The topology of software risk in scientific models

4. Scalability stress test

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", "doParallel"))

# 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))
}
```

```r
# SYNTHETIC SCALABILITY EXPERIMENT #############################################
###############################################################################

# -CREATE SYNTHETIC CALL GRAPH #################################################

make_synthetic_callgraph <- function(depth = 20, width = 200, branching = 3, seed = 1) {
  set.seed(seed)

  entry_id <- 1L
  layers <- vector("list", depth + 1L)
  layers[[1L]] <- entry_id

  id <- entry_id

  for (d in 2:(depth + 1L)) {
    layers[[d]] <- (id + 1L):(id + width)
    id <- id + width
  }
  exit_ids <- layers[[depth + 1L]]

  edges <- vector("list", depth)

  for (d in 1:depth) {
    from <- layers[[d]]
    to   <- layers[[d + 1L]]

    k <- pmax(1L, rpois(length(from), lambda = branching))
    ed <- lapply(seq_along(from), function(i) {
      cbind(from[i], sample(to, size = k[i], replace = TRUE))
    })
    edges[[d]] <- do.call(rbind, ed)
  }

  edge_mat <- do.call(rbind, edges)

  # explicit vertex names as characters-----------------------------------------

  edge_mat_chr <- matrix(as.character(edge_mat), ncol = 2)
  g <- graph_from_edgelist(edge_mat_chr, directed = TRUE)

  # clean up -------------------------------------------------------------------

  g <- delete_edges(g, E(g)[which_loop(g)])
  g <- delete_edges(g, E(g)[which_multiple(g)])

  # ensure all vertices exist --------------------------------------------------
  all_ids <- as.character(seq_len(id))
  missing <- setdiff(all_ids, V(g)$name)

  if (length(missing) > 0) {
    g <- add_vertices(g, nv = length(missing), name = missing)
  }

  list(g = g,
       entry_name = "1",
       exit_names = as.character(exit_ids))
}

# DEFINE NODE ATTRIBUTES #######################################################

assign_node_attributes <- function(g, seed = 1,
                                   cyclo_dist = c("lognormal", "gamma"),
                                   cyclo_scale = 1.0,
                                   btw_method = c("estimate", "cutoff", "exact"),
                                   btw_cutoff = 6) {
  set.seed(seed)
  cyclo_dist <- match.arg(cyclo_dist)
  btw_method <- match.arg(btw_method)

  n <- vcount(g)

  cyclo_raw <- switch(cyclo_dist,
                      lognormal = rlnorm(n, meanlog = log(3), sdlog = 0.9),
                      gamma = rgamma(n, shape = 2.0, rate = 0.5))

  cyclo_raw <- pmax(1, cyclo_raw * cyclo_scale)

  indeg_raw <- degree(g, mode = "in")

  btw_raw <- NULL

  if (btw_method == "estimate" &&
      exists("estimate_betweenness", where = asNamespace("igraph"), inherits = FALSE)) {

    btw_raw <- tryCatch(
      igraph::estimate_betweenness(g, directed = TRUE),
      error = function(e) NULL)

  }
  if (is.null(btw_raw) && btw_method %in% c("estimate", "cutoff")) {

    btw_raw <- betweenness(g, directed = TRUE, normalized = FALSE, cutoff = btw_cutoff)
  }
```

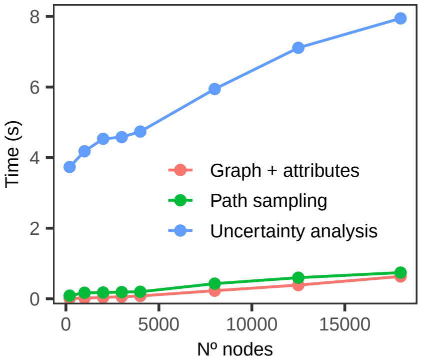
```r
  if (is.null(btw_raw)) {

    btw_raw <- betweenness(g, directed = TRUE, normalized = FALSE)
  }

  V(g)$cyclo_raw <- as.numeric(cyclo_raw)
  V(g)$indeg_raw <- as.numeric(indeg_raw)
  V(g)$btw_raw   <- as.numeric(btw_raw)

