Protocol: "Investigating DOIs classes of errors" V.2 V.3

Ricarda Boente[1], Deniz Tural[1], Cristian Santini[1], Arcangelo Massari[1]

[1]University of Bologna

Version 3

Apr 27, 2021

In Development dx.doi.org/10.17504/protocols.io.bufwntpe

Open Science 2020/2021

Arcangelo Massari

ABSTRACT

The purpose of this protocol is to provide an automated process to repair invalid DOIs that have been collected by the OpenCitations Index Of Crossref Open DOI-To-DOI References (COCI) while processing data provided by Crossref.

The data needed for this work is provided by COCI as a CSV containing pairs of valid citing DOIs and invalid cited DOIs. With the goal to determine an automated process, we first classified the errors that characterize the wrong DOIs in the list. Our starting hypothesis is that there are two main classes of errors: factual errors, such as wrong characters, and DOIs that are not yet valid at the time of processing. The first class can be furtherly divided into three classes: errors due to irrelevant strings added to the beginning (prefix-type errors) or at the end (suffix-type errors) of the correct DOI, and errors due to unwanted characters in the middle (other-type errors). Once the classes of errors are addressed, we propose automatic processes to obtain correct DOIs from wrong ones. These processes involve the use of the information returned from DOI API, as well as rule-based methods, including regular expressions. to correct invalid DOIs.

The application of this methodology produces a CSV dataset containing all the pairs of citing and cited DOIs in the original dataset, each one enriched by 5 fields: "Already_Valid", which tells if the cited DOI was already valid before cleaning, "New_DOI", which contain a clean, valid DOI (if our procedure was able to produce one), and "prefix_error", "suffix_error" and "other-type_error" fields, which contain, for each cleaned DOI the number of errors that were cleaned.

DOI

dx.doi.org/10.17504/protocols.io.bufwntpe

PROTOCOL CITATION

Ricarda Boente, Deniz Tural, Cristian Santini, Arcangelo Massari 2021. Protocol: "Investigating DOIs classes of errors" V.2. **protocols.io**
https://dx.doi.org/10.17504/protocols.io.bufwntpe
Version created by Cristian Santini

WHAT'S NEW

We added the source code of our software, as well as more procedural and contextual information related to the project and links to external resources.

KEYWORDS

Open Access, I4OC, OpenCitations, citations, DOI, Crossref, COCI, data quality

CREATED

Apr 23, 2021

LAST MODIFIED

Apr 27, 2021

OWNERSHIP HISTORY

Apr 27, 2021 Arcangelo Massari

PROTOCOL INTEGER ID

49366

MATERIALS TEXT

COCI, the OpenCitations Index of Crossref open DOI-to-DOI citations (https://w3id.org/oc/index/coci), is a collection of open citations in which citations are exposed as first-class data entities with accompanying properties. All the data available in COCI are derived from those accessible through Crossref (https://www.crossref.org/). Since Crossref does not double-check their data, many of the DOIs that they publicly provide as citations data are not valid. The data upon which our project is build is provided by COCI as result of a procedure of collection of errors and mistakes of DOIs available on Crossref and is publicly available at http://doi.org/10.5281/zenodo.4625300.

DISCLAIMER:

> This research has been realized during the University of Bologna course in Open Science held by Professor Silvio Peroni in the y.a. 2020-2021.

BEFORE STARTING

This methodology takes for granted some basic knowledge regarding the scholarly publishing domain, the nature of identifiers (e.g. DOI), the open citation movement and Python (variable, classes, methods and libraries).
Before starting, you need to make sure you have Python3.x installed on your computer, in addition, in order to correctly execute the Python-based scripts indicated in the methodology, you must install the required libraries defined in requirements.txt. Please follow the official Python guidelines at https://wiki.python.org/moin/BeginnersGuide/ to check and eventually install python and the required libraries locally on your machine.
In the rest of this protocol we will use some notation to describe the data and the processes involved in our protocol:
<var>: variables are mentioned inside angle brackets and are written in bold.
function() is a block of code to perform a specific operation. They also occur in bold.
class Object(): describe a class, a specific object which includes different functions as its methods. They occur in bold and are preceded by "class" prefix.

