



4 ▼

May 23, 2022

# Coverage of DOAJ journals' citations through OpenCitations - Protocol V.4

Constance Dami<sup>1</sup>, Alessandro Bertozzi<sup>1</sup>, Chiara Manca<sup>1</sup>, Umut Kucuk<sup>1</sup><sup>1</sup>University of Bologna

1

[dx.doi.org/10.17504/protocols.io.n92ldz598v5b/v4](https://dx.doi.org/10.17504/protocols.io.n92ldz598v5b/v4)

Open Science 2021/2022

Chiara Manca

This protocol refers to a research done for the Open Science course 21/22 of the University of Bologna.

This is the protocol for the research of the coverage of DOAJ journals' citations through OpenCitations.

Our goal is to find out:

- about the coverage of articles from open access journals in DOAJ journals as citing and cited articles,
- how many citations do DOAJ journals receive and do, and how many of these citations involve open access articles as both citing and cited entities,
- as well as the presence of trends over time of the availability of citations involving articles published in open access journals in DOAJ journals.

Our research focuses on DOAJ journals exclusively, using OpenCitations as a tool. Previous research has been made on open citations using COCI (Heibi, Peroni & Shotton 2019), and on DOAJ journals' citations (Saadat and Shabani 2012), paving the grounds for our present analysis.

After careful considerations on the best way to retrieve data from DOAJ and OpenCitations, we opted for downloading the public data dumps. Using the API resulted in a way too long running time, and the same problem arose for using the SPARQL endpoint of OpenCitations.

## Minimal Bibliography

Björk, B.-C.; Kanto-Karvonen, S.; Harviainen, J.T. "How Frequently Are Articles in Predatory Open Access Journals Cited." *Publications*, 8, 17. (2020)

<https://doi.org/10.3390/publications8020017>

Heibi, I.; Peroni, S.; Shotton, D. "Crowdsourcing open citations with CROCI – An analysis of the current status of open citations, and a proposal" arXiv:1902.02534 (2019) <https://doi.org/10.48550/arXiv.1902.02534>

Pandita, R., & Singh, S. "A Study of Distribution and Growth of Open Access Research Journals Across the World. Publishing Research Quarterly" (2022), 38(1), 131–149. <https://doi.org/10.1007/s12109-022-09860-x>

Saadat, R., A. Shabani. "Investigating the citations received by journals of Directory of Open Access Journals from ISI Web of Science's articles." *International Journal of Information Science and Management (IJISM)* 9.1 (2012): 57-74.

Solomon, D. J., Laakso, M., Björk, B.-C. "A longitudinal comparison of citation rates and growth among open access journals", *Journal of Informetrics*, 7, 3 (2013): 642-650. <https://doi.org/10.1016/j.joi.2013.03.008>.

DOI

[dx.doi.org/10.17504/protocols.io.n92ldz598v5b/v4](https://dx.doi.org/10.17504/protocols.io.n92ldz598v5b/v4)

Constance Dami, Alessandro Bertozzi, Chiara Manca, Umut Kucuk 2022.  
Coverage of DOAJ journals' citations through OpenCitations - Protocol.

**protocols.io**

<https://dx.doi.org/10.17504/protocols.io.n92ldz598v5b/v4>

Constance Dami



citations, OpenCitations, DOAJ, open access, journals, open science

protocol ,

May 18, 2022

May 23, 2022

May 23,  
2022

Chiara Manca

62797

This protocol uses the following Python libraries: tarfile, pandas, json, pickle, datetime, zipfile and plotly.

The GitHub repository for the software of our research, including all python code mentioned in the protocol is available [here](#).

:

This protocol refers to a research done for the Open Science course 21/22 of the University of Bologna.

Make sure to have Python 3.9 installed on your device.

All the dependencies of the script can be installed using the requirements.txt file stored into the github repository.

Computer technical specifications:

CPU: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz

RAM: 20,0 GB (19,9 GB usable) 2666 mhz



The time specified for each step is related to the technical specifications of the computer used.

Data Gathering: DOAJ

20m

- 1 Collecting data from DOAJ: we download data about journals and articles from the DOAJ website, and then refine it excluding all information that we are not interested in.

The following outputs are expected from this step:



**doi.json**, containing a dictionary of all the journals as key and a few information including the list of all the DOIs of the articles published in this journal as value.

**articles\_without\_dois.json**, containing all the articles excluded from our research, due to the lack of DOI provided

**dois\_articles\_journals.pickle**, a dictionary with every DOI of the articles as key and the unique identifier for the journal publishing it as value.