  V(g)$cyclo <- as.numeric(rescale(V(g)$cyclo_raw, to = c(0, 1)))
  V(g)$indeg <- as.numeric(rescale(V(g)$indeg_raw, to = c(0, 1)))
  V(g)$btw   <- as.numeric(rescale(V(g)$btw_raw, to = c(0, 1)))

  g
}

# SAMPLING OF PATHS USING ADJACENCY LISTS ######################################

sample_paths_fast <- function(g, entry_name, exit_names,
                              n_paths = 5000, max_steps = 500, seed = 1) {
  set.seed(seed)

  entry_vid <- which(V(g)$name == entry_name)
  exit_vids <- which(V(g)$name %in% exit_names)

  if (length(entry_vid) != 1) stop("Entry node not found uniquely by name.")
  if (length(exit_vids) == 0) stop("No exit nodes found by name.")

  adj <- lapply(as_adj_list(g, mode = "out"), as.integer)
  is_exit <- rep(FALSE, vcount(g))
  is_exit[exit_vids] <- TRUE

  paths <- vector("list", n_paths)
  ok <- logical(n_paths)

  for (p in seq_len(n_paths)) {
    cur <- entry_vid
    path <- integer(1 + max_steps)
    path[1L] <- cur
    len <- 1L
    steps <- 0L

    while (!is_exit[cur] && steps < max_steps) {
      nbrs <- adj[[cur]]
      if (length(nbrs) == 0L) break
      cur <- nbrs[sample.int(length(nbrs), 1L)]
      steps <- steps + 1L
      len <- len + 1L
      path[len] <- cur
    }

    if (is_exit[cur]) {
      ok[p] <- TRUE
      paths[[p]] <- path[seq_len(len)]
    } else {
      paths[[p]] <- NULL
    }
  }

  paths[ok]
}

# CREATION OF PATH RISK SCORE ##################################################

score_paths_saturating <- function(g, paths, alpha = 1/3, beta = 1/3, gamma = 1/3) {
  r_node <- alpha * V(g)$cyclo + beta * V(g)$indeg + gamma * V(g)$btw
  vapply(paths, function(p) 1 - prod(1 - r_node[p]), numeric(1))
}

# UA/SA ANALYSIS ###############################################################

ua_sa_path_stability <- function(g, paths, K = 10, n_weight_samples = 200,
                                 weight_sampler = c("dirichlet", "uniform"),
                                 seed = 1) {
  set.seed(seed)
  weight_sampler <- match.arg(weight_sampler)

  W <- switch(weight_sampler,
              dirichlet = {
                X <- matrix(rexp(n_weight_samples * 3), ncol = 3)
                X / rowSums(X)
              },
              uniform = {
                X <- matrix(runif(n_weight_samples * 3), ncol = 3)
                X / rowSums(X)
              }
  )
  colnames(W) <- c("alpha", "beta", "gamma")

  if (length(paths) == 0) {
    return(list(weights = W, freq_topK = numeric(0), stable_paths = integer(0)))
  }

  topK <- vector("list", n_weight_samples)
```

```r
  for (i in seq_len(n_weight_samples)) {
    sc <- score_paths_saturating(g, paths, alpha = W[i,1], beta = W[i,2], gamma = W[i,3])
    topK[[i]] <- order(sc, decreasing = TRUE)[seq_len(min(K, length(sc)))]
  }

  counts <- integer(length(paths))

  for (i in seq_len(n_weight_samples)) counts[topK[[i]]] <- counts[topK[[i]]] + 1L
  freq <- counts / n_weight_samples

  stable_rank <- order(freq, decreasing = TRUE)

  list(weights = W,
       freq_topK = freq,
       stable_paths = stable_rank[seq_len(min(K, length(stable_rank)))])
}

# PREPARE TO RUN IN PARALLEL ####################################################

run_scalability_experiment_parallel <- function(configs,
                                                n_paths = 5000,
                                                n_weight_samples = 200,
                                                K = 10,
                                                seed = 42,
                                                n_cores = max(1L, detectCores() - 1L),
                                                btw_method = c("estimate", "cutoff", "exact"),
                                                btw_cutoff = 6) {
  btw_method <- match.arg(btw_method)

