```

```
  palette <- palette[order(names(palette), levels = meta_color$vis_class)]
  ## set color
  meta_color$nodes_color <- palette
}else{
  meta_color$nodes_color <- palette[1:nrow(meta_color)]
}
## gather color data
meta_nodes <- merge(metadata, meta_color, by = "vis_class", all.x = T)
## ------------------------------------------------------------------------
## pick ppcp_dataset
ppcp_dataset = .MCn.ppcp_dataset[which(names(.MCn.ppcp_dataset) %in% meta_nodes$".id")]
## sort data
meta_nodes$".id" <- factor(meta_nodes$".id",
                           levels = names(ppcp_dataset))
meta_nodes <- meta_nodes[order(meta_nodes$".id"), ]
## ------------------------------------------------------------------------
## ratio_df, extra peak area data
if(plot_ratio){
  ## adjust palette_stat
  if(!is.null(names(palette_stat)[1])){
    palette_stat <- palette_stat[names(palette_stat) %in% names(ratio_df)]
  }else{
    palette_stat[1:(ncol(ratio_df)-1)]
  }
  ratio_df <- dplyr::mutate(ratio_df, .id = as.character(.id))
  ratio_df <- merge(dplyr::select(meta_nodes, .id), ratio_df, all.x = T, by = ".id", sort = F)
  ## get list data
  ratio_df_list <- by_group_as_list(ratio_df, ".id")
}else{
  ratio_df_list <- rep(0, nrow(meta_nodes))
}
## ------------------------------------------------------------------------
cat("## annotate_child_nebulae: batch_mode_nodes\n")
pbapply::pbmapply(base_vis_nodes, # function
                  ppcp_dataset, # main 1
                  meta_nodes$nodes_color, # main 2
                  names(ppcp_dataset), # main 3, key_id
                  ratio_df_list, # main 4, draw pie diagram
                  MoreArgs = list(
                                  path = normalizePath(tmp_ppcp),
                                  with_structure = with_structure,
```

```
                                    plot_ppcp = plot_ppcp,
                                    plot_ratio = plot_ratio,
                                    palette_stat = palette_stat,
                                    annotate_ppcp.class.id = annotate_ppcp.class.id,
                                    ...))
  }
## function mannually draw nodes
base_vis_nodes <-
  function(
           ppcp, ## main 1
           nodes_color, ## main 2
           key_id = NA, ## main 3
           ratio_df = NA, ## main 4, draw pie diagram
           plot_ratio = F,
           plot_nodes_id = T,
           plot_ppcp = T,
           label_color = "black",
           with_structure = 0,
           path = ".",
           class_index = unique(.MCn.nebula_index$relativeIndex),
           palette_ppcp = colorRampPalette(.MCn.palette_ppcp)(length(class_index)),
           palette_stat = .MCn.palette_stat,
           annotate_ppcp.class.id = F,
           size_adjust = 0.7,
           get_ppcp_legend = F
           ){
    ## ------------------------------------------------------------------------
    ## filter via class_index
    ppcp <- ppcp[ppcp$relativeIndex %in% class_index, ]
    ## as factor, for painting color
    ppcp$relativeIndex <- factor(ppcp$relativeIndex, levels = sort(ppcp$relativeIndex))
    ppcp$num <- seq(1, nrow(ppcp))
    if(plot_ppcp == F){
      ppcp$V1 = 0
    }
    ## ------------------------------------------------------------------------
    ## plot nodes
    p <- ggplot(ppcp, aes(x = num, y = V1)) +
      ## nodes color
      geom_ribbon(fill = nodes_color,
                          aes(ymin = -5, ymax = 0,
```

```
                            x = ifelse(num == 1, 0,
                                       ifelse(num == nrow(ppcp), num + 1, num)))) +
  ## border color
  geom_ribbon(fill = "black",
                       aes(ymin = 0, ymax = 1.1,
                           x = ifelse(num == 1, 0,
                                      ifelse(num == nrow(ppcp), num + 1, num)))) +
    ## ppcp bar plot
    geom_col(alpha = 1, aes(fill = relativeIndex), color = "white", size = 0.25) +
    ## nodes border ratio
    ylim(-5, 1.3) +
    ## Polar coordinate transformation
    coord_polar()
## ------------------------------------------------------------------------
## draw pie diagram
if(plot_ratio){
  ratio_df <- reshape2::melt(ratio_df, id.vars = ".id", variable.name = "group", value.name = "valu
  ## mutate NA as 0
  ratio_df <- dplyr::mutate(ratio_df, value = ifelse(is.na(value), 0, value))
  ## value stack
  ratio_df <- dplyr::mutate(ratio_df,
                            xend = stack_sum(ratio_df$value),
                            x = stack_sum(c(0, ratio_df$value[1:(nrow(ratio_df)-1)])))
  ## normalize x axis range and x value
  n_factor = (max(ppcp$num) + 1) / max(ratio_df$xend)
  ratio_df <- dplyr::mutate(ratio_df,
                      midd = (x + value/2) * n_factor,
                      width = value * n_factor)
  ## add pie plot into ggplot2 project
  names(palette_ppcp) <- class_index
  p <- p + geom_tile(data = ratio_df, size = 0.2, color = "white",
                      aes(y = -2.5, x = midd, width = width, height = 2.5, fill = group)) +
    ## add 'fill' palette
    scale_fill_manual(values = c(palette_ppcp, palette_stat))
}else{
  ## add 'fill' palette
  names(palette_ppcp) <- class_index
  p <- p + scale_fill_manual(values = palette_ppcp)
}
## ------------------------------------------------------------------------
## add ppcp class name id
```

```
  if(annotate_ppcp.class.id){
    ## label metadata
    label_data <- ppcp
    ## calculate angle in circle
    angle <-  90 - 360 * ((1:nrow(label_data)) - 0.5) / nrow(label_data)
    ## flip
    label_data$angle <- ifelse(angle < (-90), angle + 180, angle)
    ## mapped into plot
    p <- p + geom_text(data = label_data,
                       aes(x = num, y = 0.5, label = relativeIndex, angle = angle),
                       color = "white", fontface = "bold", alpha = 0.8,
                       size = 2, inherit.aes = FALSE)
  }
  ## ------------------------------------------------------------------------
  ## add theme
  p <- p + mc.blank_theme()
  ## ------------------------------------------------------------------------
  ## generate Graphics Device
  savepath = paste0(path, "/", key_id, ".svg")
  svglite::svglite(savepath, bg = "transparent")
  ## print nodes
  print(p)
  ## ------------------------------------------------------------------------
  ## print structure or not
  if(with_structure == 1){
    s_file = paste0(normalizePath(paste0(path, "/../structure")), "/", key_id, ".svg")
    if(file.exists(s_file)){
      ## via grImport2 import Cairo svg
      ps <- grImport2::readPicture(file = s_file)
      ## grid draw
      grImport2::grid.picture(ps, width = size_adjust, height = size_adjust)
    }
  }
  ## ------------------------------------------------------------------------
  ## grid nodes ID in nodes
  if(plot_nodes_id){
    ## a grid object
    ps <- grid::textGrob(paste0("ID:", key_id),
                         y = 0.25,
                         gp = grid::gpar(fontfamily = "Times", fontsize = 20, col = label_color))
    grid::grid.draw(ps)
```

```
    }
    ## ------------------------------------------------------------------------
    dev.off()
    # as cairo svg
    rsvg::rsvg_svg(savepath, savepath)
    ## ------------------------------------------------------------------------
    ## get ppcp legend
    if(get_ppcp_legend){
      if(requireNamespace("ggpubr", quietly = TRUE)){
        ## select the corresponding palette
        palette_ppcp <- palette_ppcp[names(palette_ppcp) %in% ppcp$relativeIndex]
        ## order according to label
        palette_ppcp <- palette_ppcp[order(factor(names(palette_ppcp), levels = ppcp$relativeIndex))]
        ## get class name metadata
        df <- .MCn.class_tree_data[, c("relativeIndex", "name")]
        df$relativeIndex <- as.factor(df$relativeIndex)
        ## merge to get class name
        ppcp <- merge(ppcp, df, by = "relativeIndex", all.x = T, sort = F)
        ## paste merge relativeIndex and name
        ppcp$name <- paste0(ppcp$relativeIndex, ": ", ppcp$name)
        ppcp$name <- stringr::str_wrap(ppcp$name, width = 30)
        ## rename palette
        names(palette_ppcp) <- ppcp$name
        ## draw legend
        legend <- ggplot(ppcp, aes(x = num, y = V1, fill = name)) +
          geom_col() +
          labs(fill = "Structural classes") +
          theme_minimal() +
          scale_fill_manual(values = palette_ppcp) +
          theme(text = element_text(family = "Times", face = "bold"),
                legend.key.height = unit(0.8, "cm")
          )
        legend <- ggpubr::get_legend(legend)
        legend <- ggpubr::as_ggplot(legend)
        ggsave(legend, filename = paste0(path, "/", "legend_", key_id, ".svg"), width = 15, height = 10
      }
    }
    ## ------------------------------------------------------------------------
  }
## function read cairo svg
base_read_cairo <-
```

```
  function(
           key_id,
           path,
           suffix = ".svg"
           ){
    prefix <- c()
    svg <- grImport2::grobify(grImport2::readPicture(paste0(path, "/", key_id, suffix)))
    svg <- gridExtra::arrangeGrob(svg)
    return(svg)
  }
stack_sum <-
  function(
           vector
           ){
    stack <- c()
    for(i in 1:length(vector)){
      stack[i] <- sum(vector[1:i])
    }
    return(stack)
  }
mc.blank_theme <-
  function(
           legend.position = "none"
           ){
    theme_minimal() +
      theme(
            text = element_text(family = "Times"),
            axis.ticks = element_blank(),
            axis.text = element_blank(),
            axis.title = element_blank(),
            panel.grid = element_blank(),
            ## remove legend
            legend.position = legend.position,
            panel.border = element_blank(),
            plot.margin =unit(c(0,0,0,0),"cm"),
            panel.spacing =unit(c(0,0,0,0),"cm")
      )
  }
```

3 File: batch_mode_structure.R

```
batch_mode_structure <-
  function(
           metadata,
           tmp_stru
           ){
    ## -----------------------------------------------------------------------
    ## collate metadata
    meta_stru <- dplyr::mutate(metadata,
                               stru_file = paste0(tmp_stru, "/", .id, ".svg"),
                               stru_check = file.exists(stru_file))
    meta_stru <- merge(meta_stru, .MCn.structure_set[, c(".id", "smiles")], by = ".id", all.x = T)
    meta_stru <- dplyr::filter(meta_stru, is.na(smiles) == F)
    cat("## STAT of structure set:",
        paste0(nrow(meta_stru), "(compounds with structure)", "/", nrow(metadata), "(all compounds)"),
        "\n")
    meta_stru <- dplyr::filter(meta_stru, stru_check == F)
    ## -----------------------------------------------------------------------
    if(nrow(meta_stru) > 0){
      pbapply::pbmapply(
                        base_vis_structure, # function
                        meta_stru$".id", # key_id
                        meta_stru$smiles, # smiles
                        MoreArgs = list( path = normalizePath(tmp_stru) ))
    }
  }
base_vis_structure <-
  function(
           key_id,
           smiles,
           path,
           to_file = paste0(path, "/", key_id, ".svg")
           ){
    ## openbabal. only support for linux
    ChemmineOB::convertToImage("SMI", "SVG", source = smiles, toFile = to_file)
    ## as transparent bg
    svg <- data.table::fread(file = to_file, sep = "", quote ="", header = F)
    svg$V1 = sub('fill="white"', 'fill="transparent"', svg$V1)
    write.table(x = svg, file = to_file, sep = "", col.names = F, row.names = F, quote = F)
    ## convert as cairo svg
    rsvg::rsvg_svg(to_file, to_file)
```

```
  return("Done")
}
```

4 File: beta.annotate_node.R

```
annotate_node <-
  function(
           node_id,
           compound_class_list = .MCn.nebula_class,
           output = paste0(.MCn.output, "/", .MCn.results),
           plot_nodes_id = T,
           plot_structure = T,
           plot_ppcp = T,
           ratio_df = NA,
           nodes_mark = NA,
           annotate_ppcp.class.id = T,
           ...
           ){
    nodes <- node_id
    ## get top compound class (nodes_color data)
    ## as well as, collate metadata
    metadata <- lapply(compound_class_list, head, n = 1) %>%
      data.table::rbindlist(idcol = T) %>% # as data.frame
      dplyr::filter(.id %in% nodes) %>% # filter via nodes
      dplyr::select(.id, name) %>%
      dplyr::rename(vis_class = name)
    ## ------------------------------------------------------------------------
    ## mark nodes in color
    if(is.data.frame(nodes_mark)){
      ## the secound col as mark col
      colnames(nodes_mark) <- c(".id", "mark")
      ## merge with metadata
      metadata <- merge(metadata, nodes_mark, by = ".id", all.x = T) %>%
        dplyr::mutate(vis_class = ifelse(is.na(mark), "Others", mark))
    }
    ## ------------------------------------
    if(is.data.frame(ratio_df)){
      plot_ratio <- T
    }else{
      plot_ratio <- F
    }
```

```
    ## ------------------------------------
    tmp_dir <- paste0(output, "/", "tmp")
    ## ------------------------------------
    ## plot 2D structure, require ChemmineOB and ChemmineR
    with_structure <- 0
    if(requireNamespace("ChemmineOB", quietly = T)){
      ## structure
      tmp_stru <- paste0(tmp_dir, "/", "structure")
      if(!file.exists(tmp_stru)){
        dir.create(tmp_stru)
      }
      if(plot_structure){
        with_structure <- 1
        batch_mode_structure(metadata = metadata, tmp_stru = tmp_stru)
      }
    }
    ## ------------------------------------
    tmp_ppcp <- paste0(tmp_dir, "/", "ppcp")
    ## draw nodes with class id number
    if(plot_ppcp | plot_structure | plot_ratio ){
      do.call(batch_mode_nodes, list(metadata = metadata,
                                     tmp_ppcp = tmp_ppcp,
                                     with_structure = with_structure,
                                     plot_ppcp = plot_ppcp,
                                     plot_ratio = plot_ratio,
                                     ratio_df = ratio_df,
                                     annotate_ppcp.class.id = annotate_ppcp.class.id,
                                     get_ppcp_legend = T,
                                     ...))
      filepath <- paste0(tmp_ppcp, "/", node_id, ".svg")
      ## mv file
      file.copy(filepath, output)
      file.copy(paste0(tmp_ppcp, "/legend_", node_id, ".svg"), output)
      ## ------------------------------------
    }
  }
```

5 File: beta.call_fun_mc.space.R

