





List.MID: A MIDI-Based Benchmark for Evaluating RDF Lists

Albert Meroño-Peñuela¹(✉)  and Enrico Daga² 

¹ Computer Science Department, Vrije Universiteit Amsterdam,
Amsterdam, The Netherlands
albert.merono@vu.nl

² Knowledge Media Institute, The Open University, Milton Keynes, UK
enrico.daga@open.ac.uk

Abstract. Linked lists represent *a countable number of ordered values*, and are among the most important abstract data types in computer science. With the advent of RDF as a highly expressive knowledge representation language for the Web, various implementations for RDF lists have been proposed. Yet, there is no benchmark so far dedicated to evaluate the performance of triple stores and SPARQL query engines on dealing with ordered linked data. Moreover, essential tasks for evaluating RDF lists, like generating datasets containing RDF lists of various sizes, or generating the same RDF list using different modelling choices, are cumbersome and unprincipled. In this paper, we propose **List.MID**, a systematic benchmark for evaluating systems serving RDF lists. **List.MID** consists of a dataset generator, which creates RDF list data in various models and of different sizes; and a set of SPARQL queries. The RDF list data is coherently generated from a large, community-curated base collection of Web MIDI files, rich in lists of musical events of arbitrary length. We describe the **List.MID** benchmark, and discuss its impact and adoption, reusability, design, and availability.

Keywords: Linked lists · RDF · Benchmarks

1 Introduction

Linked lists are data structures that represent *a countable number of ordered values*, and are one of the fundamental abstract data types in computer science [15]. They are at least basically supported, with a variety of implementations, in the core libraries of all major programming languages [20].

With the advent of the Semantic Web [4], the Resource Description Framework [23] (RDF) becomes the standard for knowledge representation on the Web. As an expressive data format designed for enabling semantic interoperability, data integration, and data modeling in all sorts of domains, many use cases demand standard ways of representing classic data structures; linked lists are among them. Consequently, Semantic Web standards such as RDF itself [23],

RDF Schema [6], and more recently JSON-LD [24] propose various implementations for RDF lists: `rdf:Seq`, based on list ordering properties; `rdf:List`, based on LISP-like `rdf:first` and `rdf:rest` pointers; or the `"@list": []` JSON-LD attribute. Moreover, the community itself has developed its own ontology design patterns [10] to implement list-like ontological structures.

With this variety of alternatives, many questions arise on practical and performance issues with respect to RDF lists. For example, it is hard to choose one such implementation in large-scale, list-based RDF datasets [18] without knowing the impact of such choice in query performance. Differently, other users may be interested in favoring list readability over performance. In order to address this, some remarkable users have reported ways to query such RDF lists.¹ However, no standard benchmark has been so far proposed in the Semantic Web in order to generate RDF list data, in all its possible modeling alternatives, in a systematic and principled way. Such a benchmark could contribute to clarify many of the open questions about RDF list modeling and publishing on the Web, such as query performance, list readability, triplestore reproducible evaluations, and so forth.

In this paper, we introduce the `List.MID` benchmark, an RDF list data generator and query template set specifically designed for the evaluation of RDF lists. The benchmark has two focus points: (a) to cover as many RDF list implementations as possible, following a systematic study that surveys and summarizes different RDF list modeling practices into *6 different RDF list modeling templates* [8]; and (b) to create such multi-model RDF lists *out of real-world data*, through the large-scale, list-rich symbolic music notation dataset of the MIDI Linked Data cloud [18]. Specifically, the contributions of the paper are:

- We list and describe 6 abstract RDF list modeling patterns recently surveyed [8] (Sect. 3.1)
- We describe the `List.MID` data generator (Sect. 3.2), which generates RDF list data according to these patterns from the MIDI Linked Data cloud dataset [18]; and a set of SPARQL query templates for retrieval (Sect. 3.3)
- We show evidence of use and potential adoption for our proposed benchmark (Sect. 4)

The rest of the paper is organized as follows: Sect. 2 covers the related work; Sect. 3 describes the `List.MID` benchmark, data generator, and queries; Sect. 4 shows evidence of use and potential adoption for the benchmark; and Sect. 5 draws our conclusions.

2 Related Work

Multiple ways of modelling RDF lists have been proposed. The RDF Schema (RDFS) recommendation [6] defines several container classes to represent

¹ See e.g. <https://stackoverflow.com/questions/16223095/sparql-queries-over-collection-and-rdfcontainers> and <http://www.snee.com/bobdc.blog/2014/04/rdf-lists-and-sparql.html>.

