Gathering Multiple Subject Queries with Large Numbers of Authors

John Muschelli and Clarke Iakovakis

2019-09-17

In `rscopus` we try to use the Scopus API to present queries about authors and affiliations. Here we will use an example from Clarke Iakovakis.

Setup

First, let's load in the packages we'll need.

```
library("rscopus")
library("dplyr")
library("tidyr")
```

Next, we need to see if we have an API key available. See the API key vignette for more information and how to set the keys up. We will use the `have_api_key()` functionality.

```
have_api_key()
#> [1] TRUE
```

Creating a query

Here we will create a query of a specific affiliation, subject area, publication year, and type of access (OA = open access). Let's look at the different types of subject areas:

```
rscopus::subject_areas()
#>  [1] "AGRI" "ARTS" "BIOC" "BUSI" "CENG" "CHEM" "COMP" "DECI" "DENT" "EART"
#> [11] "ECON" "ENER" "ENGI" "ENVI" "HEAL" "IMMU" "MATE" "MATH" "MEDI" "NEUR"
#> [21] "NURS" "PHAR" "PHYS" "PSYC" "SOCI" "VETE" "MULT"
```

These categories are helpful because to search all the documents it'd be too big of a call. We may also get rate limited. We can search each separately, store the information, save them, merge them, and then run our results.

The author of this example was analyzing data from OSU (Oklahoma State University), and uses the affiliation ID from that institution (`60006514`). If you know the institution name, but not the ID, you can use `process_affiliation_name` to retrieve it. Here we make the queries for each subject area:

```
# create the query
Query <- paste0("AF-ID(60006514) AND SUBJAREA(",
                subject_areas(),
                ") AND PUBYEAR = 2018 AND ACCESSTYPE(OA)")
```

Running the query

Let's pull the first subject area information. Note, the count may depend on your API key limits. We also are asking for a complete view, rather than the standard view. The `max_count` is set to 20000 , so this may not be enough for your query and you need to adjust.

```
if (have_api_key()) {
  make_query = function(subj_area) {
    paste0("AF-ID(60006514) AND SUBJAREA(",
           subj_area,
           ") AND PUBYEAR = 2018 AND ACCESSTYPE(OA)")
  }
  i = 3
  subj_area = subject_areas()[i]
  print(subj_area)
  completeArticle <- scopus_search(
    query = make_query(subj_area),
```

```
    view = "COMPLETE",
    count = 200)
  print(names(completeArticle))
  total_results = completeArticle$total_results
  total_results = as.numeric(total_results)
} else {
  total_results = 0
}
#> [1] "BIOC"
#> Warning in scopus_search(query = make_query(subj_area), view =
#> "COMPLETE", : STANDARD view can have a max count of 200 and COMPLETE 25
#> The query list is:
#> list(query = "AF-ID(60006514) AND SUBJAREA(BIOC) AND PUBYEAR = 2018 AND ACCESSTYPE(OA)",
#>     count = 25, start = 0, view = "COMPLETE")
#> $query
#> [1] "AF-ID(60006514) AND SUBJAREA(BIOC) AND PUBYEAR = 2018 AND ACCESSTYPE(OA)"
#>
#> $count
#> [1] 25
#>
#> $start
#> [1] 0
#>
#> $view
#> [1] "COMPLETE"
#>
#> Response [https://api.elsevier.com/content/search/scopus?query=AF-
  ID%2860006514%29%20AND%20SUBJAREA%28BIOC%29%20AND%20PUBYEAR%20%3D%202018%20AND%20ACCESSTYPE%28OA%29&count=25&start=0&vie
#>   Date: 2019-09-17 18:52
#>   Status: 200
#>   Content-Type: application/json;charset=UTF-8
#>   Size: 171 kB
#> Total Entries are 82
#> 4 runs need to be sent with current count
#>
  |
  |                                                                      |   0%
  |
  |================================                                      |  50%
  |
  |======================================================================| 100%
#> Number of Output Entries are 82
#> [1] "entries"       "total_results"  "get_statements"
```

Here we see the total results of the query. This can be useful if the `total_results = 0` or they are greater than the max count specified (not all records in Scopus are returned).

