The Journal of Open Source Software

hdt-rs: A Rust library for the Header Dictionary Triples binary RDF compression format

Konrad Höffner [1*¶] **and Tim Baccaert**[2*]

1 Institute for Medical Informatics, Statistics, and Epidemiology, Medical Faculty, Leipzig University **2** Independent Researcher, Belgium ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.05114

Software

- Review
- Repository
- Archive

Editor: Øystein Sørensen

Reviewers:

- @remram44
- @lazear

Submitted: 17 January 2023
Published: 27 April 2023

Summary

We present the Rust library hdt-rs (named "hdt" in the context of Rust libraries, such as on crates.io) for the Header Dictionary Triples (HDT) binary RDF compression format. This allows the writing of high-performance Rust applications that load and query HDT datasets using triple patterns. Existing Rust applications that use the Sophia library (Champin, 2020) can easily and greatly reduce their RAM usage by using the provided Sophia HDT adapter.

Preliminaries

RDF

The *Resource Description Framework* (RDF) is a data model that represents information using *triples*, each of which consists of a *subject*, a *predicate*, and an *object*. A set of triples is called an *RDF graph*, where the subjects and objects can be visualised as nodes and the predicates as labelled, directed edges. Predicates are always *IRIs* (Internationalised Resource Identifiers), which are generalisations of a URIs that allow additional characters. Subjects and objects can also be *blank nodes* and objects can also be *literals*. There are several text-based RDF serialisation formats with different compromises between verbosity, ease of automatic processing, and human readability. For example, the N-Triples representation of the fact *"the mayor of Leipzig is Burkhard Jung"* from DBpedia (Lehmann et al., 2015) is:

```
<http://dbpedia.org/resource/Leipzig> <http://dbpedia.org/ontology/mayor> \
    <http://dbpedia.org/resource/Burkhard_Jung> .
```

Triple Patterns

Triple patterns match a subset of a graph. Each part of the pattern is either a constant or a variable, resulting in eight different types. We denote the pattern type with all constants as SPO (*subject-predicate-object*, matches one or zero triples) and the type with all variables as ??? (matches all triples in the graph). The other triple patterns are denoted analogously.

Header Dictionary Triples

While text-based RDF serialisation formats can be read by humans, they are too verbose to be practical for large graphs. The serialised size of a graph can be drastically lowered by using the Header Dictionary Triples binary RDF format, which can be loaded into memory in compressed form while still allowing for efficient queries. The *header* contains metadata as uncompressed RDF that describes the dataset. The *dictionary* stores all the *RDF terms* (IRIs, literals, and blank nodes) in the dataset in compressed form using front-coding (Witten et al., 1999), and

assigns a unique numerical identifier (ID) to each of them. This allows the *triples* component to store the adjacency matrix of the graph using those IDs in compressed form.

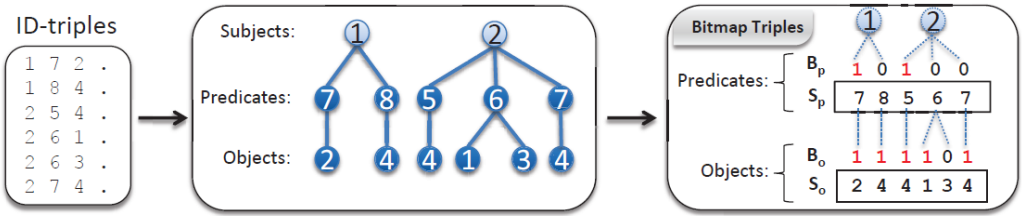

Figure 1: The Bitmap Triples structure represents the adjacency matrix of the RDF graph as trees. Image source and further information in Martínez-Prieto et al. (2012).

All patterns with constant subject (SPO, SP?, SO?, and S??) as well as the one with all variables (???) are answered using the Bitmap Triples structure (see Figure 1), while the other patterns use the HDT *Focused on Querying* (HDT-FoQ) extension, see Figure 2. As HDT is a complex format, we recommend referring to Martínez-Prieto et al. (2012) and Fernández et al. (2013) for comprehensive documentation.

