A Comparative Analysis of JSON Schema Inference Algorithms

Ivan Veinhardt Latták and Pavel Koupil[a]
Department of Software Engineering, Charles University, Prague, Czech Republic

Keywords: Schema Inference, Reverse Engineering, Document Model, JSON.

Abstract: NoSQL databases are becoming increasingly more popular due to their undeniable advantages in the context of storing and processing Big Data, mainly horizontal scalability and minimal requirement to define a schema upfront. In the absence of the explicit schema, however, an implicit schema inherent to the stored data still exists and it needs to be reverse engineered from the data. Once inferred, it is of a great value to the stakeholders and database maintainers. Nevertheless, the problem of schema inference is non-trivial and is still the subject of ongoing research. In this paper we provide a comparative analysis of five recent proposals of schema inference approaches targeting the JSON format. We provide both static and dynamic comparison of the approaches. In the former case we compare various features. In the latter case we involve both functional and performance analysis. Finally, we discuss remaining challenges and open problems.

1 INTRODUCTION

Traditional database management systems (i.e., relational, object, object-relational) enforce that the data are highly *structured* and conform to a strictly predefined schema which is designed in one of the first steps of forward engineering approaches. From the point of view of data management this approach is denoted as *schema-on-write*. On the other hand, novel NoSQL systems reflecting the V-features of Big Data (Volume, Velocity, Variety, Veracity, ...) relax this rule, since it does not reflect requirements typical for Big Data applications, which work also with *semi-structured* or *unstructured* data. However, when the data is retrieved to be processed, its structure needs to be known. We speak about *schema-on-read* approaches.

Although semi-structured and unstructured data is not bound with an explicit schema, this schema is implicitly present and can therefore be inferred, i.e. reverse engineered, from the data. Despite its limitations (given by the quality and richness of the input data), a schema inferred from a sample dataset is of great value – it can be used by stakeholders to reason about the data, by automated tools for data validation and migration, for object code generation, etc. Or, an inferred schema of newly added schema-less data can be integrated to the originally designed schema and thus enrich the knowledge of the structure of data.

[a] https://orcid.org/0000-0003-3332-3503

The inference process itself, however, is non-trivial. Several schema inference approaches already exist but many are insufficient in various aspects. Hence, our aim in this paper is to thoroughly analyze the problems of schema inference. In particular, we focus on semi-structured data, namely *document-oriented* data, due to higher complexity of the document model and the overwhelming popularity of document databases[1] compared to key/value or columnar ones.

In our previous paper (Čontoš and Svoboda, 2020), we already examined a number of existing solutions and we described their strengths and weaknesses. In addition, we provided an example of inferred JSON schema for each discussed approach. In this paper we significantly extend our work. The main contributions are as follows:

- We analyze five recent JSON schema inference approaches. Apart from a static analysis and discussion of various features, we provide an experimental analysis of the algorithms involving both functional and performance analysis.
- Implementations of all the examined approaches were acquired from the authors[2] and debugged when necessary. The datasets, as well as the used generator are available in GitHub repository[3] for

[1] https://db-engines.com/en/ranking
[2] The links to repositories are provided in Table 1.
[3] https://github.com/ivan-lattak/schema-inference

further exploitation.

- We discuss a set of open problems and challenges in the area of schema inference, including the context of currently highly popular multi-model data.
- As long as we are not authors of any of the compared approaches, we provide an unbiased comparative study.

The rest of the paper is structured as follows: Section 2 summarizes both the compared approaches and other related work. In section 3 we provide a comparison of the selected algorithms based on their static characteristics. In section 4, we present an experimental analysis of selected existing solutions, both functional and performance. In Section 5 we overview challenges and discuss possible future improvements. We conclude in Section 6.

2 SCHEMA INFERENCE APPROACHES

Research on schema inference of semi-structured data is not new as it involves both modern NoSQL databases as well as a bit older technologies such as XML[4] or RDF[5]. In this section, we introduce selected comparative works and we summarize the most prominent and relevant schema inference approaches.