collections: `rdf:Bag` to contain unordered elements; `rdf:Alt` for “alternative” containers whose typical processing will be to select one of its members; and `rdf:Seq` to contain elements ordered by the numerical order of the container membership properties. [6] also defines a collection vocabulary to describe closed collection that can have no more members, through the class `rdf:List` and the properties `rdf:first`, `rdf:rest`, and `rdf:nil`. In the more recent JSON-LD [24], ordered lists like `"@list": ["bob", "alice", "carol"]` have equivalent representations as `rdf:List`. Similarly, the RDF 1.1 Turtle [2] syntax allows for the specification of `rdf:List` instances, e.g. `:a :b ("bob" "alice" "carol")`. Besides W3C standards, various ontology design patterns [10], like the Sequence Ontology Pattern² (SOP), address the task of representing RDF lists. About relevant previous work on benchmarks, the Semantic Web community has developed a number of them for evaluating the performance of SPARQL engines. The Berlin SPARQL Benchmark (BSBM) [5] generates benchmark data about exploring products and analyzing consumer reviews. The Lehigh University Benchmark (LUBM) [13] does so on data about universities, departments, professors and students. SP²Bench [22] enables comparison of SPARQL optimization strategies, an estimation of their generality, and the prediction of their benefits in real-world scenarios; it includes a benchmark data generator based on the DBLP bibliographic database [16]. Similarly, the DBpedia SPARQL benchmark [19] proposes human-written queries that execute against non-relational schemas. The Waterloo SPARQL Diversity Test Suite (WatDiv) focuses on measuring “how an RDF data management system performs across a wide spectrum of SPARQL queries with varying structural characteristics and selectivity classes” [1]. Other approaches like Linked SPARQL queries (LSQ) [21] focus specifically on benchmark queries from SPARQL query logs, but typically do not generate data to run these queries on. More recently, frameworks to integrate and compare various benchmarks, such as IGUANA [7]³, have emerged. Other, more pragmatic approaches propose ad-hoc benchmarks supporting specific applications [25] or SPARQL features, like federation [12]. To the best of our knowledge, none of these benchmarks address specifically the evaluation of RDF lists.

3 The List.MID Benchmark

In this Section we describe the List.MID benchmark. First, we summarize the various modeling alternatives for lists in RDF (Sect. 3.1); for a complete survey, see [8]. Second, we implement these modeling alternatives in a benchmark data generator that creates RDF datasets rich in lists from a large MIDI data collection (Sect. 3.2). Finally, we propose a set of SPARQL queries to retrieve RDF list data according to the different modeling alternatives (Sect. 3.3).

All the List.MID benchmark resources are available online in a GitHub repository at <https://github.com/midi-ld/List.MID>. The benchmark is licensed under

² <http://ontologydesignpatterns.org/wiki/Submissions:Sequence>.

³ See also <https://github.com/dice-group/triplestore-benchmarks>.

Table 1. Links to key resources of the `List.MID` benchmark.

Meroño-Peñuela, A. and Daga, E. (2019). <i>List.MID: A MIDI-based benchmark for evaluating RDF lists</i> . https://doi.org/10.5281/zenodo.3265139	
Resource	Link
GitHub repository	https://github.com/midi-ld/List.MID
Benchmark queries	https://github.com/midi-ld/List.MID/tree/master/queries
Benchmark example data	https://github.com/midi-ld/List.MID/tree/master/data
Benchmark generation data	https://github.com/midi-ld/sources https://github.com/albertmeronyo/awesome-midi-sources
Full dump download	https://github.com/midi-ld/List.MID/archive/master.zip
Zenodo	https://zenodo.org/record/3265139#.XRoYXXUzaV4
Figshare	https://figshare.com/articles/List.MID_A_MIDI-Based_Benchmark_for_Evaluating_RDF_Lists/8426912
Datahub	https://datahub.ckan.io/dataset/list-mid-a-midi-based-benchmark-for-evaluating-rdf-lists

the Creative Commons Attribution-ShareAlike 4.0 International⁴ (CC BY-SA 4.0) license. The benchmark is deposited in Zenodo, Figshare, and Datahub. The open availability of the benchmark in these platforms allows for fast and frictionless contributions from other parties. All relevant URLs and canonical citation are shown in Table 1.

3.1 Modeling Lists in RDF

There are various models for representing a sequence, a *finite collection of ordered elements*, in RDF. In this section we offer a summary of such models and their properties, recalling the research in [8]. These models were surveyed by selecting them from the following sources, including W3C standards⁵ ontology design patterns [10], resource track papers in ISWC (e.g. [3, 18]), and lookups of relevant terms in Linked Open Vocabularies [28]. For a further detail and a description of the surveying methodology, see [8].

RDF Sequences. The RDF Schema (RDFS) recommendation [6] defines the container classes `rdf:Bag`, `rdf:Alt`, `rdf:Seq` to represent collections. Since `rdf:Bag` is intended for unordered elements, and `rdf:Alt` for “alternative” containers whose typical processing will be to select one of its members, these two models do not fit our sequence definition, and thus we do not include them among our candidates. Conversely, we do consider *RDF Sequences*: collections represented by `rdf:Seq` and ordered by the properties `rdf:_1`, `rdf:_2`, `rdf:_3`, ... instances of the class `rdfs:ContainerMembershipProperty` (see Fig. 1).