```
## set some empty var to local to storage var
call_fun_mc.space <-
```

```
  function(
           FUN,
           args,
           clear_start = T,
           clear_end = T
           ){
    local <- environment()
    if(clear_start){
      rm_mc.set(envir = parent.env(local))
    }
    ## ------------------------------------------------------------------------
    overall_set <- get_mc.global_meta()
    set <- overall_set[names(overall_set) == FUN]
    set <- unlist(set, use.names = F)
    ## ------------------------------------------------------------------------
    lapply(set, function(var){
             assign(var, 0, envir = parent.env(local))
    })
    ## ------------------------------------------------------------------------
    res <- do.call(match.fun(FUN), args)
    ## ------------------------------------
    res <- list(envir = parent.env(local), results = res)
    ## ------------------------------------
    if(clear_end){
      rm_mc.set(envir = parent.env(local))
    }
    return(res)
  }
get_mc.global_meta <-
  function(){
    overall_set <- list(build_classes_tree_list = c(".MCn.class_tree_list"),
                        collate_ppcp = c(".MCn.ppcp_dataset",
                                         ".MCn.class_tree_data",
                                         ".MCn.nebula_class",
                                         ".MCn.nebula_index"),
                        collate_structure = c(".MCn.formula_set",
                                              ".MCn.structure_set"),
                        generate_child_nebulae = c(".MCn.child_graph_list"),
                        generate_parent_nebula = c(".MCn.parent_graph",
                                                   ".MCn.parent_nodes",
                                                   ".MCn.parent_edges"),
```

```
                      initialize_mcnebula = c(".MCn.sirius",
                                              ".MCn.output",
                                              ".MCn.results",
                                              ".MCn.palette",
                                              ".MCn.palette_stat",
                                              ".MCn.palette_ppcp",
                                              ".MCn.palette_label"))
    return(overall_set)
  }
```

6 File: beta.classyfire.batch_get_classification.R

```
## -----------------------------------------------------------------------------
## -----------------------------------------------------------------------------
## -----------------------------------------------------------------------------
## -----------------------------------------------------------------------------
## classyfire curl classification
batch_get_classification <-
  function(
           inchikey2d,
           dir_pubchem,
           dir_classyfire,
           ...
           ){
    rdata <- paste0(dir_classyfire, "/", "class.rdata")
    classes <- extract_rdata_list(rdata)
    ## ------------------------------------
    inchikey2d <- inchikey2d[!inchikey2d %in% names(classes)]
    ## ------------------------------------
    if(length(inchikey2d) == 0)
      return()
    ## ------------------------------------
    inchikey_set <- extract_rdata_list(paste0(dir_pubchem, "/", "inchikey.rdata"), inchikey2d)
    ## ------------------------------------
    list <- lapply(inchikey_set, function(df){
                    if("InChIKey" %in% colnames(df))
                      return(df)
                    return()
           })
    ## ------------------------------------
    ## get classyfire classification
```

```
    df <- data.table::rbindlist(list)
    df <- dplyr::mutate(df, inchikey2d = stringr::str_extract(InChIKey, "^[A-Z]{1,}"))
    ## use the function which writed based on classyfireR::get_classification
    auto_classyfire(df, dir_classyfire, ...)
    ## -----------------------------------
    ## gather classes
    packing_as_rdata_list(dir_classyfire, pattern = "^[A-Z]{14}$", rdata = "class.rdata", extra = classe
  }
## ------------------------------------------------------------------------
auto_classyfire <-
  function(
           df,
           dir_classyfire,
           classyfire_cl = NULL,
           ...
           ){
    ## create access log
    log_file <- paste0(dir_classyfire, "/log")
    if(file.exists(log_file)){
      log_df <- data.table::fread(log_file)
      df <- dplyr::filter(df, !InChIKey %in% log_df$log)
      if(nrow(df) == 0)
        return()
    }
    ## -----------------------------------
    list <- by_group_as_list(df, "inchikey2d")
    ## this part can be multi-threads
    log <- pbapply::pblapply(list, base_auto_classyfire,
                             dir_classyfire = dir_classyfire,
                             cl = classyfire_cl) %>%
      unlist(use.names = F) %>%
      data.table::data.table(log = .)
    ## -----------------------------------
    if(exists("log_df"))
      log <- dplyr::bind_rows(log_df, log)
    write_tsv(log, file = log_file)
  }
base_auto_classyfire <-
  function(
           df,
           dir_classyfire
```

```
          ){
    inchikey2d <- df[1,][["inchikey2d"]]
    log <- lapply(df[["InChIKey"]], base2_classyfire,
                  inchikey2d = inchikey2d,
                  dir_classyfire = dir_classyfire)
    return(unlist(log, use.names = F))
  }
base2_classyfire <-
  function(
          inchikey,
          inchikey2d,
          dir_classyfire
          ){
    file = paste0(dir_classyfire, "/", inchikey2d)
    if(file.exists(file) == F){
      ## if not exists
      ch <- classyfireR::get_classification(inchikey)
    }else{
      return()
    }
    if(is.null(ch)){
      return(inchikey)
    }else{
      ch <- classyfireR::classification(ch)
      write_tsv(ch, file)
      return()
    }
  }
```

7 File: beta.format_quant_table.R

```
format_quant_table <-
  function(
          file,
          meta.group = c(blank = "blank", raw = "raw", pro = "pro"),
          from = "mzmine",
          get_metadata = F
          ){
    df <- data.table::fread(file) %>%
      dplyr::rename(.id = 1, mz = 2, rt = 3) %>%
      dplyr::select(1, 2, 3, contains("Peak area"))
```

```
    ## -------------------------------------
    colnames(df) <- gsub("\\.mz.{0,1}ML Peak area", "", colnames(df))
    metadata <- meta.group %>%
      lapply(function(vec){
                str <- .meta_find_and_sort(colnames(df), vec)
            })
    metadata <- mapply(metadata, names(metadata), SIMPLIFY = F,
                       FUN = function(vec, name){
                         df <- data.table::data.table(group = name, sample = vec)
                         return(df)
                       })
    metadata <- data.table::rbindlist(metadata)
    ## -------------------------------------
    if(get_metadata)
      return(metadata)
    ## -------------------------------------
    stat <- df %>%
      dplyr::select(-mz, -rt) %>%
      ## as long table
      reshape2::melt(id.vars = ".id", variable.name = "sample", value.name = "value") %>%
      merge(metadata, by = "sample", all.y = T) %>%
      ## as data.table
      data.table::data.table() %>%
      dplyr::mutate(value = as.numeric(value)) %>%
      ## calculate average
      .[, list(mean = mean(value, na.rm = T)), by = list(.id, group)] %>%
      ## NAN as 0
      dplyr::mutate(mean = ifelse(is.nan(mean), 0, mean)) %>%
      ## as wide data
      data.table::dcast(.id ~ group, value.var = "mean") %>%
      ## .id is character
      dplyr::mutate(.id = as.character(.id)) %>%
      dplyr::as_tibble()
    return(stat)
  }
## ------------------------------------------------------------------------
.meta_find_and_sort <-
  function(
           name_set,
           pattern_set
           ){
```

```
    name_set <- lapply(pattern_set, .meta_mutate_grep_get,
                       string_set = name_set) %>%
      unlist()
    return(name_set)
  }
.meta_mutate_grep_get <-
  function(
           pattern,
           string_set
           ){
    string <- string_set %>%
      .[grepl(pattern, .)]
    return(string)
  }
## ------------------------------------------------------------------------
```

8 File: beta.method__target__spec__compare.R

```
## target compare specture in classes
method_target_spec_compare <-
  function(
           nebula_name,
           nebula_index = .MCn.nebula_index,
           output = paste0(.MCn.output, "/", .MCn.results, "/", nebula_name, ".spec.tsv"),
           edge_filter = 0.5,
           ...
           ){
    idset <- dplyr::filter(nebula_index, name %in% nebula_name)$.id
    edges <- method_formula_based_spec_compare(
      target_ids = idset,
      only_identical_class = F,
      min_hierarchy = 1,
      ...
    )
    write_tsv(edges, output)
    return(output)
  }
```

9 File: beta.nebula_get_candidate.R

```
nebula_get_candidate <-
  function(
           ...,
           path = "mcnebula_results"
           ){
    path <- paste0(path, "/candidates")
    if(file.exists(path) == F)
      dir.create(path)
    args <- list(
                 ...,
                 top_n = 50,
                 match_pattern = NULL, ## or c("precursorFormula", "adduct") or NULL
                 collate_factor = NA,
                 revise_MCn_formula_set = F,
                 revise_MCn_structure_set = F,
                 only_gather_structure = T
    )
    candidates <- do.call(nebula_re_rank, args)
    write_tsv(candidates, paste0(path, "/", args[[1]], ".tsv"))
  }
```

10 File: beta.packing_as_rdata_list.R

```
extract_rdata_list <-
  function(
           rdata,
           names = NA
           ){
    if(!file.exists(rdata))
      return()
    load(rdata)
    if(!is.na(names[1])){
      list <- list[names(list) %in% names]
    }
    return(list)
  }
packing_as_rdata_list <-
  function(
           path,
```

```
          pattern,
          rdata,
          extra = NULL,
          rm_files = T,
          dedup = T
          ){
  file_set <- list.files(path, pattern = pattern)
  if(length(file_set) == 0)
    return()
  ## read as list
  list <- pbapply::pblapply(paste0(path, "/", file_set), read_tsv)
  names(list) <- file_set
  ## merge
  list <- c(extra, list)
  ## according to name, unique
  if(dedup){
    df <- data.table::data.table(name = names(list), n = 1:length(list))
    df <- dplyr::distinct(df, name, .keep_all = T)
    list <- list[df$n]
  }
  ## rm origin file sets
  if(rm_files){
    lapply(paste0(path, "/", file_set), file.remove)
  }
  ## save as rdata
  save(list, file = paste0(path, "/", rdata))
}
```

11 File: beta.pubchem_curl_inchikey.R

```
## ------------------------------------------------------------------------
## ------------------------------------------------------------------------
## ------------------------------------------------------------------------
## ------------------------------------------------------------------------
## pubchem curl inchikey
pubchem_curl_inchikey <-
  function(
          inchikey2d,
          dir,
          curl_cl = NULL,
          gather_as_rdata = T,
```

```
          ...
          ){
    rdata <- paste0(dir, "/", "inchikey.rdata")
    inchikey_set <- extract_rdata_list(rdata)
    ## ------------------------------------
    inchikey2d <- inchikey2d[!inchikey2d %in% names(inchikey_set)]
    ## ------------------------------------
    if(length(inchikey2d) == 0)
      return()
    ## ------------------------------------
    pbapply::pblapply(inchikey2d, base_pubchem_curl_inchikey,
                      dir = dir, cl = curl_cl, ...)
    ## ------------------------------------
    cat("## gather InChIKey\n")
    packing_as_rdata_list(dir, pattern = "^[A-Z]{14}$",
                          rdata = "inchikey.rdata", extra = inchikey_set)
  }
base_pubchem_curl_inchikey <-
  function(
          inchikey2d,
          dir,
          type = "inchikey",
          get = "InChIkey",
          ...
          ){
    file <- paste0(dir, "/", inchikey2d)
    ## ------------------------------------
    ## if exists and valid, return
    if(file.exists(file)){
      csv <- read_tsv(file)
      if("CID" %in% colnames(csv))
        return()
    }
    ## ------------------------------------
    ## curl via inchikey2d, which the same as InChIKey
    url_start = paste0("https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/", type, "/")
    ## to get InChIKey, and get data as csv format
    url_end = paste0("/property/", paste(get, collapse = ","), "/CSV")
    ## gather url
    url = paste0(url_start, "/", inchikey2d, "/", url_end)
    ## ------------------------------------
```

```
    csv <- RCurl::getURL(url)
    ## ------------------------------------
    ## check results, if 404, return()
    if(grepl("Status: 404", csv)){
      write_tsv(csv, file = file)
      return()
    }
    ## ------------------
    ## if 503, system busy, try again
    while(grepl("Status:    503", csv)){
      csv <- RCurl::getURL(url)
    }
    ## ------------------------------------
    csv <- data.table::fread(text = csv)
    write_tsv(csv, file = file)
  }
```

12 File: beta.pubchem_get_synonyms.R

```
pubchem_get_synonyms <-
  function(
           cid,
           dir,
           curl_cl = NULL,
           gather_as_rdata = T,
           group_number = 50,
           ...
           ){
    rdata <- paste0(dir, "/", "cid.rdata")
    ## extract as list
    cid_set <- extract_rdata_list(rdata)
    ## as data.table
    cid_set <- data.table::rbindlist(cid_set)
    ## !duplicated
    if("cid" %in% colnames(cid_set)){
      cid_set <- cid_set %>%
        dplyr::distinct(cid, syno)
    }
    ## ------------------------------------
    ## exclude existing
    cid <- cid[!cid %in% cid_set$cid]
```

