MillenniumDB Path Query Challenge

Benjamín Farías[1,2], Carlos Rojas[2] and Domagoj Vrgoč[1,2]

[1]PUC Chile, Santiago, Chile
[2]IMFD Chile, Santiago, Chile

Abstract

In this short paper we present a benchmark of regular path queries where paths are returned in addition to reachable nodes. The types of paths returned mimic the upcoming GQL graph query standard, and require returning a single path, a single shortest path, all shortest paths, all trails, or all simple paths connecting each pair of nodes in the answer to the query. We provide two challenge sets: (i) real world queries extracted from the Wikidata query logs; and (ii) a synthetic dataset with exponential behavior meant to serve as a sanity check for the termination conditions of a path algorithm. A reference implementation using MillenniumDB, a recently published open-source graph database engine, is also provided with the challenge.

Keywords

graph databases, regular path queries, query evaluation

1. Introduction

Graph databases [1] are a rapidly growing area, both in terms of academic research [2], and commercial systems being developed by multiple vendors [3, 4, 5]. Graphs databases offer flexibility in terms of design, are easily extensible, and independent of a fixed schema. In terms of querying, graph databases introduce interesting challenges as opposed to the relational setting. Following [1], graph queries could roughly be divided into two classes: (i) graph patterns; and (ii) path queries. Here we will focus on path queries.

The most widely used class of path queries are *regular path queries*, or *RPQs* for short [6]. An RPQ is specified by a regular expression, and the query looks for all pairs of nodes connected by a path whose edge labels form a word in the language of the expression. An RPQ can thus return: (a) only the connected nodes; or (b) the connected nodes together with the path(s) witnessing this connection. Option (a) is usually considered in the research literature [6], or in SPARQL systems [7]. However, option (b) is extremely relevant in practice, where multiple engines support returning paths [3, 4, 8], and the upcoming ISO standard for graph querying [9] prescribing different types of paths (shortest, trail, simple, etc.) to be returned when answering an RPQ.

In this short paper we report on ongoing work about algorithms for returning paths that witness a pair of nodes being in the answer of an RPQ, and present a resource, called *MillenniumDB Path Query Challenge*, which can be used by the research and development community

AMW'23: 15th Alberto Mendelzon International Workshop on Foundations of Data Management, May 22 –26, 2023, Santiago, Chile

bffarias@uc.cl (B. Farías); c.rojasvictoriano@gmail.com (C. Rojas); vrdomagoj@uc.cl (D. Vrgoč)

to test their implementations of path queries. The challenge contains datasets and RPQs, which are to be evaluated under different semantics for returning paths. We also provide a reference implementation using MillenniumDB [10], an open-source graph database engine developed at IMFD Chile. All the resources are freely available at [11].

1.1. Related Work

Several path query benchmarks had been proposed in the past. Most of them focused on reachable nodes, and not on returning paths. Notable examples include gMark [12], and BeSEPPI [13]. The most representative work on returning paths along with reachable nodes is The LDBC Social Network Benchmark [14], which contains several queries that use the RPQ `knows*`, and require paths with different characteristic (weighted, shortest, etc.) are to be returned. Our approach is more focused on complex RPQs, and path modes prescribed by the GQL standard [9]. The presented work heavily relies on WDBench [15], a recent Wikidata [16] query benchmark, and on examples from [17].

2. MillenniumDB Path Query Challenge

The key objective of the MillenniumDB Path Query Challenge is to test the efficiency of algorithms for finding paths that witness an answer to an RPQ. We specify RPQs using the SPARQL property path syntax [7], given that this is the only fully standardized language for RPQs (and their slight extensions such as 2RPQs [18]) to date. We consider the following three types of RPQs:

(i) `(start) propertyPath (end);`
(ii) `(start) propertyPath (?x);` and
(iii) `(?x) propertyPath (?y),`

where `(start)` and `(end)` are either fixed nodes of the graph, and `?x` and `?y` are variables. Type (i) queries are boolean and check whether `end` is reachable from `start` via `propertyPath`. Type (ii) queries look for all nodes reachable from `start`, while type (iii) queries return all pairs of nodes connected by `propertyPath`. For each such answer, we also wish to return paths.

