EvonNemo - A Symbiosis of Datalog Tracing and Proof Tree Visualization

Lukas Gerlach[1][0000−0003−4566−0224], Alex Ivliev[1][0000−0002−1604−6308], Julián Méndez[2][0000−0003−1029−7656], Simon Meusel[1], Raimund Dachselt[2][0000−0002−2176−876X], and Markus Krötzsch[1][0000−0002−9172−2601]

[1] Knowledge-Based Systems Group, TU Dresden
{lukas.gerlach, alex.ivliev, markus.kroetzsch}@tu-dresden.de,
simon.meusel@mailbox.tu-dresden.de
[2] Interactive Media Lab Dresden, TU Dresden
julian.mendez2@tu-dresden.de, dachselt@acm.org

Abstract. Datalog and other rule-based formalisms are explainable by design, but showing explanations in a digestible form to a human remains a key challenge since rule engines often work with billions of facts. As a first step in this direction, we present a prototype of the integration of two KR tools: Nemo, a fast and versatile rule reasoning toolkit, and Evonne, a tool for visualizing, analysing, and debugging Description Logic proofs. We briefly discuss challenges on both ends such as computing Datalog proof trees in Nemo and special requirements for their visualization concerning Evonne. We conclude this extended abstract by showcasing key analysis features and an elaborate discussion of our roadmap to further aid explainability as well as developer experience from both a knowledge engineering perspective and a rule engine point of view.

Keywords: Rule-based Reasoning · Datalog · Visualization

1 Introduction

Datalog and *existential rules* are widely used in knowledge representation and reasoning and provide an intuitive way of recursively constructing relations.

Example 1. We can encode a parent relation in a Datalog program, compute an ancestor relation as the transitive closure of the parent relation and find common descendants of specific individuals (Isabelle and Heinrich in this case).

```
parent(alice, bob). parent(alice, charlotte). parent(daniel, bob).
parent(daniel, charlotte). parent(charlotte, edward).
parent(edward, fabienne). parent(edward, gilbert).
parent(fabienne, heinrich). parent(gilbert, isabelle).

ancestor(?X, ?Y) :- parent(?X, ?Y).
ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z).
commonDescendant(?X, ?Y, ?Z) :- ancestor(?X, ?Y), ancestor(?X, ?Z).
commonDescendantsOfIsabelleAndHeinrich(?X) :-
    commonDescendant(?X, isabelle, heinrich).
```