```
    ## ------------------------------------
    if(length(cid) == 0)
      return()
    ## ------------------------------------
    group <- grouping_vec2list(cid, group_number = group_number)
    ## ------------------------------------
    pbapply::pblapply(group, base_pubchem_get_synonyms,
                      dir = dir, cl = curl_cl, ...)
    ## ------------------------------------
    cat("## gather synonyms\n")
    packing_as_rdata_list(dir, pattern = "^G[0-9]{1,}$",
                          dedup = F,
                          rdata = "cid.rdata",
                          ## data.table as list
                          extra = list(cid_set))
  }
base_pubchem_get_synonyms <-
  function(
           cid,
           dir,
           ...
           ){
    savename <- attr(cid, "name")
    file <- paste0(dir, "/", savename)
    ## gather cid and sep by ,
    cid <- paste(cid, collapse = ",")
    ## use cid
    url_start <- "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/"
    ## get as XML
    url_end <- "/synonyms/XML"
    ## paste as url
    url <- paste0(url_start, cid, url_end)
    ## ------------------------------------
    check <- 0
    while(check == 0 | class(check)[1] == "try-error"){
      check <- try(text <- RCurl::getURL(url), silent = T)
    }
    ## ------------------------------------
    while(grepl("Status:    503", text)){
      text <- RCurl::getURL(url)
    }
```

```
    ## "PUGREST.BadRequest"
    ## -------------------------------------
    ## convert to list
    text <- XML::xmlToList(text)
    ## only 'information'
    text <- text[names(text) == "Information"]
    ## in list to separate data
    text <- lapply(text, function(list){
                    syno <- list[names(list) == "Synonym"]
                    syno <- lapply(syno,
                                   function(char){
                                     if(is.null(char)){
                                       return(NA)
                                     }else{
                                       return(char)
                                     }
                                   })
                    data.table::data.table(cid = list$CID, syno = unlist(syno))
            })
    text <- data.table::rbindlist(text, fill = T)
    ## -------------------------------------
    ## save data
    write_tsv(text, filename = file)
  }
## -------------------------------------
grouping_vec2list <-
  function(
           vector,
           group_number,
           byrow = F
           ){
    if(length(vector) < group_number){
      attr(vector, "name") <- "G1"
      return(list(vector))
    }
    ## if grouped, the rest number
    rest <- length(vector) %% group_number
    ## assign group
    group <- matrix(vector[1:(length(vector) - rest)],
                    ncol = group_number,
                    byrow = byrow) %>%
```

```
    ## use apply to multiple list
    apply(1, c, simplify = F) %>%
    ## gather the rest vector
    c(., list(tail(vector, n = rest))) %>%
    ## add group name
    mapply(FUN = function(vec, name){
             attr(vec, "name") <- name
             return(vec)
           }, ., paste0("G", 1:length(.)),
           SIMPLIFY = F)
  ## ------------------------------------
  if(rest == 0)
    group <- group[1:(length(group) - 1)]
  return(group)
}
```

13 File: beta.vis_via_molconvert.R

```
vis_via_molconvert_nebulae <-
  function(
           nebula_name
           ){
    df <- dplyr::filter(.MCn.nebula_index, name == nebula_name)
    stru <- dplyr::filter(.MCn.structure_set, .id %in% df$".id")
    vis_via_molconvert(stru$smiles, stru$".id")
    return("Done")
  }
vis_via_molconvert <-
  function(
           smiles_set,
           id_set,
           output = paste0(.MCn.output, "/", .MCn.results, "/tmp/structure")
           ){
    if(!file.exists(output)){
      dir.create(output, recursive = T)
    }
    pbapply::pbmapply(molconvert_structure,
           smiles_set,
           id_set,
           MoreArgs = list(output = output)
    )
```

```
    return("Done")
  }
## ------------------------------------------------------------------------
molconvert_structure <-
  function(
           smiles,
           id,
           output = paste0(.MCn.output, "/", .MCn.results, "/tmp/structure")
           ){
    file = tempfile()
    system(paste0("molconvert mol \"", smiles, "\" -o ", file))
    system(paste0("obabel ", file, " -imol -osvg -O ", file, " > /dev/null 2>&1"))
    system(paste0("sed -i \'s/white/transparent/g\' ", file))
    system(paste0("cairosvg -f svg ", file, " -o ", output, "/", id, ".svg"))
    return()
  }
```

14 File: build_classes_tree_list.R

```
#' @title build_classes_tree_list
#' @description According to source data (table) of classification in SIRIUS project to build list in h
#' @param class_index Character, source data name, Default: 'canopus.tsv'
#' @param path SIRIUS project path, Default: .MCn.sirius
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[dplyr]{reexports}}
#' @rdname build_classes_tree_list
#' @export
#' @importFrom dplyr as_tibble
build_classes_tree_list <-
  function(
           class_index="canopus.tsv", path=.MCn.sirius
           ){
    data <- read_tsv(paste0(path, "/", class_index))
    ## ------------------------------------------------------------------------
```

```
    ## separate each levels of classes into sub-list
    root <- data[which(data$parentId==""), ]
    list <- list()
    n = 1
    list[[n]] <- dplyr::as_tibble(root)
    df <- data[data$parentId %in% root$id, ]
    ## ------------------------------------------------------------------------
    while(nrow(df) > 0){
      n = n + 1
      list[[n]] <- dplyr::as_tibble(df)
      df <- data[data$parentId %in% df$id, ]
    }
    .MCn.class_tree_list <<- list
    cat("INFO: Classification Index in.MCn.sirius project --->", class_index, "\nA total of 11 levels.\n
        Use following arguments to get some specific classes:
        .MCn.class_tree_list[[3]] >>> superclass
        .MCn.class_tree_list[[4]] >>> class
        .MCn.class_tree_list[[5]] >>> subclass
        .MCn.class_tree_list[[6]] >>> level 5 \n")
}
```

15 File: by_group_as_list.R

```
by_group_as_list <-
  function(
           df,
           colnames
           ){
    vector <- unique(df[[colnames]])
    list <- lapply(vector, by_group_as_list_select,
                   df = df,
                   colNames = colnames)
    names(list) <- vector
    return(list)
  }
by_group_as_list_select <-
  function(
           KEY,
           df,
           colNames
           ){
```

```
    df <- df[which(df[[colNames]] == KEY), ]
    return(df)
  }
```

16 File: collate_ppcp.R

```
#' @title collate_ppcp
#' @description Collate posterior probability of classification prediction (PPCP) from SIRIUS project
#' and conduct integration to get nebula_class and nebula-index.
#' @param dirs Vector, Default: 'all'
#' @param write_output Logic, Default: T
#' @param nebula_class Logic, Default: T
#' @param nebula_index Logic, Default: T
#' @param ... ...
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[pbapply]{pbapply}}
#'  \code{\link[dplyr]{rename}}, \code{\link[dplyr]{mutate}}, \code{\link[dplyr]{filter}}, \code{\link[
#'  \code{\link[data.table]{rbindlist}}
#' @rdname collate_ppcp
#' @export
#' @importFrom pbapply pbsapply pblapply
#' @importFrom dplyr rename mutate filter as_tibble
#' @importFrom data.table rbindlist
collate_ppcp <-
  function(
           dirs = "all",
           write_output = T,
           nebula_class = T,
           nebula_index = T,
           ...
           ){
    cat("[INFO] MCnebula run: collate_ppcp\n")
    ## ----------------------------------------------------------------------
    ## check dirs ---- canopus
```

```
  cat("## collate_ppcp: check_dir\n")
  if(length(dirs) == 1 & dirs[1] == "all"){
    dirs <- list.files(path = .MCn.sirius, pattern = "^[0-9](.*)_(.*)_(.*)$", full.names = F)
    check <- pbapply::pbsapply(dirs, check_dir, file = "canopus") %>% unname
  }else{
    check <- pbapply::pbsapply(dirs, check_dir, file = "canopus") %>% unname
  }
  ## -----------------------------------------------------------------------
  ## lock on file location
  meta_dir <- dirs[which(check == T)] %>%
    data.frame() %>%
    dplyr::rename(dir = ".") %>%
    dplyr::mutate(.id = grep_id(dir)) %>%
    merge(.MCn.formula_set, by = ".id", all.x = T, sort = F) %>%
    dplyr::mutate(adduct_trans = gsub(" ", "", adduct),
           target = paste0(precursorFormula, "_", adduct_trans, ".fpt"),
           full.name = paste0(.MCn.sirius, "/", dir, "/", "canopus", "/", target),
           ## these files need to be check and filter (whether exist)
           ## note that some formula is no fingerprint computed
           ppcp = file.exists(full.name))
    ## ------------------------------------
  meta_dir_filter <- dplyr::filter(meta_dir, ppcp == T)
  cat("## STAT of PPCP dataset:",
      paste0(nrow(meta_dir_filter), "(formula with PPCP)", "/", nrow(meta_dir), "(all formula)"),
      "\n")
  ## -----------------------------------------------------------------------
  ## load all ppcp dataset
  if(!exists(".MCn.ppcp_dataset")){
    ppcp_dataset <- pbapply::pblapply(meta_dir_filter$full.name, read_fpt)
    names(ppcp_dataset) <- meta_dir_filter$".id"
    .MCn.ppcp_dataset <<- ppcp_dataset
  }else{
    ppcp_dataset <- .MCn.ppcp_dataset
  }
  ## -----------------------------------------------------------------------
  ## summarize nebula_class
  if(nebula_class){
    cat("## collate_ppcp: method_summarize_nebula_class\n")
    metadata <- data.table::rbindlist(.MCn.class_tree_list, idcol = T) %>%
      dplyr::rename(hierarchy = .id)
    ## ------------------------------------
```

```
    ## set as global var
    .MCn.class_tree_data <<- dplyr::as_tibble(metadata)
    ## ------------------------------------
    assign("envir_meta", environment(), envir = parent.env(environment()))
    ## get nebula classes
    nebula_class <- pbapply::pblapply(ppcp_dataset, method_summarize_nebula_class,
                          class_data_type = "classes_tree_data",
                          ...)
    ## ------------------------------------
    .MCn.nebula_class <<- nebula_class
  }
  ## ------------------------------------------------------------------------
  if(nebula_index){
    cat("## collate_ppcp: method_summarize_nebula_index.\n")
    ## gather all nebula classes
    nebula_index <- method_summarize_nebula_index(ppcp_dataset,
                                                  ...)
    .MCn.nebula_index <<- nebula_index
  ## ------------------------------------------------------------------------
    if(write_output){
      output = paste0(.MCn.output, "/", .MCn.results)
      write_tsv(nebula_index, file = paste0(output, "/", "nebula_index.tsv"))
    }
  }
  ## ------------------------------------------------------------------------
  rm("envir_meta", envir = parent.env(environment()))
  ## ------------------------------------------------------------------------
  cat("[INFO] MCnebula Job Done: collate_ppcp.\n")
  return(nebula_index)
}
```

17 File: collate_structure.R

```
#' @title collate_structure
#' @description Collate chemical structure data from SIRIUS project
#' @param dirs Vector, Default: 'all'
#' @param write_output Logic, Default: T
#' @param ... ...
#' @details DETAILS
#' @examples
#' \dontrun{
```

```
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[pbapply]{pbapply}}
#'  \code{\link[data.table]{rbindlist}}
#'  \code{\link[dplyr]{mutate}}, \code{\link[dplyr]{relocate}}, \code{\link[dplyr]{reexports}}
#' @rdname collate_structure
#' @export
#' @importFrom pbapply pbsapply pbmapply
#' @importFrom data.table rbindlist
#' @importFrom dplyr mutate relocate as_tibble
collate_structure <-
  function(
           dirs = "all",
           write_output = T,
           ...
           ){
  cat( paste0("[INFO] MCnebula run: collate_structure\n") )
  ## ------------------------------------------------------------------------
  ## check dirs
  cat("## collate_structure: check_dir\n")
  if(length(dirs) == 1 | dirs[1] == "all"){
    dirs <- list.files(path = .MCn.sirius, pattern="^[0-9](.*)_(.*)_(.*)$", full.names = F)
    check <- pbapply::pbsapply(dirs, check_dir) %>%
      unname()
  }else{
    check <- pbapply::pbsapply(dirs, check_dir) %>%
      unname()
  }
  dirs <- dirs[which(check == T)]
  ## ------------------------------------------------------------------------
  cat("## collate_structure: method_pick_formula_excellent\n")
  formula_set <- method_pick_formula_excellent(dir = dirs, ...)
  ## ------------------------------------
  ## set as global var
  .MCn.formula_set <<- formula_set
  ## ------------------------------------------------------------------------
  ## structure collate
  cat("## collate_structure: re-collate structure\n")
```

```
  structure_dataset <- pbapply::pbmapply(mutate2_get_structure,
                                         formula_set$.id,
                                         formula_set$precursorFormula,
                                         formula_set$adduct,
                                         SIMPLIFY = F)
  structure_dataset <- data.table::rbindlist(structure_dataset, fill = T) %>%
    ## debug
    dplyr::mutate(tanimotoSimilarity = as.numeric(tanimotoSimilarity)) %>%
    dplyr::relocate(.id) %>%
    dplyr::as_tibble()
  ## ------------------------------------
  ## set as global var
  .MCn.structure_set <<- structure_dataset
  ## ------------------------------------------------------------------------
  ## write output
  if(write_output == T){
    output = paste0(.MCn.output, "/", .MCn.results)
    write_tsv(structure_dataset, paste0(output, "/", "method_pick_formula_excellent", ".structure.tsv"))
    write_tsv(formula_set, paste0(output, "/", "method_pick_formula_excellent", ".tsv"))
  }
  ## ------------------------------------------------------------------------
  cat( paste0("[INFO] MCnebula Job Done: collate_structure.\n") )
}
grep_id <- function(x){
  stringr::str_extract(x, "(?<=_)[^_]{1,}$")
}
check_dir <- function(dir, path = .MCn.sirius, file = "compound.info"){
  if(file.exists(paste0(path, "/", dir, "/", file)) == T){
    check = T
  }else{
    check = F
  }
  return(check)
}
mutate2_get_structure <-
  function(
           key_id,
           formula,
           adduct
           ){
    df <- try(silent = T, get_structure(key_id,
```

```
                                        precursor_formula = formula,
                                        adduct = adduct,
                                        return_row = 1))
    if(class(df)[1] == "try-error"){
      return()
    }else{
      df$".id" <- key_id
      return(df)
    }
  }
```

18 File: generate_child_nebulae.R