Getting the content

The `gen_entries_to_df` function is an attempt at turning the parsed JSON to something more manageable from the API output. You may want to go over the list elements `get_statements` in the output of `completeArticle`. The original content can be extracted using `httr::content()` and the `"type"` can be specified, such as `"text"` and then `jsonlite::toJSON` can be used explicitly on the JSON output. Alternatively, any arguments to `jsonlite::toJSON` can be passed directly into `httr::content()`, such as `flatten` or `simplifyDataFrame`.

These are all alternative options, but we will use `rscopous::gen_entries_to_df`. The output is a list of `data.frame`s after we pass in the `entries` elements from the list.

```
if (have_api_key()) {

  # areas = subject_areas()[12:13]
  areas = c("ENER", "ENGI")
  names(areas) = areas
  results = purrr::map(
    areas,
    function(subj_area) {
      print(subj_area)
      completeArticle <- scopus_search(
        query = make_query(subj_area),
        view = "COMPLETE",
        count = 200,
        verbose = FALSE)
```

```
    return(completeArticle)
  })
entries = purrr::map(results, function(x) {
  x$entries
})
total_results = purrr::map_dbl(results, function(x) {
  as.numeric(x$total_results)
})
total_results = sum(total_results, na.rm = TRUE)

df = purrr::map(entries, gen_entries_to_df)
MainEntry = purrr::map_df(df, function(x) {
  x$df
}, .id = "subj_area")

ddf = MainEntry %>%
            filter(as.numeric(`author-count.$`) > 99)
if ("message" %in% colnames(ddf)) {
  ddf =  ddf %>%
            select(message, `author-count.$`)
  print(head(ddf))
}

MainEntry = MainEntry %>%
  mutate(
    scopus_id = sub("SCOPUS_ID:", "", `dc:identifier`),
    entry_number = as.numeric(entry_number),
    doi = `prism:doi`)
################################
# remove duplicated entries
################################
MainEntry = MainEntry %>%
  filter(!duplicated(scopus_id))

Authors = purrr::map_df(df, function(x) {
  x$author
}, .id = "subj_area")
Authors$`afid.@_fa` = NULL

Affiliation = purrr::map_df(df, function(x) {
  x$affiliation
}, .id = "subj_area")
Affiliation$`@_fa` = NULL

# keep only these non-duplicated records
MainEntry_id = MainEntry %>%
  select(entry_number, subj_area)
Authors = Authors %>%
  mutate(entry_number = as.numeric(entry_number))

Affiliation = Affiliation %>%
  mutate(entry_number = as.numeric(entry_number))

Authors = left_join(MainEntry_id, Authors)
Affiliation = left_join(MainEntry_id, Affiliation)


# first filter to get only OSU authors
osuauth <- Authors %>%
  filter(`afid.$` == "60006514")

}
#> [1] "ENER"
#> Warning in scopus_search(query = make_query(subj_area), view =
#> "COMPLETE", : STANDARD view can have a max count of 200 and COMPLETE 25
#> [1] "ENGI"
#> Warning in scopus_search(query = make_query(subj_area), view =
#> "COMPLETE", : STANDARD view can have a max count of 200 and COMPLETE 25
#> Joining, by = c("entry_number", "subj_area")
#> Joining, by = c("entry_number", "subj_area")
```

At the end of the day, we have the author-level information for each paper. The `entry_number` will join these `data.frame`s if necessary. The `df` element has the paper-level information in this example, the `author` `data.frame` has author information, including affiliations. There can be multiple affiliations, even within institution, such as multiple department affiliations within an institution affiliation. The `affiliation` information relates to the affiliations and can be merged with the author information.