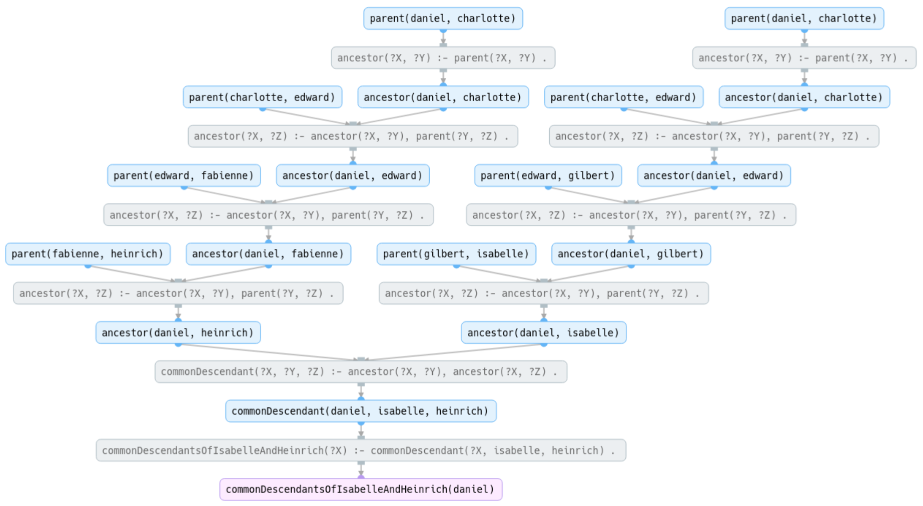

parent(daniel, charlotte)
parent(daniel, charlotte)
ancestor(?X, ?Y) :- parent(?X, ?Y) .
ancestor(?X, ?Y) :- parent(?X, ?Y) .
parent(charlotte, edward)
ancestor(daniel, charlotte)
parent(charlotte, edward)
ancestor(daniel, charlotte)
ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z) .
ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z) .
parent(edward, fabienne)
ancestor(daniel, edward)
parent(edward, gilbert)
ancestor(daniel, edward)
ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z) .
ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z) .
parent(fabienne, heinrich)
ancestor(daniel, fabienne)
parent(gilbert, isabelle)
ancestor(daniel, gilbert)
ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z) .
ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z) .
ancestor(daniel, heinrich)
ancestor(daniel, isabelle)
commonDescendant(?X, ?Y, ?Z) :- ancestor(?X, ?Y), ancestor(?X, ?Z) .
commonDescendant(daniel, isabelle, heinrich)
commonDescendantsOfIsabelleAndHeinrich(?X) :- commonDescendant(?X, isabelle, heinrich) .
commonDescendantsOfIsabelleAndHeinrich(daniel)

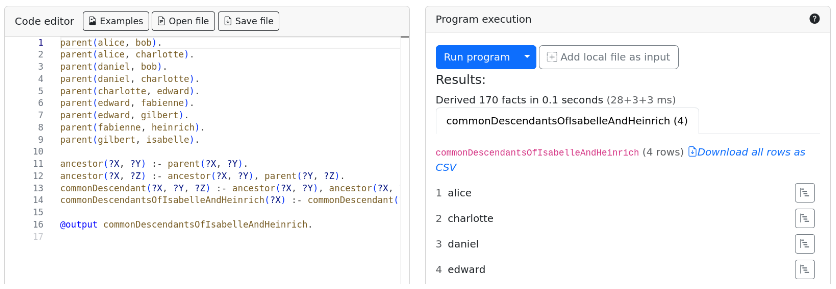
Fig. 1: Proof Tree for `commonDescendantsOfIsabelleAndHeinrich(daniel)`

A clear advantage of representing knowledge with such rules is that reasoning is explainable by design: for each logical fact, one can give (all) reasons, why and how the fact follows from the rules. A whole field of research tackles problems around efficiently computing the *provenance* of facts in Datalog settings [6,7,8].

In practice, Datalog systems work with billions of facts. While in principle every inference can still be explained, the reasons for a fact to exist might become very convoluted. Furthermore, for working with such amounts of data, state of the art systems employ involved optimization techniques [2,3,16,9,17,11,12]. In this respect, explaining should more closely reflect the actual rule applications that were performed by a particular Database system to arrive at the conclusion. We refer to this as the *trace* of a fact. Such a trace can naturally be represented as a proof tree. Figure 1 shows the trace for Daniel being a common descendant of Isabelle and Heinrich given by the rule engine Nemo [11,12].

Representing such trees in a digestible manner is challenging. Fortunately, the tool Evonne [14] already tackles similar problems occuring in the setting of *Description Logics* (DLs). Going beyond capabilities of ontology editing tools [15,10,13], Evonne provides a comfortable tree reading experience through multiple layout modes, with free-form exploration features like bidirectional collapsing/expanding of the proof and subproof inspection. We integrate Nemo and Evonne to be able to use the existing proof tree visualization capabilities for Datalog traces. This imposes additional requirements on both ends, which we discuss in Sections 2 and 3, respectively.