```
#' @title generate_child_nebulae
#' @description According to nodes and edges data of parent-nebula and nebula-index to generate child-n
#' @param nodes A data.frame, Default: .MCn.parent_nodes
#' @param edges A data.frame, Default: .MCn.parent_edges
#' @param max_edges A number, Default: 5
#' @param nebula_index A data.frame, Default: .MCn.nebula_index
#' @param output Character, Default: paste0(.MCn.output, "/", .MCn.results)
#' @param output_format Character, 'igraph' supported format, Default: 'graphml'
#' @param ... ...
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[pbapply]{pbapply}}
#' @rdname generate_child_nebulae
#' @export
#' @importFrom pbapply pblapply
generate_child_nebulae <-
  function(
           nodes = .MCn.parent_nodes,
           edges = .MCn.parent_edges,
           max_edges = 5,
           nebula_index = .MCn.nebula_index,
           output = paste0(.MCn.output, "/", .MCn.results),
```

```
            output_format = "graphml",
            ...
            ){
    ##
    cat("[INFO] MCnebula run: generate_child_nebulae\n")
    assign("envir_nebula", environment(), envir = parent.env(environment()))
    ## get names of all classes
    names <- unique(nebula_index$name)
    ## for using lapply, first, trans the data.frame into list
    nebula_index <- by_group_as_list(nebula_index, "relativeIndex")
    ## push names
    names(nebula_index) <- names
    ## facet parent nebula
    ## create dir for placing graph file
    dir = paste0(output, "/", "child_nebula")
    if(file.exists(dir) == F){
      dir.create(dir)
    }
    .MCn.child_graph_list <<- pbapply::pblapply(nebula_index, separate_nebula,
                      output = dir,
                      max_edges = max_edges,
                      output_format = output_format,
                      ...)
    rm("envir_nebula", envir = parent.env(environment()))
    cat("[INFO] MCnebula Job Done: generate_child_nebulae\n")
  }
separate_nebula <-
  function(
            df,
            nodes = get("nodes", envir = get("envir_nebula")),
            edges = get("edges", envir = get("envir_nebula")),
            write_output = T,
            output = paste0(.MCn.output, "/", .MCn.results, "/", "child_nebula"),
            output_format = "graphml",
            max_edges = 5,
            write_extra = F
            ){
    id <- unique(df$".id")
    ## get the child nebula name
    ## note that some character in name caused fail to write as file into dir
    name <- gsub("/", "#", df[1,]$"name")
```

```
    nodes <- nodes[nodes$".id" %in% id, ]
    edges <- edges[edges$".id_1" %in% id & edges$".id_2" %in% id, ]
    ## an edges number cut-off
    edges <- better_vis_nebula(edges, max_edges = max_edges)
    child_nebula <- igraph::graph_from_data_frame(edges, directed = T, vertices = nodes)
    if(write_output == T){
      igraph::write_graph(child_nebula,
                  file = paste0(output, "/", name, ".", output_format),
                  format = output_format)
      if(write_extra == T){
        write_tsv(edges, paste0(output, "/", name, "_edges.tsv"))
        write_tsv(nodes, paste0(output, "/", name, "_nodes.tsv"))
      }
    }
    return(child_nebula)
  }
better_vis_nebula <-
  function(
           edges,
           max_edges = 5
           ){
    ## order
    edges <- dplyr::arrange(edges, desc(edges[,3]))
    ta <- table(c(edges[[1]], edges[[2]]))
    ## at least loop number
    n <- length(which(ta > max_edges))
    if(n == 0){
      return(edges)
    }
    ## ------------------------------------------------------------------------
    ## copy data for override
    df <- dplyr::mutate(edges[, 1:2], SEQ = 1:nrow(edges))
    assign("envir_meta", environment(), envir = parent.env(environment()))
    ## use sapply instead of while loop
    continue = 1
    sapply(1:n, edges_cut_off, max_edges = max_edges)
    ## ------------------------------------------------------------------------
    edges <- edges[df$SEQ, ]
    rm("envir_meta", envir = parent.env(environment()))
    return(edges)
  }
```

```r
edges_cut_off <-
  function(
           i,
           max_edges = 5
           ){
    continue = get("continue", envir = get("envir_meta"))
    if(continue == 1){
      edges = get("df", envir = get("envir_meta"))
      ## stat edges number of an id
      ta <- table(c(edges[[1]], edges[[2]]))
      ## ----------------------------------------------------------------------
      if(max(ta) > max_edges){ ## greater than threshold, hence perform exclude
        ## select an id to exclude its excess edges
        key_id <- names(ta[which(ta == max(ta))])[1]
        ## get SEQ of the edges which need to be excluded
        incude_id_edges <- edges[which(edges[[1]] == key_id | edges[[2]] == key_id),]
        exclude_edges_seq <- incude_id_edges[-(1:max_edges), ]$SEQ
        ## exclude edges
        edges <- edges[which(!edges$SEQ %in% exclude_edges_seq), ]
        assign("df", edges, envir = get("envir_meta"))
      }else{
      ## ----------------------------------------------------------------------
        ## signature of stop exclude
        assign("continue", 0, envir = get("envir_meta"))
      }
    }else{
      return()
    }
  }
```

19 File: generate_parent_nebula.R

```r
#' @title generate_parent_nebula
#' @description According to formula dataset and structure dataset to generate parent-nebula graph
#' @param write_output Logic, Default: T
#' @param output_format Character, 'igraph' supported format, Default: 'graphml'
#' @param output Character, Default: paste0(.MCn.output, "/", .MCn.results)
#' @param edges_file Character, path to edges file, Default: paste0(output, "/parent_nebula/parent_nebu
#' @param edges_method Character, Default: 'method_formula_based_spec_compare'
#' @param nodes_attributes A data.frame, formula dataset, Default: .MCn.formula_set
#' @param nodes_other_attributes A data.frame, structure dataset, Default: .MCn.structure_set
```

```
#' @param rm_parent_isolate_nodes Logic, Default: F
#' @param edge_filter A number, Default: 0.5
#' @param cpu_cores A number, thread count, Default: 8
#' @param ... ...
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[dplyr]{reexports}}, \code{\link[dplyr]{mutate}}, \code{\link[dplyr]{mutate_all}}, \code
#'  \code{\link[igraph]{as_data_frame}}, \code{\link[igraph]{write_graph}}
#' @rdname generate_parent_nebula
#' @export
#' @importFrom dplyr as_tibble mutate mutate_at rename filter
#' @importFrom igraph graph_from_data_frame write_graph
generate_parent_nebula <-
  function(
           write_output = T,
           output_format = "graphml",
           output = paste0(.MCn.output, "/", .MCn.results),
           # exists edges file
           edges_file = paste0(output, "/parent_nebula/parent_nebula_edges.tsv"),
           # or NULL
           edges_method = "method_formula_based_spec_compare",
           nodes_attributes = .MCn.formula_set,
           nodes_other_attributes = .MCn.structure_set,
           rm_parent_isolate_nodes = F,
           edge_filter = 0.5,
           cpu_cores = 8,
           ...
           ){
    cat("[INFO] MCnebula run: generate_parent_nebula\n")
    ## main body
    ## ----------------------------------------------------------------------
    ## generate edges data
    if(is.null(edges_method)){
      ## no edges_method
      cat("## generate_parent_nebula: no edges_uethod used\n")
```

```
  edges <- dplyr::as_tibble(cbind(".id_1" = nodes_other_attributes$".id",
                ".id_2" = nodes_other_attributes$".id")) %>%
    dplyr::mutate(dotproduct = 1, mass_diff = 0)
}else if(edges_method == "method_formula_based_spec_compare"){
  ## with edges_method
  if(!is.null(edges_file) & file.exists(edges_file)){
    cat("## generate_parent_nebula: file.exists(edges_file) == T.\nEscape from time-consuming comput
    edges <- read_tsv(edges_file) %>%
      dplyr::mutate_at(c(".id_1", ".id_2"), as.character) %>%
      dplyr::mutate_at(c(colnames(.)[3:4]), as.numeric)
  }else{
    cat("## generate_parent_nebula: method_formula_based_spec_compare\n")
    edges = method_formula_based_spec_compare(edge_filter = edge_filter, cpu_cores = cpu_cores, ...)
  }
}
## ------------------------------------------------------------------------
## generate nodes data
nodes <- nodes_attributes
## additional nodes attributes
if(is.data.frame(nodes_other_attributes)){
  nodes <- merge(nodes, nodes_other_attributes, by = ".id", all.x = T, sort = T) %>%
    ## rename the column name, otherwise the column will be choosed as key column in igraph
    dplyr::rename(compound_name = name)
}
if(rm_parent_isolate_nodes){
  non_iso_nodes <- unique(c(edges$.id_1, edges$.id_2))
  nodes.parent <- dplyr::filter(nodes, .id %in% all_of(non_iso_nodes))
}else{
  nodes.parent <- nodes
}
## ------------------------------------------------------------------------
## graph
parent_nebula <- igraph::graph_from_data_frame(edges, directed = T, vertices = nodes.parent)
if(write_output){
  dir = paste0(output, "/", "parent_nebula")
  if(!file.exists(dir)){
    dir.create(dir)
  }
  igraph::write_graph(parent_nebula,
              file = paste0(dir, "/", "parent_nebula.", output_format),
              format = output_format)
```

```
    write_tsv(edges, paste0(dir, "/", "parent_nebula_edges.tsv"))
    write_tsv(nodes, paste0(dir, "/", "parent_nebula_nodes.tsv"))
  }
  ## ------------------------------------------------------------------------
  ## set as global var for next stage
  .MCn.parent_graph <<- parent_nebula
  .MCn.parent_nodes <<- as_tibble(nodes)
  .MCn.parent_edges <<- as_tibble(edges)
  cat("[INFO] MCnebula Job Done: generate_parent_nebula\n")
}
```

20 File: get_formula.R

```
#' @title get_formula
#' @description A function to read table of formula data of feature in SIRIUS project.
#' @param key_id Character.
#' @param exclude_element Vector, Default: NULL
#' @param formula_method Character, Default: 'top_zodiac'
#' @param rank Vector of number, Default: 1:5
#' @param ppm_error A bumber, Default: 20
#' @param return_col Vector of character, Default: c("rank", "precursorFormula",
# '   "molecularFormula", "adduct", "ZodiacScore",
#'    "massErrorPrecursor(ppm)")
#' @param ... ...
#' @return A data.frame
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @rdname get_formula
#' @export
get_formula <-
  function(
           key_id,
           exclude_element = NULL, ## e.g., c("S", "B", "P", "Si")
           formula_method = "top_zodiac",
           rank = 1:5, # or "all"
           ppm_error = 20,
```

```
          return_col = c("rank", "precursorFormula", "molecularFormula",
                         "adduct", "ZodiacScore", "massErrorPrecursor(ppm)"),
          ...
          ){
    path <- list.files(path = .MCn.sirius, pattern=paste0("*_", key_id, "$"), full.names=T)
    file <- read_tsv(paste0(path, "/", "formula_candidates.tsv"))
    file$rank <- as.numeric(file$rank)
    ## ------------------------------------------------------------------------
    if("ZodiacScore" %in% colnames(file) == F){
      file$ZodiacScore = 0
    }
    ## ------------------------------------------------------------------------
    if(is.null(exclude_element) == F){
      file <- file[!unname(sapply(file$precursorFormula, grep_element,
                                  exclude_element = exclude_element)), ]
    }
    ## ------------------------------------------------------------------------
    if(formula_method == "top_zodiac"){
      if(rank[1] == "all"){
        rank <- unique(file$rank)
      }
      ## to escape the key in data.table
      filter = rank
      file <- file[abs(file$"massErrorPrecursor(ppm)") <= ppm_error &
                   file$"rank" %in% filter,
                   return_col, with = F]
    }
    return(file)
}
## ------------------------------------------------------------------------
grep_element <-
  function(
          formula,
          exclude_element = c("S", "P", "B")
          ){
    check <- grepl(paste(exclude_element, collapse = "|"), formula)
    return(check)
  }
```

21 File: get_ppcp.R

```r
#' @title FUNCTION_TITLE
#' @description FUNCTION_DESCRIPTION
#' @param key_id PARAM_DESCRIPTION, Default: NULL
#' @param dir PARAM_DESCRIPTION, Default: NULL
#' @param precursor_formula PARAM_DESCRIPTION, Default: 'method_pick_formula_excellent'
#' @param adduct PARAM_DESCRIPTION, Default: NULL
#' @param reformat PARAM_DESCRIPTION, Default: T
#' @param filter PARAM_DESCRIPTION, Default: T
#' @param filter_threshold PARAM_DESCRIPTION, Default: 0.1
#' @param class_index PARAM_DESCRIPTION, Default: 'canopus.tsv'
#' @param ... PARAM_DESCRIPTION
#' @return OUTPUT_DESCRIPTION
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @rdname get_ppcp
#' @export
get_ppcp <-
  function(
           key_id = NULL,
           dir = NULL,
           precursor_formula = "method_pick_formula_excellent",
           adduct = NULL,
           reformat = T,
           filter = T,
           filter_threshold = 0.1,
           class_index = "canopus.tsv",
           ...
           ){
    ## get dir path
    if(is.null(dir) & is.null(key_id)){
      return()
    }else if(is.null(dir)){
      dir <- get_dir(key_id)
    }
    ## ------------------------------------------------
```

```
    ## aquire formula via the method
    if( precursor_formula == "method_pick_formula_excellent" ){
      meta <- method_pick_formula_excellent(dir = dir)
      precursor_formula <- meta$precursorFormula
      adduct <- meta$adduct
    }
    ## ---------------------------------------------------
    ## read ppcp data
    file <- list.files(path = paste0(.MCn.sirius, "/", dir, "/", "canopus"),
                       pattern = paste0("^", precursor_formula, "(.*)", escape_ch(adduct), "(.*)", ".fp
                       full.names = T)
    ppcp <- read_fpt(file)
    ## ---------------------------------------------------
    ## reformat section
    if(!reformat){
      return(ppcp)
    }
    ## check meta list
    if(!exists(".MCn.class_tree_list")){
      build_classes_tree_list(class_index = class_index)
    }
    ## merge with meta table, and filter
    ppcp <- lapply(.MCn.class_tree_list, merge_class_ppcp,
                   ## parameter
                   key_id = key_id,
                   values = ppcp,
                   filter = filter,
                   filter_threshold = filter_threshold)
    return(ppcp)
  }
## a small function to get data of ppcp
read_fpt <- function(file){
  fpt = data.table::fread(input = file, header = F, quote = "")
  fpt$relativeIndex = seq(0, nrow(fpt) - 1)
  return(fpt)
}
## specific character in adduct description need to be revise, for pattern matching
escape_ch <- function(x){
  x <- gsub("\\[", "\\\\\\[", x)
  x <- gsub("\\]", "\\\\\\]", x)
  x <- gsub("\\+", "\\\\\\+", x)
```