  # Create parallel end ---------------------------------------------------------

  cl <- makeCluster(n_cores)
  registerDoParallel(cl)
  on.exit(stopCluster(cl), add = TRUE)

  res <- foreach(i = seq_along(configs),
                 .combine = rbind,
                 .packages = c("igraph", "scales"),
                 .export = c("make_synthetic_callgraph",
                             "assign_node_attributes",
                             "sample_paths_fast",
                             "score_paths_saturating",
                             "ua_sa_path_stability")) %dopar% {

    cfg <- configs[[i]]

    # Warm up -------------------------------------------------------------------
    tmp_g <- igraph::make_ring(50, directed = TRUE)

    # Warm-up exact betweenness
    igraph::betweenness(tmp_g, directed = TRUE, normalized = FALSE)

    # Warm-up estimate_betweenness too (this is the one that can cause ~10s first-call spikes)

    if (exists("estimate_betweenness", where = asNamespace("igraph"), inherits = FALSE)) {
      tryCatch(igraph::estimate_betweenness(tmp_g, directed = TRUE), error = function(e) NULL)
}

    # Build graph and allocated attributes -------------------------------------

    t0 <- proc.time()[["elapsed"]]

    cg <- make_synthetic_callgraph(depth = cfg$depth,
                                   width = cfg$width,
                                   branching = cfg$branching,
                                   seed = seed + i)

    g <- assign_node_attributes(cg$g,
                                seed = seed + 1000 + i,
                                btw_method = btw_method,
                                btw_cutoff = btw_cutoff)

    t_build <- proc.time()[["elapsed"]] - t0

    # Sample paths -------------------------------------------------------------

    t1 <- proc.time()[["elapsed"]]

    paths <- sample_paths_fast(g, cg$entry_name, cg$exit_names, n_paths = n_paths,
                               max_steps = max(500, cfg$depth * 5),
                               seed = seed + 2000 + i)

    t_paths <- proc.time()[["elapsed"]] - t1

    # UA/SA --------------------------------------------------------------------

    t2 <- proc.time()[["elapsed"]]

    uasa <- ua_sa_path_stability(g, paths,
                                 K = K,
                                 n_weight_samples = n_weight_samples,
                                 seed = seed + 3000 + i)

    t_uasa <- proc.time()[["elapsed"]] - t2
    # Summary ------------------------------------------------------------------

    if (length(uasa$freq_topK) == 0) {

      top1_stability <- NA_real_
      topK_median_stability <- NA_real_
      topK_mean_stability <- NA_real_

    } else {

      stable_freqs <- sort(uasa$freq_topK, decreasing = TRUE)
      top1_stability <- stable_freqs[1L]
      topK_vec <- stable_freqs[seq_len(min(K, length(stable_freqs)))]
      topK_median_stability <- median(topK_vec)
      topK_mean_stability <- mean(topK_vec)
    }

    data.frame(depth = cfg$depth,
               width = cfg$width,
               branching = cfg$branching,
               n_nodes = vcount(g),
               n_edges = ecount(g),
               n_sampled_paths = length(paths),
               time_build = t_build,
               time_sample_paths = t_paths,
               time_uasa = t_uasa,
               top1_stability = top1_stability,
               topK_median_stability = topK_median_stability,
               topK_mean_stability = topK_mean_stability,
               btw_method = btw_method,
               btw_cutoff = btw_cutoff,
               stringsAsFactors = FALSE)
  }

  res
}

# CONFIGURATIONS ###############################################################

configs <- list(list(depth = 10, width = 20,  branching = 2.5),
                list(depth = 20, width = 50,  branching = 2.5),
                list(depth = 20, width = 100, branching = 2.5),
                list(depth = 20, width = 150, branching = 2.5),
                list(depth = 20, width = 200, branching = 2.5),
                list(depth = 40, width = 200, branching = 3.0),
                list(depth = 50, width = 250, branching = 3.0),
                list(depth = 60, width = 300, branching = 3.0))
```