DATA IMPORT

1 We import a CSV containing invalid DOI-to-DOI citations. The CSV is provided on public license by Peroni, Silvio. (2021). Citations to invalid DOI-identified entities obtained from processing DOI-to-DOI citations to add in COCI (Version 1.0) [Data set]. *Zenodo*. http://doi.org/10.5281/zenodo.4625300.

> Citations to invalid DOIs obtained from Crossref

The CSV is organized as follows: the first column contains valid citing DOIs and the second contains invalid cited DOIs.

Valid_citing_DOI	Invalid_Cited_DOI
10.14778/1920841.1920954	10.5555/646836.708343
10.5406/ethnomusicology.59.2.0202	10.2307/20184517
10.1161/01.cir.63.6.1391	10.1161/circ.37.4.509

ERROR ANALYSIS

2 Taking as a reference the DOI error's taxonomy proposed in "Errors in DOI indexing by bibliometric databases" (Franceschini et al. 2015), there are two main classes of errors: author errors, made by authors when creating the list of cited articles for their publication, and database mapping errors, related to a data-entry error. This protocol deals only with the second kind of error, which can be further divided between prefix errors, suffix errors, and other-type errors, as proposed in "DOI errors and possible solutions for Web of Science" (Xu et al., 2019). In order to solve our problem, we manually isolated from a subset of 100 DOIs recurrent strings at the beginning and at the end with corrupted DOI prefix and suffix respectively. In addition, we found other types of errors, like double underscores, double periods, XML tags, spaces, and forward slashes that could be removed at the end of the data cleaning process.

Citation: Ricarda Boente, Deniz Tural, Cristian Santini, Arcangelo Massari (04/27/2021). Protocol: "Investigating DOIs classes of errors" V.2. https://dx.doi.org/10.17504/protocols.io.bufwntpe

2.1 For cleaning **prefix-type errors**, we took a modified version of the regular expression from (Xu et al., 2019) in order to remove unwanted characters at the beginning of a DOI. The regular expression for matching prefix-type errors is the following:

```
"(.*?)(?:\.)?(?:HTTP:\/\/DX\.D[0|O]I\.[0|O]RG\/|HTTPS:\/\/D[0|O]I\.[0|O]RG\/)(.*)"
```

2.2 For capturing **suffix-type errors**, we enriched the approach provided by (Xu et al., 2019) and we devised a set of 17 alternative suffix regular expressions. The regular expression are the following:

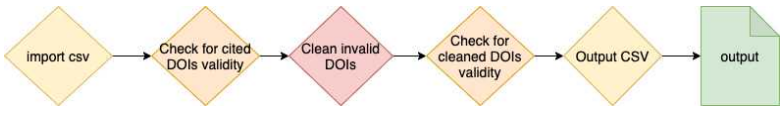
```
1) "(.*?)(?:\/-\/DCSUPPLEMENTAL)$"
2) "(.*?)(?:SUPPINF[0|O](\.)?)$"
3) "(.*?)(?:[\.|\(|,|;]?PMID:\d+.*?)$"
4) "(.*?)(?:[\.|\(|,|;]?PMCID:PMC\d+.*?)$"
5) "(.*?)(?:[\(|\[]EPUBAHEADOFPRINT[\)\]])$"
6) "(.*?)(?:[\.|\(|,|;]?ARTICLEPUBLISHEDONLINE.*?\d{4})$"
7) "(.*?)(?:[\.|\(|,|;]*HTTP:\/\/.*?)$"
8) "(.*?)(?:\/(META|ABSTRACT|FULL|EPDF|PDF|SUMMARY)([>|\)](LAST)?ACCESSED\d+)?)$"
9) "(.*?)(?:[>|)](LAST)?ACCESSED\d+)$"
10) "(.*?)(?:[\.|(|,|;]?[A-Z]*\.?SAGEPUB.*)$?"
11) "(.*?)(?:\.{5}.*?)$"
12) "(.*?)(?:[\.|,|<|&|(|;])$"
13) "(.*?)(?:[\.|(|,|;]10.\d{4}\/.*?)$"
14) "(.*?)(?:\[DOI\].*?)$"
15) "(.*?)(?:\(\d{4}\)?)$"
16) "(.*?)(?:\?.*?=.*?)$"
17) "(.*?)(?:#.*?)$"
```