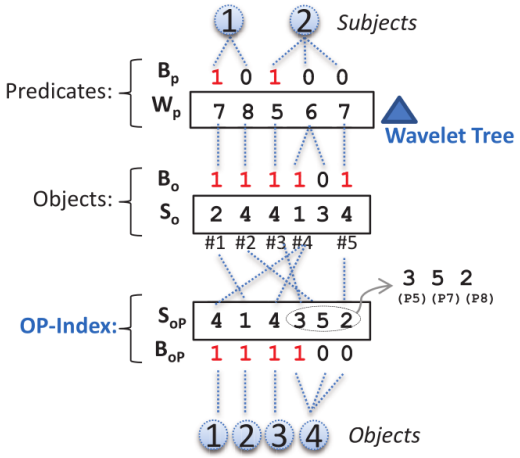

Figure 2: The HDT *Focused on Querying* (HDT-FoQ) extension allows efficient queries with ?PO, ?P?, and ??O patterns. Image source and further information in Martínez-Prieto et al. (2012).

Statement of need

Semantic Web technologies have been adopted by major tech companies in recent years but widespread use is still inhibited by a lack of freely available performant, accessible, robust, and adaptable tooling (Hitzler, 2021). SPARQL endpoints provide a standard publication channel and API to any RDF graph but they are not suitable for all use cases. On small graphs, there is a large relative overhead in both memory and CPU resources. On large graphs, on the other hand, query complexity and shared access may cause an overload of the server, causing delayed or missed responses. The long-term availability of SPARQL endpoints is often compromised (Buil-Aranda et al., 2013), which impacts all applications that depend on them.

To insulate against such problems, Semantic Web applications can integrate and query an RDF graph using libraries such as Apache Jena (Carroll et al., 2004) for Java, RDFlib (Swartz et al., 2023) for Python, librdf (Beckett et al., 2015) for C, or Sophia (Champin, 2020) for Rust. However these libraries do not scale to large RDF graphs due to their excessive memory

usage, see Figure 3. To complement hdt-cpp (Arias et al., 2023) and hdt-java (Torres et al., 2022), we implement HDT in Rust, which is a popular modern, statically typed high-level programming language that allows writing performant software while ensuring memory safety, which meets the challenges of Semantic Web adoption. hdt-rs is used by the RDF browser RickView (Höffner, 2023) via the included Sophia adapter to publish large graphs, for example LinkedSpending (Höffner et al., 2016) at https://linkedspending.aksw.org, which previously suffered from frequent downtime when based on a SPARQL endpoint.