This extended abstract presents the integration of Nemo and Evonne as a first step of visualizing traces of Datalog systems shining a light on the visualization requirements identified for the Datalog setting. We conclude with a roadmap of features to add to make the Nemo and Evonne symbiosis even more fruitful.



```
parent(alice, bob).
parent(alice, charlotte).
parent(daniel, bob).
parent(daniel, charlotte).
parent(charlotte, edward).
parent(edward, fabienne).
parent(edward, gilbert).
parent(fabienne, heinrich).
parent(gilbert, isabelle).

ancestor(?X, ?Y) :- parent(?X, ?Y).
ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z).
commonDescendant(?X, ?Y, ?Z) :- ancestor(?X, ?Y), ancestor(?X,
commonDescendantsOfIsabelleAndHeinrich(?X) :- commonDescendant(

@output commonDescendantsOfIsabelleAndHeinrich.
```

Results:

Derived 170 facts in 0.1 seconds (28+3+3 ms)

commonDescendantsOfIsabelleAndHeinrich (4)

commonDescendantsOfIsabelleAndHeinrich (4 rows) Download all rows as CSV

```
1 alice
2 charlotte
3 daniel
4 edward
```

Fig. 2: Nemo's Web Interface with Code Editor and Result View for Example 1

2 Computing Traces in Nemo

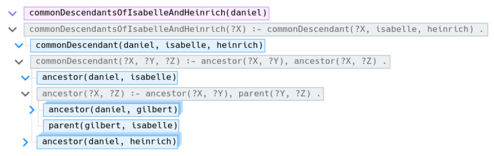
Nemo is a versatile rule-based reasoning system designed for large-scale data processing [11,12]. Its rule language is based on Datalog and extended with various features of modern query languages, including support for datatypes, aggregates, negation, existential rules, and numerous built-in functions and operators. Nemo is available as a web application[3], making it possible to present information graphically to the user. Figure 2 shows Example 1 in Nemo's web interface with both the code editor and the result table for the target predicate.

The core reasoning task of Nemo is to derive all logical consequences of a knowledge base consisting of a set of facts and rules. Rules are applied exhaustively using semi-naive evaluation [1] and the restricted chase [4]. Each step selects a rule, finds all matches that satisfy its body and computes the new facts according to the rule's head. Such facts are grouped by predicate and stored in separate tables for each rule application [18]. This allows us to provide tracing with no additional cost, as for each fact the exact proof tree used to derive it can be reconstructed by recursively applying the rules backwards as follows.

1. Given a fact, find the table storing it to determine the derivation rule
2. For each head atom of the same predicate in that rule:
   (a) Unify the fact with the current head atom (skip if not possible).
   (b) Query the database with the rule's body after applying the unifier, resulting in a set of ground body atoms.
   (c) Decent recursively for each such body atom.

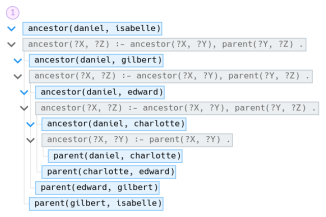
This process continues recursively until an input fact is reached in each branch. We avoid recomputing traces for the same fact by caching each result, which internally creates a directed acyclic graph (DAG) representing the proof. The implementation of step 2b is optimized by utilizing Nemo's trie-based architecture [19] that offers an efficient way to obtain only the first result of a query. For step 2a, some minor rewriting of the rule is required to accommodate features such as built-in functions and the minimum and maximum aggregates. Providing insightful traces for all of Nemo's features, specifically when negation and other aggregates are involved, is challenging and still ongoing work in the Nemo team.

[3] `https://tools.iccl.inf.tu-dresden.de/nemo/evonne-demo/`



```
commonDescendantsOfIsabelleAndHeinrich(daniel)
commonDescendantsOfIsabelleAndHeinrich(?X) :- commonDescendant(?X, isabelle, heinrich) .
 commonDescendant(daniel, isabelle, heinrich)
 commonDescendant(?X, ?Y, ?Z) :- ancestor(?X, ?Y), ancestor(?X, ?Z) .
  ancestor(daniel, isabelle)
  ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z) .
   ancestor(daniel, gilbert)
   parent(gilbert, isabelle)
  ancestor(daniel, heinrich)
```

(a) Example of collapsing



```
ancestor(daniel, isabelle)
ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z) .
 ancestor(daniel, gilbert)
 ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z) .
  ancestor(daniel, edward)
  ancestor(?X, ?Z) :- ancestor(?X, ?Y), parent(?Y, ?Z) .
   ancestor(daniel, charlotte)
   ancestor(?X, ?Y) :- parent(?X, ?Y) .
    parent(daniel, charlotte)
   parent(charlotte, edward)
  parent(edward, gilbert)
 parent(gilbert, isabelle)
```

