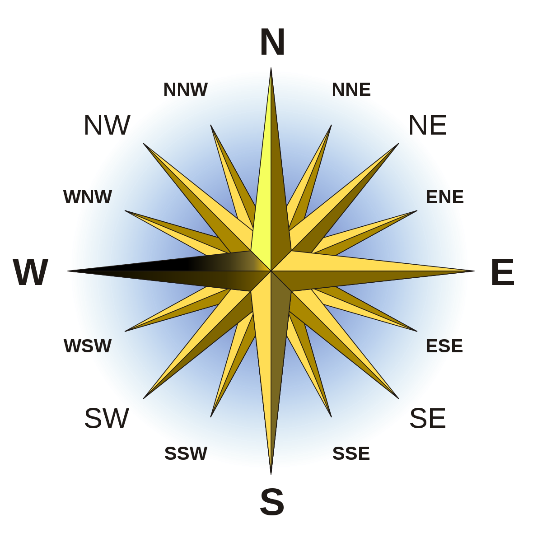
1. Pre-processing
   1. Checks
      1. Check a cell DF + other tests
      2. Validate cell DF
   2. Delete <Empty>
2. Data Cell and Attribute Cell Separation

**Definitive function**

1. get\_group\_id
2. get\_direction\_df



**Ai function**

1. ai\_get\_data\_attr\_map\_main
   1. ai\_get\_data\_attr\_map
      1. get\_direction\_df (this I think can be linked to intra block dist)
   2. ~~if crude\_join~~ (this is problematic) (delinked)
      1. ~~ai\_crude\_data\_block\_joins~~
   3. ai\_relative\_data\_split\_attr (main place to work)
   4. ~~ai\_get\_data\_attr\_map\_details~~ renamed to ai\_get\_dimention\_analysis\_details
2. ai\_data\_gid\_join
   1. is\_attachable
   2. ~~get\_links\_df~~ (this seems unnecessarily complicated) removed as gid\_map\_link\_tune introduced.
   3. get\_group\_id\_join\_gids

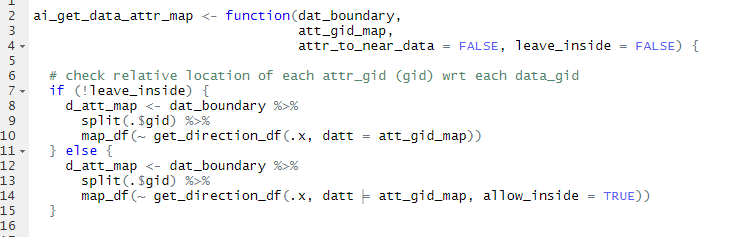
**ai\_get\_data\_attr\_map**

ai\_get\_data\_attr\_map <- function(dat\_boundary,

att\_gid\_map,

attr\_to\_near\_data = FALSE, leave\_inside = FALSE)

Need to fix (can be kept without if block)



d\_att\_map has all maps wrt each data boundary

d\_gid\_att\_map gets md min dist between each attr\_gid and data\_gid and direction group

d\_gid\_att\_map\_min\_d attach each data\_gid to nearest attr\_gid for all directions.

map -> mapping (data\_gid to nearest attr\_gid + attr\_gid to nearest data\_gid if attr\_to\_near\_data)

all\_map -> all other mapping

raw -> attr gid’s cell level information

lo <- list(map = d\_gid\_att\_map\_min\_d, all\_map = d\_gid\_att\_map, raw = d\_att\_map)

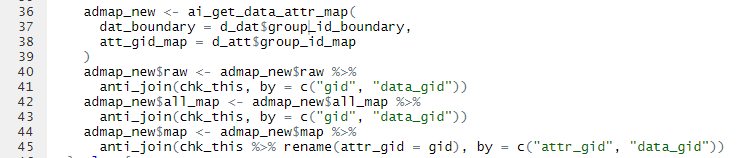
md need to change - done

leave inside ???

ai\_relative\_data\_split\_attr

This may not be required

(Changed)



ai\_get\_data\_attr\_map\_details

major minor classification

name change suggested ai\_get\_data\_attr\_map\_details done

ai\_data\_gid\_join

nearby gids first

ai\_get\_data\_attr\_map\_main <- function(d\_dat, d\_att) {

# start with simple attr data map

admap0 <- ai\_get\_data\_attr\_map(

dat\_boundary = get\_group\_id\_boundary(d\_dat),

att\_gid\_map = d\_att

)