Benchmark

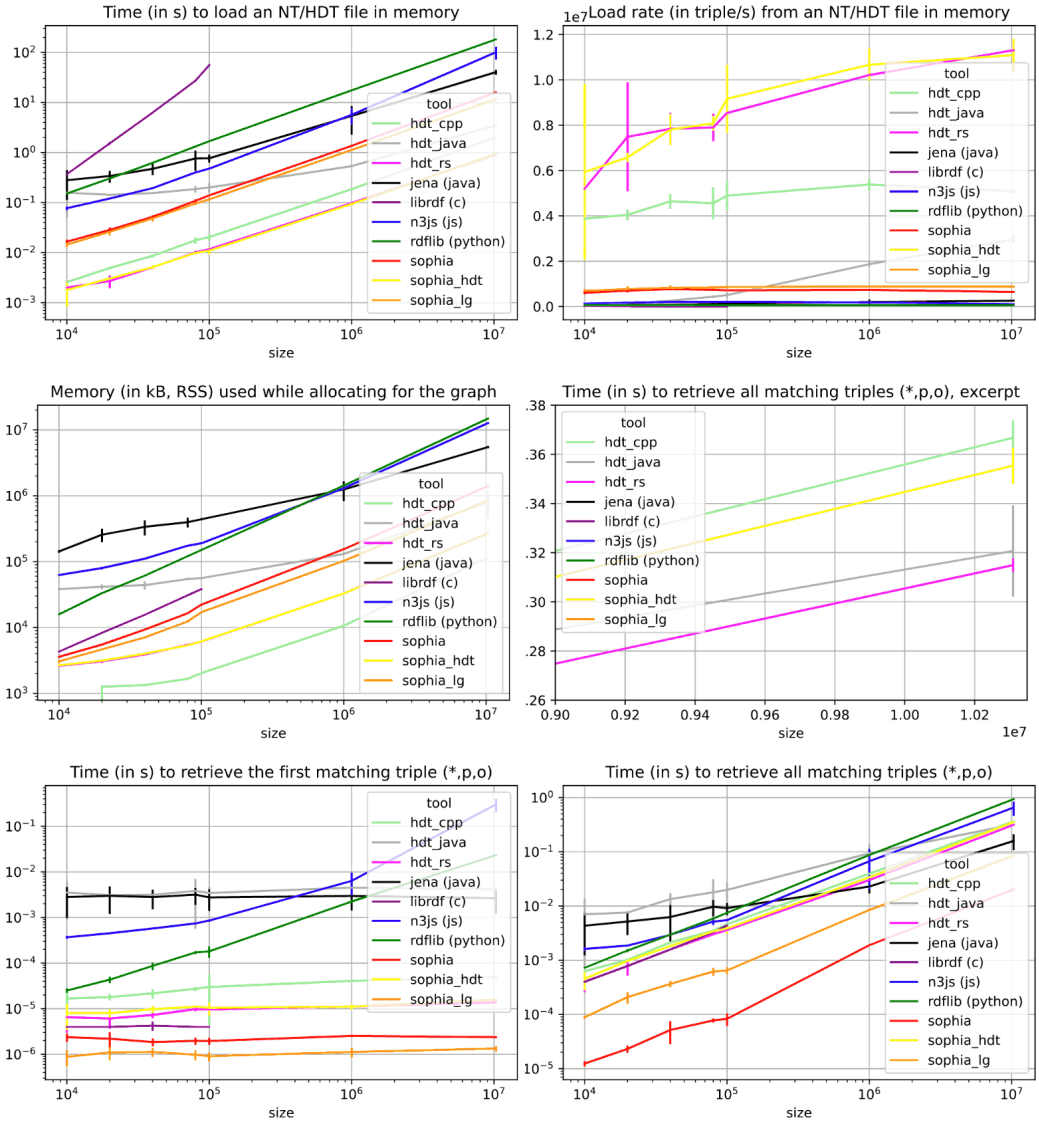

Figure 3: Dataset load time, memory usage (resident set size), and ?PO triple pattern query time of different RDF libraries on an Intel i9-12900k CPU based on the benchmark suite of Champin (2020). librdf was not benchmarked on 10^6 triples and beyond due to graph loading times exceeding several hours. hdt-java produces `DelayedString` instances that are converted to strings to account for the time that would otherwise be spent later. The index files created by hdt-java and hdt-cpp produce are deleted before each run. Versions: Apache Jena 4.6.1, n3.js 1.6.3, librdf 1.0.17, RDFlib 6.2.0, sophia 0.8.0-alpha, hdt-rs 0.0.13-alpha, hdt-java 3.0.9, hdt-cpp master fbcb31a, OpenJDK 19, Node.js 16.18.0, clang 14.0.6, Python 3.10.8, rustc 1.69.0-nightly (target-cpu=native), GCC 12.2.1.

Table 1: Rounded averages over four runs on the complete person data dataset containing 10310105 triples (rightmost points in Figure 3) serialised as a 90 MB HDT and 1.2 GB RDF Turtle file. Sorted by memory usage of the graph. For better comparison, results for hdt_java are given both with and without calling `DelayedString::toString` on the results. The measured values are subject to considerable fluctuations, see the vertical bars in Figure 3.

