Supplementary Material
A Multi-Phase Reference Matching Algorithm for Bibliometric Analysis: Design, Implementation, and Evaluation

Massimo Aria[1*], Luca D'Aniello[1] and Maria Spano[1]

[1*]Department of Economics and Statistics, University of Naples Federico II, Via Cinthia 26, C:sso Monte S.Angelo, Naples, 80126, Italy.

*Corresponding author(s). E-mail(s): massimo.aria@unina.it;
Contributing authors: luca.daniello@unina.it; maria.spano@unina.it;

This supplementary material provides the complete R code used for the reference matching workflow, the gold standard construction, the synthetic variant generation, and the evaluation protocol described in the main paper. All code is contained in the accompanying file `supplementary_code.R` and is reproduced below for reference.

S1: Using the Reference Matching Functions

The following example demonstrates the typical workflow for applying reference matching to a bibliographic dataset exported from Scopus.

Listing 1 Reference matching workflow.

```
library(bibliometrix)

# Load bibliographic data from Scopus
M <- convert2df("scopus.csv", dbsource = "scopus", format = "csv")

# Apply reference matching with default settings
results <- applyReferenceMatching(M, threshold = 0.90, method = "jw")

# Inspect results
```

```r
str(results, max.level = 1)
# $full_data     : citation-level results (CR_original, CR_
    canonical, cluster_id)
# $summary       : cluster-level summary (n_variants, format,
    metadata)
# $matched_citations : raw matching output
# $CR_normalized     : reconstructed CR field per document

# View top matched citations
head(results$summary, 10)

# Apply normalized citations back to the dataset
M$CR <- results$CR_normalized$CR
```

Return structure.

The `normalize_citations` function returns a named list with four components:

1. `full_data`: a data frame containing each original citation string, its normalised form, cluster identifier, detected format, extracted metadata fields, and the canonical representative of its cluster.
2. `summary`: aggregate statistics including the number of original citations, the number of unique clusters, the reduction percentage, and the distribution of cluster sizes.
3. `matched_citations`: a data frame listing only those citations that were merged with at least one other citation, together with their cluster assignments.
4. `CR_normalized`: a character vector of the same length as the input, in which each citation string has been replaced by the canonical representative of its cluster. This vector can be directly substituted into the `CR` field of a bibliometrix data frame.

S2: Gold Standard Construction

The gold standard is built from 1,064 Web of Science articles. Each record is converted into a Scopus-format reference string, with the journal name randomly assigned as either the full title or the ISO 4 abbreviation.

Listing 2 Gold standard construction from WoS metadata.

```r
# Load WoS export and extract core metadata fields
wos <- read_xls(file.path(project_dir, "gold_standard", "
    savedrecs.xls"))

gold <- wos %>%
  transmute(
    article_id  = row_number(),
    authors_raw = `Author Full Names`,
    authors_short = Authors,
    title       = `Article Title`,
    source_full = `Source Title`,
    source_iso4 = `Journal ISO Abbreviation`,
```

```
    year = as.character(`Publication Year`),
    volume = as.character(Volume),
    issue  = as.character(Issue),
    page_start = as.character(`Start Page`),
    page_end   = as.character(`End Page`),
    doi = DOI
  ) %>%
  filter(!is.na(title), !is.na(year), !is.na(source_full))

# Convert WoS author names to Scopus style:
# "Aria, Massimo; Cuccurullo, Corrado" -> "Aria M., Cuccurullo C."
format_authors_scopus <- function(authors_full, authors_short) {
  auth_str <- ifelse(!is.na(authors_full), authors_full, authors_short)
  if (is.na(auth_str) || auth_str == "") return(NA_character_)
  authors <- trimws(strsplit(auth_str, ";")[[1]])
  formatted <- sapply(authors, function(a) {
    parts <- trimws(strsplit(a, ",")[[1]])
    if (length(parts) >= 2) {
      surname <- trimws(parts[1])
      given_parts <- strsplit(trimws(parts[2]), "\\s+")[[1]]
      initials <- paste0(substr(given_parts, 1, 1), ".", collapse = "")
      paste0(surname, "␣", initials)
    } else trimws(a)
  }, USE.NAMES = FALSE)
  paste(formatted, collapse = ",␣")
}

# Randomly assign full or ISO4 journal name (~50/50)
gold <- gold %>%
  mutate(
    use_iso4 = sample(c(TRUE, FALSE), n(), replace = TRUE),
    journal_used = ifelse(use_iso4 & !is.na(source_iso4),
                          source_iso4, source_full)
  )

# Build Scopus-format reference string:
# "Authors, Title (Year) Journal, Volume, Issue, Pages"
gold <- gold %>%
  mutate(ref_scopus = paste0(
    authors_scopus,
    ifelse(!is.na(title), paste0(",␣", title, "␣(", year, ")"), ""),
    ifelse(!is.na(journal_used), paste0("␣", journal_used), ""),
    ifelse(!is.na(volume), paste0(",␣", volume), ""),
    ifelse(!is.na(issue), paste0(",␣", issue), ""),
    ifelse(!is.na(pages_str), paste0(",␣", pages_str), "")
  ))
```

```
58
59
60 ## ============================================================
```

S3: Synthetic Variant Generation

Perturbation functions modify only non-blocking fields (title, journal, issue, pages). Blocking keys—first-author surname, year, and journal-name prefix (first 15 characters)—are never altered.