There exist several surveys dealing with schema inference approaches in document databases. Mlýnková et al. (Mlýnková and Nečaský, 2013) provide an overview of the field of heuristic XML Schema inference and summarize existing approaches and open problems. Morales in his dissertation thesis (Morales, 2017) statically compares several schema extraction algorithms over multiple NoSQL stores.

We believe that due to the versatility and popularity of the JSON format, JSON schema inference approaches are the most promising, especially in terms of extensibility towards schema inference in popular multi-model systems. Hence, in this paper, we focus on these approaches. In particular, we analyze the following ones:

- Sevilla et al. (Sevilla Ruiz et al., 2015b) present an approach for inferring versioned schemas from document NoSQL databases based on *Model-Driven Engineering* (MDE) along with sample applications created from such inferred schemas. This research is furthered by Morales in his dissertation thesis (Morales, 2017) and by Hernandez et al. who tackle the issues of visualization of schemas of *aggregate-oriented* NoSQL databases and propose desired features which should be supported in visualization tools (Chillón et al., 2017). Most recently, Fernandez et al. expand upon the meta-model from paper (Sevilla Ruiz et al., 2015b) by introducing a unified meta-model capable of modeling both NoSQL and relational data (Candel et al., 2021).
- Scherzinger et al. (Scherzinger et al., 2013) introduce a platform-agnostic NoSQL data evolution management and schema maintenance solution. The same research group later proposes an approach for extraction of a schema from JSON data stores, measuring the degree of heterogeneity in the data and detecting structural outliers (Klettke et al., 2015). They also introduce an approach for reconstructing schema evolution history of *data lakes* (Klettke et al., 2017b). Additionally, Moller et al. present *jHound* (Möller et al., 2019), a JSON data profiling tool which can be used to report key characteristics of a dataset, find structural outliers, or detect documents violating best practices of data modeling. Finally, Fruth et al. present *Josch* (Fruth et al., 2021), a tool that enables NoSQL database maintainers to extract a schema from JSON data more easily, refactor it, and then validate it against the original dataset.
- Baazizi et al. (Baazizi et al., 2019b) propose a distributed approach for parameterized schema inference of massive JSON datasets and introduce a simple but expressive JSON type language to represent the schema.
- Izquierdo and Cabot provide an MDE-based approach for discovering schema of multiple JSON web-based services (Izquierdo and Cabot, 2013a) and later put it in practice as a web-based application along with a visualization tool (Izquierdo and Cabot, 2016).
- Frozza et al. introduce a graph-based approach for schema extraction of JSON and BSON[6] document collections (Frozza et al., 2018a) and another inference process for columnar NoSQL databases (Frozza et al., 2020), specifically HBase[7].

In addition, there exists a number of approaches that deal with different data models. For example:

- Bex et al. (Bex et al., 2010) introduce a method of inference of a concise XML *DTD*[8] by reduc-

[4]https://www.w3.org/TR/xml/
[5]https://www.w3.org/RDF/
[6]https://bsonspec.org
[7]http://hbase.apache.org
[8]https://www.w3.org/XML/

ing the problem to learning concise regular expressions from positive examples (i.e., documents valid against the target DTD).

- DiScala and Abadi (DiScala and Abadi, 2016) present an algorithm for automatic generation of a relational database schema from JSON data along with subsequent transformation of the data itself.
- Galinucci et al. (Gallinucci et al., 2018) propose a way how to enable non-technical users to enrich RDF data cubes by recognizing recurring patterns in *linked open data*.
- Finally, Bouhamoum et al. (Bouhamoum et al., 2018) deal with the issues of horizontal scaling of existing RDF schema discovery approaches and present a method based on extracting a condensed representation of the initial dataset.

Nevertheless, in the rest of the comparative study we will focus only on the following JSON schema inference approaches: Sevilla et al. (Sevilla Ruiz et al., 2015b), Klettke et al. (Klettke et al., 2017b), Baazizi et al. (Baazizi et al., 2019b), Izquierdo and Cabot (Izquierdo and Cabot, 2013a), and Frozza et al. (Frozza et al., 2018a).