Library	Memory in MB	Load Time in ms	Query Time in ms
hdt_cpp	**112**	1985	362
sophia_hdt	263	930	355
hdt_rs	264	**912**	315
hdt_java (DelayedString)	738	3170	**214**
hdt_java (String)	785	3476	**321**
sophia_lg	**834**	**11656**	85
sophia	1371	15990	**20**
jena (java)	5352	40400	159
n3js (js)	12404	100820	654
rdflib (python)	14481	182002	940
librdf (c)	–	–	–

Table 1 demonstrates the advantage of HDT libraries in memory usage, with hdt_cpp using only 112 MB compared to 834 MB for the most memory-efficient non-HDT RDF library tested, sophia_lg (LightGraph). When comparing only Rust libraries, sophia_lg still uses over three times as much memory as hdt_rs. The memory consumption is calculated by comparing the resident set size before and after graph loading and index generation, with the caveat that the memory usage may be higher during graph loading. Converting other formats to HDT in the first place is also a time and memory-intensive process. The uncompressed and fully indexed Sophia FastGraph (sophia) strongly outperforms the HDT libraries in ?PO query time, with 20ms compared to 214ms respectively 321ms for hdt_java. While being the fastest querying HDT library in this test, hdt_java has a large memory usage for an HDT library placing it closer to the much faster sophia_lg. The large overhead on small graph sizes for hdt_java in Figure 3 suggests that with larger graph sizes, these considerations might yield different results. In fact, HDT allows loading much larger datasets, but at that point, several of the tested libraries could not have been included, such as rdflib, which already uses over 14 GB of memory to load the ~10 million triples. hdt_rs achieves the lowest graph-loading time with 912ms compared to more than 11s for the fastest-loading non-HDT library sophia_lg. hdt_cpp and hdt_java can speed up loading by reusing previously saved indexes, but these were deleted between runs to achieve consistent measurements.

Examples

Further examples are available in the API documentation and in the code repository.

Add the dependency to a Rust application

```
$ cargo add hdt
```

Load an HDT file

```rust
use hdt::Hdt;
use std::{fs::File,io::BufReader};
let f = File::open("example.hdt").expect("error opening file");
let hdt = Hdt::new(BufReader::new(f)).expect("error loading HDT");
```

Query SP? pattern

Find the mayor of Leipzig from DBpedia using an SP? triple pattern:

```
hdt.triples_with_pattern(
    Some("http://dbpedia.org/resource/Leipzig"),
    Some("http://dbpedia.org/ontology/mayor"),
    None).next();
```

Query ?PO pattern

Which city has Burkhard Jung as the mayor?

```
hdt.triples_with_pattern(
    None,
    Some("http://dbpedia.org/ontology/mayor"),
    Some("http://dbpedia.org/resource/Burkhard_Jung")).next();
```

Use HDT with the Sophia library

```
use hdt::{Hdt,HdtGraph};
use hdt::sophia::api::graph::Graph;
use hdt::sophia::api::term::{IriRef, SimpleTerm, matcher::Any};
use std::{fs::File,io::BufReader};

let file = File::open("dbpedia.hdt").expect("error opening file");
let hdt = Hdt::new(BufReader::new(file)).expect("error loading HDT");
let graph = HdtGraph::new(hdt);
// now Sophia can be used as usual
let s = SimpleTerm::Iri(
    IriRef::new_unchecked("http://dbpedia.org/resource/Leipzig".into()));
let p = SimpleTerm::Iri(
    IriRef::new_unchecked("http://dbpedia.org/ontology/mayor".into()));
let mayors = graph.triples_matching(Some(s),Some(p),Any);
```

Limitations

HDT is read-only, so for querying and modifying large graphs, we recommend to use a separate SPARQL endpoint. We do not supply command line tools for converting other formats to and from HDT. Instead, the tools of hdt-cpp and hdt-java can be used. Extensions such as HDT++ (Hernández-Illera et al., 2015) or iHDT++ (Hernández-Illera et al., 2020) are unsupported.

Acknowledgements

We express our gratitude to Pierre-Antoine Champin for explaining the intricacies of Sophia and for developing the benchmark suite that the HDT benchmarks are based on and for the thorough code review. We extend our thanks to Edgard Marx for proofreading the paper.