Properties. RDF Sequences indicate membership through various *properties*, which are used in triples in *predicate position*. Ordering of elements is *absolute* in such predicates through an integer index after an underscore (“_”).

⁴ <https://creativecommons.org/licenses/by-sa/4.0/>.

⁵ <https://www.w3.org/standards/>.

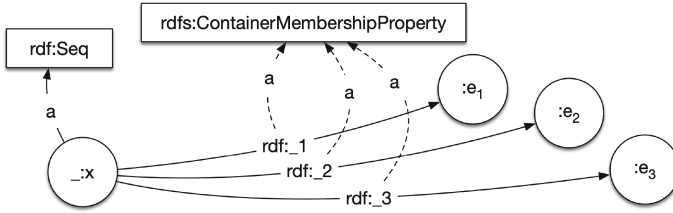


Fig. 1. The RDF Sequence model.

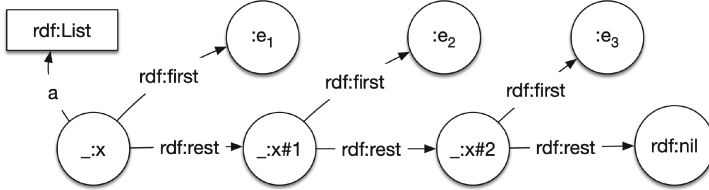


Fig. 2. The RDF List model.

RDF Lists. The RDFS recommendation [6] also defines a vocabulary to describe closed collections or *RDF Lists*. Such lists are members of the class `rdf:List`. Resembling LISP lists, every element of an RDF List is represented by two triples: $\langle L_k \text{ rdf:first } E_k \rangle$, where E_k is the k -th element of the list; and $\langle L_k \text{ rdf:rest } L_{k+1} \rangle$, representing the rest of the list (in particular, `rdf:nil` to end the list) (see Fig. 2).

Properties. RDF Lists indicate membership through the use of a *unique property* `rdf:first` in *predicate position*. Ordering of elements is *relative* to the use of the `rdf:rest` property, and given by the sequential forward traversal of the list.

URI-Based Lists. A more practical approach followed by many RDF datasets [3, 18] consists of establishing list membership through an explicit property or class membership, and assigning order by a unique identifier embedded in the element's URI. For instance, the triple $\langle \text{http://ld.zdb-services.de/resource/1480923-0} \rangle \text{ a } \langle \text{http://purl.org/ontology/bibo/Periodical} \rangle$ indicates that the subject belongs to a list of periodicals with list order 14809234; the triple $\langle \text{http://purl.org/midi-ld/piece/8cf9897/track00} \rangle \text{ midi:hasEvent } \langle \text{http://purl.org/midi-ld/piece/8cf9897/track00/event0006} \rangle$ identifies the 7th event in a MIDI track [18] (see Fig. 3).

Properties. URI-based lists indicate membership through the use of *class membership* or through *properties*. Order is *absolute* and given by URI-embedded sequential identifiers.



Fig. 3. The URI-based list model.

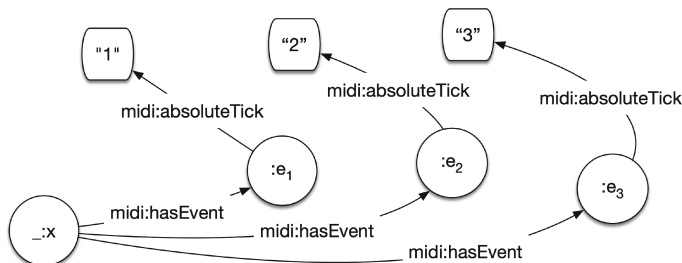


Fig. 4. The Number-based list model.

Number-Based Lists. Another practical model, used e.g. in the Sequence Ontology/Molecular Sequence Ontology (MSO) [9],⁶ also uses class membership or object properties to specify the elements that belong to a list, but use a *literal value* in a separate property to indicate order. For instance, the triple `<http://purl.org/midi-ld/piece/8cf9897/track00> midi:hasEvent <http://purl.org/midi-ld/piece/8cf9897/track00/event0006>` indicates that the object belongs to a list of events; and the additional triple `<http://purl.org/midi-ld/piece/8cf9897/track00/event0006> midi:absoluteTick 6` indicates that the event has index 6 (see Fig. 4).

Properties. Number-based lists indicate membership through the use of *class membership* or through *properties*. Order is *absolute* and given by an integer index in a literal as an object of an additional property.

Timestamp-Based Lists. Similarly to Number-based lists, other lists modeled by e.g. the Simple Event Model (SEM) [27], use timestamp markers instead of integer indexes to indicate the time in which the element of the list occurs. This is particularly useful in event-based applications, in which order clashes in the list are of lesser importance, as long as the timestamp order is preserved. For instance, the triple `<http://purl.org/midi-ld/piece/8cf9897535d79e68c33a3076aa06d073/track00/event0006> midi:absoluteTime 0e+00` indicates that the 7th event occurs at the start of the list, possibly simultaneously with other events (see Fig. 5).