3 STATIC ANALYSIS

First of all, we focus on the statically determinable features. Table 1 summarizes the comparison of the key characteristics discussed in the following paragraphs.

- *Input Format:* All selected approaches support schema extraction from JSON data. Frozza et al. approach supports also BSON data. In addition, approaches by Sevilla et al. and Klettke et al. mention how other aggregate-oriented data can be trivially converted to JSON, so that in practice any schema inference approach for JSON data can be used also for key/value and columnar data.
- *Multiple Collections:* The majority of algorithms support inference only from a single document collection at a time, while merging of these schemas is left up to the user. Approaches that can process an entire JSON database are Sevilla et al. and Izquierdo and Cabot exploiting MDE to do so.
- *Inference Process:* Most of the approaches extract schema information from all the documents stored in the input collection. Approaches by Sevilla et al. and Frozza et al. first select a minimal collection of mutually distinct documents such that they still bear all the different input cases. A common feature of all the approaches is the replacement of values of properties by names of the primitive types encountered. In addition, this step is usually parallelized using MapReduce or Apache Spark which greatly improves the scalability.
- *Output Format:* The majority of approaches output the inferred schema in a textual format. Klettke et al. and Frozza et al. use *JSON Schema*[9] and Baazizi et al. use their own type description language. The approaches by Sevilla et al. and Izquierdo and Cabot output the inferred schema as a data model, both based on *UML*[10].
- *Implementation:* The approaches of Sevilla et al. and Izquierdo and Cabot were implemented as Java applications running on the Eclipse platform. Both of them offer a Java API and a simple web application wrapper. Klettke et al. implemented their approach as Java application running on the Spring Boot platform. Baazizi et al. implemented their approach as a Scala application designed to run in the Apache Spark environment. Finally, the approach by Frozza et al. is implemented as a JavaScript web application. The front end is written in TypeScript and provides the user with a presentation layer, while the back end is written using *Node.js* and contains the implementation of the inference approach.
- *Structural Components:* All the approaches are capable of inference of various structural components of JSON documents, including simple data types (String, Number, Boolean) and some complex data types (array, object). Parent/child relationships (within aggregates) are also captured in inferred schemas, either as nesting in textual form or an arrow in the graphical form of an inferred schema. On the other hand, there is no approach capable of inference of complex data structures, including sets, maps and tuples.
- *Integrity Constraints:* Only one specific type of integrity constraints (ICs), namely simple referential integrity, is detected by the inference approach by Sevilla et al. No other ICs are inferred by any of the researched inference approaches.
- *Optional Properties:* All approaches, except for Izquierdo and Cabot, are able to describe optional properties in their schemas. JSON Schema-based approaches, i.e., Klettke et al. and Frozza et al., use keyword `required` to enumerate the required properties, while others are optional. Approach of Baazizi et al. uses the optionality modifier (i.e.,

[9]https://json-schema.org

[10]https://www.omg.org/spec/UML/

Table 1: Comparison of selected schema inference approaches.

	Sevilla	**Klettke**	**Baazizi**	**Izquierdo**	**Frozza**
Repository	(Sevilla Ruiz et al., 2015a)	(Klettke et al., 2017a)	(Baazizi et al., 2019a)	(Izquierdo and Cabot, 2013b)	(Frozza et al., 2018b)
Algorithm	MapReduce + MDE	Fold into graph	Type reduction in Apache Spark	MDE	Aggregation + fold into graph
Input format	Aggregate-oriented NoSQL data	JSON	JSON	JSON web service responses	Extended JSON
Input type	Multiple collections	Single collection	Single collection	Multiple collections	Single collection
Output format	NoSQL Schema model	JSON Schema	Custom textual type language	Ecore model	JSON Schema
Schema root	Entities	Documents	Documents	Entities	Documents
Implementation	Eclipse bundle	Spring Boot application	Apache Spark application in Scala	Eclipse bundle	Node.js web application
Optional	Yes	Yes	Yes	No	Yes
Entity versions	Yes	No	Yes	No	No
Union type	No	Yes	Yes	No	Yes
References	Yes	No	No	No	No
Tuple	No	No	No	No	No
Set	No	No	No	No	No
Map	No	No	No	No	No
Extended JSON	No	No	No	No	Yes
Complex IC	No	No	No	No	No
Scalable design	Yes	Yes	Yes	Yes	No
Scalable implementation	Yes	No	Yes	No	No

`?`) to describe optional properties when a kind-equivalence relation is used in the reduction. Approach by Sevilla et al. can infer optional properties by merging all versions of a single entity together, marking each property as required if it is present in all of them, and optional if not.