All the suffix regular expressions can be combined in a single regular expression by using alternation within a non-capturing group. This is how the regular expression for suffix-type errors looks like:

```
"(.*?)(?:\/-\/DCSUPPLEMENTAL|SUPPINF[0|O](\.)?|[\.|\(|,|;]?PMID:\d+.*?|[\.|\(|,|;]?
PMCID:PMC\d+.*?|[\(|\[]EPUBAHEADOFPRINT[\)\]]|[\.|\(|,|;]?ARTICLEPUBLISHEDONLINE.*?\d{4}|
[\.|\(|,|;]*HTTP:\/\/.*?|\/(META|ABSTRACT|FULL|EPDF|PDF|SUMMARY)([>|\)](LAST)?
ACCESSED\d+)?|[>|\)](LAST)?ACCESSED\d+|[\.|\(|,|;]?[A-Z]*\.?SAGEPUB.*?|\.{5}.*?|[\.|,|<|&|\(|;]+|
[\.|\(|,|;]10.\d{4}\/.*?|\[DOI\].*?|\(\d{4}\)?|\?.*?=.*?|#.*?)$"
```

DATA CLEANING PROCEDURE

3 Here we describe the steps through which we carried out the cleaning procedure of factually invalid DOIs.
The workflow of our procedure is organized as follows:



Workflow of our data cleaning procedure

We created a Python software for handling the whole procedure. The source code is publicly available on ISC license at http://doi.org/10.5281/zenodo.4723984.
For our data cleaning system, we devised two class objects: one called **class Clean_DOIs()**, with all the methods required to carry our cleaning/validation procedure, and **class Support()**, with all the methods required to process and dump CSV as well as carry HTTP requests. In this protocol, we describe the use of the different methods that can be imported from our classes. If you want to reproduce the procedure described in the next step, once you have verified to have Python 3.x installed and the required libraries, you can clone our Github repository and launch the *tutorial.py* file in a terminal inside the folder or you can execute the commands listed in the following sub-sections.

Citation: Ricarda Boente, Deniz Tural, Cristian Santini, Arcangelo Massari (04/27/2021). Protocol: "Investigating DOIs classes of errors" V.2. https://dx.doi.org/10.17504/protocols.io.bufwntpe

Launch tutorial.py

python tutorial.py

3.1 First, with the function **process_csv_input()** from our **class Support()** we import our data in CSV format and we store it in a *DictReader*, an object that maps the information in each row to a dictionary whose keys are given by the optional fieldnames parameter.

```
def process_csv_input(path:str) -> list:
        print(f"[Support:INFO Proccessing csv at path {path}]")
        with open(path, 'r', encoding='utf-8') as csvfile:
            reader = csv.DictReader(csvfile)
            return list(reader)
```

On our procedure, we perform this method on our input csv and we store it as a **<data>** variable. You can reproduce the experiment by launching:

```
python3
>>> import csv
>>> from support import Support
>>> from clean_dois import Clean_DOIs
>>> data = Support().process_csv_input(path="./dataset/invalid_dois.csv")
```

After this procedure, our **<data>** variable will consist of a list of dictionaries as shown below:

[{"Valid_citing_DOI": "10.14778/1920841.1920954", "Invalid_cited_DOI": "10.5555/646836.708343"}, {"Valid_citing_DOI": "10.5406/ethnomusicology.59.2.0202", "Invalid_cited_DOI": "10.2307/20184517"},
...]