What should my implementation return? We prescribe evaluating the challenge queries under the following semantics:

- **Endpoints:** For type (i) queries this is just true/false; for type (ii) we return all nodes reachable from `(start)`; for type (iii) we require all pairs of nodes connected by `propertyPath`.
- **Any path:** For each answer returned by **enpoints**, we also provide a single path witnessing this result.
- **Shortest path:** For each answer returned by **enpoints**, we also provide a single *shortest path* witnessing this result.
- **All shortest paths:** For each answer returned by **enpoints**, we also return *all* shortest paths witnessing this result.

- **All trails:** For each answer returned by **enpoints**, we also return *all* trails (i.e. paths not repeating any edges) witnessing this result.
- **All simple paths:** For each answer returned by **enpoints**, we also return *all* simple paths (i.e. paths not repeating any nodes) witnessing this result.

How many results? We recommend the number of results to be limited to 100,000. This number includes different paths for the same pair of connected nodes in the query answer. For instance, if a fixed pair `(start, end)` is connected by 100,000 shortest paths, returning all these paths already reaches the query limit. The limit is to keep the number of results manageable, and to allow checking whether the algorithms terminate efficiently.

How to run the queries? For each challenge dataset, all the queries should be run in succession without any warmup for each implementation mode that the user wishes to test. For instance, if we are testing how the algorithms for returning all shortest paths runs, we should run *all* the queries under the all shortest-path semantics. Each return mode should be tested in isolation. A timeout of 1 minute should be imposed on all the queries.

What should I report? The main metric we wish to track is *execution time*. Additionally, *memory usage* is recommended to be reported. Depending on the actual implementation, the data management mode should be categorized as:

1. **On-disk:** when the graph database is stored on disk (e.g. in B+trees) and buffered into main memory.
2. **In-memory:** when all the data is available in main memory.
3. **Hybrid:** when the disk data is loaded into specialized memory structures.

Reference implementation A reference implementation is made available at [11]. We remark that at this point queries of type (iii) are not fully supported. All the semantics modes and data management modes are available in the reference implementation, with repository readme files explaining their usage.

We next describe the data/queries provided in the challenge.

2.1. Challenge #1: real-world

The first challenge set tests the performance on real world data and user posted queries. For this, we use the Wikidata [16] dataset and its SPARQL query logs [19], from which property path patterns were extracted. More precisely, we base ourselves on the recently published WDBench [15] benchmark. The same dataset as in WDBench is used; namely, we take the truthy dump of Wikidata, and keep only direct properties from this RDF dataset[1], and remove Wikidata labels and descriptions. This results in an (RDF) dataset consisting of 1.257 billion triples. Transforming this dataset into a graph results in 364 million nodes and 1.257 billion edges. The dataset can be downloaded at [20].

[1]This allows us to have the graph structure in the dataset, which is the only thing explored by the RPQs.

In terms of queries, WDBench extracted several different query sets according to their characteristics. We took the RPQ query set from WDBench, and classified the RPQs in this set according to the three types described above. The queries are to be evaluated according to the modes described above (endpoints, trail, etc.). In total there are 659 RPQs in our dataset. Of these 6 are of type (i); 586 are of type (ii), and 67 of type (iii). These are to be evaluated according to one of the semantics modes described above. Remember that a limit of 100,000 results applies. The queries can be found at [11].

In order to facilitate usage in other engines, at [11] we include the Cypher [21] version of queries whenever possible. Notice that not all RPQs can be expressed in Cypher, and some semantics modes are not supported.