- *Union Type:* Approaches by Klettke et al., Frozza et al., and Baazizi et al. can infer and express union types in the schema. The former two use JSON Schema keyword `oneOf`, while the latter one defines for this purpose the union type constructor (i.e., +). Approaches by Sevilla et al. and Izquierdo and Cabot do not support union types of properties. Approach by Sevilla et al. uses entity versioning instead while approach by Izquierdo and Cabot uses the alternative approach of reducing different types to their most generic type, like `EString`.
- *Scalability:* Despite the fact that the majority of approaches has a scalable design, the implementations are not parallelized, therefore they do not scale horizontally. Sevilla et al. use MapReduce to decrease the number of input documents that are considered in the rest of the schema inference process. Similarly, Apache Spark is used in approach of Baazizi et al.

4 DYNAMIC ANALYSIS

Next we design, execute, and evaluate experiments which demonstrate the behavior of individual approaches. We identify points of failure and illustrate differences between the approaches. The first phase involves a *functional analysis* of the given approaches. It exemplifies the functional behavior of the approaches when applied on datasets containing different schema features. The second phase, *performance analysis*, compares the relative runtime performance of the approaches by executing them in an identical environment and against identical datasets.

4.1 Functional Analysis

For this analysis we have created manually 8 separate datasets with self-descriptive names, each focusing on a different schema feature, namely *PrimitiveTypes*, *SimpleArrays*, *SimpleObjects*, *ComplexArrays*, *ComplexObjects*, *Optional*, *Union*, and *References*. They are located in the `/experiment` directory within the root of the GitHub repository[11] as JSON files. These datasets were imported to a collection named `articles`, each within a separate MongoDB database. Then, all 5 inference approaches were run over the databases. Since the majority of the approaches behaved according to the expectation with a majority of the datasets, we will only discuss the detected abnormalities.

Dataset *SimpleArrays* features an empty JSON array in property `nothings`. The empty array is not handled correctly by the implementation of Sevilla et al. – it throws an uncaught exception during the inference. The implementation by Frozza et al. also has a problem with this edge case, although not so severe one – the resulting JSON Schema is invalid as it contains an invalid definition for the array element type.

Dataset *ComplexArrays* contains a two-dimensional array in property `nested_arrays`. The dimensionality is not handled correctly by the approach by Izquierdo and Cabot, which models the property as a simple one-to-many relationship.

[11] https://github.com/ivan-lattak/schema-inference

In dataset *Optional* the optional properties were not modeled by the approach by Izquierdo and Cabot For an unknown reason, optional property `body.compressed` was not inferred at all.

Approach by Sevilla et al. inferred the union types in dataset *Union* as versioned entities instead. However, the union type is used as the element type of an heterogeneous array in property `comments`. This heterogeneous array is modeled by Sevilla et al. as a tuple containing a string and a number instead. In the schema inferred by Izquierdo and Cabot the heterogeneous types are reduced to the most generic type, i.e., String.

Finally, dataset *References* contains references to entities in two forms: a property named `article_id` and a property containing BSON `DBRef`. These were difficult to handle for most inference approaches – Sevilla et al. and Izquierdo and Cabot output an empty schema and an empty package definition, respectively, while approaches by Klettke et al. and Frozza et al. end with an error and do not output any schema.

4.2 Performance Analysis

Next we compare the approaches in terms of runtime performance. This was done using a series of performance experiments, running the existing implementations against a number of datasets.