### 1.1 Firstly we download the data dumps from DOAJ in .tar.gz. format.

DOAJ articles public data dump

DOAJ journals public data dump

Both datasets contain metadata that is not useful for our research, so we need to filter only the necessary data.

### 1.2 From the **DOAJ dump** we create **doi.json** by running **create\_json\_dois.py**. Inside this JSON file we create a unique key for each journal by concatenating the issn and the eissn, having as values: the issn (if it is present), eissn (if it is present), the title of the journal, the list of all the articles' DOIs.

Firstly, we open the tarfile containing the data:

```
journals=set()  
  
# Extracting data from the DOAJ journals dump (May 7th,  
2022)  
tar_journals =  
tarfile.open("./data/imported/doaj_journal_data_2022-05-  
07.tar.gz", "r:gz")  
    for tarinfo in tar_journals:  
  
        file = tar_journals.extractfile(tarinfo)  
        # Extracting the data in json format  
        p = json.load(file)
```

For every journal, we extract only the information about **issn** and **eissn**, verifying first that there is always at least one of the two for each record in the dump:

```

for journal in p:
    try:
        if journal["bibjson"]["pissn"]:
            journal_issn = journal["bibjson"]["pissn"]
    except KeyError:
        journal_issn=""
    try:
        if journal["bibjson"]["eissn"]:
            journal_eissn = journal["bibjson"]["eissn"]
    except KeyError:
        journal_eissn=""

```

We then create add to the set of journals our unique identifier "**issn+eissn**"

```

key_dict = f"{journal_issn}{journal_eissn}"
journals.add(key_dict)

```

- 1.3 The extraction of data for the articles is the same as above, opening the file with tarfile, then for each article, we collect the information about **issn**, **eissn** of the journal publishing it, as well as the **DOI** of the article:

```

for article in p:
    for el in article["bibjson"]["identifier"]:
        if el["type"] == "pissn":
            journal_issn = el["id"]
        if el["type"] == "eissn":
            journal_eissn = el["id"]
        if el["type"] == "doi" or el["type"] == "DOI":
            try:
                art_doi = el["id"]
            except KeyError:
                art_doi = ""

```

If the article doesn't have any DOI registered, we add it to a list that we will store separately.

Otherwise we handle cases where the issn and eissn have been wrongly registered in the articles dump by aligning data with the journals set previously created.

```

if art_doi=="":
    art_without_doi.append(article)
else:
    # Collecting the title of the journal
    journal_title=article["bibjson"]["journal"]["title"]

```

```

key_dict = f"{journal_issn}{journal_eissn}"

# Handling cases where the issn and/or eissn from
the articles dump don't match the journals dump
if key_dict not in journals:

    # Aligning with the journals metadata if there
    is only the issn registered
    if journal_issn in journals:
        key_dict = journal_issn
    # Aligning with the journals metadata if there
    is only the eissn registered
    elif journal_eissn in journals:
        key_dict = journal_eissn
    else:
        for issn in journals:
            if journal_issn != "" and
journal_issn in issn:
                key_dict = issn
                break
            elif journal_eissn != "" and
journal_eissn in issn:
                key_dict = issn
                break

```

Once all of the information are collected, we add them to our final json, adding a new key if it doesn't exist or adding it to the list of dois for the journal

```

if key_dict in doi_json:
    doi_json[key_dict]["dois"].append(art_doi)
else:
    doi_json[key_dict]={"title":journal_title,
"pissn":journal_issn, "eissn":journal_eissn, "dois":
[art_doi]}

```

We save the dictionary completed in **doi.json** and the list of articles without dois in a separate file **articles\_without\_dois.json**

An example of an element in the **doi.json**:

```

{
  "1779-627X1779-6288": {
    "title": "International Journal for Simulation and Multidisciplinary
Design    Optimization",

```

```

        "pissn": "1779-627X",
        "eissn": "1779-6288",
        "dois": [
            "10.1051/ijsmdo:2008025",
            "10.1051/smdo/2019012",
            "10.1051/smdo/2020004",
            "10.1051/smdo/2020001",
            "10.1051/smdo/2016003",
            ...
        ]
    },
    ...
}

```

- 1.4 We also create another file **dois\_articles\_journals.pickle** containing a dictionary with all DOAJ articles' DOIs from DOAJ as keys and the "issn+eissn" identifier of the journal who published it as value, which has just the aim to simplify the next steps.

```

all_dois={}
with open(".\data\queried\DOAJ\doi.json", 'r',
encoding="utf-8") as json_file:
    p = json.load(json_file)
    for i in p:
        for doi in p[i]["dois"]:
            all_dois[doi]=i

with
open('./data/queried/DOAJ/doi_articles_journals.pickle',
'wb') as pickle_file:
    pickle.dump(all_dois, pickle_file,
protocol=pickle.HIGHEST_PROTOCOL)

```

Data Gathering: OpenCitations

2h

- 2 **Collecting and filtering data from OpenCitations:** we take the data from the download section, on the OpenCitations website, and then refine them using the files obtained from the previous step.

