The topology of software risk in scientific models

1. The synthetic example

Arnald Puy

Contents

1 Preliminary

```r
# PRELIMINARY FUNCTIONS ##########################################################
##################################################################################

sensobol::load_packages(c("data.table", "tidyverse", "openxlsx", "scales",
                          "cowplot", "readxl", "ggrepel", "tidytext", "here",
                          "tidygraph", "igraph", "foreach", "parallel", "ggraph",
                          "tools", "purrr", "sensobol", "benchmarkme"))

# Create custom theme ----------------------------------------------------------

theme_AP <- function() {
  theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          legend.background = element_rect(fill = "transparent", color = NA),
          legend.key = element_rect(fill = "transparent", color = NA),
          strip.background = element_rect(fill = "white"),
          legend.text = element_text(size = 7.3),
          axis.title = element_text(size = 10),
          legend.key.width = unit(0.4, "cm"),
          legend.key.height = unit(0.4, "cm"),
          legend.key.spacing.y = unit(0, "lines"),
          legend.box.spacing = unit(0, "pt"),
          legend.title = element_text(size = 7.3),
          axis.text.x = element_text(size = 7),
          axis.text.y = element_text(size = 7),
          axis.title.x = element_text(size = 7.3),
          axis.title.y = element_text(size = 7.3),
          plot.title = element_text(size = 8),
          strip.text.x = element_text(size = 7.4),
          strip.text.y = element_text(size = 7.4))
}

# Source all .R files in the "functions" folder --------------------------------

r_functions <- list.files(path = here("functions"),
                          pattern = "\\.R$", full.names = TRUE)

lapply(r_functions, source)

# Set seed ---------------------------------------------------------------------

seed <- 123

# Define labels for better plotting --------------------------------------------
```

```r
lab_expr <- c(b1 = expression(C %in% "(" * 0 * ", 10" * "]"),
              b2 = expression(C %in% "(" * 10 * ", 20" * "]"),
              b3 = expression(C %in% "(" * 20 * ", 50" * "]"),
              b4 = expression(C %in% "(" * 50 * ", " * infinity * ")"))
```