Listing 3 Perturbation functions and example scenario generation.

```
1  # keeping blocking fields from the gold standard.
2
3  build_ref <- function(authors, title, year, journal, volume,
       issue, pages) {
4    parts <- authors
5    has_title <- !is.na(title) & title != ""
6    parts <- ifelse(has_title,
7                    paste0(parts, ", ", title, " (", year, ")"),
8                    paste0(parts, " (", year, ")"))
9    parts <- ifelse(!is.na(journal), paste0(parts, " ", journal),
         parts)
10   parts <- ifelse(!is.na(volume),  paste0(parts, ", ", volume),
         parts)
11   parts <- ifelse(!is.na(issue),   paste0(parts, ", ", issue),
         parts)
12   parts <- ifelse(!is.na(pages),   paste0(parts, ", ", pages),
         parts)
13   parts
14 }
15
16 # --- Typographical perturbation ---
17 # Introduces random substitutions, transpositions, deletions,
18 # or insertions at a specified error rate.
19
20 perturb_typo <- function(s, rate = 0.03) {
21   if (is.na(s) || s == "") return(s)
22   chars <- strsplit(s, "")[[1]]
23   n_errors <- max(1, round(length(chars) * rate))
24   positions <- sample(seq_along(chars), min(n_errors, length(
         chars)))
25   for (pos in positions) {
26     op <- sample(c("substitute","transpose","delete","insert"),
           1,
27                  prob = c(0.4, 0.3, 0.15, 0.15))
28     if (op == "substitute") chars[pos] <- sample(letters, 1)
29     else if (op == "transpose" && pos < length(chars)) {
30       tmp <- chars[pos]; chars[pos] <- chars[pos+1]; chars[pos+1]
             <- tmp
```

```
    } else if (op == "delete") chars[pos] <- ""
    else chars[pos] <- paste0(chars[pos], sample(letters, 1))
  }
  paste(chars, collapse = "")
}

# --- Non-standard journal abbreviation ---
# Strips stop words and truncates remaining words to random
# lengths between 3 and 5 characters.

perturb_journal_nonstandard <- function(journal) {
  sapply(journal, function(j) {
    if (is.na(j)) return(j)
    words <- strsplit(toupper(j), "\\s+")[[1]]
    stops <- c("OF","THE","AND","IN","FOR","ON","A","AN","&")
    words <- words[!words %in% stops]
    abbr <- sapply(words, function(w)
      if (nchar(w) > 4) substr(w, 1, sample(3:5, 1)) else w)
    paste(abbr, collapse = "␣")
  }, USE.NAMES = FALSE)
}

# --- Example scenario generation ---
# Each scenario applies one or more perturbation functions
# and rebuilds the reference string.

# E2: Fuzzy matching test with 3% title typos
scenarios[["E2_fuzzy_3pct"]] <- gold %>%
  rowwise() %>%
  mutate(
    title_p = perturb_typo(title, rate = 0.03),
    ref_perturbed = build_ref(authors_scopus, title_p, year,
                              journal_used, volume, issue, pages_
                                 str)
  ) %>% ungroup() %>%
  mutate(scenario = "E2_fuzzy_3pct", severity = "low",
         perturbation_type = "fuzzy_test")

# D3: Hard combined -- non-standard journal + 5% typos + missing
    metadata
scenarios[["D3_hard_combined"]] <- gold %>%
  rowwise() %>%
  mutate(
    journal_p = perturb_journal_nonstandard(journal_used),
```

S4: Evaluation Protocol

For each scenario, the original gold standard references and their perturbed counterparts are pooled into a single vector. After normalization, the evaluation checks

whether each original–perturbed pair is assigned to the same cluster. Pairwise precision, recall, F_1, and the Adjusted Rand Index are computed over the full clustering.

Listing 4 Evaluation protocol and grid evaluation.

```
         perturbation_type = "combined")


## ============================================================
## Appendix D: Evaluation Protocol
## ============================================================

evaluate_scenario <- function(originals, perturbeds, article_ids,
                              threshold, method = "jw") {
  n <- length(originals)

  # Pool originals and perturbed variants into one vector
  all_refs <- c(originals, perturbeds)
  all_ids  <- c(article_ids, article_ids)
  all_type <- c(rep("original", n), rep("perturbed", n))

  # Run the reference matching algorithm
  matched <- normalize_citations(all_refs, threshold = threshold,
                                 method = method, min_chars = 15)

  # Map each reference back to its article and type
  lookup <- tibble(CR_input = all_refs, article_id = all_ids,
                   type = all_type)
  result <- matched %>%
    select(CR_original, cluster_id) %>% distinct() %>%
    left_join(lookup, by = c("CR_original" = "CR_input"))

  # Check if original and perturbed land in the same cluster
  orig_clusters <- result %>% filter(type == "original") %>%
    select(article_id, cluster_orig = cluster_id)
  pert_clusters <- result %>% filter(type == "perturbed") %>%
    select(article_id, cluster_pert = cluster_id)

  comparison <- orig_clusters %>%
    inner_join(pert_clusters, by = "article_id") %>%
    mutate(matched = cluster_orig == cluster_pert)

  match_rate <- sum(comparison$matched) / nrow(comparison)
```