These are our expected results from this section:



A **general\_dict\_count.json**, containing all journals with the number of citing entites, cited entites, open\_cited entites (inside DOAJ) and open\_citing entites (inside DOAJ).

An **open\_cit repository with several JSON files**, which are obtained by the filter operation on the CSV extract from zip repository, dumped form OpenCitations.

- 2.1 We unzip the first level of files and obtain a series of zip repositories. In this way, we avoid extending the required memory for storing files.

COCI public data dump (March 2022)

Then we iterate over all the files inside the zip directory, following these nested steps:

1. Collect all zip files in a list
2. Iterate all over zip files
3. iterate all over CSVs into zip file

Moreover we initialize an empty dictionary (**general\_count\_dict**), which will be populated in the next steps.

```
list_zip_file = [f for f in
listdir(input_path_zip_file)]
general_count_dict = dict()

for zip_file in list_zip_file:
    with zipfile.ZipFile(f"
{input_path_zip_file}/{zip_file}", "r") as zfile:

        for name_file in zfile.namelist():
```

- 2.2 We decode the file obtaining a file in CSV format.

```
zfiledata = BytesIO(zfile.read(name_file))
```

Using the [Python library Pandas](#), we transform the CSV file into a dataframe.



```
df = pd.read_csv(zfiledata)
```

2.3 We create a dictionary with several filtered versions of the same CSV file, with a [dictionary comprehension](#) and the pickle file obtained from the previous step about the DOAJ data (**dois\_articles\_journals.pickle**):

1. A **cited** version with only records which have as value in the column *cited* a DOI from DOAJ.
2. A **citing** version with only records which have as value in the column *citing* a DOI from DOAJ.
3. An **open\_cit** version with only records which have as value in both columns, *citing* and *cited*, a DOI from DOAJ.

```
my_dfs = {"cited":
df[df["cited"].isin(data_json.keys())],
        "citing":
df[df["citing"].isin(data_json.keys())],
        "open_cit":
df[(df["cited"].isin(data_json.keys())) &
(df["citing"].isin(data_json.keys()))]}
```



```
my_dfs = {'cited':                                oci ... author_sc

169  0200100000736090708630463040301630505060909630...
...   no
449  0200100000736090708630801630302026302060302630...
...   no
453  0200100000736090708630801630302026302060302630...
...   no
547  0200100000736090708630801630302026302060701630...
...   no
673  0200100000736090708630801630302026302070301630...
...   no

[64443 rows x 7 columns],

'citing':                                          oci ... author_sc

23640 020010000073621272763020001056302-020010000073...
...   no
59993 0200100000736280102010802630001076300020101630...
```

```

...    no
59994 0200100000736280102010802630001076300020101630...
...    no
59995 0200100000736280102010802630001076300020103633...
...    no
59996 0200100000736280102010802630001076300020104633...
...    no

```

[13577 rows x 7 columns],

```

'open_cit':          oci ... author_sc

```

```

59999 0200100000736280102010802630001086300020200630...
...    no
64857 0200100000736280103020001630001086300070204630...
...    no
64859 0200100000736280103020001630001086300070305630...
...    no
64906 0200100000736280103020002630001086300050207630...
...    no
67115 0200100000736280400000903630001076300010902630...
...    no

```

[768 rows x 7 columns]}

2.4 Then we take the previous initialized **general\_count\_dict** and we iterate all over these selected dataframes.

```

for name_df, df in my_dfs.items():

```

For each dataframe we follow these steps:

1. We create another column with the id of the journal defined inside the file pickle.
2. We group\_by the column previous selected for filtering operation:

```

if name_df == 'cited':
    df["journal"] = df["cited"].apply(lambda x:
data_json[x])
    df_count = df.groupby(["journal"], as_index=False)
["cited"].count()
    general_count_dict =
counter_function(df_count.to_dict(orient="records"),
general_count_dict, "cited")