2 The synthetic example

```r
# SYNTHETIC EXAMPLE ##################################################################
#####################################################################################

# EXAMPLE: BUILD ACYCLIC GRAPH ######################################################

# Set seed ----------------------------------------------------------------------

seed <- 123

## Build a layered DAG: entries -> mid -> sinks ------------------------------

n_entry <- 5
n_mid   <- 35
n_sink  <- 15

entry_ids <- paste0("E", 1:n_entry)
mid_ids   <- paste0("M", 1:n_mid)
sink_ids  <- paste0("S", 1:n_sink)

nodes <- c(entry_ids, mid_ids, sink_ids)

edges <- list()

## ---- E -> M. Each entry calls 2-5 mid-level functions ----------------------
# Top level drivers typically fan out to multiple subroutines. This example is
# meant to reproduce the fact that each entry is not just a one-to-one call. ---

set.seed(seed)

for (e in entry_ids) {

  k <- sample(2:5, 1)  # desired fan-out
  targets <- sample_safe_fun(mid_ids, k)

  if (length(targets) > 0) {
    edges[[length(edges) + 1]] <- cbind(e, targets)
  }
}
```

```r
## ---- M -> M (forward only by index to avoid cycles). Our example is an
# acyclic graph and so there are no loops or recursions.- ----------------------

set.seed(seed)

for (i in seq_along(mid_ids)) {

  from <- mid_ids[i]

  if (i < length(mid_ids)) {

    candidates <- mid_ids[(i+1):length(mid_ids)]
    candidates <- candidates[!is.na(candidates)]

  } else {

    candidates <- character(0)
  }

  if (length(candidates) > 0) {

    k <- sample(0:3, 1)      # some may have 0 mid->mid edges
    targets <- sample_safe_fun(candidates, k)

    if (length(targets) > 0) {
      edges[[length(edges) + 1]] <- cbind(from, targets)
    }
  }
}

## ---- M -> S. Each mid function calls 1-3 sinks ----------------------------

set.seed(seed)

for (m in mid_ids) {

  k <- sample(1:3, 1)

  targets <- sample(sink_ids, k, replace = TRUE)
  edges[[length(edges) + 1]] <- cbind(m, targets)
}

edge_mat <- do.call(rbind, edges)
colnames(edge_mat) <- c("from", "to")

# Create graph ------------------------------------------------------------
```

```r
g <- as_tbl_graph(edge_mat, directed = TRUE)

# DEFINE NODE METRICS ##########################################################
################################################################################

# Fake cyclomatic complexity (1 - 80; log-normal distribution)
# for toy simulation ----------------------------------------------------------

n <- length(V(g))

set.seed(seed)

raw_cyclo <- rlnorm(n, meanlog = 1.2, sdlog = 0.4)  # heavier tail
raw_cyclo <- raw_cyclo ^ 1.7 # Pareto-like amplification

# Define weights and compute node metrics-------------------------------------

alpha_weight <- 0.6
beta_weight <- 0.3
gamma_weight <- 0.1

g <- g %>%
  activate(nodes) %>%
  mutate(cyclo = round(scales::rescale(raw_cyclo, to = c(1, 60))),
         indeg = degree(g, mode = "in"),
         outdeg = degree(g, mode = "out"),
         btw = betweenness(g, directed = TRUE, weights = NULL,
                           normalized = FALSE),
         cyclo_sc = rescale(cyclo),
         indeg_sc = rescale(indeg),
         btw_sc = rescale(btw),
         risk_score =  alpha_weight * cyclo_sc + beta_weight * indeg_sc +
           gamma_weight * btw_sc,
         cyclo_class = case_when(cyclo <= 10 ~ "green",
                                 cyclo <= 20 ~ "orange",
                                 cyclo <= 50 ~ "red",
                                 cyclo >  50 ~ "purple",
                                 TRUE ~ "grey"),
         complexity_category = cut(cyclo,
                                   breaks = c(-Inf, 10, 20, 50, Inf),
                                   labels = c("b1","b2","b3","b4")))

# Create data frame of nodes --------------------------------------------------

node_df <- g %>%
  activate(nodes) %>%
  as_tibble()
```

```r
# DEFINE PATH METRICS ##########################################################
################################################################################

paths_tbl <- all_paths_fun(node_df = node_df, graph = g)

# MONTE CARLO CHECK ############################################################
################################################################################

# Define number of montecarlo runs ---------------------------------------------

n_runs <- 10^5

set.seed(seed)
path_choices <- sample(paths_tbl$path_id, size = n_runs, replace = TRUE)

# Each run picks a path uniformly and simulates independent node failures
# according to risk_score. A run fails if any node fails. Fail_pos records  where
# in the chain the first failure happens (for slope analysis) ------------------

sim_results <- map_dfr(seq_len(n_runs), function(i) {

  pid <- path_choices[i]
  v_names <- paths_tbl$path_nodes[[pid]]
  ps <- node_df$risk_score[match(v_names, node_df$name)]
  fails <- rbinom(length(ps), size = 1, prob = ps)
  run_fail <- any(fails == 1)
  fail_pos <- if (run_fail) which(fails == 1)[1] else NA_integer_
  tibble(run_id = i,
         path_id = pid,
         fail = run_fail,
         fail_pos = fail_pos)
})

# Aggregate the results --------------------------------------------------------

empirical_tbl <- sim_results %>%
  group_by(path_id) %>%
  summarise(runs = n(),
            fails = sum(fail),
            emp_fail_rate = fails / runs,
            mean_fail_pos = if (any(fail)) mean(fail_pos[fail], na.rm = TRUE) else NA_real_,
            .groups = "drop")

paths_eval <- paths_tbl %>%
  left_join(empirical_tbl, by = "path_id")

cat("Correlation(P_k, empirical failure):",
```

```
    cor(paths_eval$p_path_fail, paths_eval$emp_fail_rate, use = "complete.obs"), "\n")
```

```
## Correlation(P_k, empirical failure): 0.9954597
```

```
# ILLUSTRATION OF THE GINI INDEX: FIXING TOP-RISK NODES VS RANDOM NODES ########

# Identify the highest risk node per path ------------------------------------

top_node_tbl <- paths_tbl %>%
  mutate(p_vec = map(path_nodes, ~ node_df$risk_score[match(.x, node_df$name)])) %>%
  mutate(idx_top = map_int(p_vec, which.max),
         top_node = map2_chr(path_nodes, idx_top, ~ .x[.y]),
         top_p = map2_dbl(p_vec, idx_top, ~ .x[.y])) %>%
  select(path_id, path_nodes, p_vec, gini_node_risk, idx_top, top_node, top_p)

recompute_Pk <- function(p_vec, kill_index) {
  if (length(p_vec) == 0) return(0)
  p_new <- p_vec
  p_new[kill_index] <- 0
  1 - prod(1 - p_new)
}

set.seed(42)

gini_effect_tbl <- top_node_tbl %>%
  mutate(# Original path failure probability ---------------------------------------
    P_orig = map_dbl(p_vec, ~ if (length(.x) == 0) 0 else 1 - prod(1 - .x)),

    # Path failure if we set the risk of the top-risk node at 0 ----------------
    P_fix_top = map2_dbl(p_vec, idx_top, recompute_Pk),
    rand_idx = map_int(p_vec, ~ sample(seq_along(.x), 1)),

    # Path failure if we randomly fix any node ---------------------------------
    P_fix_rand = map2_dbl(p_vec, rand_idx, recompute_Pk)) %>%
  mutate(drop_top  = P_orig - P_fix_top,
         drop_rand = P_orig - P_fix_rand)


# Summary of the effect that fixing a node has on gini -------------------------

gini_effect_tbl %>%
  summarise(cor_gini_drop_top  = cor(gini_node_risk, drop_top),
            cor_gini_drop_rand = cor(gini_node_risk, drop_rand)) %>%
  print()
```

```
## # A tibble: 1 x 2
##   cor_gini_drop_top cor_gini_drop_rand
##               <dbl>              <dbl>
```

```
## 1             0.554               0.122
```

```
# ILLUSTRATION OF THE SLOPE ###########################################################

# Where do failures tend to occur? ------------------------------------------

slope_eval <- paths_eval %>%
  filter(fails > 0) %>%
  select(path_id, risk_slope, mean_fail_pos, p_path_fail)

# WHICH PATHS IMPROVE IF A NODE'S RISK IS FIXED TO 0? ##########################

# how many nodes and paths to show in the heatmap ---------------------------

number_of_interest <- 20
N_nodes <- number_of_interest
K_paths <- number_of_interest

# top-N nodes by failure probability ----------------------------------------

top_nodes <- node_df %>%
  arrange(desc(risk_score)) %>%
  slice_head(n = N_nodes)

# top-K paths by path failure probability -----------------------------------

top_paths <- paths_tbl %>%
  arrange(desc(p_path_fail)) %>%
  slice_head(n = K_paths)

# COMPUTE ##############################################################################

# make a named vector for fast lookup of risk_score by node name ------------------

p_map <- setNames(node_df$risk_score, node_df$name)

# Compute -------------------------------------------------------------------

delta_tbl <- map_dfr(seq_len(nrow(top_nodes)), function(i) {
  node_name <- top_nodes$name[i]

  map_dfr(seq_len(nrow(top_paths)), function(j) {
    pid <- top_paths$path_id[j]
    nodes_vec <- top_paths$path_nodes[[j]]
    P_orig <- top_paths$p_path_fail[j]

    # if node not in this path, improvement is zero ---------------------------
```