(b) Example of subproof

Fig. 3: Inspecting Example 1 by collapsing and exploring subproofs in Evonne

3 Evonne for Visualization of Traces

Evonne [14] is a standalone web tool for visualizing reasoning with OWL ontologies. OWL (Web Ontology Language) is based on Description Logics (DLs), a family of logics targeted at describing concepts and relationships between them. Given an ontology and a *consequence* (i.e., an axiom that follows from the ontology), Evonne computes and displays a proof that indicates how the consequence was inferred from the axioms of the ontology. More specifically, a proof is a tree where nodes represent axioms or DL rules, and the edges indicate the flow of logical inferences leading to the consequence at the root. Evonne is tailored for DL experts and was designed to complement their existing work environments. These environments typically include Protégé as the default editor for OWL ontologies. Protégé includes some basic visualization components, such as compact indented tree view for class hierarchies. It is also possible to develop plug-ins for Protégé, where we highlight the proof explanation plug-in [13], which shows proofs as indented trees. Since indented tree views were available elsewhere, the published version of Evonne did not include such a view. Instead, Evonne is meant to provide a more comfortable reading experience, multiple layout modes, bidirectional collapsing and expanding of the proof, subproof inspection, and features to support debugging of erroneous reasoning. In order to make Evonne a more complete toolkit for its integration with Nemo, we update it by: (1) turning it into a library which can be embedded in other web projects, and (2) incorporating a compact indented tree view (similar to Protégé's) that better supports longer traces and rules generated by Nemo. A default layout of Evonne is visible in Figure 1, and the new compact mode can be seen in Figure 3.

4 Integration and Roadmap

Integration. With both tracing in Nemo and the generalized API for Evonne in place, it remains merely a technical effort to integrate both tools. Evonne accepts trees in a GraphML format [5], modified to include node types and alternative labels. Figure 2 already shows the Nemo Web view for our running example. Each row in the result view on the right features a small tracing button. Clicking the button opens a popup showing the trace for the specific fact using Evonne. For example, the trace in Figure 1 corresponds to `commonDescendantsOfIsabelleAndHeinrich(daniel)`. Since the full tree is already quite big, even for our toy example, we can leverage Evonne's features for interacting with the tree. We show the new compact layout in Figure 3. It is possible to collapse out-of-interest branches of the tree by clicking small arrows on the nodes that appear when hovering (Figure 3a), or to isolate subproofs for inspection by ctrl-clicking a node. For instance, Figure 3b showcases the subproof for `ancestor(daniel, fabienne)`. The view can further be simplified by clicking a rule node to highlight the nodes involved in the particular application.

Roadmap. Based on our first prototype, we identified a couple of needs to further improve user experience for analyzing Datalog proofs with Evonne. In an immediate next step, we aim to support proof DAGs, which can already be generated by Nemo. One can see in Figure 1, that the subtree for `ancestor(daniel, edward)` occurs twice in the tree since this fact is required for two different rule applications. Nemo produces this subproof in a DAG which is then output as a tree with redundancies. However, these redundancies can be hard to note, and users may benefit from a DAG view instead. This poses new layout challenges as, e.g., edges are likely to cross. To migitate this, it would also be possible to keep the tree layout and to highlight common subtrees. We also aim to connect the Evonne view to the code editor, to e.g., be able to jump directly to a rule or highlight the lines of code that are involved in deriving a fact chosen in Evonne.

Focusing on the development of Nemo itself, the integration with Evonne also lays groundwork for profiling visualization. For example, the proof tree could encode the execution times for certain rules to see where most time is spent during reasoning. Since Nemo is built to work with billions of facts, understanding how to further boost performance is crucial. Connected views, showing multiple inferences for a given rule or even breaking down the application of a single rule into substeps all including performance metrics promise to be helpful in understanding Nemo's performance and also pose interesting visualization challenges.