2.2. Challenge #2: synthetic

Here we present a synthetic dataset with a large number of paths between two nodes. The purpose of this challenge is to check whether the path algorithms can detect choking points and stop execution. The graph used in this challenge, illustrated in Figure 1, is taken from [17]. The single RPQ of type (i) used here has the pattern `(A) a* (B)`. Notice that in this graph all paths from A to B conforming to `a*` are shortest, trails and simple paths at the same time, and there are 2^n of them, while the graph has only $3n + 1$ nodes and $4n$ edges. Notice that for any semantics mode finding 100,000 paths should be easy enough, however, a poorly designed algorithm might try and compute all paths, which is unfeasible even for moderate values of n.

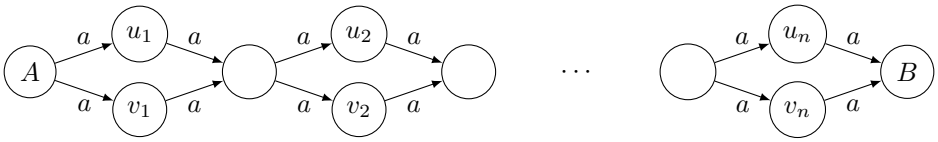

Figure 1: Graph for challenge #2. Here n is a parameter.

3. Conclusions

We present a resource and reference implementation for testing out algorithms for returning paths conforming to regular expressions over graph databases. We also perform initial experiments available at [11]. We hope that this resource can be of use to researchers and developers interested in implementing algorithms for returning paths in graph database query answers.

Acknowledgments

Work supported by the ANID – Millennium Science Initiative Program – Code ICN17_002 and ANID Fondecyt Regular nr. 1221799.

References

[1] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. L. Reutter, D. Vrgoč, Foundations of Modern Query Languages for Graph Databases, ACM Comput. Surv. 50 (2017) 68:1–68:40. URL: https://doi.org/10.1145/3104031. doi:10.1145/3104031.

[2] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutiérrez, J. E. L. Gayo, S. Kirrane, S. Neumaier, A. Polleres, R. Navigli, A. N. Ngomo, S. M. Rashid, A. Rula, L. Schmelzeisen, J. F. Sequeda, S. Staab, A. Zimmermann, Knowledge Graphs, CoRR abs/2003.02320 (2020). URL: https://arxiv.org/abs/2003.02320. arXiv:2003.02320.

[3] J. Webber, A programmatic introduction to Neo4j, in: G. T. Leavens (Ed.), Conference on Systems, Programming, and Applications: Software for Humanity, SPLASH '12, Tucson, AZ, USA, October 21-25, 2012, ACM, 2012, pp. 217–218. URL: https://doi.org/10.1145/2384716.2384777. doi:10.1145/2384716.2384777.

[4] T. Team, TigerGraph Documentation – version 3.1, 2021. URL: https://docs.tigergraph.com/.

[5] A. N. Team, What Is Amazon Neptune?, 2021. URL: https://docs.aws.amazon.com/neptune/latest/userguide/intro.html.

[6] P. B. Baeza, Querying graph databases, in: PODS 2013, 2013, pp. 175–188.

[7] S. Harris, A. Seaborne, E. Prud'hommeaux, SPARQL 1.1 Query Language, W3C Recommendation, 2013. URL: https://www.w3.org/TR/sparql11-query/.

[8] D. Vrgoč, C. Rojas, R. Angles, M. Arenas, D. Arroyuelo, C. B. Aranda, A. Hogan, G. Navarro, C. Riveros, J. Romero, Millenniumdb: A persistent, open-source, graph database, CoRR abs/2111.01540 (2021). URL: https://arxiv.org/abs/2111.01540.