# split attr gid relative to data\_gid

rel\_chk <- ai\_relative\_data\_split\_attr(basic\_map = admap0, d\_att = d\_att)

if (rel\_chk$done) {

d\_att <- rel\_chk$d\_att

admap0 <- rel\_chk$admap

}

# dimension analysis done here (major minor classification)

# phase 1

admap1\_major\_minor <- admap0$all\_map %>%

filter(direction\_group != "corner") %>%

ai\_get\_dimention\_analysis\_details(d\_dat, d\_att)

# @Dev

# why comapting required?

admap1\_major\_compact <- admap1\_major\_minor$map %>%

filter(attr\_group == "major") %>%

group\_by(data\_gid, direction\_group) %>%

filter(dist == min(dist)) %>%

ungroup()

admap1\_major\_compact <- admap1\_major\_compact %>%

# @Dev this filter seems redundant

# filter(direction\_group != "corner") %>%

unique() %>%

select(-attr\_group)

# dimension analysis done here (major minor classification)

# phase 2

admap1 <- admap1\_major\_compact %>%

ai\_get\_dimention\_analysis\_details(d\_dat, d\_att)

##########################

# @Dev why do we need two phase??

# really reaquired ? if so why two why not iterative ?? Check it..

if(!identical(admap1\_major\_minor, admap1)){

cat("you need to see it now!!\n")

browser()

# this is the situation

}

##########################

# @Dev

# d\_dat does not change

list(admap = admap1, d\_dat = d\_dat, d\_att = d\_att)

}

Analyze Cells

#' Analyze Cells

#'

#' @description After [`Value Attribute Classification`][value\_attribute\_classify()] done on a [`cell\_df`][cell\_df-class] next task to do is

#' analyze it's contents for data block detection, attribute orientation identification etc. The function `analyze\_cells` (and also `analyse\_cells`)

#' does the same for you.

#'

#' \*\*Note\*\*:

#' \if{html}{If you are not sure about what package functions actually do or how they work together,

#' please start with \href{../doc/tidycells-intro.html}{\code{vignette("tidycells-intro")}}.}

#' \if{latex}{If you are not sure about what package functions actually do or how they work together,

#' please start with \code{vignette("tidycells-intro")}.}

#'

#' @param d A [`cell\_df`][cell\_df-class] after [`Value Attribute Classification`][value\_attribute\_classify()] done

#' @param silent logical scalar indicating whether to raise a warning if heuristic detection fails. (Default TRUE).

#'

#' @details it returns detailed analysis of the data structure including data block detection, attribute orientation detection etc.

#' The argument `silent` is set to `TRUE` by default, as the warning will be given whenever the [`cell\_analysis`][cell\_analysis-class] is printed.

#'

#' After this step one may like to do :

#' \* [`compose\_cells`][compose\_cells()]

#'

#' If in an interactive session, following additional functions can be helpful for interactive visualizations:

#' \* [`visual\_data\_block\_inspection`][visual\_data\_block\_inspection()]

#' \* [`visual\_orientation\_modification`][visual\_orientation\_modification()]

#' \* [`visual\_traceback`][visual\_traceback()]

#'

#' @return Detailed analysis of the cell data structure.

#' Which will be a [`cell\_analysis`][cell\_analysis-class] class object.

#'

#' @seealso [`compose\_cells`][compose\_cells()], [`collate\_columns`][collate\_columns()]

#' @export

#' @examples

#' d <- structure(c(

#' "block 1", "", "C", "D", "", "block 2", "", "C",

#' "D", "", "A", "1", "2", "", "", "A", "10", "20", "", "B", "3",

#' "4", "", "", "B", "30", "40"

#' ), .Dim = c(9L, 3L))

#' d <- as.data.frame(d)

#' cd <- as\_cell\_df(d) %>% numeric\_values\_classifier()

#'

#' # see it

#' cd %>% plot(adaptive\_txt\_size = FALSE)

#' ca <- analyze\_cells(cd)

#'

#' # look at the plot for detected directions

#' plot(ca)

analyze\_cells <- function(d, silent = TRUE) {

analyze\_cells\_raw(d = d, silent = silent)

}

analyze\_cells\_raw <- function(d, silent = TRUE) {

if (!is\_cell\_df(d)) {

abort("A Cell DF expected")

}

if (!hasName(d, "type")) {

abort(paste("The type column not found.",

"(You may like to do 'Value Attribute Classification'.",

"Check basic\_classifier, sample\_based\_classifier, numeric\_values\_classifier for details.",

sep = "\n"

))