```
  x <- gsub("\\-", "\\\\\\-", x)
  x <- gsub(" ", "", x)
  return(x)
}
## the function to merge raw ppcp with meta list
merge_class_ppcp <-
  function(
           class,
           values,
           filter = T,
           filter_threshold = 0.1,
           key_id = NULL,
           filter_col = "V1"
           ){
    df <- merge(class, values, all.x = T, by = "relativeIndex", sort = F)
    df <- df[which(df[[filter_col]] > ifelse(filter == T, filter_threshold, 0)),] %>%
      dplyr::as_tibble()
    return(df)
  }
```

22 File: get_structure.R

```
#' @title FUNCTION_TITLE
#' @description FUNCTION_DESCRIPTION
#' @param key_id PARAM_DESCRIPTION
#' @param precursor_formula PARAM_DESCRIPTION, Default: NULL
#' @param adduct PARAM_DESCRIPTION, Default: NULL
#' @param structure_method PARAM_DESCRIPTION, Default: 'top_score'
#' @param order PARAM_DESCRIPTION, Default: T
#' @param return_row PARAM_DESCRIPTION, Default: 1:10
#' @param as_tibble PARAM_DESCRIPTION, Default: F
#' @param ... PARAM_DESCRIPTION
#' @return OUTPUT_DESCRIPTION
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
```

```
#'  \code{\link[data.table]{rbindlist}}
#'  \code{\link[dplyr]{reexports}}
#' @rdname get_structure
#' @export
#' @importFrom data.table rbindlist
#' @importFrom dplyr as_tibble
get_structure <-
  function(
           key_id,
           precursor_formula = NULL,
           adduct = NULL,
           structure_method = "top_score", # or top_similarity
           order = T,
           return_row = 1:10, # or "all"
           as_tibble = F,
           ...
           ){
    path <- list.files(path = .MCn.sirius, pattern = paste0("*_", key_id, "$"), full.names = T)
    files <- list.files(paste0(path, "/fingerid"),
                        pattern = paste0(precursor_formula, "(.*)", escape_ch(adduct), "(.*).tsv$"),
                        full.names = F)
    ## ----------------------------------------------------
    ## read file
    list <- lapply(paste0(path, "/fingerid/", files), read_tsv)
    names(list) <- gsub(".tsv", "", files)
    ## ----------------------------------------------------
    ## reformat the data
    if(order == T){
      ## bind row as data frame
      df <- data.table::rbindlist(list, idcol = T)
      colnames(df)[which(colnames(df) == ".id")] <- "file_name"
      if(nrow(df) == 0){
        return(df)
      }
      ## ----------------------------------------------------
      ## order upon CSI:fingerID score
      if(structure_method == "top_score"){
        df <- df[order(-df$score),]
        df$structure_rank = 1:nrow(df)
      ## order upon tanimoto similarity
      }else if(structure_method == "top_similarity"){
```

```
        df$tanimotoSimilarity <- as.numeric(df$tanimotoSimilarity)
        df <- df[order(-df$tanimotoSimilarity),]
        df$structure_rank = 1:nrow(df)
      }
      ## ------------------------------------------------
      if(as_tibble == T){
        df <- dplyr::as_tibble(df)
      }
      # return with top n
      if(return_row[1] != "all"){
        return(df[which(1:nrow(df) %in% return_row),])
      }else{
        return(df)
      }
    }
    return(list)
}
```

23 File: grid_child_nebula.R

```
grid_child_nebula <-
  function(
           graph,
           anno = c(nebula_index = "classification"),
           class,
           layout = "fr",
           title_palette = .MCn.palette_label,
           palette = .MCn.palette,
           print_into = T,
           save_layout_df = NA,
           remove_nodes = F,
           legend_fill = F,
           legend_size = F,
           remove_legend_lab = F,
           theme_args = NA,
           ...
           ){
    ## reformat graph, add with class
    graph <- tidygraph::as_tbl_graph(graph)
    ## nodes ------------------------------------------------------
    nodes <- merge(graph, class, by.x = "name" , by.y = ".id", all.x = TRUE, sort = F)
```

```
  nodes <- dplyr::as_tibble(nodes)
  ## edges -----------------------------------------------------
  edges <- dplyr::as_tibble(tidygraph::activate(graph, edges))
  if(nrow(edges) >= 1){
    ## "dotproduct" or other attributes of compare spectra method.
    edges <- dplyr::rename(edges, similarity = 3)
  }else{
    edges <- dplyr::mutate(edges, similarity = NA)
  }
  ## ----------------------------------------------------------
  ## gather nodes and edges
  graph <- tidygraph::tbl_graph(nodes = nodes, edges = edges)
  ## create network layout
  if(layout == "fr" & nrow(nodes) >= 500)
    layout = "kk"
  layout_n <- create_layout(graph, layout = layout)
  if(is.environment(save_layout_df)){
    assign("layout_n", layout_n, envir = save_layout_df)
  }
  ## plot
  p <- base_vis_c_nebula(layout_n, palette,
                         title = anno[["nebula_index"]],
                         title_fill = title_palette[as.numeric(anno[["hierarchy"]])],
                         remove_nodes = remove_nodes,
                         theme_args = theme_args,
                         ...)
  ## if print into grid panel
  if(!print_into){
    return(p)
  }
  ## remove legend or not
  p <- p + guides(size = ifelse(legend_size, "legend", "none"),
                           fill = ifelse(legend_fill, "legend", "none"))
  ## color bar
  if(class(class$vis_class) == "numeric" & legend_fill){
    p <- p + guides(fill = guide_colorbar(direction = "horizontal", barheight = 0.3))
  }
  ## rm legend labal
  if(remove_legend_lab){
    p <- p + labs(size = "", fill = "")
  }
```

```
    print(p, vp = grid::viewport(layout.pos.row = anno[["row"]],
                                 layout.pos.col = anno[["col"]]))
  }
base_vis_c_nebula <-
  function(
           nebula,
           title = NULL,
           palette = .MCn.palette,
           title_fill = "grey",
           nodes_size_range = c(3, 7),
           nodes_stroke = 0.2,
           edges_width_range = c(0.1, 0.7),
           edges_color = "black",
           title_size = 20,
           remove_nodes = F,
           legend_position = "right",
           scale_fill_expression = "scale_fill_manual(values = palette)",
           theme_args = NA,
           ...
           ){
    if(is.vector(attr(palette, "name")))
      palette <- palette[which(names(palette) %in% nebula$vis_class)]
    ## ------------------------------------
    if(remove_nodes){
      nodes_size_range = 0
    }
    ## ------------------------------------
    theme_args.default <- list(
        text = element_text(family = "Times"),
        axis.ticks = element_blank(),
        axis.text = element_blank(),
        axis.title = element_blank(),
        panel.background = element_rect(fill = "white"),
        panel.grid = element_blank(),
        ## cunstom defined legend position
        legend.position = legend_position,
        plot.title = ggtext::element_textbox(
          ## nebula name textbox
          size = title_size,
          color = "white", fill = title_fill, box.color = "white",
          halign = 0.5, linetype = 1, r = unit(5, "pt"), width = unit(1, "npc"),
```

```
        padding = margin(2, 0, 1, 0), margin = margin(3, 3, 3, 3)
      ))
  ## -------------------------------------
  if(is.list(theme_args)){
    theme_args.default <- theme_args.default[!names(theme_args.default) %in% names(theme_args)]
    theme_args <- c(theme_args.default, theme_args)
  }else{
    theme_args <- theme_args.default
  }
  ## -------------------------------------
  p <- ggraph(nebula) +
    ## compound MS2 similarity mapping as edge width
    geom_edge_fan(aes(edge_width = similarity), color = edges_color, show.legend = F) +
    ## nodes size mapping as compound idenfication similarity
    geom_node_point(aes(size = ifelse(!is.na(tanimotoSimilarity),
                                      ## if the tanimotoSimilarity is NA, set to 0.2
                                      tanimotoSimilarity, 0.2),
                        ## nodes fill. mapping as classification or custom mark
                        fill = vis_class),
                    shape = 21,
                    stroke = nodes_stroke) +
    ## for custum commound expression
    eval(parse(text = scale_fill_expression)) +
    ## set range for edge width
    scale_edge_width(range = edges_width_range) +
    ## for this setting, if range set to 0, remove the nodes
    scale_size(range = nodes_size_range) +
    ## if the nodes is removed, the override.aes setting will retain nodes shape and color in legend
    guides(fill = guide_legend(override.aes = list(size = 5))) +
    ## the title is the annotation of classification
    ggtitle(stringr::str_wrap(title, width = 30)) +
    labs(size = "Tanimoto\nsimilarity", fill = "Compound mark") +
    theme_grey() +
    do.call(theme, theme_args)
  return(p)
}
```

24 File: initialize_mcnebula.R

```
#' @title FUNCTION_TITLE
#' @description FUNCTION_DESCRIPTION
```

```r
#' @param sirius_path PARAM_DESCRIPTION
#' @param output_path PARAM_DESCRIPTION, Default: sirius_path
#' @param output_file PARAM_DESCRIPTION, Default: 'mcnebula_results'
#' @param palette PARAM_DESCRIPTION, Default: unique(c((ggsci::pal_simpsons())(16), (ggsci::pal_igv("de
#' @param palette_stat PARAM_DESCRIPTION, Default: palette
#' @param palette_label PARAM_DESCRIPTION, Default: colorRampPalette(c("#C6DBEFFF", "#3182BDFF", "red")
#' @param rm_var PARAM_DESCRIPTION, Default: F
#' @return OUTPUT_DESCRIPTION
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[ggsci]{pal_simpsons}}, \code{\link[ggsci]{pal_igv}}
#' @rdname initialize_mcnebula
#' @export
#' @importFrom ggsci pal_simpsons pal_igv

initialize_mcnebula <-
  function(
           sirius_path,
           output_path = sirius_path,
           output_file = "mcnebula_results",
           palette = unique(c(ggsci::pal_simpsons()(16),
                              ggsci::pal_igv("default")(51),
                              ggsci::pal_ucscgb()(26),
                              ggsci::pal_d3("category20")(20)
                              )),
           palette_stat = palette,
           palette_ppcp = palette,
           palette_label = colorRampPalette(c("#C6DBEFFF", "#3182BDFF", "red"))(10),
           rm_mc.set = F
           ){
    if(rm_mc.set){
      rm_mc.set(envir = .GlobalEnv)
    }
    if(!file.exists(sirius_path) | !file.exists(output_path)){
      cat("File path not find.\n")
```

```
      return()
    }
    if(!file.exists(paste0(sirius_path, "/", ".format"))){
      cat("SIRIUS project not find.\n")
      return()
    }
    .MCn.sirius <<- sirius_path
    .MCn.output <<- output_path
    .MCn.results <<- output_file
    .MCn.palette <<- palette
    .MCn.palette_stat <<- palette_stat
    .MCn.palette_ppcp <<- palette_ppcp
    .MCn.palette_label <<- palette_label
    dir.create(paste0(.MCn.output, "/", .MCn.results))
    cat("MCnebula project has initialized at ->", .MCn.output, "\n")
  }
## ------------------------------------------------------------------------
rm_mc.set <-
  function(
           envir,
           pattern = "^\\.MCn\\..*"
           ){
    mc.set <- ls(pattern = pattern, envir = envir, all.names = T)
    rm(list = mc.set, envir = envir)
  }
```

25 File: method_formula_based_spec_compare.R

```
#' @title FUNCTION_TITLE
#' @description FUNCTION_DESCRIPTION
#' @param path PARAM_DESCRIPTION, Default: .MCn.sirius
#' @param dirs PARAM_DESCRIPTION, Default: 'all'
#' @param cpu_cores PARAM_DESCRIPTION, Default: 8
#' @param compare_fun PARAM_DESCRIPTION, Default: 'dotproduct'
#' @param precursor_mass_diff PARAM_DESCRIPTION, Default: T
#' @param edge_filter PARAM_DESCRIPTION, Default: 0.3
#' @param only_identical_class PARAM_DESCRIPTION, Default: T
#' @param min_hierarchy PARAM_DESCRIPTION, Default: 5
#' @param filter_only_max PARAM_DESCRIPTION, Default: 2000
#' @param min_zodiac PARAM_DESCRIPTION, Default: 0.9
#' @param min_tanimoto PARAM_DESCRIPTION, Default: 0.4
```

```r
#' @param ... PARAM_DESCRIPTION
#' @return OUTPUT_DESCRIPTION
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[pbapply]{pbapply}}
#'  \code{\link[dplyr]{rename}}, \code{\link[dplyr]{mutate}}, \code{\link[dplyr]{filter}}, \code{\link[
#' @rdname method_formula_based_spec_compare
#' @export
#' @importFrom pbapply pbsapply pbmapply pbapply pblapply
#' @importFrom dplyr rename mutate filter group_by summarise_at ungroup select distinct as_tibble

method_formula_based_spec_compare <-
  function(
           path = .MCn.sirius,
           dirs = "all",
           cpu_cores = 8,
           compare_fun = "dotproduct",
           precursor_mass_diff = T,
           edge_filter = 0.3,
           ## only identical classes will be compared (according to results of function:collate_ppcp
           only_identical_class = T,
           # only hierarchy >= 5 (class, subclass...) will be considered)
           min_hierarchy = 5,
           # only nodes number >= 2000, do zoidacScore and tanimotoSimilarity filter
           filter_only_max = 2000,
           # only ZodiacScore >= 0.5 ...
           min_zodiac = 0.9,
           # only the top structure tanimotoSimilarity >= 0.4 ...
           min_tanimoto = 0.4,
           ## ------------------------------------
           formula_set = .MCn.formula_set,
           structure_set = .MCn.structure_set,
           target_ids = NULL,
           get_meta_dir = F,
           ...
```