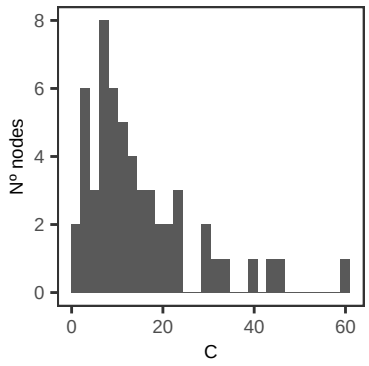
```
    if (!node_name %in% nodes_vec) {
      tibble(node = node_name,
             path_id = pid,
             deltaP = 0)
    } else {

      # failure probs along the path ----------------------------------------

      p_vec <- p_map[nodes_vec]

      # if node appears multiple times, fixing ANY of its appearances is enough:

      idxs <- which(nodes_vec == node_name)

      # we fix all occurrences (set all to zero) ------------

      p_vec_fix <- p_vec
      p_vec_fix[idxs] <- 0

      P_fix <- if (length(p_vec_fix) == 0) 0 else 1 - prod(1 - p_vec_fix)
      tibble(node = node_name,
             path_id = pid,
             deltaP = P_orig - P_fix)
    }
  })
})

# order factors so tha highest risk nodes are at the top and the highest
# risk paths are on the left ------------------------------------------------

# Bright cells: nodes that are chokepoints for a given path.
# Rows with many bright cells: nodes whose refactoring will improve many
# risk paths (global chokepoints)
# Columns with few very bright cells: paths dominated by a single risky node
# Rows/columns with dark colors: diffuse risk.

delta_tbl <- delta_tbl %>%
  mutate(node = factor(node, levels = rev(top_nodes$name)),
         path_id = factor(path_id, levels = top_paths$path_id))
```

```
# SAVE NODES AND PATHS DATASET FOR THE UNCERTAINTY ANALYSIS IN NEXT FILE #######

saveRDS(list(nodes_df = node_df, paths_tbl = paths_tbl), "graph_objects.rds")
```

```
################################################################################
################################################################################

# PLOT THE FIGURES OF THE SIMULATION ###########################################
```