4.2.1 Execution

To the best of our knowledge, there is no open JSON dataset covering all the structural aspects we want to test. So, first, we have generated a dataset of 500,000 JSON documents, serving as the master dataset for our experiments. For this generation we have used the *jsongenerator* open-source library, whose source code is available in GitHub[12]. This library is able to generate JSON documents according to a given JSON Schema. As the schema for the generation, we used a manually created schema that covers all the aspects of JSON data, i.e., simple types, complex types, nesting, union types, optional fields, etc.

We conducted 8 experiments differing in the size of input data. Namely the chosen sample sizes were 1k, 2k, 4k, 8k, 16k, 32k, 64k, and 128k documents. Each experiment was conducted using the following steps:

1. Extract a randomly sampled subset of the given size.
2. Run each algorithm on the extracted subset 30 times.
3. Repeat the previous steps a total of 30 times.

[12]https://github.com/jimblackler/jsongenerator

Experiments of different sizes were chosen, because we wanted to measure the performance of a given implementation as it changes depending on the number of input documents. If we had extracted just one random sample of a given size for an experiment, the results could be distorted as the runtime performance of the algorithms could become dependent on the particularities of each random selection. 30 different random samples of a given size were extracted from the master dataset to mitigate this distortion. Furthermore, if only a single run of each algorithm was performed for a given size, the results could be distorted by the runtime cost of algorithm initialization and would not reflect the sustained performance of the algorithm. Additional distortion could be caused by momentary decrease of system resources caused by random external influences. To mitigate both of them, 30 runs of each algorithm were performed on each extracted subset.

The experiments were performed on a virtual machine running on VMware[13] infrastructure with 64 gigabytes of memory and 8 single-thread processor cores. When performing the measurements, the first, so-called *warm-up* run for each random sample for each approach was significantly longer than the rest. Measurements for these warm-up runs were removed not to skew the results.

4.2.2 Results

Figure 1 depicts the behavior of each of the approaches in all experiments. The *x*-axis represents the different experiment sizes. The *y*-axis represents the average runtime of each approach. The legend above explains the meaning of the different colors of the lines.

The linear scale shows the differences in average runtimes for large data. Frozza et al. performed the best for larger datasets, twice as fast as the slowest approaches. In the 16k, 32k, and 64k experiments Sevilla et. al., Baazizi et al., and Izquierdo and Cabot kept roughly identical performance. However, in the 128k experiment, Baazizi had significantly better performance compared to the other two.

Figure 2 contains a line chart almost identical to the previous one, only the *y*-axis is logarithmic. This chart can better express the performance behavior for smaller-size experiments since the logarithmic scale emphasizes relative differences in small numbers and shrinks them in large numbers. More importantly, though, this chart demonstrates the linear scalability

[13]https://www.vmware.com/

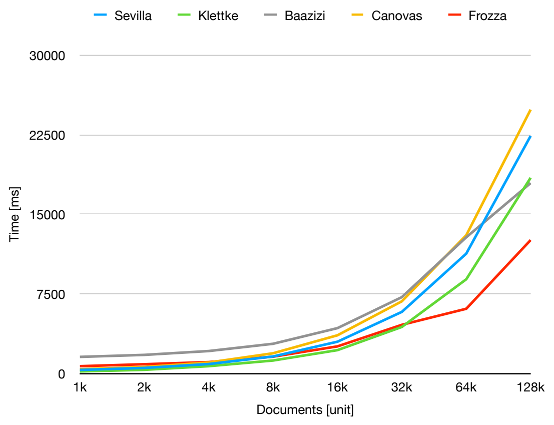

Figure 1: Average runtimes of measured inference approaches across all experiment sizes, linear scale.

of each of the measured approaches. All five approaches exhibit their performance as more-or-less straight lines on the chart.