Our first prototype of integrating Nemo and Evonne is one tool in our belt for the vision of building more developer friendly and explainable knowledge engineering tools. We think that profiling tools built on top will furthermore catalyze the development of Nemo in particular and fast rule engines in general.

Acknowledgments. This work is funded by Deutsche Forschungsgemeinschaft (DFG) under Germany's Excellence Strategy: EXC-2068, 390729961 – "Physics of Life" and EXC 2050/1, 390696704 – "Centre for Tactile Internet" (CeTI), by DFG grant 389792660 (TRR 248 – CPEC); by Bundesministerium für Bildung und Forschung (BMBF) under European ITEA project 01IS21084 (InnoSale); by BMBF and Saxon State Ministry for Science, Culture and Tourism (SMWK) in Center for Scalable Data Analytics and Artificial Intelligence (ScaDS.AI, SCADS22B); by BMBF and German Academic Exchange Service (DAAD) in project 57616814 (SECAI, School of Embedded and Composite AI); and by the Center for Advancing Electronics Dresden (cfaed).

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison Wesley (1994)
2. Aref, M., ten Cate, B., Green, T.J., Kimelfeld, B., Olteanu, D., Pasalic, E., Veldhuizen, T.L., Washburn, G.: Design and implementation of the logicblox system. In: Sellis, T.K., Davidson, S.B., Ives, Z.G. (eds.) Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015. pp. 1371–1382. ACM (2015). https://doi.org/10.1145/2723372.2742796, https://doi.org/10.1145/2723372.2742796
3. Bellomarini, L., Sallinger, E., Gottlob, G.: The vadalog system: Datalog-based reasoning for knowledge graphs. Proc. VLDB Endow. **11**(9), 975–987 (2018). https://doi.org/10.14778/3213880.3213888, http://www.vldb.org/pvldb/vol11/p975-bellomarini.pdf
4. Benedikt, M., Konstantinidis, G., Mecca, G., Motik, B., Papotti, P., Santoro, D., Tsamoura, E.: Benchmarking the chase. In: Sallinger, E., den Bussche, J.V., Geerts, F. (eds.) Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017, Chicago, IL, USA, May 14-19, 2017. pp. 37–52. ACM (2017). https://doi.org/10.1145/3034786.3034796, https://doi.org/10.1145/3034786.3034796
5. Brandes, U., Eiglsperger, M., Herman, I., Himsolt, M., Marshall, M.S.: Graphml progress report. In: Mutzel, P., Jünger, M., Leipert, S. (eds.) Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria, September 23-26, 2001, Revised Papers. Lecture Notes in Computer Science, vol. 2265, pp. 501–512. Springer (2001). https://doi.org/10.1007/3-540-45848-4_59, https://doi.org/10.1007/3-540-45848-4_59
6. Buneman, P., Khanna, S., Tan, W.C.: Why and where: A characterization of data provenance. In: den Bussche, J.V., Vianu, V. (eds.) Database Theory - ICDT 2001, 8th International Conference, London, UK, January 4-6, 2001, Proceedings. Lecture Notes in Computer Science, vol. 1973, pp. 316–330. Springer (2001). https://doi.org/10.1007/3-540-44503-X_20, https://doi.org/10.1007/3-540-44503-X_20
7. Cheney, J., Chiticariu, L., Tan, W.C.: Provenance in databases: Why, how, and where. Found. Trends Databases **1**(4), 379–474 (2009). https://doi.org/10.1561/1900000006, https://doi.org/10.1561/1900000006
8. Elhalawati, A., Krötzsch, M., Mennicke, S.: An existential rule framework for computing why-provenance on-demand for datalog. In: Governatori, G., Turhan, A. (eds.) Rules and Reasoning - 6th International Joint Conference on Rules and Reasoning, RuleML+RR 2022, Berlin, Germany, September 26-28, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13752, pp. 146–163. Springer (2022). https://doi.org/10.1007/978-3-031-21541-4_10, https://doi.org/10.1007/978-3-031-21541-4_10
9. Fan, Z., Zhu, J., Zhang, Z., Albarghouthi, A., Koutris, P., Patel, J.M.: Scaling-up in-memory datalog processing: Observations and techniques. Proc. VLDB Endow. **12**(6), 695–708 (2019). https://doi.org/10.14778/3311880.3311886, http://www.vldb.org/pvldb/vol12/p695-fan.pdf
10. Horridge, M., Gonçalves, R.S., Nyulas, C.I., Tudorache, T., Musen, M.A.: Webprotégé: A cloud-based ontology editor. In: Companion Proc. of WWW'19. p. 686–689. Association for Computing Machinery (2019). https://doi.org/10.1145/3308560.3317707