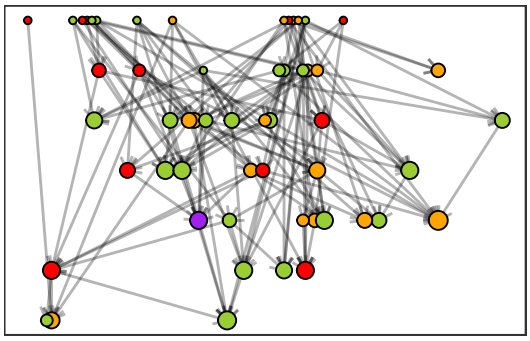
```
################################################################################
################################################################################

# Define size of points in graphs ----------------------------------------------

size_points <- 1

#Plot the distribution of cyclomatic complexity in this toy model --------------

distribution_cc <- node_df %>%
  ggplot(., aes(cyclo)) +
  geom_histogram() +
  labels(x = "C", y = "Nº nodes") +
  theme_AP()

distribution_cc
```

```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```

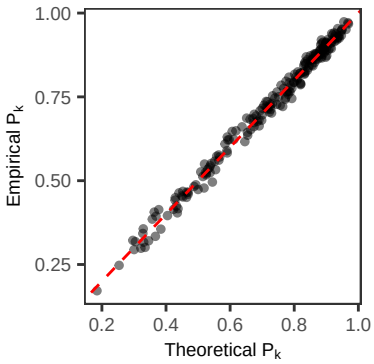