⁶ <https://github.com/The-Sequence-Ontology/Specifications/blob/master/gff3.md>.

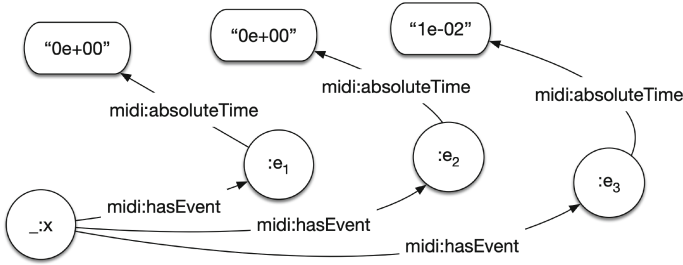


Fig. 5. The Timestamp-based list model.

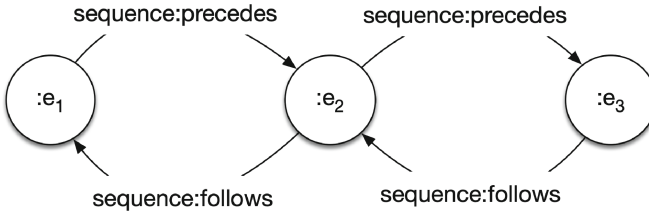


Fig. 6. The Sequence Ontology Pattern model.

Properties. Timestamp-based lists indicate membership through the use of *class membership* or through *properties*. Order is *absolute* and given by a timestamp in a literal as an object of an additional property.

Sequence Ontology Pattern. A number of models use RDF, RDFS and OWL to model sequences in domain specific ways. For example, the Time Ontology [14] and the Timeline Ontology⁷ offer a number of classes and properties to model temporality and order, including timestamps (see Sect. 3.1), but importantly also before/after relations. The *Sequence Ontology Pattern*⁸ (SOP) is an ontology design pattern [10] that “represents the ‘path’ cognitive schema, which underlies many different conceptualizations: spatial paths, time lines, event sequences, organizational hierarchies, graph paths, etc.”. We select SOP as an abstract model representing this group of list models (see Fig. 6).

Properties. SOP lists indicate list membership through *properties*. Order is *relative* and given by the sequential forward or backward traversal of the sequence.

3.2 Data Generator

The first component of the List.MID benchmark is an algorithm to generate RDF datasets with lists according to the modeling patterns discussed above.

⁷ <http://motools.sourceforge.net/timeline/timeline.html>.

⁸ <http://ontologydesignpatterns.org/wiki/Submissions:Sequence>.

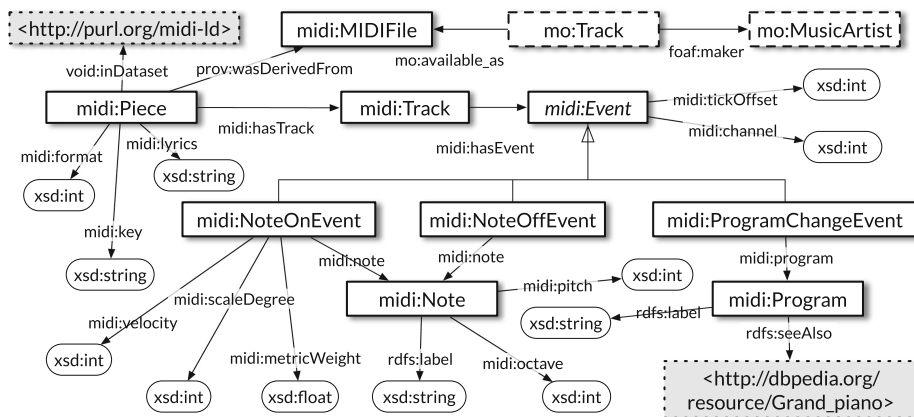


Fig. 7. Excerpt of the MIDI ontology. Tracks contain lists of sequential MIDI events.

The source code and all documentation are available on GitHub at <https://github.com/midi-ld/List.MID>.

In order to root our benchmark within real-world data, we propose to generate data using MIDI files [26], a symbolic music encoding, as a basis. The reason for this is that MIDI files, and symbolic music notations in general, must encode musical events (the start of a note, the end of a note, the switching of one instrument for another, etc.) in strict sequential order to preserve musical coherence. Consequently, we use the `midir2rdf` algorithm proposed in [17] to generate RDF graphs from MIDI files; and we extend this algorithm here in order to encode RDF lists of musical events supporting the list data models discussed in Sect. 3.1.

Figure 7 shows an excerpt of the MIDI ontology used by the original `midi2rdf` algorithm. The relevant elements here are `midi:Track`, each containing a sequence of related musical events (e.g. notes played by one single instrument); and `midi:Event`, each representing a musical event that happens in a strict order within the track (e.g. the start of a note, the end of a note). For more details on MIDI event encoding see [17, 18, 26].