11. Ivliev, A., Ellmauthaler, S., Gerlach, L., Marx, M., Meißner, M., Meusel, S., Krötzsch, M.: Nemo: First glimpse of a new rule engine. In: Pontelli, E., Costantini, S., Dodaro, C., Gaggl, S.A., Calegari, R., d'Avila Garcez, A.S., Fabiano, F., Mileo, A., Russo, A., Toni, F. (eds.) Proceedings 39th International Conference on Logic Programming, ICLP 2023, Imperial College London, UK, 9th July 2023 - 15th July 2023. EPTCS, vol. 385, pp. 333–335 (2023). `https://doi.org/10.4204/EPTCS.385.35`, `https://doi.org/10.4204/EPTCS.385.35`
12. Ivliev, A., Gerlach, L., Meusel, S., Steinberg, J., Krötzsch, M.: Nemo: Your friendly and versatile rule reasoning toolkit. In: Proceedings of the 21st International Conference on Principles of Knowledge Representation and Reasoning (2024), `https://iccl.inf.tu-dresden.de/web/Inproceedings3390`, to appear
13. Kazakov, Y., Klinov, P., Stupnikov, A.: Towards reusable explanation services in Protégé. In: Proc. 30th Int. Workshop on Description Logics (DL). CEUR Workshop Proceedings, vol. 1879 (2017), `http://www.ceur-ws.org/Vol-1879/paper31.pdf`
14. Méndez, J., Alrabbaa, C., Koopmann, P., Langner, R., Baader, F., Dachselt, R.: Evonne: A visual tool for explaining reasoning with owl ontologies and supporting interactive debugging. Computer Graphics Forum (3 2023). `https://doi.org/10.1111/cgf.14730`
15. Musen, M.A.: The Protégé project: a look back and a look forward. AI Matters **1**(4), 4–12 (2015). `https://doi.org/10.1145/2757001.2757003`
16. Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., Banerjee, J.: Rdfox: A highly-scalable RDF store. In: Arenas, M., Corcho, Ó., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P., Dumontier, M., Heflin, J., Thirunarayan, K., Staab, S. (eds.) The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9367, pp. 3–20. Springer (2015), `https://doi.org/10.1007/978-3-319-25010-6_1`
17. Seo, J., Guo, S., Lam, M.S.: Socialite: An efficient graph query language based on datalog. IEEE Trans. Knowl. Data Eng. **27**(7), 1824–1837 (2015). `https://doi.org/10.1109/TKDE.2015.2405562`, `https://doi.org/10.1109/TKDE.2015.2405562`
18. Urbani, J., Jacobs, C.J.H., Krötzsch, M.: Column-oriented datalog materialization for large knowledge graphs. In: Schuurmans, D., Wellman, M.P. (eds.) Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA. pp. 258–264. AAAI Press (2016). `https://doi.org/10.1609/AAAI.V30I1.9993`, `https://doi.org/10.1609/aaai.v30i1.9993`
19. Veldhuizen, T.L.: Triejoin: A simple, worst-case optimal join algorithm. In: Schweikardt, N., Christophides, V., Leroy, V. (eds.) Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014. pp. 96–106. OpenProceedings.org (2014). `https://doi.org/10.5441/002/ICDT.2014.13`, `https://doi.org/10.5441/002/icdt.2014.13`