```
# Plot the call graph ----------------------------------------------------------

set.seed(seed)

toy_graph <- ggraph(g, layout = "sugiyama") +
  geom_edge_link(alpha = 0.3, arrow = arrow(length = unit(2, "mm"))) +
  geom_node_point(aes(size = indeg, fill = complexity_category),
                  shape = 21, colour = "black", show.legend = TRUE) +
  scale_size_continuous(range = c(1, 3), guide = "none") +
  scale_fill_manual(values = c("yellowgreen", "orange", "red", "purple"),
                    labels = lab_expr,
                    name = "") +
  labs(x = "", y = "") +
  theme_AP() +
  theme(axis.text.y = element_blank(),
        axis.ticks.y = element_blank(),
        axis.text.x = element_blank(),
        axis.ticks.x = element_blank(),
```

```
        legend.position = "none")

legend <- get_legend(toy_graph + theme(legend.position = "top"))

toy_graph
```

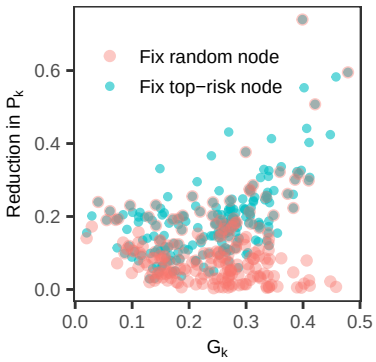


```
# Plot the trend between theoretical and empirical failure ---------------------

fig_Pk <- ggplot(paths_eval, aes(x = p_path_fail, y = emp_fail_rate)) +
  geom_point(alpha = 0.5, size = size_points) +
  geom_abline(slope = 1, intercept = 0, colour = "red", linetype = 2) +
  labs(x = expression("Theoretical " * P[k]),
       y = expression("Empirical " * P[k])) +
  theme_AP()

fig_Pk
```

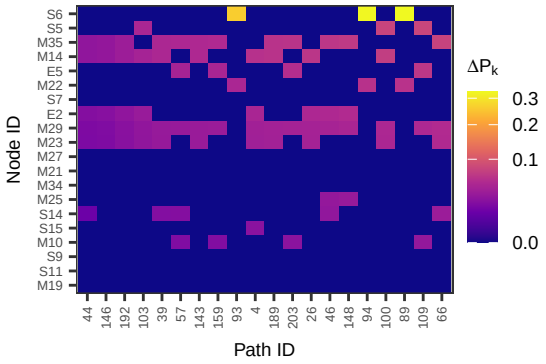



```
# Plot the gini figure ---------------------------------------------------------

fig_gini <- ggplot(gini_effect_tbl, aes(x = gini_node_risk)) +
  geom_point(aes(y = drop_top,  colour = "Fix top-risk node"), alpha = 0.6, size = size_points)
  geom_point(aes(y = drop_rand, colour = "Fix random node"), alpha = 0.4) +
  labs(x = expression(G[k]),
       y = expression("Reduction in " * P[k]), colour = NULL) +
  theme_AP() +
  theme(legend.position = c(0.4, 0.78))
```

```
fig_gini
```

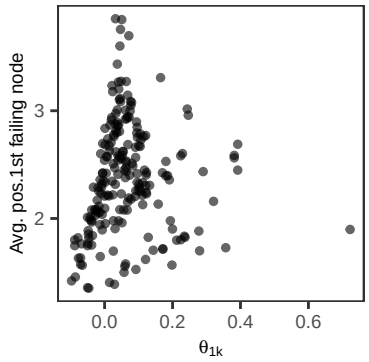



```
# Plote the tile figure ------------------------------------------------------

minP <- min(delta_tbl$deltaP, na.rm = TRUE)
medP <- median(delta_tbl$deltaP, na.rm = TRUE)
maxP <- max(delta_tbl$deltaP, na.rm = TRUE)

fig_heat <- ggplot(delta_tbl, aes(x = path_id, y = node, fill = deltaP)) +
  geom_tile() +
  scale_fill_viridis_c(option = "C",
                       name = expression(Delta * P[k]),
                       trans  = "sqrt") +
  theme_AP() +
  theme(axis.text.x  = element_text(angle = 90, vjust = 0.5, hjust = 1, size = 5),
        axis.text.y = element_text(size = 5),
        axis.title.x = element_text(margin = margin(t = 5)),
        axis.title.y = element_text(margin = margin(r = 5))) +
  labs(x = "Path ID", y = "Node ID")