```r
# RUN THE STRESS TEST ##########################################################

res <- run_scalability_experiment_parallel(configs = configs,
                                           n_paths = 2000,
                                           n_weight_samples = 2000,
                                           K = 10,
                                           seed = 123,
                                           n_cores = 8,
                                           btw_method = "estimate",
                                           btw_cutoff = 6)

print(res)
```

```
##   depth width branching n_nodes n_edges n_sampled_paths time_build
## 1    10    20       2.5     201     417            2000      0.009
## 2    20    50       2.5    1001    2329            2000      0.019
## 3    20   100       2.5    2001    4896            2000      0.039
## 4    20   150       2.5    3001    7232            2000      0.057
## 5    20   200       2.5    4001    9688            2000      0.080
## 6    40   200       3.0    8001   23537            2000      0.229
## 7    50   250       3.0   12501   36877            2000      0.389
## 8    60   300       3.0   18001   54070            2000      0.633
##   time_sample_paths time_uasa top1_stability topK_median_stability
## 1             0.089     3.734         0.7785               0.54550
## 2             0.171     4.180         0.8725               0.63800
## 3             0.178     4.532         0.8430               0.40150
## 4             0.190     4.580         0.6215               0.43050
## 5             0.199     4.737         0.7225               0.46850
## 6             0.429     5.943         0.7670               0.49925
## 7             0.598     7.111         0.9330               0.59475
## 8             0.742     7.945         0.7890               0.51000
##   topK_mean_stability btw_method btw_cutoff
## 1             0.55400   estimate          6
## 2             0.61555   estimate          6
## 3             0.46865   estimate          6
## 4             0.46115   estimate          6
## 5             0.50710   estimate          6
## 6             0.53920   estimate          6
## 7             0.58845   estimate          6
## 8             0.53810   estimate          6
```

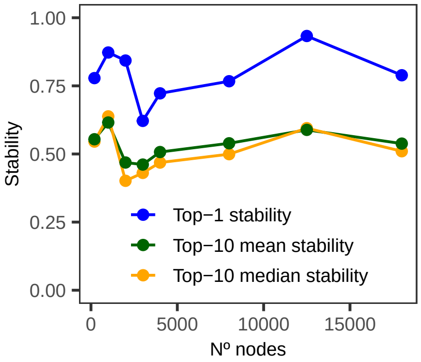
```r
# Time vs n_nodes --------------------------------------------------------------

res_time_long <- res %>%
  select(n_nodes, time_build, time_sample_paths, time_uasa) %>%
  pivot_longer(cols = starts_with("time_"),
               names_to = "stage",
                values_to = "seconds") %>%
  mutate(stage = recode(stage,
                         time_build = "Graph + attributes",
                         time_sample_paths = "Path sampling",
                         time_uasa = "Uncertainty analysis"))

p_time <- ggplot(res_time_long, aes(x = n_nodes, y = seconds, group = stage, color = stage)) +
  geom_line() +
  geom_point() +
  scale_y_continuous(expand = expansion(mult = c(0.02, 0.05))) +
  labs(x = "Nº nodes", y = "Time (s)") +
  scale_color_discrete(name = "") +
  theme_AP() +
  theme(legend.position = c(0.6,0.4))

p_time
```



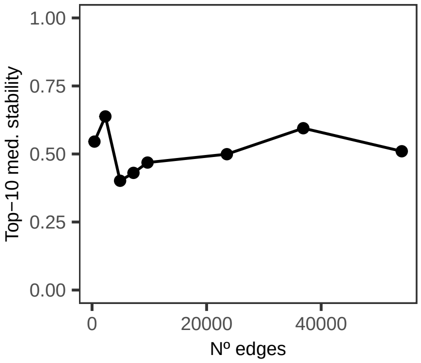
```
# Stability vs n_nodes ---------------------------------------------------------

res_stab_long <- res %>%
  select(n_nodes, top1_stability, topK_median_stability, topK_mean_stability) %>%
  pivot_longer(cols = c(top1_stability, topK_median_stability, topK_mean_stability),
               names_to = "metric", values_to = "stability") %>%
  mutate(metric = recode(metric,
                          top1_stability = "Top-1 stability",
                          topK_median_stability = "Top-10 median stability",
                          topK_mean_stability = "Top-10 mean stability"))

p_stab <- ggplot(res_stab_long, aes(x = n_nodes, y = stability, group = metric, color = metric
  geom_line() +
  geom_point() +
  coord_cartesian(ylim = c(0, 1)) +
  labs(x = "Nº nodes", y = "Stability", shape = "Metric") +
  theme_AP() +
  scale_color_manual(values = c("blue", "darkgreen", "orange"),
                     name = "") +
  theme(legend.position = c(0.5, 0.25))
```