The original `mid2rdf` algorithm generates *implicit* lists of events by encoding their order in the URI of the event (e.g. `ex:track00/event02` happens immediately before `ex:track00/event01` and immediately after `ex:track00/event01`), and hence adhering to the *URI-based Lists* pattern discussed in Sect. 3.1. We extend this generation to the remaining patterns.

Usage. The first step is to find a MIDI file with the desired *list size*. The MIDI Linked Data cloud API⁹ incorporates a query¹⁰ to retrieve all track sizes in number of events in descending order from the dataset [18]. Since this query is expensive, we include a resulting dump in the benchmark. An inspection of this

⁹ See <http://grlc.io/api/midi-ld/queries/>.

¹⁰ http://grlc.io/api/midi-ld/queries/#/default/get_events_count_per_track_piece.

result allows users to select a MIDI identifier of the chosen size; this identifier can be used in a second query¹¹ to download an RDF dump for the MIDI file. This dump can be transformed into an input MIDI file with the included `rdf2midi` command [17].

Once the chosen input MIDI file has been generated, the `midi2rdf` CLI tool of the `List.MID` benchmark can be used to generate its RDF graph according to the requested list pattern. The syntax is:

```
midi2rdf [-h]
          [--format [{xml,n3,turtle,nt,pretty-xml,trix,trig,nquads,
                    json-ld}]]
          [--gz] [--order [{uri,prop_number,prop_time,seq,list,
                           sop}]]
          [--version]
          filename [outfile]
```

The relevant introduced argument is `order`, which lets the user select the RDF list modeling to use for data generation. The mapping for the values of this argument with the patterns of Sect. 3.1 is: *RDF Sequences* \rightarrow `seq`, *RDF Lists* \rightarrow `list`, *URI-based Lists* \rightarrow `uri`, *Number-based Lists* \rightarrow `prop_number`, *Timestamp-based Lists* \rightarrow `prop_time`, *Sequence Ontology Pattern* \rightarrow `sop`. For example, to generate benchmark data of a preselected <http://purl.org/midi-ld/pattern/bc7d9c25f81a4d90c000c30b6efc887d> MIDI with 16,638 list elements using the RDF List pattern, we do:

```
midi2rdf --format turtle --order list
         bc7d9c25f81a4d90c000c30b6efc887d.mid benchmark.ttl
```

The output `benchmark.ttl` file is ready to be used in a standard compliant RDF store. As shown in the syntax above, the benchmark is agnostic with respect to serialization formats, and the most frequent (including JSON-LD) are supported.

3.3 Queries

In this section we propose a set of SPARQL query templates for retrieval of elements of lists, according to the patterns described in Sect. 3.1. Since the full coverage of list operations in SPARQL is cumbersome, here we restrict ourselves to typical *data publishing* functionality. Therefore, we consider minimal and atomic *read* operations; and we do not consider *management* operations (edit, merge, split of lists, etc.). The implementation of management operations is possible, but depend on implementations of read operations; thus, we focus here on read operations, and leave management operations for future work.

Therefore, the currently supported operations in `List.MID` consist of (a) orderly retrieve all elements of the list; and (b) access the n -th element of the list.

¹¹ http://grlc.io/api/midi-ld/queries/#/default/get_pattern_graph.

In order to systematically do this in datasets following one of the RDF list modeling patterns (Sect. 3.1), we include corresponding SPARQL query templates in the benchmark. The queries can be found online in the GitHub repository of the benchmark,¹² and are summarized in Table 2.

Table 2. SPARQL query templates of the benchmark.