References

Arias, M., Smith, A. W. S., Diefenbach, D., & Sande, M. V. (2023). HDT library, Java implementation. In *GitHub repository*. GitHub. https://github.com/rdfhdt/hdt-cpp

Beckett, D., Frederiksen, M., Robillard, D., & Aalto, L. (2015). Redland librdf RDF API and triple stores. In *GitHub repository*. GitHub. https://github.com/dajobe/librdf

Buil-Aranda, C., Hogan, A., Umbrich, J., & Vandenbussche, P.-Y. (2013). SPARQL web-querying infrastructure: Ready for action? In H. Alani, L. Kagal, A. Fokoue, P. Groth, C. Biemann, J. X. Parreira, et al. (Eds.), *The semantic web – ISWC 2013* (pp. 277–293). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-41338-4_18

Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., & Wilkinson, K. (2004). Jena: Implementing the semantic web recommendations. *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*, 74–83. https://doi.org/10.1145/1013367.1013381

Champin, P.-A. (2020). *Sophia: A Linked Data and Semantic Web toolkit for Rust* (E. Wilde & M. Amundsen, Eds.). The Web Conference 2020: Developers Track. https://www2020devtrack.github.io/site/schedule

Fernández, J. D., Martínez-Prieto, M. A., Gutiérrez, C., Polleres, A., & Arias, M. (2013). Binary RDF representation for publication and exchange (HDT). *Web Semantics: Science, Services and Agents on the World Wide Web*, *19*, 22–41. https://doi.org/10.2139/ssrn.3198999

Hernández-Illera, A., Martínez-Prieto, M. A., & Fernández, J. D. (2015). Serializing RDF in compressed space. *Proceedings of the Data Compression Conference 2015*, 363–372. https://doi.org/10.1109/dcc.2015.16

Hernández-Illera, A., Martínez-Prieto, M. A., Fernández, J. D., & Fariña, A. (2020). iHDT++: Improving HDT for SPARQL triple pattern resolution. *Journal of Intelligent & Fuzzy Systems*, *39*(2), 2249–2261. https://doi.org/10.3233/JIFS-179888

Hitzler, P. (2021). A review of the semantic web field. *Communications of the ACM*, *64*(2), 76–83. https://doi.org/10.1145/3397512

Höffner, K. (2023). RickView. In *GitHub repository*. GitHub. https://github.com/KonradHoeffner/rickview

Höffner, K., Martin, M., & Lehmann, J. (2016). LinkedSpending: OpenSpending becomes Linked Open Data. *Semantic Web*, *7*(1), 95–104. https://doi.org/10.3233/sw-150172

Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P. N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S., & Bizer, C. (2015). DBpedia – A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web*, *6*(2), 167–195. https://doi.org/10.3233/SW-140134

Martínez-Prieto, M. A., Arias, M., & Fernández, J. D. (2012). Exchange and consumption of huge RDF data. *The Semantic Web: Research and Applications*, 437–452. https://doi.org/10.1007/978-3-642-30284-8_36

Swartz, A., Eland, A., Nelson, A., Kuchling, A., Sommer, A., Knudsen, A., Cogrel, B., Pelakh, B., Ogbuji, C., Markiewicz, C., Mungall, C., Scott, D., Krech, D., Jones, D. H., Bowman, D., Winston, D., Perttula, D., Chuc, E., Torres, E., … Waites, W. (2023). RDFLib. In *GitHub repository*. GitHub. https://github.com/RDFLib/rdflib

Torres, P., Arias, M., Verbogh, R., Nikolopoulos, D., Robillard, D., Bendiken, A., Rietveld, L., & Beek, W. (2022). C++ implementation of the HDT compression format. In *GitHub repository*. GitHub. https://github.com/rdfhdt/hdt-java

Witten, I. H., Witten, I. H., Moffat, A., Bell, T. C., Bell, T. C., Fox, E., & Bell, T. C. (1999). *Managing gigabytes: Compressing and indexing documents and images*. Morgan Kaufmann.