T. Grust et al. (Hrsg.): Datenbanksysteme für Business, Technologie und Web (BTW 2019),
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2019 555

jHound: Large-Scale Profiling of Open JSON Data

Mark Lukas Möller[1] Nicolas Berton[2] Meike Klettke[1] Stefanie Scherzinger[3] Uta Störl[4]

Abstract: We present jHound, a tool for profiling large collections of JSON data, and apply it to thousands of data sets holding open government data. jHound reports key characteristics of JSON documents, such as their nesting depth. As we show, jHound can help detect structural outliers, and most importantly, badly encoded documents: jHound can pinpoint certain cases of documents that use string-typed values where other native JSON datatypes would have been a better match. Moreover, we can detect certain cases of maladaptively structured JSON documents, which obviously do not comply with good data modeling practices. By interactively exploring particular example documents, we hope to inspire discussions in the community about what makes a good JSON encoding.

1 Introduction

In an effort towards facilitating government transparency, the German government passed the e-government law in 2013. In particular, this law regulates the provision of *machine readable data* by government institutions. However, a W3C working draft on publishing open government data[5] recommends to use *human-readable* data formats instead. In this demo, we focus on the data exchange format JSON, since it is both machine and human-readable, and becoming increasingly popular: At the time of writing, the site www.data.gov alone is hosting over 14,000 JSON documents[6], capturing just about anything from census data to consumer complaints. However, the documents come without a proper schema declaration (such as a JSON Schema description). Many documents even come without *any* meta-data regarding the utmost basics, such as the semantics of property names. For data consumers who intend to programmatically process this data, this poses a serious challenge.

To understand the document structure and to detect bad encoding (more on this later), we introduce jHound, a profiler for large collections of JSON data. While our main motivation is to analyze open government data, our tool is not restricted to this use case.

[1] University of Rostock, mark.moeller2|meike.klettke@uni-rostock.de
[2] ENSEIRB-MATMECA, Bordeaux, France
[3] OTH Regensburg, stefanie.scherzinger@oth-regensburg.de
[4] University of Applied Sciences Darmstadt, uta.stoerl@h-da.de
[5] Publishing Open Data, https://www.w3.org/TR/gov-data/, W3C Working Draft September 2009.
[6] According to our experience, open government data changes frequently on the hosting sites, due to additions and removals of files. We have noticed that statements regarding the total number of files hosted come with a short expiration date. The number reported here dates back to December 6th, 2018.

doi:10.18420/btw2019-44

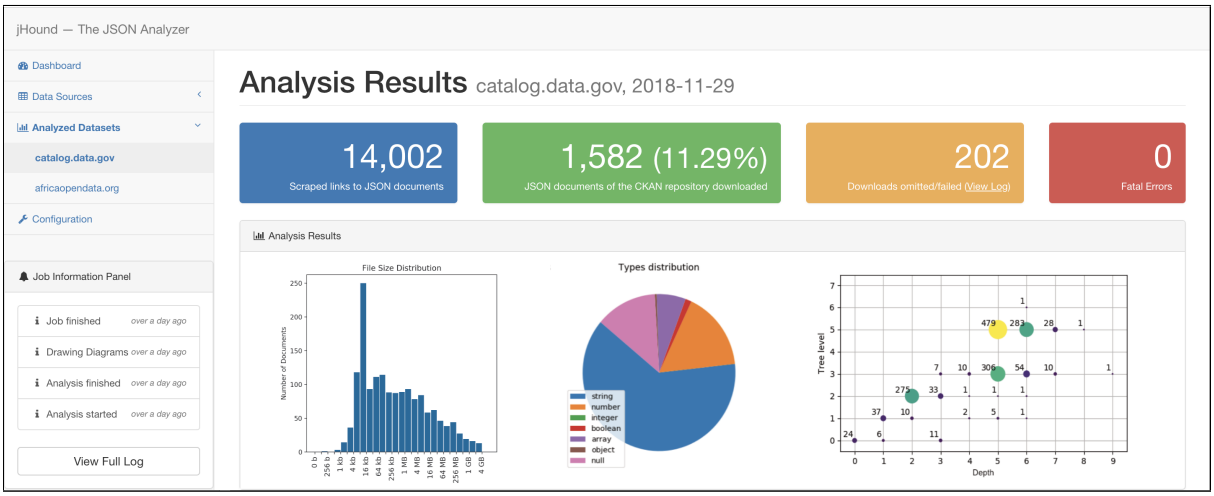


Fig. 1: The web frontend of jHound, showing the analysis results page.

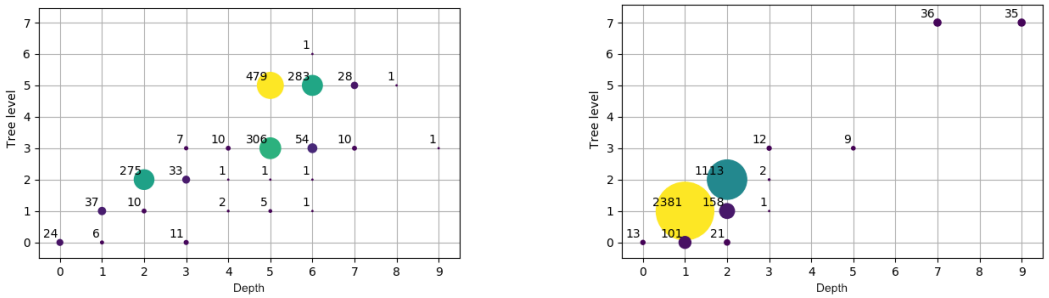
Related Work. Profiling relational data is an established research area (cf. [Ab18] for an overview), and there is a range of established commercial tools. However, tooling for JSON data has not yet matured to the same extent. One active direction of research is schema extraction for JSON document collections (cf. [FSS18]). While this is certainly related, a profiler additionally extracts descriptive statistics such as distributions and outliers.