ID	RDF list model	Access	SPARQL
Q1	RDF Sequences	Full list	WHERE { [] a midi:Track ; midi:hasEvents [?seq ?event] . BIND (xsd:integer(SUBSTR(str(?seq), 45)) AS ?index) } ORDER BY ?index
Q2	RDF Sequences	n -th item	WHERE { [] a midi:Track ; midi:hasEvents [?seq ?event] . BIND (xsd:integer(SUBSTR(str(?seq), 45)) AS ?index) } ORDER BY ?index OFFSET n LIMIT 1
Q3	RDF Lists	Full list	WHERE { [] a midi:Track ; midi:hasEvents ?events . ?events rdf:rest*/rdf:first ?event . BIND (xsd:integer(SUBSTR(str(?event), 77)) AS ?id) } ORDER BY ?id
Q4	RDF Lists	n -th item	WHERE { [] a midi:Track ; midi:hasEvents ?events . ?events rdf:rest*/rdf:first ?event . BIND (xsd:integer(SUBSTR(str(?event), 77)) AS ?id) } ORDER BY ?id OFFSET n LIMIT 1
Q5	URI-based	Full list	WHERE { [] a midi:Track ; midi:hasEvent ?event . BIND (xsd:integer(SUBSTR(str(?event), 77)) AS ?id) } ORDER BY ?id
Q6	URI-based	n -th item	WHERE { [] a midi:Track ; midi:hasEvent ?event . BIND (xsd:integer(SUBSTR(str(?event), 77)) AS ?id) } ORDER BY ?id OFFSET n LIMIT 1
Q7	Number-based	Full list	WHERE { [] a midi:Track ; midi:hasEvent ?event . ?event midi:absoluteTick ?tick . } ORDER BY ?tick
Q8	Number-based	n -th item	WHERE { [] a midi:Track ; midi:hasEvent ?event . ?event midi:absoluteTick ?tick . } ORDER BY ?tick OFFSET n LIMIT 1
Q9	Timestamp-based	Full list	WHERE { [] a midi:Track ; midi:hasEvent ?event . ?event midi:absoluteTick ?tick . } ORDER BY ?tick
Q10	Timestamp-based	n -th item	WHERE { [] a midi:Track ; midi:hasEvent ?event . ?event midi:absoluteTick ?tick . } ORDER BY ?tick OFFSET n LIMIT 1
Q11	Sequence Ontology Pattern	Full list	WHERE { [] a midi:Track ; midi:hasEvent ?event . ?event sequence:precedes? ?next.event . ?next.event sequence:follows? ?event . BIND (xsd:integer(SUBSTR(str(?event), 77)) AS ?id) } ORDER BY ?time
Q12	Sequence Ontology Pattern	n -th item	WHERE { [] a midi:Track ; midi:hasEvent ?event . ?event sequence:precedes? ?next.event . ?next.event sequence:follows? ?event . BIND (xsd:integer(SUBSTR(str(?event), 77)) AS ?id) } ORDER BY ?time OFFSET n LIMIT 1

4 Experiments and Reuse

In this Section we discuss current use and potential for reuse of our proposed benchmark in research.

4.1 First Experiment

The List.MID benchmark has been used in a first Semantic Web research experiment [8]. The purpose of this work is to understand the impact of different

¹² See <https://github.com/midi-ld/List.MID>.

RDF list modeling patterns (see Sect. 3.1) in the performance and availability of sequential retrieval of Linked Data. This crucially includes basic list operations such as orderly getting all elements of the list; randomly accessing one element of the list; and randomly accessing a sublist contained in a list. The most important findings quantify the impact of different list modeling choices in retrieval; and show that this impact is triplestore-invariant to a great degree. For a full report on such experiments, see [8]. These experiments demonstrate the applicability and usefulness of the benchmark, and can be easily reproduced with `List.MID` and the supplementary materials at https://www.dropbox.com/sh/m98115y7ah2nqcv/AAAxkGsWuiPaLf6X7c_uM0yWa.

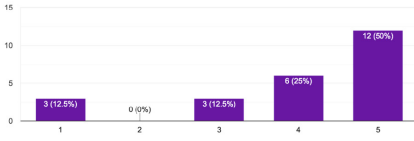
4.2 Online Survey

Since the `List.MID` benchmark is a *new resource* for the Semantic Web community, we discuss here evidence for potential adoption. To gather such evidence, we perform an **online survey** in which we directly ask the community of potential adopters 8 questions regarding their background, relevance, and interest in benchmarking RDF lists. The online survey was distributed in the **semantic-web** and **public-lod** public mailing lists of the W3C; and in the internal mailing lists of the affiliation labs of the authors. In total we gathered $N = 24$ responses. The survey can be found online.¹³ Fig. 8 shows the results.

Except for question 3 (Fig. 8c), all questions ask the respondents to quantify the agreement with the statement made from 1 (absolutely disagree) to 5 (absolutely agree), being 3 a neutral response (no agree nor disagree). In the first two questions (Fig. 8a, 8b) we assess the background of the respondents, finding that 75% of them have experience in modeling and publishing RDF, and 54.2% have experience or interest in RDF benchmarking; and thus proving adequacy of the population sample. Among the various RDF list modeling practices (Fig. 8c), `rdf:List` is the most popular, known by 2/3 of the respondents. Other practices like `rdf:Seq` (37.5%), implicit RDF elements as proxies (URIs, properties, etc.; 25%) and ontology design patterns (20.8%) are also familiar. Some respondents express here other less known approaches that could fit the broader categories (e.g. using a `xyz:nextitem`). Figure 8d shows that the community is divided in whether expressing lists in RDF is a real need; conversely, Fig. 8e shows that the impact of list modeling choices in query performance is a real concern (0% disagree; 83.3% agree or strongly agree). Figure 8f signals that current benchmarks might be missing coverage for RDF lists (only 8.3% find them somewhat covered). Most importantly, the community feels the **need of new benchmarks specifically designed for the evaluation of RDF lists** (Fig. 8g, 70.9%). Asking directly on their interest as potential users of a new RDF list benchmark, the community seems divided (Fig. 8h), although this could be attributed to different research interests. **29.1% of the respondents would be interested in reusing an RDF list benchmark like the one here proposed.**

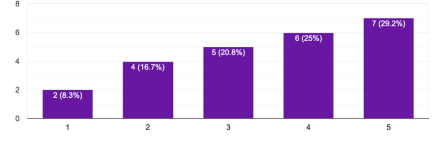
¹³ See <https://forms.gle/SwkCdFFFVGXWCgCp7>.