```
p_stab
```

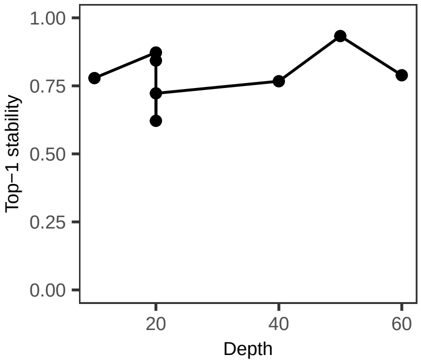
```
## Ignoring unknown labels:
## * shape : "Metric"
```



```
# Top-K median stability vs n_edges------------------------------------------------

p_stab_edges <- ggplot(res, aes(x = n_edges, y = topK_median_stability)) +
  geom_line() +
  geom_point() +
  coord_cartesian(ylim = c(0, 1)) +
  scale_x_continuous(breaks = breaks_pretty(n = 3)) +
  labs(x = "Nº edges", y = "Top-10 med. stability") +
  theme_AP()

p_stab_edges
```



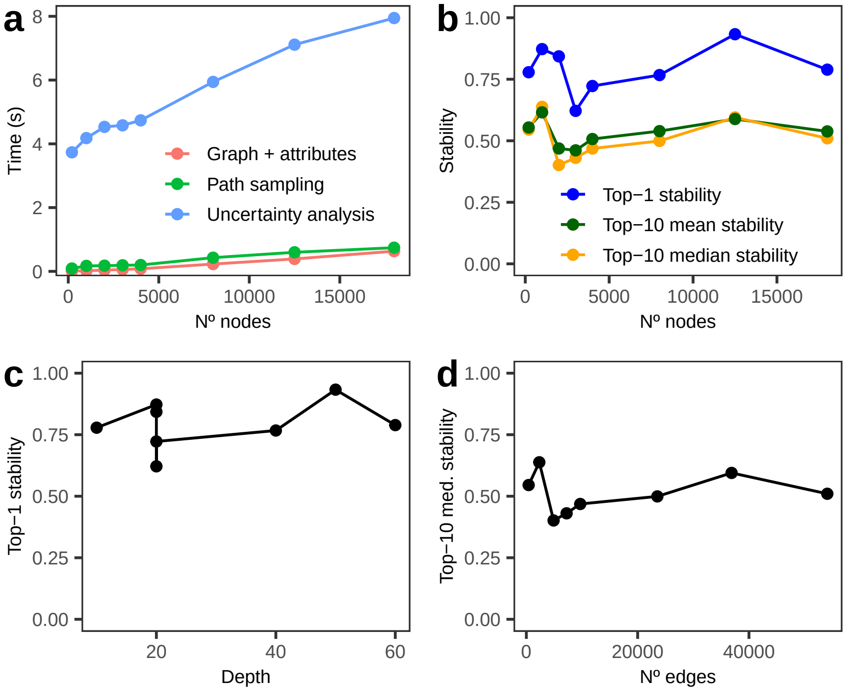
```
# Stability vs depth (architecture) ---------------------------------------------

p_stab_depth <- ggplot(res, aes(x = depth, y = top1_stability)) +
  geom_line() +
  geom_point() +
  coord_cartesian(ylim = c(0, 1)) +
  labs(x = "Depth", y = "Top-1 stability") +
  theme_AP()

p_stab_depth
```



```
# Merge ######################################################################

top <- plot_grid(p_time, p_stab, ncol = 2, labels = "auto")
```

```
## Ignoring unknown labels:
## * shape : "Metric"
```

```
bottom <- plot_grid(p_stab_depth, p_stab_edges, ncol = 2, labels = c("c", "d"))
plot_grid(top, bottom, ncol = 1)
```


a
Time (s)
Graph + attributes
Path sampling
Uncertainty analysis
Nº nodes
b
Stability
Top−1 stability
Top−10 mean stability
Top−10 median stability
Nº nodes
c
Top−1 stability
Depth
d
Top−10 med. stability
Nº edges