Funding agencies

Here we look at the funding agencies listed on all the papers. This can show us if there is a pattern in the funding sponsor and the open-access publications. Overall, though, we would like to see the funding of all the papers if a specific funder requires open access. This checking allows libraries and researchers ensure they are following the guidelines of the funding agency.

```
if (total_results > 0) {
  cn = colnames(MainEntry)
  cn[grep("fund", tolower(cn))]

  tail(sort(table(MainEntry$`fund-sponsor`)))

  funderPoland <- filter(
    MainEntry,
    `fund-sponsor` == "Ministerstwo Nauki i Szkolnictwa Wyższego" )
  dim(funderPoland)

  osuFunders <- MainEntry %>%
    group_by(`fund-sponsor`) %>%
    tally() %>%
    arrange(desc(n))
  osuFunders
}
#> # A tibble: 25 x 2
#>    `fund-sponsor`                                    n
#>    <chr>                                         <int>
#>  1 Ministerstwo Nauki i Szkolnictwa Wyższego        22
#>  2 <NA>                                             14
#>  3 National Science Foundation                       4
#>  4 Australian Research Council                       2
#>  5 Natural Science Foundation of Fujian Province     2
#>  6 Oklahoma State University                         2
#>  7 University of Oklahoma Health Sciences Center     2
#>  8 Advanced Scientific Computing Research            1
#>  9 Austrian Science Fund                             1
#> 10 China Scholarship Council                         1
#> # … with 15 more rows
```

Number of author limits

In the Scopus API, if there are > 100 authors on a paper, it only will retrieve the first 100 authors. For those cases, we must use the `abstract_retrieval` to get all the author-level information. Here we make this information into an integer so that we can filter the rows we need to run based on author count.

```
if (total_results > 0) {

  # if there are 100+ authors, you have to use the abstract_retrieval function to get the full author
  data
  # coerce to integer first
  MainEntry <- MainEntry %>%
    mutate(`author-count.$` = as.integer(`author-count.$`))
  run_multi = any(MainEntry$`author-count.$` > 99)
  print(run_multi)
}
#> [1] TRUE
```

In this case, we see there are some articles with author counts > 99 and we must get all author information for those.

More than 100 authors

Now, the `abstract_retrieval` function can take a number of identifiers for papers, such as DOI, PubMed ID, and Scopus ID. Here we will use the Scopus ID, as it is given for all results, but we could also use DOI.

```
if (total_results > 0) {
  if (run_multi) {

    MainEntry_99auth <- MainEntry %>%
      filter(`author-count.$` > 99)

    MainEntry_99auth_id = MainEntry_99auth %>%
      select(entry_number, subj_area)
    auth99 = left_join(MainEntry_99auth_id, Authors)
    affil99 = left_join(MainEntry_99auth_id, Affiliation)


    missing_table = MainEntry %>%
      ungroup() %>%
      mutate_at( vars(scopus_id, doi), .funs = is.na) %>%
      summarize_at(.vars = vars(scopus_id, doi), .funs = sum)
    print(missing_table)
  }
}
#> Joining, by = c("entry_number", "subj_area")
#> Joining, by = c("entry_number", "subj_area")
#>   scopus_id doi
#> 1         0   0
```