}

val <- validate\_cells(d)

if (!val) {

abort(attr(val, "msg") %>% paste0(collapse = "\n"))

}

# remove empty cells

d\_orig <- d

d <- d %>% filter(type != "empty")

# the term 'data' and 'value' are interchangeably used going forward

data\_cells <- d %>%

filter(type == "value") %>%

as\_rc\_df()

attr\_cells <- d %>%

filter(type == "attribute") %>%

as\_rc\_df()

if (nrow(data\_cells) == 0) {

abort("No `value` cells found")

}

if (nrow(attr\_cells) == 0) {

abort("No `attribute` cells found")

}

d\_dat <- get\_group\_id(data\_cells, gid\_tag = "d")

d\_att <- get\_group\_id(attr\_cells, gid\_tag = "a")

#@Dev

# this function name may be changed

step1 <- ai\_get\_data\_attr\_map\_main(d\_dat, d\_att)

d\_att <- step1$d\_att

admap1 <- step1$admap

# data\_gid join (if possible)

if (length(unique(d\_dat$gid)) > 1) {

d\_dat\_potential\_joins <- ai\_data\_gid\_join(

d\_dat, d\_att,

data\_attr\_map = admap1,

full\_data = d %>% as\_tibble()

)

if (d\_dat\_potential\_joins$done) {

# @Dev need to fix

# this means results has been invalidated

d\_dat <- d\_dat0

# revert back to original form

d\_att <- d\_att\_orig

step2 <- ai\_get\_data\_attr\_map\_main(d\_dat, d\_att, crude\_join = FALSE)

d\_dat <- step2$d\_dat

d\_att <- step2$d\_att

admap1 <- step2$admap

}

}

# join attr based on block merges (potentially possible)

rel\_chk <- ai\_relative\_data\_join\_attr(admap\_main = admap1, d\_att = d\_att)

if (rel\_chk$done) {

d\_att <- rel\_chk$d\_att %>% map(unique)

admap1 <- rel\_chk$admap

}

# now time for corners (potential) / minor / less strong mappings

# information blocks are introduced from here

# @Dev check and delink functions

# d\_dat$group\_id\_extended\_boundary <- extend\_data\_block(d\_dat$group\_id\_boundary, admap1$map, d\_att)

info\_block <- ai\_get\_information\_blocks(admap1, d\_dat, d\_att)

admap1 <- info\_block$map

d\_inf <- info\_block$d\_inf

unmapped\_attr\_gids <- d\_att$gid %>% unique() %>%

setdiff(admap1$attr\_gid) %>%

# @Dev

# following may not be perfect as it discards the <missed\_block\_connections> $ <attr\_gid> without seeeing

# <data\_gid> <- this is stale now (for this we may opt for <row, col> sample in <missed\_block\_connections>)

# for now solving purpose but may be dropped later

#

setdiff(common\_knowledge("missed\_block\_connections")$attr\_gid)

if(length(unmapped\_attr\_gids)>0){

# attach unmapped\_attr\_gids to info\_blocks

# fr: for rest attr\_ids (this actually mean <unmapped\_attr\_gids> and can include corners and non-corners also)

admap\_fr0 <- ai\_get\_data\_attr\_map(

dat\_boundary = get\_group\_id\_boundary(d\_inf),

att\_gid\_map = d\_att %>% filter(gid %in% unmapped\_attr\_gids),

attr\_to\_near\_data = TRUE,

leave\_inside = TRUE

)

unmapped\_attr\_gids <-

admap1$map$attr\_gid %>%

c(admap\_fc0$map$attr\_gid) %>%

setdiff(d\_att$group\_id\_boundary$gid, .) %>%

setdiff(d\_att$missed\_blocks$gid)

admap\_fc1 <- admap\_fc0$map %>%

ai\_get\_dimention\_analysis\_details(d\_dat, d\_att, major\_direction\_relax = FALSE)

# try to attach rest attr\_gid if any to nearest data\_gid [on data\_gid boundary]

if (length(unmapped\_attr\_gids) > 0) {

admap\_other0 <- ai\_get\_data\_attr\_map(

dat\_boundary = d\_dat$group\_id\_boundary,

att\_gid\_map = d\_att$group\_id\_map %>% filter(gid %in% unmapped\_attr\_gids),

attr\_to\_near\_data = TRUE

)

unmapped\_attr\_gids <-

unmapped\_attr\_gids %>%

setdiff(admap\_other0$map$attr\_gid)

admap\_other1 <- admap\_other0$map %>%

ai\_get\_dimention\_analysis\_details(d\_dat, d\_att, major\_direction\_relax = FALSE)

admap\_fc1 <- merge\_admaps(admap\_fc1, admap\_other1)