```

```

elif name_df == 'citing':
    df["journal"] = df["citing"].apply(lambda x:
data_json[x])
    df_count = df.groupby(["journal"], as_index=False)
["citing"].count()
    general_count_dict =
counter_function(df_count.to_dict(orient="records"),
general_count_dict, "citing")

elif name_df == "open_cit":
    copy_df = df.copy()
    df["journal"] = df["cited"].apply(lambda x:
data_json[x])
    df = df.rename(columns={'journal': 'journal',
'cited': 'open_cited', 'citing': 'citing'})
    df_count = df.groupby(["journal"], as_index=False)
["open_cited"].count()
    general_count_dict =
counter_function(df_count.to_dict(orient="records"),
general_count_dict, "open_cited")

    copy_df["journal"] = copy_df["citing"].apply(lambda
x: data_json[x])
    copy_df = copy_df.rename(columns={'journal':
'journal', 'cited': 'cited', 'citing': 'open_citing'})
    df_count = copy_df.groupby(["journal"],
as_index=False)["open_citing"].count()
    general_count_dict =
counter_function(df_count.to_dict(orient="records"),
general_count_dict, "open_citing")

```

3. finally with another function (**counter\_function**) we add all the records to the **general\_count\_dict**, defined before:

```

def counter_function(group_by_dict, json_file, name):
    for record in group_by_dict:

        if record["journal"] not in json_file:
            json_file[record["journal"]] = dict()
            json_file[record["journal"]]["cited"] = 0
            json_file[record["journal"]]["citing"] = 0
            json_file[record["journal"]]["open_cited"] =
0
            json_file[record["journal"]]["open_citing"]
= 0

```

```

        json_file[record["journal"]][name] +=
record[name]

    return json_file

```

- 2.5 At each iteration on files from the zipped repository, after the open\_cit dataframe, we save the open\_cit version of the CSV as a JSON file inside the data/queried/OC/open\_cit repository. These will be used for the plotting part.

```

result = df.to_dict(orient="records")
json_object = json.dumps(result)
with open(f"
{output_file_path_open_cit_json}/{name_df}/{name_file.re
place('.csv','')}
.replace(':', '_').json", "w") as outfile:
    outfile.write(json_object)
    outfile.close()

```




```

[
  {
    "oci": "02001000007362801020108026300010863000202006307-020010001063619372514292122370200010737000337000004",
    "citing": "10.1007/s12182-018-0220-7", "open_cited": "10.1016/j.petlm.2017.03.004", "creation": "2018-03-27", "timespan": "P0Y3M", "journal_sc": "no", "author_sc": "no", "journal": "2405-65612405-5816"},
    {
      "oci": "02001000007362801030200016300010863000702046308-020010509003600010000630209040563020701360104", "citing": "10.1007/s13201-018-0724-8", "open_cited": "10.1590/0100-2945-271/14", "creation": "2018-06", "timespan": "P2Y4M", "journal_sc": "no", "author_sc": "no", "journal": "0100-29451806-9967"},
      ...
    ]

```

Moreover, at each iteration, we save the updated version of the general\_count\_dict inside the data/queried repository into a JSON file format.

```
with open(f"
{output_file_path_general_dict}/general_count_dict_prova
.json","w") as outfile:
    json_object = json.dumps(general_count_dict)
    outfile.write(json_object)
    outfile.close
```



```
{
  '0001-37651678-2690': {
    'cited': 6,
    'citing': 0,
    'open_cited': 0,
    'open_citing': 0},
  '0001-69772083-9480': {
    'cited': 12,
    'citing': 0,
    'open_cited': 0,
    'open_citing': 0},
  '0003-94382363-9822': {
    ...
  },
  ...
}
```

Data Gathering: merge

- 3 **Merging the data from DOAJ and from OC:** we add to the **doi.json** file, obtained from the first step, the information obtained from step 2 stored in the **general\_count\_dict**. In this dictionary the information is divided in the same way as the **doi.json** file: by journals.