```
      ){
## ------------------------------------------------------------------------------------------
## check dirs ---- spectra
if(length(dirs) == 1 & dirs[1] == "all"){
  dirs <- list.files(path = path, pattern="^[0-9](.*)_(.*)_(.*)$", full.names = F)
}
cat("## Method part: check_dir\n")
check <- pbapply::pbsapply(dirs, check_dir, file = "spectra") %>% unname
## ------------------------------------------------------------------------------------------
## lock on file location
meta_dir <- dirs[which(check)] %>%
  data.frame() %>%
  dplyr::rename(dir = ".") %>%
  dplyr::mutate(.id = grep_id(dir)) %>%
  merge(formula_set, by = ".id", all.x = T) %>%
  merge(structure_set[, c(".id", "tanimotoSimilarity")], by = ".id", all.x = T)
if(is.vector(target_ids)){
  meta_dir <- dplyr::filter(meta_dir, .id %in% target_ids)
}
## some .id were Avoid time-consuming calculation
if(nrow(meta_dir) >= filter_only_max){
  meta_dir <- dplyr::filter(meta_dir, ZodiacScore >= min_zodiac, tanimotoSimilarity >= min_tanimoto)
}
meta_dir <- dplyr::mutate(meta_dir, adduct_trans = gsub(" ", "", adduct),
                          target = paste0(precursorFormula, "_", adduct_trans, ".tsv"),
                          full.name = paste0(path, "/", dir, "/", "spectra", "/", target),
                          ## these files need to be check and filter (whether exist)
                          spectra = file.exists(full.name))
meta_dir_filter <- dplyr::filter(meta_dir, spectra)
cat("## STAT of spectra dataset:",
    paste0(nrow(meta_dir_filter), "(formula with match spectra)", "/", nrow(meta_dir), "(all formula
    "\n")
if(get_meta_dir)
  return(meta_dir_filter)
## ------------------------------------------------------------------------------------------
## load all spectra dataset
spectra_cache <- new.env()
pbapply::pbmapply(read_as_spectrum2, # function
                  meta_dir_filter$full.name,
                  meta_dir_filter$".id",
                  MoreArgs = list(
```

```
                                    cache = spectra_cache
                                    ))
  ## ------------------------------------------------------------------------------------------
  ## enumeration combination
  if(only_identical_class){
    ## enumeration combination in each hierarchy
    combn <- dplyr::filter(.MCn.nebula_index,
                           hierarchy >= min_hierarchy,
                           .id %in% meta_dir_filter$".id") %>%
      dplyr::group_by(hierarchy) %>%
      ## dispose in each group
      dplyr::summarise_at(c(".id"), unique) %>%
      dplyr::summarise_at(c(".id"), sort) %>%
      ## enumerate possible
      dplyr::summarise_at(c(".id"), combn_edge) %>%
      dplyr::ungroup() %>%
      dplyr::select(.id) %>%
      dplyr::distinct()
    ## this column has two sub-colunm
    combn <- dplyr::as_tibble(combn$".id")
  ## ------------------------------------------------------------------------------------------
  ## this cost too much time !!!
  }else{
    combn <- combn_edge(meta_dir_filter$".id")
  }
  ## ------------------------------------------------------------------------------------------
  ## compareSpectra (ms2) (via MSnbase)
  cat("## Method part: compare_spectra: sum:", nrow(combn), "\n")
  combn[[compare_fun]] <- pbapply::pbapply(combn, 1, couple_ms2_compare,
                                           fun = compare_fun,
                                           cl = cpu_cores,
                                           cache = spectra_cache,
                                           ...)
  ## filter via spectra similarity (edge_filter)
  ## ------------------------------------------------------------------------------------------
  combn <- combn[which(combn[[compare_fun]] >= edge_filter), ]
  if(precursor_mass_diff == T){
    ## dir
    info_path = paste0(path, "/", meta_dir_filter$dir, "/", "compound.info")
    ## load file
    cat("## Method part: load compound info file\n")
```

```
      meta_dir_filter$compound_mass <- pbapply::pblapply(info_path, get_precursor_mass) %>%
        unlist()
      ## compute mass difference
      cat("## Method part: diff_precursor_mass: sum:", nrow(combn), "\n")
      combn[["mass_diff"]] <- pbapply::pbapply(combn[,1:2], 1, precursor_mass_diff,
                                               df = data.table::data.table(meta_dir_filter))
    }
    combn <- dplyr::as_tibble(combn)
    return(combn)
    ## ----------------------------------------------------------------------------------------
  }
read_as_spectrum2 <-
  function(filename,
           key_id,
           cache = spectra_cache){
    file <- read_tsv(filename)
    file <- new("Spectrum2", mz = file$mz, intensity = file$rel.intensity)
    assign(paste0(key_id), file, envir = cache)
    return()
  }
combn_edge <-
  function(x){
    combn <- combn(x, 2)
      combn <- t(combn)
      combn <- data.frame(combn)
      colnames(combn) <- c(".id_1", ".id_2")
      return(combn)
  }
couple_ms2_compare <-
  function(x,
           fun = "dotproduct",
           cache = spectra_cache,
           ...){
    simi <- MSnbase::compareSpectra(get(x[1], envir = cache),
                                    get(x[2], envir = cache),
                                    fun = fun,
                                    ...)
    return(simi)
  }
get_precursor_mass <-
  function(
```

```
          path
          ){
    df <- read_tsv(path)
    mass <- df[index == "ionMass", 2]
    mass <- as.numeric(mass)
    return(mass)
  }
precursor_mass_diff <-
  function(
          x,
          df
          ){
    ##
    x1 = df[.id == x[1], ]$"compound_mass"
    x2 = df[.id == x[2], ]$"compound_mass"
    x = x2 - x1
    return(x)
  }
```

26 File: method_pick_formula_excellent.R

```
#' @title FUNCTION_TITLE
#' @description FUNCTION_DESCRIPTION
#' @param dir PARAM_DESCRIPTION, Default: NULL
#' @param key_id PARAM_DESCRIPTION, Default: NULL
#' @param exclude_element PARAM_DESCRIPTION, Default: NULL
#' @param ppm_error PARAM_DESCRIPTION, Default: 20
#' @param fc PARAM_DESCRIPTION, Default: 1.5
#' @return OUTPUT_DESCRIPTION
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[dplyr]{mutate}}
#' @rdname method_pick_formula_excellent
#' @export
#' @importFrom dplyr mutate
```

```r
method_pick_formula_excellent <-
  function(
           key_id = NULL,
           dir = NULL,
           exclude_element = NULL,
           ppm_error = 20,
           fc = NA
           ){
    ## ---------------------------------------------------------------------
    if(length(dir) >= 1){
      meta <- data.table::data.table(dir = dir)
      meta <- dplyr::mutate(meta, key_id = grep_id(dir))
    }else{
      meta <- data.table::data.table(key_id = key_id, dir = get_dir(key_id))
    }
    ## ---------------------------------------------------------------------
    meta <- dplyr::mutate(meta,
                          formula_dir = paste0(.MCn.sirius, "/", dir, "/",
                                               "formula_candidates.tsv"),
                          structure_dir = paste0(.MCn.sirius, "/", dir, "/",
                                                 "structure_candidates.tsv"),
                          fingerid = paste0(.MCn.sirius, "/", dir, "/", "fingerid"))
    ## -------------------------------------
    cat("## Method part: batch_get_formula\n")
    formula_list <- pbapply::pblapply(meta$key_id, mutate_get_formula,
                                      ppm_error = ppm_error, exclude_element = exclude_element)
    formula_df <- data.table::rbindlist(formula_list)
    ## -------------------------------------
    ## get top ZodiacScore within a key_id formula set
    df_fz <- lapply(formula_list, head, n = 1)
    df_fz <- data.table::rbindlist(df_fz)
    ## -------------------------------------
    cat("## Method part: batch_get_structure\n")
    structure_list <- pbapply::pbmapply(mutate_get_structure, meta$structure_dir, meta$key_id,
                                        SIMPLIFY = F)
    structure_df <- data.table::rbindlist(structure_list)
    ## -------------------------------------
    ## merge structure_df with formula_df, to get ZodiacScore of top structure
    structure_df <- merge(structure_df, formula_df, by = c(".id", "molecularFormula", "adduct"))
    ## ---------------------------------------------------------------------
    ## ---------------------------------------------------------------------
```

```
    if(is.na(fc) == F){
      df_sz <- dplyr::rename(structure_df, sz_score = ZodiacScore)
      ## ------------------------------------
      ## compare fz_score with sz_score
      compare <- merge(df_fz[, c(".id", "ZodiacScore"), with = F],
                       df_sz[, c(".id", "sz_score"), with = F],
                       all.x = T, by = ".id")
      compare <- dplyr::mutate(compare,
                               use_zodiac = ifelse(is.na(sz_score), T,
                                                   ifelse(ZodiacScore >= sz_score * fc, T, F)
                                                   ))
      ## ------------------------------------
      ## the .id which formula of use_zodiac or not
      fz <- dplyr::filter(compare, use_zodiac == T)$".id"
      sz <- dplyr::filter(compare, use_zodiac == F)$".id"
    }else{
      sz <- structure_df$".id"
      fz <- df_fz$".id"[which(!df_fz$".id" %in% sz)]
    }
    ## ---------------------------------------------------------------------
    ## ---------------------------------------------------------------------
    df_fz <- mutate(df_fz[.id %in% fz, ], use_zodiac = T)
    df_sz <- mutate(structure_df[.id %in% sz, ], use_zodiac = F)
    ## ------------------
    formula_adduct <- dplyr::bind_rows(df_fz, df_sz)
    formula_adduct <- dplyr::as_tibble(formula_adduct) %>%
      dplyr::select(.id, colnames(.))
    ## ---------------------------------------------------------------------
    return(formula_adduct)
  }
mutate_get_formula <-
  function(
           key_id,
           ppm_error,
           exclude_element
           ){
    formula_df <- try(silent = T, get_formula(key_id, rank = "all", ppm_error = ppm_error,
                                              exclude_element = exclude_element))
    if(class(formula_df)[1] == "try-error"){
      return()
    }else{
```

```
      formula_df <- dplyr::mutate(formula_df,
                                  ZodiacScore = ifelse(grepl("[0-9]", ZodiacScore),
                                                       ZodiacScore, 0),
                                  ZodiacScore = as.numeric(ZodiacScore))
      formula_df$".id" <- key_id
      return(formula_df)
    }
  }
mutate_get_structure <-
  function(
           structure_dir,
           key_id
           ){
    structure_df <- try(silent = T, read_tsv(structure_dir))
    if(class(structure_df)[1] == "try-error"){
      return()
    }
    if(nrow(structure_df) == 0){
      return()
    }
    max <- max(structure_df$"CSI:FingerIDScore")
    structure_df <- structure_df[`CSI:FingerIDScore` == max, c("molecularFormula", "adduct"), with = F]
    if(nrow(structure_df) > 1){
      structure_df <- head(structure_df, n = 1)
    }
    structure_df$".id" <- key_id
    return(structure_df)
  }
get_dir <- function(
                    key_id,
                    path = .MCn.sirius
                    ){
  dir <- list.files(path = path,
                    pattern=paste0("^[0-9](.*)_(.*)_", key_id, "$"),
                    full.names = F)
  check <- check_dir(dir)
  if(check == T){
    return(dir)
  }
}
```

27 File: method_summarize_nebula_class.R

```r
#' @title FUNCTION_TITLE
#' @description FUNCTION_DESCRIPTION
#' @param data PPCP dataset
#' @param ppcp_threshold PARAM_DESCRIPTION, Default: 0.5
#' @param max_classes PARAM_DESCRIPTION, Default: 5
#' @param hierarchy_priority PARAM_DESCRIPTION, Default: c(6, 5, 4, 3)
#' @param class_data_type PARAM_DESCRIPTION, Default: 'classes_tree_list'
#' @param ... PARAM_DESCRIPTION
#' @return OUTPUT_DESCRIPTION
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[data.table]{rbindlist}}
#'  \code{\link[dplyr]{rename}}, \code{\link[dplyr]{filter}}
#' @rdname method_summarize_nebula_class
#' @export
#' @importFrom data.table rbindlist
#' @importFrom dplyr rename filter
method_summarize_nebula_class <-
  function(
           data,
           ppcp_threshold = 0.5,
           max_classes = NA,
           hierarchy_priority = c(6, 5, 4, 3), ## level 5, subclass, class, superclass
           class_data_type = "classes_tree_list", ## or "classes_tree_data"
           ...
           ){
    ## input data
    if(class_data_type == "classes_tree_list"){
      class_data = .MCn.class_tree_list
      metadata <- data.table::rbindlist(class_data, idcol = T)
      metadata <- dplyr::rename(metadata, hierarchy = .id)
    }else if(class_data_type == "classes_tree_data"){
      metadata <- class_data <- get("metadata", envir = get("envir_meta"))
    }
```

```
    ## main body
    df <- dplyr::filter(data, V1 >= ppcp_threshold)
    df <- merge(df, metadata[, 1:5], all.x = T, by = "relativeIndex", sort = F)
    df <- dplyr::filter(df, hierarchy %in% hierarchy_priority)
    df <- df[order(factor(df$hierarchy, levels = hierarchy_priority), -df$V1), ]
    if(is.na(max_classes) == F)
      df <- head(df, n = max_classes)
    return(df)
  }
```

28 File: method_summarize_nebula_index.R