fig_heat
```




```
# Plot the slope figure ------------------------------------------------------

fig_slope <- ggplot(slope_eval, aes(x = risk_slope, y = mean_fail_pos)) +
  geom_point(alpha = 0.6, size = size_points) +
  labs(x = expression(theta[1*k]),
```

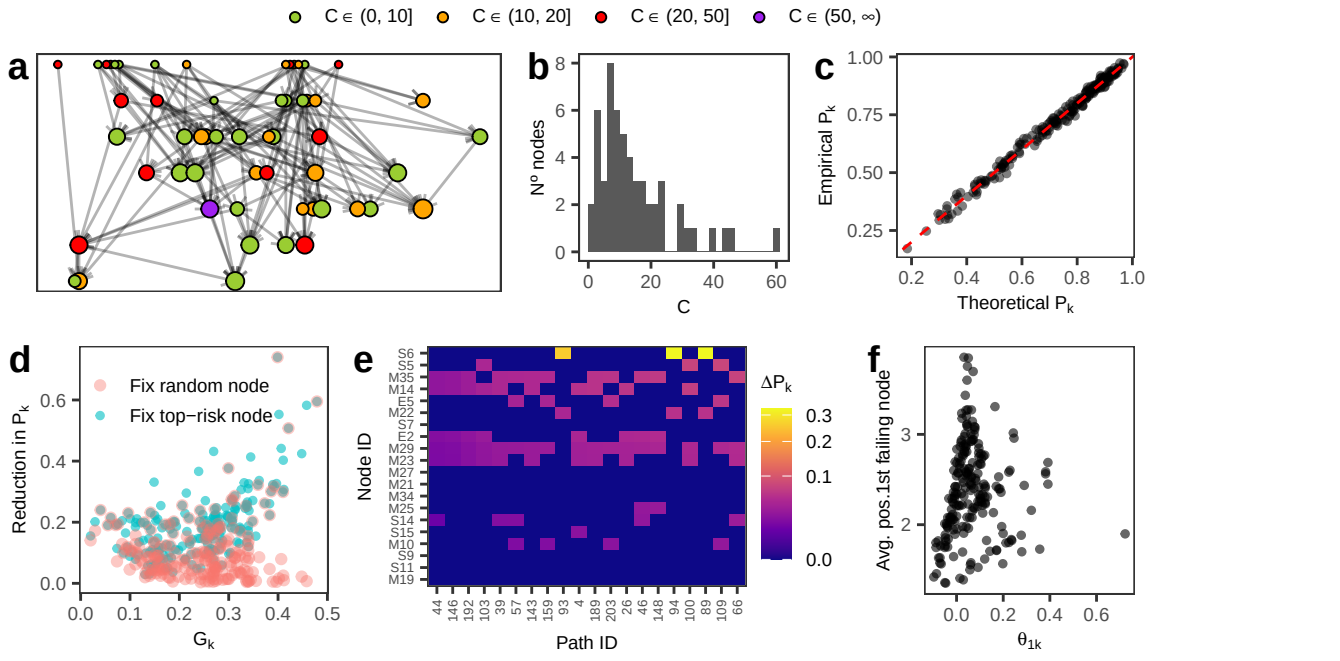
```
       y = "Avg. pos.1st failing node") +
  theme_AP()

fig_slope
```




```
# MERGE ALL FIGURES ##############################################################

top <- plot_grid(toy_graph, distribution_cc, fig_Pk, ncol = 3, labels = "auto",
                 rel_widths = c(0.45, 0.25, 0.3))
```