3.2 Our procedure first checks for each cited DOI if it is factually invalid or it has become valid in the meanwhile. For this purpose, we use the function **check_dois_validity()**, from our **class Clean_DOIs()**, which takes in input a list of dictionaries like the one described above and does a GET request to the DOI Proxy with the cited DOI (https://www.doi.org/factsheets/DOIProxy.html) of each row of the CSV: if the status code corresponding to that specific DOI is different from 1, it means that it is not valid; otherwise, that DOI has become valid in the meanwhile. The function outputs an enriched version of the list of dictionaries in input, with two additional fields: *"Valid_DOI"*, which stores a validated DOI, and *"Already_valid"*, which stores 1 if it has become valid in the meanwhile, 0 otherwise.

```
def check_dois_validity(self, data:list) -> list:
        checked_dois = list()
        pbar = tqdm(total=len(data))
        for row in data:
            invalid_dictionary = {
                "Valid_citing_DOI": row["Valid_citing_DOI"],
                "Invalid_cited_DOI": row["Invalid_cited_DOI"],
                "Valid_DOI": "",
                "Already_valid": 0
            }
            handle =
Support().handle_request(url=f"https://doi.org/api/handles/{row['Invalid_cited_
DOI']}", cache_path=self.cache_path, error_log_dict=self.logs)
            if handle is not None:
                if handle["responseCode"] == 1:
                    checked_dois.append(
                        {"Valid_citing_DOI": row["Valid_citing_DOI"],
```


Citation: Ricarda Boente, Deniz Tural, Cristian Santini, Arcangelo Massari (04/27/2021). Protocol: "Investigating DOIs classes of errors" V.2. https://dx.doi.org/10.17504/protocols.io.bufwntpe

```
                        "Invalid_cited_DOI": row["Invalid_cited_DOI"],
                        "Valid_DOI": row['Invalid_cited_DOI'],
                        "Already_valid": 1
                        })
                else:
                    checked_dois.append(invalid_dictionary)
            else:
                checked_dois.append(invalid_dictionary)
            pbar.update(1)
        pbar.close()
        return checked_dois
```

For performing the HTTP request, the function uses the **handle_request()** method of the **class Support()**.

```
def _requests_retry_session(
        self,
        tries=1,
        status_forcelist=(500, 502, 504, 520, 521),
        session=None
    ) -> Session:
        session = session or requests.Session()
        retry = Retry(
            total=tries,
            read=tries,
            connect=tries,
            status_forcelist=status_forcelist,
        )
        adapter = HTTPAdapter(max_retries=retry)
        session.mount('http://', adapter)
        session.mount('https://', adapter)
        return session

def handle_request(self, url:str, cache_path:str="",
error_log_dict:dict=dict()):
        if cache_path != "":
            requests_cache.install_cache(cache_path)
        try:
            data = self._requests_retry_session().get(url, timeout=10)
            if data.status_code == 200:
                return data.json()
            else:
                error_log_dict[url] = data.status_code
        except Exception as e:
            error_log_dict[url] = str(e)
```

In our procedure we applied our **check_dois_validity()** function to our **<data>** and we store the results in a **<checked_dois>** variable:

```
>>> checked_dois=clean_dois.check_dois_validity(data=data)
```

This is how the **<checked_dois>** variable appears:

[{"Valid_citing_DOI": "10.1007/s11771-020-4410-2", "Invalid_cited_DOI": "10.13745/j.esf.2016.02.011", "Valid_DOI": "10.13745/j.esf.2016.02.011", Already_valid": 1}, ...]