```
#' @title FUNCTION_TITLE
#' @description FUNCTION_DESCRIPTION
#' @param ppcp_dataset PARAM_DESCRIPTION
#' @param nebula_class PARAM_DESCRIPTION, Default: .MCn.nebula_class
#' @param ppcp_threshold PARAM_DESCRIPTION, Default: 0.5
#' @param min_possess PARAM_DESCRIPTION, Default: 10
#' @param max_possess_pct PARAM_DESCRIPTION, Default: 0.2
#' @param identical_factor PARAM_DESCRIPTION, Default: 0.8
#' @param filter_identical PARAM_DESCRIPTION, Default: c(top_hierarchy = 4)
#' @param ... PARAM_DESCRIPTION
#' @return OUTPUT_DESCRIPTION
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[data.table]{rbindlist}}
#'  \code{\link[dplyr]{distinct}}, \code{\link[dplyr]{filter}}, \code{\link[dplyr]{rename}}, \code{\lin
#'  \code{\link[pbapply]{pbapply}}
#' @rdname method_summarize_nebula_index
#' @export
#' @importFrom data.table rbindlist
#' @importFrom dplyr distinct filter rename select group_by
#' @importFrom pbapply pblapply
method_summarize_nebula_index <-
  function(
```

```
        ppcp_dataset = .MCn.ppcp_dataset,
        nebula_class = .MCn.nebula_class,
        ppcp_threshold = 0.5,
        # min number of compounds allowed exist in a child-nebula. if less than, filter the nebula
        min_possess = 10,
        ## max percentage of compounds allowed exist in a child-nebula
        max_possess_pct = 0.2,
        ## identical filter
        filter_identical = c("top_hierarchy" = 4),
        identical_factor = 0.7,
        rm_position_describe_class = T,
        ## in the nebula, if too many structure score is too low, filter the nebula.
        ## or NA
        filter_via_struc_score = "tanimotoSimilarity",
        struc_score_cutoff = 0.3,
        min_reached_pct = 0.6,
        target_classes = NA,
        ...
        ){
  if(is.list(nebula_class)){
    classes <- data.table::rbindlist(nebula_class)
    if(rm_position_describe_class){
      classes <- dplyr::filter(classes, !grepl("[0-9]", name))
    }
    ## classes is merely a classes index number set
    classes <- dplyr::distinct(classes, relativeIndex)$relativeIndex
  }else{
    classes <- c()
  }
  ## user defined classes
  if(is.vector(target_classes)){
    target_classes <- dplyr::filter(.MCn.class_tree_data, name %in% target_classes)
    ## merge
    classes <- c(classes, target_classes$relativeIndex)
  }
  ## get classes
  cat("## Method part: class_retrieve\n")
  index_list <- pbapply::pblapply(ppcp_dataset, class_retrieve,
                                  the_relativeIndex = classes,
                                  ...)
  index_df <- data.table::rbindlist(index_list, idcol = T)
```

```
  ## ------------------------------------------------------------------------
  ## filter via max_possess and min_possess
  stat <- table(index_df$relativeIndex)
  stat <- stat[which(stat >= min_possess & stat <= max_possess_pct * length(unique(index_df$".id")))]
  index_df <- index_df %>%
    dplyr::filter(relativeIndex %in% names(stat))
  ## ------------------------------------
  ## gather with classes annotation
  index_df <- data.table::rbindlist(.MCn.class_tree_list, idcol = T) %>%
    dplyr::rename(hierarchy = .id) %>%
    dplyr::select(relativeIndex, name, hierarchy) %>%
    merge(index_df, by = "relativeIndex", all.y = T, sort = F) %>%
    data.table::data.table()
  ## ------------------------------------------------------------------------
  if(!is.na(identical_factor)){
    ## filter identical or similar classes
    ## enumerate combination
    class_for_merge <- index_df %>%
      dplyr::filter(hierarchy >= filter_identical[["top_hierarchy"]]) %>%
      dplyr::distinct(relativeIndex) %>%
      unlist() %>% combn(m = 2) %>%
      t() %>% data.frame()
    ## ------------------------------------
    cat("## Method part: identical_filter\n")
    discard = pbapply::pbapply(class_for_merge, 1, identical_filter,
                               index_df = index_df,
                               identical_factor = identical_factor,
                               ...) %>%
      unlist() %>% unique()
    index_df <- dplyr::filter(index_df, !relativeIndex %in% discard)
  }
  ## ------------------------------------------------------------------------
  if(!is.na(filter_via_struc_score)){
    df <- merge(index_df, .MCn.structure_set[, c(".id", filter_via_struc_score)],
                by = ".id", all.x = T)
    list <- by_group_as_list(df, "relativeIndex")
    ## ------------------------------------
    cat("## Method part: fun_filter_via_struc_score\n")
    select_index <- pbapply::pblapply(list, fun_filter_via_struc_score,
                                      filter_via_struc_score,
                                      struc_score_cutoff,
```

```
                                        min_reached_pct)
      select_index <- unlist(select_index)
      index_df <- dplyr::filter(index_df, relativeIndex %in% all_of(select_index))
    }
    ## ------------------------------------------------------------------------
    ## cluster id in each classes
    nebula_index <- dplyr::group_by(index_df, relativeIndex)
    return(nebula_index)
  }
class_retrieve <-
  function(
           data,
           the_relativeIndex,
           ppcp_threshold = 0.5,
           ...
           ){
    ##
    classes <- the_relativeIndex
    data <- dplyr::filter(data, relativeIndex %in% classes, V1 >= ppcp_threshold)
    return(data)
  }
identical_filter <-
  function(
           couple,
           index_df,
           identical_factor = 0.7,
           ...
           ){
    ## index_df is a data.table project
    x = unique(index_df[relativeIndex %in% couple[1], ]$".id")
    y = unique(index_df[relativeIndex %in% couple[2], ]$".id")
    p_x = table(x %in% y)
    p_y = table(y %in% x)
    ## ------------------------------------
    if("TRUE" %in% names(p_x) == F | "TRUE" %in% names(p_y) == F){
      return()
    }
    p_x = prop.table(p_x)[["TRUE"]]
    p_y = prop.table(p_y)[["TRUE"]]
    ## ------------------------------------
    if(p_x >= identical_factor & p_y >= identical_factor){
```

```
      idn = ifelse(length(x) >= length(y), couple[2], couple[1])
      return(idn)
    }else{
      return()
    }
  }
fun_filter_via_struc_score <-
  function(
           df,
           score = "tanimotoSimilarity",
           cutoff = 0.4,
           min_reached_pct = 0.5
           ){
    x <- df[[score]]
    df <- dplyr::mutate(df, reach = ifelse(x >= cutoff &
                                           is.na(x) == F,
                                         T, F))
    check <- prop.table(table(df[["reach"]]))
    if("TRUE" %in% names(check) == F)
      return()
    if(check[["TRUE"]] >= min_reached_pct){
      return(df[1, ]$relativeIndex)
    }else{
      return()
    }
  }
## ------------------------------------------------------------------------
method_summarize_target_index <-
  function(
           target_classes
           ){
    target_index <- method_summarize_nebula_index(nebula_class = NA,
                                                  target_classes = target_classes,
                                                  identical_factor = NA,
                                                  filter_via_struc_score = NA,
                                                  max_possess_pct = 1,
                                                  min_possess = 1
    )
    return(target_index)
  }
```

29 File: mutate_get_parent_class.R

```
mutate_get_parent_class <-
  function(
           classes,
           class_cutoff = 4,
           meta = .MCn.class_tree_data,
           this_class = F
           ){
    ## ------------------------------------
    cat("## get_parent_class\n")
    db <- dplyr::filter(meta, hierarchy >= class_cutoff)
    ## ------------------------------------
    ## for name to get id
    db_id <- lapply(db$id, c)
    names(db_id) <- db$name
    ## for id to get parentId
    db_parent <- lapply(db$parentId, c)
    names(db_parent) <- db$id
    ## for id to get name
    db_name <- lapply(db$name, c)
    names(db_name) <- db$id
    ## ------------------------------------
    set_list <- pbapply::pblapply(classes, get_parent_class,
                                  db_id = db_id,
                                  db_parent = db_parent,
                                  db_name = db_name,
                                  this_class = this_class)
    names(set_list) <- classes
    return(set_list)
  }
get_parent_class <-
  function(
           class,
           db_id,
           db_parent,
           db_name,
           this_class = F
           ){
    set <- c()
    parent <- 0
    id <- db_id[[class]]
```

```
    test <- try(db_parent[[id]], silent = T)
    if (inherits(test, "try-error"))
      return()
    while(is.null(parent) == F){
      if(parent != 0){
        set <- c(set, db_name[[parent]])
        id <- parent
      }
      parent <- db_parent[[id]]
    }
    if(length(set) == 0){
      if(this_class == T)
        return(class)
    }
    return(set)
  }
```

30 File: nebula_re_rank.R

```
#' @title FUNCTION_TITLE
#' @description FUNCTION_DESCRIPTION
#' @param nebula_name PARAM_DESCRIPTION
#' @param top_n PARAM_DESCRIPTION, Default: 10
#' @param match_pattern PARAM_DESCRIPTION, Default: c("precursorFormula")
#' @param collate_factor PARAM_DESCRIPTION, Default: 0.85
#' @param revise_MCn_formula_set PARAM_DESCRIPTION, Default: T
#' @param revise_MCn_structure_set PARAM_DESCRIPTION, Default: T
#' @param ... PARAM_DESCRIPTION
#' @return OUTPUT_DESCRIPTION
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[dplyr]{filter}}, \code{\link[dplyr]{reexports}}, \code{\link[dplyr]{arrange}}, \code{\l
#'  \code{\link[pbapply]{pbapply}}
#'  \code{\link[data.table]{rbindlist}}
#'  \code{\link[tidyr]{separate}}
```

```r
#' @rdname nebula_re_rank
#' @export
#' @importFrom dplyr filter as_tibble arrange mutate select distinct
#' @importFrom pbapply pblapply
#' @importFrom data.table rbindlist
#' @importFrom tidyr separate
nebula_re_rank <-
  function(
           nebula_name,
           top_n = 50,
           match_pattern = NULL, # c("precursorFormula"), or c("precursorFormula", "adduct")
           collate_factor = NA,
           only_gather_structure = F,
           ## -------------------------------------
           reference_compound = NA,
           reference_ratio = 0.5,
           ## -------------------------------------
           cluster_method = NA,
           csi_score_weight = 0.6,
           class_similarity_weight = 0.3,
           ## -------------------------------------
           filter_via_classification = F,
           ## -------------------------------------
           rt_set = NA,
           rt_weight = 0.1,
           rt_window = 1.5,
           ## -------------------------------------
           revise_MCn_formula_set = F,
           revise_MCn_structure_set = F,
           ...
           ){
    cat("[INFO] MCnebula run: nebula_re_rank\n")
    ## ---------------------------------------------------------------------
    structure_set <- get_nebula_structure_candidates(nebula_name, top_n, match_pattern, collate_factor,
                                                     ...)
    if(only_gather_structure == T){
      return(dplyr::as_tibble(structure_set))
    }
    ## ---------------------------------------------------------------------
    if(is.data.frame(reference_compound) == F){
      reference_compound <- set_reference_compound(structure_set, reference_ratio)
```

```
    }else{
      reference_compound <- reference_compound
    }
    ## ------------------------------------------------------------------------
    ## cluster method
    if(is.na(cluster_method) == F){
      method_fun <- match.fun(cluster_method)
      cat("## netbula_re_rank:", paste0(cluster_method), "\n")
      structure_set <- method_fun(structure_set, reference_compound, csi_score_weight = csi_score_weigh
                                  class_similarity_weight = class_similarity_weight,
                                  ...)
    }
    ## ------------------------------------------------------------------------
    ## retrive class of candidates via classyfire
    if(filter_via_classification == T){
      cat("## netbula_re_rank: method_filter_candidates_upon_classyfire\n")
      structure_set <- method_filter_candidates_upon_classyfire(structure_set, nebula_name, ...)
    }
    ## ------------------------------------------------------------------------
    ## rt prediction
    if(is.data.frame(rt_set)){
      cat("## netbula_re_rank: method_predict_candidates_rt\n")
      structure_set <- method_predict_candidates_rt(structure_set, reference_compound, rt_set,
                                                    rt_weight = rt_weight, rt_window = rt_window, ...)
    }
    ## ------------------------------------------------------------------------
    ## revise .GlobalVar .MCn.formula_set
    if(revise_MCn_formula_set == T){
      revise_MCn_formula_set(structure_set)
    }
    ## ------------------------------------------------------------------------
    ## revise .GlobalVar .MCn.structure_set -------
    if(revise_MCn_structure_set == T){
      revise_MCn_structure_set(structure_set)
    }
    ## ------------------------------------------------------------------------
    cat("[INFO] MCnebula Job Done: nebula_re_rank\n")
    return(dplyr::as_tibble(structure_set))
  }
## ----------------------------------------------------------------------
smiles_to_sdfset <-
```

```
  function(
          structure_set
          ){
    ##
    smiles_set <- structure_set$smiles
    names(smiles_set) <- paste0(structure_set$".id", "_", structure_set$structure_rank)
    ## this function automaticly set the vector name as name of each subset
    sdf_set <- ChemmineR::smiles2sdf(smiles_set)
    return(sdf_set)
  }
df_get_structure <-
  function(
          x,
          top_n = 10,
          collate_factor = 0.85,
          ...
          ){
    df <- get_structure(
                        x[[".id"]],
                        x[["precursorFormula"]],
                        x[["adduct"]],
                        return_row = 1:top_n,
                        ...)
    if(nrow(df) == 0){
      return(df)
    }
    df <- dplyr::mutate(df, .id = x[[".id"]]) ## add key_id
    if(is.na(collate_factor) == F){
      top_simi <- df[1, "tanimotoSimilarity"]
      df <- dplyr::filter(df, tanimotoSimilarity >= top_simi * collate_factor)
    }
    return(df)
  }
rename_file <-
  function(
          file,
          suffix = "prefix"
          ){
    if(file.exists(file) == T){
      file.rename(file, paste0(file, ".", suffix))
    }
```