}

}

# last stage of analysis

# this is not required

# d\_dat$group\_id\_whole\_boundary <- extend\_data\_block(d\_dat$group\_id\_extended\_boundary, admap\_fc1$map, d\_att)

admap2 <- merge\_admaps(admap1, admap\_fc1)

# join attr based on block merges possible (one more time)

rel\_chk <- ai\_relative\_data\_join\_attr(admap\_main = admap2, d\_att = d\_att)

if (rel\_chk$done) {

d\_att <- rel\_chk$d\_att %>% map(unique)

admap2 <- rel\_chk$admap

}

cmp <- compact\_gid\_maps(d\_att, admap2)

d\_att <- cmp$gid\_map

admap2 <- cmp$admap

admap3 <- admap2$map %>%

select(-attr\_group) %>%

ai\_get\_dimention\_analysis\_details(d\_dat, d\_att)

if (!identical(admap3$map, admap2$map)) {

# I think this can be iterated

# KFL

admap3\_pass <- admap3$map %>%

rename(md = dist) %>%

group\_by(data\_gid, direction\_group, attr\_group) %>%

mutate(m\_dist = min(md)) %>%

ungroup() %>%

filter(md == m\_dist) %>%

select(-md) %>%

rename(dist = m\_dist)

admap <- admap3\_pass %>%

select(-attr\_group) %>%

ai\_get\_dimention\_analysis\_details(d\_dat, d\_att)

} else {

admap <- admap3

}

# once admap is done

d\_dat$group\_id\_extended\_boundary <- NULL

d\_dat$group\_id\_whole\_boundary <- extend\_data\_block(d\_dat$group\_id\_boundary, admap$map, d\_att)

# str-detection done

this\_cells <- get\_cells\_from\_admap(admap, d\_dat, d\_att)

# natural gid for easier understanding

gid\_ngid <- d\_dat$group\_id\_map %>%

distinct(gid) %>%

mutate(natural\_gid = gid %>% as.factor() %>% as.numeric())

# attach natural gid

this\_cells <- this\_cells %>%

left\_join(gid\_ngid, by = "gid")

admap$raw\_map <- admap$raw\_map %>%

mutate(gid = data\_gid) %>%

left\_join(gid\_ngid, by = "gid")

admap$map <- admap$map %>%

mutate(gid = data\_gid) %>%

left\_join(gid\_ngid, by = "gid") %>%

select(-gid)

d\_dat$group\_id\_whole\_boundary <- d\_dat$group\_id\_whole\_boundary %>%

left\_join(gid\_ngid, by = "gid")

# attach directions to it

admap$raw\_map <- ai\_attach\_direction(admap$raw\_map)

df\_details <- get\_definiteness\_details(admap$raw\_map,

all\_attr\_gids = d\_att$group\_id\_boundary$gid %>%

setdiff(d\_att$missed\_blocks$gid)

)

definiteness\_checks <- get\_definiteness\_checks(df\_details, silent = silent)

obj <- list(

cells = this\_cells,

sections = d\_dat$group\_id\_whole\_boundary,

details = list(

attr\_details = d\_att,

data\_details = d\_dat,

data\_attr\_map\_raw = admap$raw\_map,

definiteness\_checks = definiteness\_checks

),

cell\_df = d\_orig

)

# attach cell\_df\_analysis class

class(obj) <- cell\_df\_analysis\_class

obj

}

ai\_attach\_direction <- function(d\_att\_dat\_map\_raw) {

# asp: attr split

d\_att\_dat\_map\_raw\_asp <- d\_att\_dat\_map\_raw %>%

# kept for tracking

mutate(direction\_basic = direction) %>%

mutate(attr\_gid\_split = ifelse(direction\_group == "NS" & attr\_group == "major", row\_a,

ifelse(direction\_group == "WE" & attr\_group == "major", col\_a, 0)

))

d\_att\_dat\_map\_raw\_asp <- d\_att\_dat\_map\_raw\_asp %>%

mutate(attr\_gid\_split = ifelse(direction\_group == "corner", col\_a, attr\_gid\_split))

d\_att\_dat\_map\_raw\_asp %>%

group\_by(data\_gid, attr\_gid, direction, attr\_gid\_split) %>%

group\_split() %>%

map\_df(~ .x %>% mutate(direction = get\_direction(.x)))

}

ai\_attr\_gid\_micro\_splits