3.3 We then pass the **<checked_dois>** executed by the previous function to another function called **procedure()** aimed to clean the DOIs with the regular expressions described above and to verify the validity of the cleaned DOI.

```
def clean_doi(self, doi:str) -> str:
        tmp_doi = doi.upper().replace(" ", "")
```


Citation: Ricarda Boente, Deniz Tural, Cristian Santini, Arcangelo Massari (04/27/2021). Protocol: "Investigating DOIs classes of errors" V.2. https://dx.doi.org/10.17504/protocols.io.bufwntpe

```
        prefix_match = re.search(self.prefix_regex, tmp_doi)
        classes_of_errors = {
            "prefix": 0,
            "suffix": 0,
            "other-type": 0
        }
        if prefix_match:
            tmp_doi = prefix_match.group(1)
            classes_of_errors["prefix"] += 1
        suffix_match = re.search(self.suffix_regex, tmp_doi)
        if suffix_match:
            tmp_doi = suffix_match.group(1)
            classes_of_errors["suffix"] += 1
        new_doi = re.sub("\\\\", "", tmp_doi)
        new_doi = re.sub("__", "_", tmp_doi)
        new_doi = re.sub("\\.\\.", ".", tmp_doi)
        new_doi = re.sub("<.*?>.*?", "", tmp_doi)
        new_doi = re.sub("<.*?/>", "", tmp_doi)
        if new_doi != tmp_doi:
            classes_of_errors["other-type"] += 1
        return new_doi, classes_of_errors

def procedure(self, data:list) -> list:
        output = list()
        pbar = tqdm(total=len(data))
        for row in data:
            invalid_doi = row["Invalid_cited_DOI"]
            invalid_dictionary = {
                "Valid_citing_DOI": row["Valid_citing_DOI"],
                "Invalid_cited_DOI": row["Invalid_cited_DOI"],
                "Valid_DOI": row["Valid_DOI"],
                "Already_valid": row["Already_valid"],
                "Prefix_error": 0,
                "Suffix_error": 0,
                "Other-type_error": 0
            }
            if row["Already_valid"] != 1:
                new_doi, classes_of_errors =
self.clean_doi(row["Invalid_cited_DOI"])
                valid_dictionary = {
                    "Valid_citing_DOI": row["Valid_citing_DOI"],
                    "Invalid_cited_DOI": row["Invalid_cited_DOI"],
                    "Valid_DOI": new_doi,
                    "Already_valid": row["Already_valid"],
                    "Prefix_error": classes_of_errors["prefix"],
                    "Suffix_error": classes_of_errors["suffix"],
                    "Other-type_error": classes_of_errors["other-type"]
                }
                if new_doi != row["Invalid_cited_DOI"]:
                    handle =
Support().handle_request(url=f"https://doi.org/api/handles/{new_doi}",
cache_path=self.cache_path, error_log_dict=self.logs)
                    if handle is not None:
                        output.append(valid_dictionary)
                    else:
                        output.append(invalid_dictionary)
                else:
                    output.append(invalid_dictionary)
            else:
                output.append(invalid_dictionary)
            pbar.update(1)
        pbar.close()
        return output
```

In our data cleaning procedure we store the output of the **procedure()** function in the **<output>** variable:

Citation: Ricarda Boente, Deniz Tural, Cristian Santini, Arcangelo Massari (04/27/2021). Protocol: "Investigating DOIs classes of errors" V.2. https://dx.doi.org/10.17504/protocols.io.bufwntpe

```
>>> output = clean_dois.procedure(data=checked_dois)
```

For each cited DOI, the function **procedure()** performs five actions:
1) it removes white spaces in the string
2) it applies the prefix regular expression to catch eventual unwanted characters that precede a DOI. If there's a match, the regular expression removes the unwanted characters and the DOI is replaced with the cleaned one.
3) it applies the suffix regular expressions as a list of alternative unwanted patterns that may occur at the end of the string. If there's a match, the regular expression removes the unwanted characters and the DOI is replaced with the cleaned one.
4) finally, it removes eventual unwanted characters like double slashes, double points, xml tags and backward slashes.
5) Finally the algorithm checks if the cited DOI after the cleaning procedure is different from the one in **<checked_dois>** and it also checks if it is returned as valid by the DOI Handle API. If it is different and is validated by the DOI API, the algorithm stores the cleaned DOI in the "*Valid_DOI*" field of the dictionary; if it is not different, the dictionary will store no new DOI.
By applying **procedure()** to our **<checked_dois>** we obtain a list of dictionaries containing 7 fields: the four already present in the input list ("*Valid_citing_DOI*", *"Invalid_cited_DOI"*, *"Valid_DOI"*, "*Already_Valid*") and, in addition, *"prefix_error"*, *"suffix_error"*, *"other-type*" fields, which contain, for each cleaned DOI the number of errors that were cleaned. This is how the **<output>** variable looks like:

[{"Valid_citing_DOI": "10.1007/s40617-018-00299-1", "Invalid_cited_DOI": "10.1901/jaba.2012.45-657", "Valid_DOI": "10.1901/JABA.2012.45-657", Already_valid": 1, "prefix_error": 0, "suffix_error" 0:, "other-type": 0 },
{"Valid_citing_DOI": "10.1074/jbc.m508416200", "Invalid_cited_DOI": "10.1059/0003-4819-100-4-483", "Valid_DOI": "", Already_valid": 0, "prefix_error": 0, "suffix_error" 0:, "other-type": 0 },
{"Valid_citing_DOI": "10.1177/2054358119836124", "Invalid_cited_DOI": "10.1016/j.amepre.2015.07.017.", "Valid_DOI": "10.1016/J.AMEPRE.2015.07.017", Already_valid": 0, "prefix_error": 0, "suffix_error" 1:, "other-type": 0 },
...]

3.4 At the end of our procedure we dump our **<output>** in a csv file through the **dump_csv()** method of the class **Support()**:

```
def dump_csv(data:list, path:str):
        print(f"[Support:INFO Writing csv at path {path}]")
        with open(path, 'w', newline='', encoding='utf8')  as output_file:
            keys = data[0].keys()
            dict_writer = csv.DictWriter(output_file, keys)
            dict_writer.writeheader()
            dict_writer.writerows(data)
```

This is how the method is used in our data cleaning procedure:

```
>>> Support().dump_csv(data=output, path="./output.csv")
```

The output of the function is then stored in a CSV file, organized in 7 fields as the dictionary mentioned above:

Valid_citing_DOI	Invalid_cited_DOI	Valid_DOI	Already_valid	Prefix_error	Suffix_error	Other-type_error
10.1007/s40617-018-00299-1	10.1901/jaba.2012.45-657	10.1901/JABA.2012.45-657	1	0	0	0
10.1074/jbc.m508416200	10.1059/0003-4819-100-4-483		0	0	0	0
10.1177/2054358119836124	10.1016/j.amepre.2015.07.017.	10.1016/J.AMEPRE.2015.07.017	0	0	1	0

A sample of the output data is available in our code release (http://doi.org/10.5281/zenodo.4723984).
Further information about the possible uses of research data generated by this project, and the

Citation: Ricarda Boente, Deniz Tural, Cristian Santini, Arcangelo Massari (04/27/2021). Protocol: "Investigating DOIs classes of errors" V.2. https://dx.doi.org/10.17504/protocols.io.bufwntpe

support provided by the authors for reuse, are available in the DMP provided by the authors at https://doi.org/10.5281/zenodo.4665853.

STATISTICS AND VISUALIZATION

4 Finally, a visualization was produced starting from the output file. For this purpose, d3.js v6.7.0, an open-source Javascript library for manipulating data-driven documents, was used. In particular, a barchart was created that compares the total number of DOIs with the number of valid DOIs after cleaning, the number of those that were already valid and the number of those that presented the various types of errors, i.e. errors of prefix, suffix and other-type. The barchart can be sorted in ascending or descending order and you can hover the mouse over the bars to view the exact represented value. It is possible to interact with the visualization at the following address: https://open-sci.github.io/2020-2021-grasshoppers-code/.

Citation: Ricarda Boente, Deniz Tural, Cristian Santini, Arcangelo Massari (04/27/2021). Protocol: "Investigating DOIs classes of errors" V.2. https://dx.doi.org/10.17504/protocols.io.bufwntpe