```
  }
revise_MCn_formula_set <-
  function(
           structure_set
           ){
    ## prepare replace data
    rp <- dplyr::arrange(structure_set, .id) %>%
      tidyr::separate(col = "file_name", sep = "_", into = c("precursorFormula", "adduct")) %>%
      dplyr::mutate(adduct = gsub("\\+(?!$)", " \\+ ", adduct, perl = T),
                    adduct = gsub("\\-(?!$)", " \\- ", adduct, perl = T)) %>%
      dplyr::select(.id, precursorFormula, adduct, molecularFormula)
    ## replace
    fset <- dplyr::arrange(.MCn.formula_set, .id)
    fset[fset$".id" %in% rp$".id", c(".id", "precursorFormula", "adduct", "molecularFormula")] <- rp
    .MCn.formula_set <<- fset
    return()
  }
revise_MCn_structure_set <-
  function(
           structure_set
           ){
    sset <- dplyr::arrange(.MCn.structure_set, .id)
    ## prepare replace data
    rp <- dplyr::arrange(structure_set, .id) %>%
      dplyr::select(colnames(sset))
    ## replace
    sset <- dplyr::distinct(rbind(rp, sset), .id, .keep_all = T)
    .MCn.structure_set <<- sset
    ## rename exist structure picture -------
    tmp_stru <- paste0(.MCn.output, "/", .MCn.results, "/tmp/structure")
    if(file.exists(tmp_stru) == T){
      lapply(paste0(tmp_stru, "/", rp$".id", ".svg"), rename_file)
    }
  }
get_nebula_structure_candidates <-
  function(
           nebula_name,
           top_n = 50,
           match_pattern = NULL,
           collate_factor = NA,
           ...
```

```
          ){
    ## get formula
    id_set <- dplyr::filter(.MCn.nebula_index, name == nebula_name)
    formula_adduct <- dplyr::filter(.MCn.formula_set, .id %in% id_set$".id")
    ## ----------------------------------------------------------------------
    ## match patern
    if("precursorFormula" %in% match_pattern == F){
      formula_adduct$precursorFormula = NULL
    }
    if("adduct" %in% match_pattern == F){
      formula_adduct$adduct = NULL
    }
    ## ----------------------------------------------------------------------
    ## catch file
    formula_adduct <- by_group_as_list(formula_adduct, ".id")
    ## then, use lapply match file
    cat("## netbula_re_rank: get_structure\n")
    structure_set <- pbapply::pblapply(formula_adduct, df_get_structure,
                                       top_n = top_n,
                                       collate_factor = collate_factor,
                                       ...)
    structure_set <- data.table::rbindlist(structure_set, fill = T)
    cat("## STAT of structure_set:",
        paste0(nrow(structure_set), " (structure sum)/", length(unique(structure_set$".id")), "(.id sum
    return(structure_set)
  }
set_reference_compound <-
  function(
          structure_set,
          reference_ratio = 0.5
          ){
    reference_compound <- dplyr::filter(structure_set, structure_rank == 1) %>%
      dplyr::select(.id, structure_rank, tanimotoSimilarity) %>%
      dplyr::arrange(tanimotoSimilarity) %>%
      dplyr::slice(1:(round(reference_ratio * nrow(.))))
    return(reference_compound)
  }
```

31 File: read_tsv.R

```
read_tsv <- function(path){
  file <- data.table::fread(input=path, sep="\t", header=T, quote="", check.names=F)
  return(file)
}
write_tsv <-
  function(x, filename, col.names = T, row.names = F){
    write.table(x, file = filename, sep = "\t", col.names = col.names, row.names = row.names, quote = F
  }
```

32 File: visualize_child_nebulae.R

```
#' @title FUNCTION_TITLE
#' @description FUNCTION_DESCRIPTION
#' @param graph_list PARAM_DESCRIPTION, Default: .MCn.child_graph_list
#' @param compound_class_list PARAM_DESCRIPTION, Default: .MCn.nebula_class
#' @param output PARAM_DESCRIPTION, Default: paste0(.MCn.output, "/", .MCn.results)
#' @param layout PARAM_DESCRIPTION, Default: 'fr'
#' @param width PARAM_DESCRIPTION, Default: 23
#' @param height PARAM_DESCRIPTION, Default: 30
#' @param ... PARAM_DESCRIPTION
#' @return OUTPUT_DESCRIPTION
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[data.table]{rbindlist}}
#'  \code{\link[dplyr]{select}}, \code{\link[dplyr]{rename}}, \code{\link[dplyr]{reexports}}, \code{\li
#'  \code{\link[svglite]{svglite}}
#'  \code{\link[grid]{grid.newpage}}, \code{\link[grid]{Working with Viewports}}
#'  \code{\link[pbapply]{pbapply}}
#' @rdname visualize_child_nebulae
#' @export
#' @importFrom data.table rbindlist
#' @importFrom dplyr select rename as_tibble arrange mutate
#' @importFrom svglite svglite
```

```r
#' @importFrom grid grid.newpage pushViewport
#' @importFrom pbapply pbmapply
visualize_child_nebulae <-
  function(
           graph_list = .MCn.child_graph_list,
           compound_class_list = .MCn.nebula_class,
           output = paste0(.MCn.output, "/", .MCn.results),
           layout = "fr",
           width = 23,
           height = 30,
           nodes_mark = NA,
           ...
           ){
    cat("[INFO] MCnebula run: visualize_child_nebulae\n")
    ## get top compound class (nodes_color data)
    metadata <- lapply(compound_class_list, head, n = 1) %>%
      data.table::rbindlist(idcol = T) %>%
      dplyr::select(.id, name) %>%
      dplyr::rename(vis_class = name)
    ## ------------------------------------
    if(is.data.frame(nodes_mark)){
      ## the secound col as mark col
      colnames(nodes_mark) <- c(".id", "mark")
      ## merge with metadata
      metadata <- merge(metadata, nodes_mark, by = ".id", all.x = T) %>%
        dplyr::mutate(vis_class = ifelse(!is.na(mark), mark,
                                         ifelse(is.numeric(mark), NA, "Others")))
    }
    ## ------------------------------------------------------------------------
    ## draw network via ggplot, and print into grid palette
    ## number of child_nebulae
    n = length(graph_list)
    ## specification of grid (cols * rows)
    cols = n^(1/2)
    if(round(cols) != cols){
      cols = round(cols)
      rows = cols + 1
    }else{
      rows = cols
    }
    ## ------------------------------------
```

```
  ## grid position of all child_nebulae
  graph_anno <- names(graph_list) %>% # names
    dplyr::as_tibble() %>%
    dplyr::rename(nebula_index = value) %>%
    merge(.MCn.class_tree_data[,c("name", "hierarchy")], by.x = "nebula_index", by.y = "name", all.x =
    dplyr::arrange(desc(hierarchy)) %>%
    ## calculate position
    dplyr::mutate(seq = 1:n,
                  col = ifelse(seq %% cols != 0, seq %% cols, cols),
                  row = (seq - col)/cols + 1)
  ## ------------------------------------
  ## re-set rows
  rows <- max(graph_anno$row)
  ## as list
  nebula_index <- graph_anno$nebula_index
  graph_anno <- by_group_as_list(graph_anno, "nebula_index")
  ## re-order the graph list according to annotation
  graph_list <- lapply(nebula_index, function(x){
                         graph_list[[x]]
                       })
  ## ------------------------------------
  ## prepare grid panel
  svglite::svglite(paste0(output, "/", "child_nebulae.svg"), width = width, height = height)
  grid::grid.newpage()
  grid::pushViewport(viewport(layout = grid.layout(rows, cols)))
  ## draw child_nebulae in grid
  pbapply::pbmapply(grid_child_nebula, ## function
                    graph_list, ## graph list
                    graph_anno, ## graph annotation
                    MoreArgs = list( ## args
                                    class = metadata,
                                    layout = layout,
                                    ...
                                    ))
  dev.off()
  cat("[INFO] MCnebula Job Done: visualize_child_nebulae\n")
}
```

33 File: visualize_nebula.R

```r
#' @title FUNCTION_TITLE
#' @description FUNCTION_DESCRIPTION
#' @param nebula_name PARAM_DESCRIPTION
#' @param ... PARAM_DESCRIPTION
#' @return OUTPUT_DESCRIPTION
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @rdname visualize_nebula
#' @export
visualize_nebula <-
  function(
           nebula_name,
           ...
           ){
    p <- annotate_child_nebulae(
                                nebula_name = nebula_name,
                                write_output = F,
                                plot_structure = F,
                                plot_ppcp = F,
                                merge_image = F,
                                return_plot = T,
                                ...)
    return(p)
  }
```

34 File: visualize_parent_nebula.R

```r
#' @title FUNCTION_TITLE
#' @description FUNCTION_DESCRIPTION
#' @param graph PARAM_DESCRIPTION, Default: .MCn.parent_graph
#' @param write_output PARAM_DESCRIPTION, Default: T
#' @param output PARAM_DESCRIPTION, Default: paste0(.MCn.output, "/", .MCn.results)
#' @param layout PARAM_DESCRIPTION, Default: 'mds'
#' @param nodes_color PARAM_DESCRIPTION, Default: c(hierarchy = 4)
```

```
#' @param width PARAM_DESCRIPTION, Default: 15
#' @param height PARAM_DESCRIPTION, Default: 12
#' @param return_plot PARAM_DESCRIPTION, Default: F
#' @param ... PARAM_DESCRIPTION
#' @return OUTPUT_DESCRIPTION
#' @details DETAILS
#' @examples
#' \dontrun{
#' if(interactive()){
#'  #EXAMPLE1
#'  }
#' }
#' @seealso
#'  \code{\link[pbapply]{pbapply}}
#'  \code{\link[data.table]{rbindlist}}
#'  \code{\link[dplyr]{select}}, \code{\link[dplyr]{rename}}, \code{\link[dplyr]{mutate}}, \code{\link[
#'  \code{\link[tidygraph]{as_tbl_graph.data.frame}}, \code{\link[tidygraph]{activate}}
#'  \code{\link[ggraph]{ggraph}}
#'  \code{\link[ggplot2]{ggsave}}
#' @rdname visualize_parent_nebula
#' @export
#' @importFrom pbapply pblapply
#' @importFrom data.table rbindlist
#' @importFrom dplyr select rename mutate as_tibble
#' @importFrom tidygraph as_tbl_graph activate tbl_graph
#' @importFrom ggraph create_layout
#' @importFrom ggplot2 ggsave
visualize_parent_nebula <-
  function(
           graph = .MCn.parent_graph,
           write_output = T,
           output = paste0(.MCn.output, "/", .MCn.results),
           layout = "mds",
           nodes_color = c("hierarchy" = 3), ## default, use superclass as color.
           width = 15,
           height = 12,
           return_plot = F,
           ...
           ){
    cat("[INFO] MCnebula run: visualize_parent_nebula\n")
    ## get nodes_color data
```

```
  metadata = .MCn.class_tree_data
  assign("envir_meta", environment(), envir = parent.env(environment()))
  cat("## visualize_parent_nebula: method_summarize_nebula_index\n")
  class <- pbapply::pblapply(.MCn.ppcp_dataset, method_summarize_nebula_class,
                             class_data_type = "classes_tree_data",
                             max_number = 1,
                             hierarchy_priority = nodes_color[["hierarchy"]] )
  class <- data.table::rbindlist(class, idcol = T) %>%
    dplyr::select(.id, name) %>%
    dplyr::rename(vis_class = name)
  ## reformat graph, add with class
  graph <- tidygraph::as_tbl_graph(graph)
  ## -----------------------------------------------------------------------
  nodes <- graph %>%
    tidygraph::activate(nodes) %>%
    merge(class, by.x = "name" , by.y = ".id", all.x = TRUE, sort = F) %>%
    dplyr::mutate(vis_class = ifelse(is.na(vis_class) == T, "Unknown", vis_class)) %>%
    dplyr::as_tibble()
  edges <- graph %>%
    tidygraph::activate(edges) %>%
    ## rename the col of value of compare spectra
    dplyr::rename(similarity = 3) %>%
    dplyr::as_tibble()
  ## -----------------------------------------------------------------------
  graph <- tidygraph::tbl_graph(nodes = nodes, edges = edges)
  ## create network layout
  layout_n <- create_layout(graph, layout = layout, ...)
  ## palette
  palette <- .MCn.palette
  ## draw network via ggraph
  p <- base_vis_p_nebula(layout_n, palette)
  ## write_output
  if(write_output == T){
    ggsave(p, file = paste0(output, "/", "parent_nebula", "/", "parent_nebula.svg"),
           width = width, height = height)
  }
  ## -----------------------------------------------------------------------
  rm("envir_meta", envir = parent.env(environment()))
  ## -----------------------------------------------------------------------
  cat("[INFO] MCnebula Job Done: visualize_parent_nebula\n")
  if(return_plot == T){
```

```
      return(p)
    }
  }
base_vis_p_nebula <-
  function(
           layout_n,
           palette = .MCn.palette,
           ...
           ){
    p <- ggraph(layout_n) +
      geom_edge_fan(aes(edge_width = similarity),
                    color = "lightblue", show.legend = F) +
      geom_node_point(aes(size = ifelse(is.na(tanimotoSimilarity) == F,
                                        tanimotoSimilarity, 0.2),
                          fill = stringr::str_wrap(vis_class, width = 25)
                          ),
                      shape = 21
                      ) +
      scale_fill_manual(values = palette) +
      scale_edge_width(range = c(0.1, 0.7)) +
      guides(fill = guide_legend(override.aes = list(size = 5))) +
      labs(fill = "Class", size = "Tanimoto similarity") +
      ## ------------------------------------
      theme_grey() +
      theme(
            text = element_text(family = "Times"),
            axis.ticks = element_blank(),
            axis.text = element_blank(),
            axis.title = element_blank(),
            panel.background = element_rect(fill = "white"),
            legend.key.width = unit(1, "cm"),
            legend.key.height = unit(1.8, "cm"),
            legend.title = element_text(size = 20, face = "bold"),
            legend.text = element_text(size = 20),
            legend.background = element_rect(fill = "transparent"),
            panel.grid = element_blank(),
            strip.text = element_text(size = 20, face = "bold")
      )
    return(p)
  }
```