I have experience in modeling and publishing RDF datasets as Linked Data
24 responses



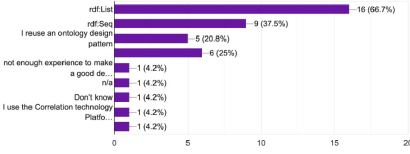
(a)

I have experience or interest in benchmarking triplestores
24 responses



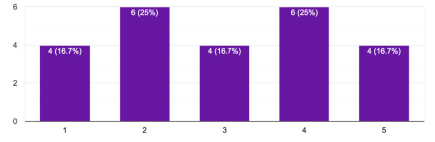
(b)

Which of the following RDF list modelling practices are you familiar with?
(check as many as needed)
24 responses



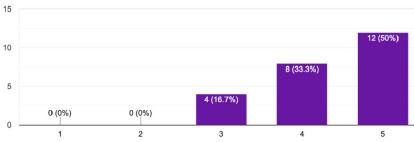
(c)

I have encountered the need for modeling the resources in my RDF datasets as ordered lists of resources
24 responses



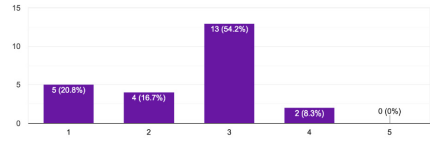
(d)

I think that different list modelling choices might have an impact in query performance
24 responses



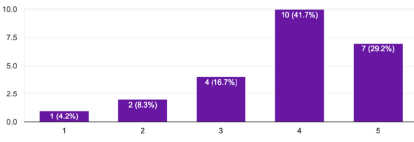
(e)

I think that current datasets and queries in benchmarks cover RDF lists appropriately
24 responses



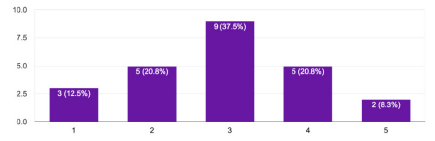
(f)

I think that the Semantic Web community would benefit from a benchmark specifically designed for the evaluation of RDF lists
24 responses



(g)

I would be interested in using a benchmark specifically designed for the evaluation of RDF lists in my own systems/data
24 responses



(h)

Fig. 8. Results of the online survey

5 Conclusions

Lists are fundamental data structures in computer science, and various models implementing them in the Semantic Web —using RDF, standards, and community best practices— have been proposed. So far, studying the differences, and trade-offs, in features and performance of these RDF list models has been done only in a superficial and exploratory manner. To address this, in this paper we contribute two important findings. First, we show evidence that the Semantic

Web community feels the need for a benchmark specifically designed for the evaluation of RDF lists; and that a number of researchers would be interested in reusing such a benchmark. Second, we propose the benchmark to precisely address this issue, enabling a systematic and principled way of generating, and querying, RDF list data from real-world datasets according to dominant RDF list models in the Semantic Web. We feel that, by adopting this benchmark, researchers will be able to understand better the implications of different list-modeling practices; and developers will find a first building block to construct more varied and performant solutions for RDF lists. We expect both researchers and developers to fundamentally contribute, through their research and software, in making the `List.MID` benchmark better.

This room for improvement can be observed from various prisms. First, in next iterations we will include more real-world use cases and base datasets from which to generate the benchmark data. Similarly, we will include additional list operations regarding list management, such as inserting a new element, and swapping two elements, taking inspiration from array operations in programming [11]. If more, alternative models for modeling RDF lists become a need for our users, we will support them too. Finally, we will continue working to deploy a more automated and usable infrastructure and tools for RDF list benchmarking.

Acknowledgements. This work was partially funded by the CLARIAH project of the Dutch Science Foundation (NWO). We are grateful to all participants of the online survey.

References

1. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified stress testing of RDF data management systems. In: Mika, P., et al. (eds.) ISWC 2014. LNCS, vol. 8796, pp. 197–212. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11964-9_13
2. Beckett, D., Berners-Lee, T., Prud'hommeaux, E., Carothers, G.: RDF 1.1 turtle - terse RDF triple language. Technical report, World Wide Web Consortium (2014). <https://www.w3.org/TR/turtle/>
3. Beek, W., Rietveld, L., Bazoobandi, H.R., Wilemaker, J., Schlobach, S.: LOD laundromat: a uniform way of publishing other people's dirty data. In: Mika, P., et al. (eds.) ISWC 2014. LNCS, vol. 8796, pp. 213–228. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11964-9_14
4. Berners-Lee, T., Hendler, J., Lassila, O., et al.: The semantic web. *Sci. Am.* **284**(5), 28–37 (2001)
5. Bizer, C., Schultz, A.: The Berlin SPARQL benchmark. *Int. J. Semant. Web Inf. Syst.* **5**(2), 1–24 (2009)
6. Brickley, D., Guha, R.: RDF schema 1.1. Technical report, World Wide Web Consortium (2014). <https://www.w3.org/TR/rdf-schema/>
7. Conrads, F., Lehmann, J., Saleem, M., Morsey, M., Ngonga Ngomo, A.-C.: IGUANA: a generic framework for benchmarking the read-write performance of triple stores. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10588, pp. 48–65. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68204-4_5