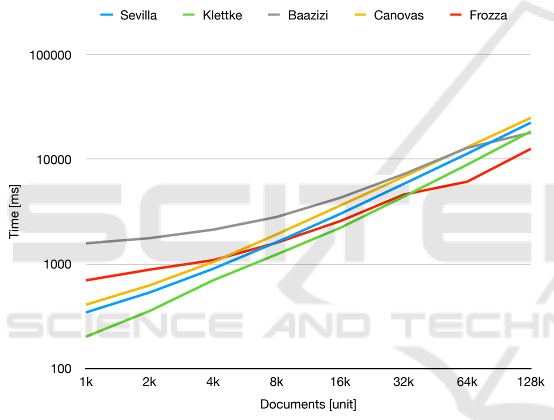

Figure 2: Average runtimes of measured inference approaches across all experiment sizes, logarithmic scale.

Looking at the logarithmic line chart, we can see that Sevilla et al., Izquierdo and Cabot and especially Klettke et al. approaches performed significantly better for smaller sample sizes. This can be attributed mainly to the high overhead cost of using Apache Spark in the approach by Baazizi et al. This difference between the approaches by Klettke et al. versus the approach by Baazizi et al. becomes less pronounced for larger sample sizes. Baazizi et al. even started performing better in the largest experiment.

4.2.3 Evaluation

The results of the experiment provide useful insight to the performance of each approach and the dependence of the performance on the size of the input data.

Different approaches are preferable for usage depending on the size of the input data. For small datasets, Sevilla et al., Klettke et al., Izquierdo and Cabot, and even Frozza et al. are advisable due to the fact that they do not incur high flat overhead costs of the Apache Spark framework unlike Baazizi et al. Out of these, Sevilla et al. had the best performance. Combining that with the most interesting feature set (especially the unique ability to infer references) makes it the best option for small datasets.

As far as large datasets are considered, Frozza et al. is the best option if inference is to be run on a single machine. However, as MongoDB datasets can span multiple database nodes and can contain upwards of millions of documents, horizontal scaling of schema inference may be desired. In that case, Frozza et al. is unsuitable, as it cannot be horizontally scaled.

Sevilla et al. and Izquierdo and Cabot can scale horizontally by decreasing the input size using MapReduce built into MongoDB, in which case the number of MongoDB cluster nodes is the scaling factor. However in the worst-case scenario where every (or almost every) document has a unique raw schema this may not significantly decrease the input size, in which case the algorithm will run slowly.

Baazizi et al. can linearly scale even for the described worst-case scenario simply by adding nodes to the Apache Spark cluster on which the approach is run.

5 CHALLENGES AND OPEN PROBLEMS

After a detailed analysis of existing approaches, we provide a discussion of important open problems and challenges in the area of schema inference.

In general, a schema describes the structure of the data, i.e., it is a set of named (ordered) sets of possibly hierarchical properties. Additionally, a schema may contain a list of integrity constraints, e.g., describing complex business rules or references between data. All these features need to be inferred.

- *Complex Types:* There are complex types which can not be inferred by any examined approach and whose consideration could increase the usability of the approach. *Maps*, sometimes also called *dictionaries*, are similar in structure to JSON objects but semantically different, because the key of the set is not a part of the schematic information (metadata), but it is part of actual data. Similarly *tuples* are special cases of arrays – they have a fixed size and the types of their positional elements must be modeled separately, not as a union. (Tuples in this sense are supported only by Sevilla et al.) Finally, *sets*, i.e., unordered arrays, should be modeled distinctly from standard ordered arrays.

- *References:* Another area of the approach that could be improved is modeling of entity references. Currently, to confirm that a property is an entity reference, the inferred entities are searched to find one with a matching name. To make this heuristic stronger, the reference itself could be checked against existing objects of the given entity type. Additionally, a support for other than primitive-typed entity references, such as references with composite keys, could be beneficial. In general, various types of references are commonly used in document databases.
- *Complex Integrity Constraints:* There is a room for improvements in the field of inference of complex integrity constraints for semi-structured data in general. At first sight, this goes beyond the limits of JSON, since JSON Schema does not consider complex integrity constraints. On the contrary, e.g., XML allows one to model not only keys and references, but also conditional expressions and complex integrity constraints. Or, the combination with (a subset of) the *Object Constraints Language*[14] may extend the expressive power of JSON Schema for practical purposes.