**doi\_with\_count.json**, containing all the information obtained from merging the two files, created into the two previous steps (**doi.json**, **general\_count\_dict**).

```
def final_merge(all_doi, count):
    new_dict = dict()
    list_dict = [all_doi, count]


    # Iterate over the two dict in input
    for listed_dict in list_dict:
        for journal_id, value in listed_dict.items():

            # if the key of the journal don't exist, add the key
            # with as value a dict
            if journal_id not in new_dict:
                new_dict[journal_id] = dict()

            # if the key of the journal exist or after
            # creating the new key, add inside this latter key
            # a new key of the other dict with the correspondent
            # value
            for name_field, value_field in value.items():
                new_dict[journal_id][name_field] = value_field
    return new_dict
```

We save the output as a JSON file (**doi\_with\_count.json**)

```
with open('data/doi_with_count.json', 'w', encoding='utf8') as
    json_file:
        json.dump(data, json_file, ensure_ascii=False)
```



```
doi_with_count = {
  "2674-46002674-4619":
    {
      "title": "TalTech Journal of European Studies",
      "pissn": "",
      "eissn": "2674-4619",
      "dois": ["10.2478/bjes-2021-0001", "10.2478/bjes-2021-0005", "10.2478/bjes-2021-0006", "10.2478/bjes-2021-0009", "10.2478/bjes-2021-0015", "10.2478/bjes-2021-0017", "10.2478/bjes-2021-0020", "10.2478/bjes-2021-0004", "10.2478/bjes-2021-0011", "10.2478/bjes-2021-0012", "10.2478/bjes-2021-0018", "10.2478/bjes-2021-0003", "10.2478/bjes-2021-0007", "10.2478/bjes-2021-0010", "10.2478/bjes-2021-0013", "10.2478/bjes-2021-0014", "10.2478/bjes-2021-0016", "10.2478/bjes-2021-0002", "10.2478/bjes-2021-0008", "10.2478/bjes-2021-0019"],
      "cited": 5,
      "citing": 301,
      "open_cit": 3
    },
}
```

```
...  
}
```

#### Data Gathering: grouping by year

- 4 We grouped and counted the collection of citations coming and going to DOAJ journals (for simplicity we called them *open* citations) by **years**.

```
# grouping by year and then counting  
df['citation_count'] = df.groupby('year')  
['year'].transform('count')  
df = df[['citation_count',  
        'year']].reset_index(drop=True).convert_dtypes().drop_duplicates()  
  
# dropping nans and sorting by year  
df =  
df.dropna().astype({"citation_count": "int", "year": "int"}).reset_index(drop=True)  
df = df.sort_values(by=['year']).reset_index(drop=True)  
  
# getting just the rows without errors in the dates  
df = df[df['year'] <= 2100].dropna().reset_index(drop=True)
```

The result was **open\_cit\_in\_years.json** of all of these citations for every year that was found.



```
[{"citation_count": 4, "year": 1954}, {"citation_count": 7, "year": 1955},  
{"citation_count": 5, "year": 1956}, {"citation_count": 9, "year": 1957},  
{"citation_count": 14, "year": 1958}, {"citation_count": 7, "year": 1959},  
{"citation_count": 18, "year": 1960}, {"citation_count": 13, "year": 1961},  
{"citation_count": 31, "year": 1962}, {"citation_count": 24, "year": 1963},  
{"citation_count": 39, "year": 1964}, {"citation_count": 26, "year": 1965},  
{"citation_count": 3, "year": 1966}, {"citation_count": 27, "year": 1967},  
{"citation_count": 25, "year": 1968},  
...]
```

We also encountered some citations with either a *NaN* creation date or a wrong creation date (ex. "2109"). We collected them in a **open\_cit\_w\_date\_err.json** with this code:

```
df_errors = df[(df['year'] > 2024) | (df['year'].isnull())]
df_errors.to_json('open_cit_w_date_err.json', orient="records")
```



```
[
{"oci":"0200100010636193733252724370200020037010000000207-
0200100030836280401040607630001086300060508046305","citing":"10.1016V/j.x
pro.2020.100027","cited":"10.1038Vs41467-018-06584-
5","creation":null,"timespan":null,"journal_sc":"no","author_sc":"no","year":null},
{"oci":"0200100010636193733252724370200020037010000000207-
02001010806360107040963080100046306630300","citing":"10.1016V/j.xpro.2020.1
00027","cited":"10.1186V1749-8104-6-
30","creation":null,"timespan":null,"journal_sc":"no","author_sc":"no","year":null},
...]
```

## Data Visualization

- 5 We visualize our results in *bar* and *scatter* graphs with the use of the [plotly Python library](#). We compute graphs for the top 10 most citing and cited articles in number of citations, normal or involving open access journals as both citing and cited entities.

We also plot the number of these *open* citations by **year**, thanks to the data collected in **open\_cit\_in\_years.json** at step 4.

## Publishing data

- 6 We publish the following JSON and CSV files in Zenodo and also in our Github repository (**data** folder).