```
## `stat_bin()` using `bins = 30`. Pick better value `binwidth`.
```

```
top_final <- plot_grid(legend, top, ncol = 1, rel_heights = c(0.13, 0.87))
bottom <- plot_grid(fig_gini, fig_heat, fig_slope, ncol = 3, labels = c("d", "e", "f"),
                    rel_widths = c(0.3, 0.45, 0.25))
final_plot <- plot_grid(top_final, bottom, ncol = 1)
final_plot
```



3 Session information

```
# SESSION INFORMATION ############################################################

sessionInfo()
```

```
## R version 4.5.2 (2025-10-31)
## Platform: aarch64-apple-darwin20
## Running under: macOS Sequoia 15.6.1
##
## Matrix products: default
## BLAS:   /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framew
## LAPACK: /Library/Frameworks/R.framework/Versions/4.5-arm64/Resources/lib/libRlapack.dylib;
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## time zone: Europe/London
## tzcode source: internal
##
## attached base packages:
## [1] tools     parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] benchmarkme_1.0.8  sensobol_1.1.6     ggraph_2.2.2       foreach_1.5.2
##  [5] igraph_2.1.4       tidygraph_1.3.1    here_1.0.2         tidytext_0.4.3
##  [9] ggrepel_0.9.6      readxl_1.4.5       cowplot_1.2.0.9000 scales_1.4.0
## [13] openxlsx_4.2.8     lubridate_1.9.4    forcats_1.0.1      stringr_1.5.2
## [17] dplyr_1.1.4        purrr_1.1.0        readr_2.1.5        tidyr_1.3.1
## [21] tibble_3.3.0       ggplot2_4.0.0.9000 tidyverse_2.0.0    data.table_1.17.8
##
## loaded via a namespace (and not attached):
##  [1] benchmarkmeData_1.0.4 gtable_0.3.6          xfun_0.53
##  [4] lattice_0.22-7        tzdb_0.5.0            Rdpack_2.6.4
##  [7] vctrs_0.6.5           generics_0.1.4        janeaustenr_1.0.0
## [10] pkgconfig_2.0.3       tokenizers_0.3.0      Matrix_1.7-4
## [13] RColorBrewer_1.1-3    S7_0.2.0              lifecycle_1.0.4
## [16] compiler_4.5.2        farver_2.1.2          ggforce_0.5.0
## [19] graphlayouts_1.2.2    codetools_0.2-20      htmltools_0.5.8.1
## [22] SnowballC_0.7.1       yaml_2.3.10           pillar_1.11.1
## [25] MASS_7.3-65           cachem_1.1.0          viridis_0.6.5
## [28] iterators_1.0.14      tidyselect_1.2.1      zip_2.3.3
## [31] digest_0.6.37         stringi_1.8.7         polyclip_1.10-7
## [34] rprojroot_2.1.1       fastmap_1.2.0         grid_4.5.2
## [37] cli_3.6.5             magrittr_2.0.4        withr_3.0.2
## [40] timechange_0.3.0      httr_1.4.7            rmarkdown_2.30
## [43] gridExtra_2.3         cellranger_1.1.0      hms_1.1.3
```

```
## [46] memoise_2.0.1         evaluate_1.0.5        knitr_1.50
## [49] rbibutils_2.3         doParallel_1.0.17     viridisLite_0.4.2
## [52] rlang_1.1.6           Rcpp_1.1.0            glue_1.8.0
## [55] tweenr_2.0.3          rstudioapi_0.17.1     R6_2.6.1
```

```
## Return the machine CPU ------------------------------------------------

cat("Machine:     "); print(get_cpu()$model_name)
```

```
## Machine:
## [1] "Apple M1 Max"
```

```
## Return number of true cores -------------------------------------------

cat("Num cores:   "); print(detectCores(logical = FALSE))
```

```
## Num cores:
## [1] 10
```

```
## Return number of threads ----------------------------------------------

cat("Num threads: "); print(detectCores(logical = FALSE))
```

```
## Num threads:
## [1] 10
```