Contributions and Structure. We next describe the jHound workflow. We highlight some key findings of jHound in real-world data, and conclude with an outline of our demo.

2 Analyzing Open Government Data

To analyze JSON data with jHound, we point the tool to a CKAN repository, catalog.data.gov for instance. As seen in the top left of the screenshot in Figure 1, jHound identifies over 14 thousand documents. We used jHound to download a sample with over 1.5 thousand JSON documents. An additional 202 documents could not be downloaded, causing warnings due to timeouts or broken links. We then exclude any documents that have been miscategorized, i.e., they have been labeled as JSON, but do not comply to the format (49 in this scenario).

The results of the analysis are shown in the lower half of Figure 1: (1) jHound reports the distribution of file sizes in a histogram. Exploring the extremes reveals peculiar cases. For instance, among the very small files are documents that merely consist of an empty JSON array. Also, there are files in the gigabyte range. This is a vital information for anybody planning to programmatically process this data, since they need to allocate enough main memory, or choose streaming algorithms. In fact, in our explorations of open government data, we have encountered JSON documents with 60GB in size. (2) A pie chart visualizes the overall distribution of data types among the property values. These types are derived from the encoding syntax, so the value `"3.14"` is categorized as type `string`, even though it encodes a numerical value. jHound does not yet derive types based on semantic analysis.



(a) 1,565 documents from https://www.data.gov/.

(b) 3,882 documents from https://data.europa.eu/.

Fig. 2: The maximum document depth versus the level where most of the nodes resides.

```
[ { "indicator": {
        "id": "SH.DTH.IMRT",
        "value": "Number of infant deaths" },
    "country": {
        "id": "ZW",
        "value": "Zimbabwe" },
    "value": "19375",
    "decimal": "0",
    "date": "2017" },
...]
```

(a) Infant deaths in Zimbabwe.

```
[ [ "organization",
    "interaction_intervention_id",
    "org_intervention_id",
    "project_tags",
    "project_name", ... ], ...
  [ "PROJECT C.U.R.E.",
    "59-21401",
    "11711",
    "",
    "Panama In Kind Donations ", ... ], ... ]
```

(b) Activities in Panama.

Fig. 3: Excerpts of open data documents provided at https://data.humdata.org/.

(3) A bubble chart relates the maximal nesting depth of documents and the tree level at which most of the nodes resides. Figure 2(a) shows the chart in isolation. For instance, there are 24 documents that are not nested, among them structural anomalies, like the documents that merely contain an empty JSON array, and therefore no payload data.

In Figure 2, we compare data from different hosting sites. In Fig. 2(b), the majority of documents has a nesting depth of one. These documents are not nested, but encode flat tuples via key-value pairs. Thus, these documents might have just as well been stored as relational tuples. It is a mere conjecture at this point, but we suspect that these JSON documents were generated from relational data.

Figure 3 shows excerpts of JSON documents with maladaptive structure that we have discovered with jHound: In the left document, all property values are encoded as strings, even though the values in lines 7 through 9 would be more naturally encoded using JSON type `number`. Similar with the document to the right. Structurally, both documents are unusual. For the left document, we could not find any documentation on the nesting strategy, or on the

meaning of the decimal-property in line 8. This problem of piecing together information is even more striking in the right document, which seems to be a blunt line-by-line translation of CSV data into JSON format. By matching positions in the nested arrays (lines 3 and 8), we painstakingly extract that the intervention id is 11711. It is possible to detect malformed documents by inspecting their structure and property values.

3 Demo Outline

Our demo shows the interactive workflow of analyzing open government data with jHound:

1. We start with documents downloaded from CKAN repositories (e.g. from https://data.humdata.org/, https://www.data.gov/, and https://www.govdata.de/).
2. In dialogue with our audience, we drill down into the analysis results, starting with the analysis results page, as shown in Figure 1:
   - We configure filters for the pie chart showing the type distribution, e.g. considering only JSON documents with over 90% string-typed values.
   - Having identified interesting bubbles in the bubble chart, we can further identify and inspect the raw JSON datasets. In doing so, we come across maladaptive encodings, like those shown in Figure 3.
   - For individual datasets, we can extract a schema description, using an implementation from our earlier work [Kl17].

4 Conclusion

jHound is a tool for scraping, downloading, and analyzing JSON documents. jHound compiles descriptive statistics, allowing us to systematically explore large collections of JSON documents. We plan to open-source our *python*-implementation of jHound.

Acknowledgements: The article is published in the scope of the project *"NoSQL Schema Evolution und Big Data Migration at Scale"* which is funded by the *Deutsche Forschungsgemeinschaft (DFG)* under the number 385808805.

References

[Ab18] Abedjan, Ziawasch; Golab, Lukasz; Naumann, Felix; Papenbrock, Thorsten: Data Profiling. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2018.

[FSS18] Frozza, Angelo Augusto; dos Santos Mello, Ronaldo; de Souza da Costa, Felipe: An Approach for Schema Extraction of JSON and Extended JSON Document Collections. In: Proc. IEEE IRI'18. 2018.

[Kl17] Klettke, Meike; Awolin, Hannes; Störl, Uta; Müller, Daniel; Scherzinger, Stefanie: Uncovering the evolution history of data lakes. In: Proc. SCDM'07. 2017.