From a more general point of view a JSON Schema inference approach could serve as a basis for a multi-model schema inference approach capable of processing data from relational, graph, key/value, document, columnar, and other logical models within a single schema. However, from this point of view the problem of multi-model schema inference involves the following issues:

- *Fetching Data:* The currently existing schema inference approaches are closely bound to a particular database system. They use specific ways of retrieving data and thus are not applicable to other database systems or data models in general. An optimal algorithm should allow for different ways of reading data and be independent of a particular system. Additionally, and for performance reasons of the approach, it should allow a choice of the framework for data retrieval, e.g., a choice between MapReduce, Apache Spark, etc.
- *Unified Inference Process:* There are at least two approaches to multi-model schema inference: 1) Since JSON is a complex format, it allows us to model constructs of various data models. Thus, existing inference approaches can be applied to multi-model environment if other data models are first converted to a collection of JSON documents (similarly to ArangoDB which stores even the graph model in JSON documents) and then their schema is inferred. 2) An alternative approach can create a universal schema inference approach that uses data-model specific wrappers implementing properties of particular data models.
- *Schema Representation:* Taking a look at this criterion with the multi-model context in mind, there is a concern how the output schema should be appropriately represented. For example, we have proposed a schema description format (Svoboda et al., 2021) suitable for both *semi-structured* data (i.e., the document model) and data from other models, both aggregate-oriented and aggregate-ignorant.

[14]https://www.omg.org/spec/OCL/2.4/PDF

6 CONCLUSION

The purpose of this paper was to provide an unbiased comparative analysis of recent proposals of JSON schema inference approaches. Our aim was to introduce interesting research directions for scientists, as well as to describe possibilities and limitations of existing solutions for practitioners.

To summarize our findings, despite the fact that there have recently been numerous attempts to devise an approach for schema inference from JSON documents, there are still many areas in which they need to be improved. From the ability to model even deeply nested JSON structures to various issues with the necessary horizontal scalability, there is a significant number of aspects in which the existing solutions are still limited. Primarily, as far as we know, there still does not exist a schema inference approach for JSON data able to infer and detect other than basic integrity constraints.

The most promising and at the same time probably the most applicable seems to be the area of multi-model schema inference. Today, it often makes sense for a company to store different parts of their data using different storage technologies and, consequently, different logical models. For this purpose there exist tens of multi-model databases[15] originally both relational and NoSQL. This multi-model context is especially difficult to grasp and develop a suitably robust schema inference approach. The JSON document model is robust and complex, as are schema inference approaches based on it. Hence, they can serve as a good starting point. However, the problem of de-

[15]https://db-engines.com/en/ranking

tection and modeling of inter-model entity references and other integrity constraints becomes significantly more challenging.

ACKNOWLEDGEMENTS

This paper is based on Ivan Veinhardt Latták's Master thesis (Veinhardt Latták, 2021). This work was supported by the GAČR project no. 20-22276S.

REFERENCES

Baazizi, M.-A., Colazzo, D., Ghelli, G., and Sartiani, C. (2019a). https://gitlab.lip6.fr/collab/pstl2020. (unavailable).

Baazizi, M.-A., Colazzo, D., Ghelli, G., and Sartiani, C. (2019b). Parametric Schema Inference for Massive JSON Datasets. *The VLDB Journal*.

Bex, G. J., Neven, F., Schwentick, T., and Vansummeren, S. (2010). Inference of Concise Regular Expressions and DTDs. *ACM Trans. Database Syst.*, 35(2):11:1–11:47.

Bouhamoum, R., Kellou-Menouer, K., Lopes, S., and Kedad, Z. (2018). Scaling up Schema Discovery for RDF Datasets. In *2018 IEEE ICDEW*, pages 84–89. IEEE.

Candel, C. J. F., Ruiz, D. S., and García-Molina, J. (2021). A Unified Metamodel for NoSQL and Relational Databases. *CoRR*.