Get all the authors

Here we will go through each Scopus ID and get the article information. We will create an affiliation `data.frame` and an author `data.frame`. The non-relevant columns will be deleted, such as `entry_number` since it refers to a different set of elements from a list now. The column names will be harmonized with the `Authors` and `Affiliation` data sets. The respective data is removed from the `Authors` and `Affiliation` data set and joined with the new data with the richer information.

```
if (total_results > 0) {
  # ids = MainEntry_99auth$scopus_id[1:3]
  ids = MainEntry_99auth$scopus_id
  names(ids) = ids
  big_list = purrr::map(
    ids,
    abstract_retrieval,
    identifier = "scopus_id",
    verbose = FALSE)

  all_affil_df = purrr::map_df(
    big_list, function(x) {
      d = gen_entries_to_df(
        x$content$`abstracts-retrieval-response`$affiliation)
      d$df
    }, .id = "scopus_id")

  all_df = purrr::map_df(
    big_list, function(x) {
      d = gen_entries_to_df(
        x$content$`abstracts-retrieval-response`$authors$author)
      d$df
    }, .id = "scopus_id")

  ############################
  # Remove prefix ce: for harmonization
  ############################
  no_ce = function(x) {
    sub("^ce:", "", x)
  }
  all_df = all_df %>%
    rename_all(.funs = no_ce) %>%
```

```
    rename(authid = "@auid",
           `afid.$` = `affiliation.@id`,
           authname = "indexed-name")

  all_df$entry_number = NULL
  all_affil_df$entry_number = NULL

  author_table = all_df %>%
    group_by(scopus_id) %>%
    distinct(authid) %>%
    tally()
  head(author_table)
  stopifnot(all(ids %in% author_table$scopus_id))
  # harmonizing with MainEntry
  author_table = author_table %>%
    rename(`author-count.$` = n)
  MainEntry_99auth$`author-count.$` = NULL
  MainEntry_99auth = left_join(MainEntry_99auth, author_table)


  ########################
  # Harmonized
  ########################
  all_df = MainEntry_99auth %>%
    select(entry_number, subj_area, scopus_id) %>%
    left_join(all_df)
  print(setdiff(colnames(Authors), colnames(all_df)))

  # grab only relevant columns
  all_df = all_df[, colnames(Authors)]

  # remove the old entries
  Authors = anti_join(Authors, MainEntry_99auth_id)
  # put the new data in
  Authors = full_join(Authors, all_df)

  ########################
  # Harmonized
  ########################
  all_affil_df = all_affil_df %>%
    rename(`affiliation-url` = "@href",
           afid = "@id")
  all_affil_df = MainEntry_99auth %>%
    select(entry_number, subj_area, scopus_id) %>%
    left_join(all_affil_df)
  setdiff(colnames(Affiliation), colnames(all_affil_df))

  # remove the old entries
  Affiliation = anti_join(Affiliation, MainEntry_99auth_id)
  # put the new data in
  Affiliation = full_join(Affiliation, all_affil_df)

  MainEntry = anti_join(MainEntry, MainEntry_99auth_id)
  MainEntry = full_join(MainEntry, MainEntry_99auth)
}
#> Joining, by = "scopus_id"
#> Joining, by = "scopus_id"
#> character(0)
#> Joining, by = c("entry_number", "subj_area")
#> Joining, by = c("entry_number", "subj_area", "@_fa", "@seq", "author-url", "authid", "authname",
  "surname", "given-name", "initials", "afid.$")
#> Joining, by = "scopus_id"
#> Joining, by = c("entry_number", "subj_area")
#> Joining, by = c("entry_number", "subj_area", "affiliation-url", "afid", "affilname", "affiliation-
  city", "affiliation-country")
#> Joining, by = c("subj_area", "entry_number")
#> Joining, by = c("subj_area", "@_fa", "prism:url", "dc:identifier", "eid", "dc:title", "dc:creator",
  "prism:publicationName", "prism:issn", "prism:eIssn", "prism:volume", "prism:issueIdentifier",
  "prism:pageRange", "prism:coverDate", "prism:coverDisplayDate", "prism:doi", "dc:description",
  "citedby-count", "prism:aggregationType", "subtype", "subtypeDescription", "author-count.@limit",
  "author-count.@total", "author-count.$", "authkeywords", "source-id", "fund-no", "openaccess",
  "openaccessFlag", "entry_number", "article-number", "fund-sponsor", "pii", "fund-acr", "pubmed-id",
  "scopus_id", "doi")
```

Conclusion

The Scopus API has limits for different searches and calls. Using a combination of APIs, we can gather all the information on authors that we would like. This gives us a full picture of the authors and co-authorship at a specific institution in specific scenarios, such as the open access publications from 2018.