[9] A. Deutsch, N. Francis, A. Green, K. Hare, B. Li, L. Libkin, T. Lindaaker, V. Marsault, W. Martens, J. Michels, F. Murlak, S. Plantikow, P. Selmer, O. van Rest, H. Voigt, D. Vrgoc, M. Wu, F. Zemke, Graph Pattern Matching in GQL and SQL/PGQ, in: Z. Ives, A. Bonifati, A. E. Abbadi (Eds.), SIGMOD '22, ACM, 2022, pp. 2246–2258.

[10] M. Team, MillenniumDB Source Code, 2021. URL: https://github.com/MillenniumDB/MillenniumDB.

[11] C. R. Benjamín Farías, D. Vrgoč, MillenniumDB Path Query Challenge, 2023. URL: https://github.com/MillenniumDB/path-query-challenge.

[12] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, N. Advokaat, gMark: Schema-Driven Generation of Graphs and Queries, IEEE Trans. Knowl. Data Eng. 29 (2017) 856–869.

[13] A. Skubella, D. Janke, S. Staab, BeSEPPI: Semantic-Based Benchmarking of Property Path Implementations, in: P. Hitzler, M. Fernández, K. Janowicz, A. Zaveri, A. J. G. Gray, V. López, A. Haller, K. Hammar (Eds.), ESWC 2019, volume 11503, Springer, 2019, pp. 475–490.

[14] R. Angles, J. B. Antal, A. Averbuch, P. A. Boncz, O. Erling, A. Gubichev, V. Haprian, M. Kaufmann, J. L. Larriba-Pey, N. Martínez-Bazan, J. Marton, M. Paradies, M. Pham, A. Prat-Pérez, M. Spasic, B. A. Steer, G. Szárnyas, J. Waudby, The LDBC Social Network Benchmark, CoRR abs/2001.02299 (2020). URL: http://arxiv.org/abs/2001.02299.

[15] R. Angles, C. B. Aranda, A. Hogan, C. Rojas, D. Vrgoc, WDBench: A Wikidata Graph Query Benchmark, in: U. Sattler, A. Hogan, C. M. Keet, V. Presutti, J. P. A. Almeida, H. Takeda, P. Monnin, G. Pirrò, C. d'Amato (Eds.), ISWC 2022, volume 13489, Springer,

2022, pp. 714–731.
[16] D. Vrandecic, M. Krötzsch, Wikidata: a free collaborative knowledgebase, Commun. ACM 57 (2014) 78–85. URL: https://doi.org/10.1145/2629489. doi:10.1145/2629489.
[17] W. Martens, M. Niewerth, T. Popp, S. Vansummeren, D. Vrgoč, Representing paths in graph database pattern matching, CoRR abs/2207.13541 (2022). URL: https://doi.org/10.48550/arXiv.2207.13541. doi:10.48550/arXiv.2207.13541. arXiv:2207.13541.
[18] D. Calvanese, G. D. Giacomo, M. Lenzerini, M. Y. Vardi, Rewriting of regular expressions and regular path queries, J. Comput. Syst. Sci. 64 (2002) 443–465.
[19] A. Bonifati, W. Martens, T. Timm, An analytical study of large SPARQL query logs, VLDB J. 29 (2020) 655–679. URL: https://doi.org/10.1007/s00778-019-00558-9. doi:10.1007/s00778-019-00558-9.
[20] R. Angles, C. B. Aranda, A. Hogan, C. Rojas, D. Vrgoč, WDBench: A Wikidata Graph Query Benchmark, 2022. doi:10.6084/m9.figshare.19599589, https://figshare.com/s/50b7544ad6b1f51de060.
[21] N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, A. Taylor, Cypher: An Evolving Query Language for Property Graphs, in: G. Das, C. M. Jermaine, P. A. Bernstein (Eds.), SIGMOD 2018, ACM, 2018, pp. 1433–1445. URL: https://doi.org/10.1145/3183713.3190657. doi:10.1145/3183713.3190657.

A. Online Resources

Resources needed to run the challenge (datasets, queries, scripts, etc.) can be found at:

- MillenniumDB Path Query Challenge