Chillón, A. H., Morales, S. F., Sevilla, D., and Molina, J. G. (2017). Exploring the Visualization of Schemas for Aggregate-Oriented NoSQL Databases. In *ER Forum/Demos 1979*, volume 1979 of *CEUR*, pages 72–85.

Čontoš, P. and Svoboda, M. (2020). JSON Schema Inference Approaches. In *ER Workshops*, pages 173–183. Springer.

DiScala, M. and Abadi, D. J. (2016). Automatic Generation of Normalized Relational Schemas from Nested Key-Value Data. In *SIGMOD '16*, pages 295–310.

Frozza, A. A., Defreyn, E. D., and dos Santos Mello, R. (2020). A process for inference of columnar nosql database schemas. In *Anais do XXXV Simpósio Brasileiro de Bancos de Dados*, pages 175–180. SBC.

Frozza, A. A., dos Santos Mello, R., and da Costa, F. d. S. (2018a). An Approach for Schema Extraction of JSON and Extended JSON Document Collections. In *IRI 2018*, pages 356–363. IEEE.

Frozza, A. A., dos Santos Mello, R., and da Costa, F. d. S. (2018b). https://github.com/gbd-ufsc/jsonschemadiscovery.

Fruth, M., Dauberschmidt, K., and Scherzinger, S. (2021). Josch: Managing Schemas for NoSQL Document Stores. In *ICDE '21*, pages 2693–2696. IEEE.

Gallinucci, E., Golfarelli, M., Rizzi, S., Abelló, A., and Romero, O. (2018). Interactive Multidimensional Modeling of Linked Data for Exploratory OLAP. *Inf. Syst.*, 77:86–104.

Izquierdo, J. L. C. and Cabot, J. (2013a). Discovering Implicit Schemas in JSON Data. In *ICWE '13*, pages 68–83. Springer.

Izquierdo, J. L. C. and Cabot, J. (2013b). https://github.com/som-research/jsondiscoverer.

Izquierdo, J. L. C. and Cabot, J. (2016). JSONDiscoverer: Visualizing the Schema Lurking behind JSON Documents. *Knowledge-Based Systems*, 103:52–55.

Klettke, M., Awolin, H., Storl, U., Muller, D., and Scherzinger, S. (2017a). https://github.com/dbishagen/darwin.

Klettke, M., Awolin, H., Storl, U., Muller, D., and Scherzinger, S. (2017b). Uncovering the Evolution History of Data Lakes. In *2017 IEEE International Conference on Big Data*, pages 2380–2389, New York, United States. IEEE.

Klettke, M., Störl, U., and Scherzinger, S. (2015). Schema Extraction and Structural Outlier Detection for JSON-based NoSQL Data Stores. In *DBIS '15*, pages 425–444.

Mlýnková, I. and Nečaský, M. (2013). Heuristic Methods for Inference of XML Schemas: Lessons Learned and Open Issues. *Informatica*, 24(4):577–602.

Möller, M. L., Berton, N., Klettke, M., Scherzinger, S., and Störl, U. (2019). jhound: Large-scale profiling of open json data. *BTW 2019*.

Morales, S. F. (2017). *Inferring NoSQL Data Schemas with Model-Driven Engineering Techniques*. PhD thesis, University of Murcia, Murcia, Spain.

Scherzinger, S., Klettke, M., and Störl, U. (2013). Managing schema evolution in NoSQL data stores. In *DBPL '13*.

Sevilla Ruiz, D., Morales, S. F., and García Molina, J. (2015a). https://github.com/catedrasaes-umu/nosqldataengineering.

Sevilla Ruiz, D., Morales, S. F., and García Molina, J. (2015b). Inferring versioned schemas from NoSQL databases and its applications. In *Conceptual Modeling*, pages 467–480. Springer.

Svoboda, M., Contos, P., and Holubova, I. (2021). Categorical Modeling of Multi-Model Data: One Model to Rule Them All. In *MEDI '21*, pages 1–8. Springer.

Veinhardt Latták, I. (2021). *Schema Inference for NoSQL Databases*. Master thesis, Charles University in Prague, Czech Republic.