8. Daga, E., Meroño-Peñuela, A., Motta, E.: Modelling and querying lists in RDF. A pragmatic study. In: 3rd Workshop on Querying and Benchmarking the Web of Data (QuWeDa 2019), ISWC 2019 (2019)
9. Eilbeck, K., et al.: The sequence ontology: a tool for the unification of genome annotations. *Genome Biol.* **6**(5), R44 (2005)
10. Gangemi, A.: Ontology design patterns for semantic web content. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 262–276. Springer, Heidelberg (2005). https://doi.org/10.1007/11574620_21
11. Gopan, D., Reps, T., Sagiv, M.: A framework for numeric analysis of array operations. *SIGPLAN Not.* **40**(1), 338–350 (2005). <http://doi.acm.org.vu-nl.idm.oclc.org/10.1145/1047659.1040333>
12. Görlitz, O., Thimm, M., Staab, S.: SPODGE: systematic generation of SPARQL benchmark queries for linked open data. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012. LNCS, vol. 7649, pp. 116–132. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35176-1_8
13. Guo, Y., Pan, Z., Heflin, J.: LUBM: a benchmark for OWL knowledge base systems. *J. Web Semant. Sci. Serv. Agents World Wide Web* **3**(2), 158–182 (2005)
14. Hobbs, J.R., Pan, F.: Time ontology in OWL. W3C working draft **27**, 133 (2006)
15. Hopcroft, J.E., Ullman, J.D.: Data structures and algorithms (1983)
16. Ley, M.: The DBLP computer science bibliography: evolution, research issues, perspectives. In: Laender, A.H.F., Oliveira, A.L. (eds.) SPIRE 2002. LNCS, vol. 2476, pp. 1–10. Springer, Heidelberg (2002). https://doi.org/10.1007/3-540-45735-6_1
17. Meroño-Peñuela, A., Hoekstra, R.: The song remains the same: lossless conversion and streaming of MIDI to RDF and back. In: Sack, H., Rizzo, G., Steinmetz, N., Mladenović, D., Auer, S., Lange, C. (eds.) ESWC 2016. LNCS, vol. 9989, pp. 194–199. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-47602-5_38
18. Meroño-Peñuela, A., et al.: The MIDI linked data cloud. In: d’Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10588, pp. 156–164. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68204-4_16
19. Morsey, M., Lehmann, J., Auer, S., Ngonga Ngomo, A.-C.: DBpedia SPARQL benchmark – performance assessment with real queries on real data. In: Aroyo, L., et al. (eds.) ISWC 2011. LNCS, vol. 7031, pp. 454–469. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25073-6_29
20. Reingold, E.M., Nievergelt, J., Deo, N.: Combinatorial Algorithms: Theory and Practice. Prentice Hall College Div, Englewood Cliffs (1977)
21. Saleem, M., Ali, M.I., Hogan, A., Mehmood, Q., Ngomo, A.-C.N.: LSQ: the linked SPARQL queries dataset. In: Arenas, M., et al. (eds.) ISWC 2015. LNCS, vol. 9367, pp. 261–269. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-25010-6_15
22. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP²Bench: a SPARQL performance benchmark. In: 2009 IEEE 25th International Conference on Data Engineering, ICDE 2009, pp. 222–233. IEEE (2009)
23. Schreiber, G., Raimond, Y.: RDF 1.1 primer. Technical report, World Wide Web Consortium (2014). <https://www.w3.org/TR/rdf11-primer/>
24. Sporny, M., Kellogg, G., Lanthaler, M.: JSON-LD 1.0. Technical report, World Wide Web Consortium (2014). <https://www.w3.org/TR/2014/REC-jsonld-20140116/>
25. Thakker, D., Osman, T., Gohil, S., Lakin, P.: A pragmatic approach to semantic repositories benchmarking. In: Aroyo, L., et al. (eds.) ESWC 2010. LNCS, vol. 6088, pp. 379–393. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-13486-9_26

26. The MIDI Manufacturers Association: MIDI 1.0 detailed specification. Technical report, Los Angeles, CA (1996–2014). <https://www.midi.org/specifications>
27. Van Hage, W.R., Malaisé, V., Segers, R., Hollink, L., Schreiber, G.: Design and use of the simple event model (SEM). *Web Semant. Sci. Serv. Agents World Wide Web* **9**(2), 128–136 (2011)
28. Vandenbussche, P.Y., Ateazing, G.A., Poveda-Villalón, M., Vatant, B.: Linked open vocabularies (LOV): a gateway to reusable semantic vocabularies on the web. *Semant. Web* **8**(3), 437–452 (2017)