Journal of Information Systems Engineering and Management
2025, 10(59s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Benchmarking Reachability Techniques for the Optimization of RDF Queries

Abdallah KHELIL[1] , Rachid KHALLADI[1] , Mostafa BOUFERA, Liza RAAB
[1] Laboratory of Informatics and Intelligent Systems(LISYS), Department of Computer Science, University Of Mustapha Stambouli, Mascara, Algeria.

ARTICLE INFO

Received: 29 Dec 2024

Revised: 12 Feb 2025

Accepted: 27 Feb 2025

ABSTRACT

Today, a large amount of data is shared online, and much of it is connected like a network.

RDF (Resource Description Framework) graphs are used to describe this data using triples (subject, predicate, object), and they are important in the Semantic Web and Linked Open Data. However, as RDF graphs become very large, it becomes difficult to answer queries quickly, especially reachability queries which check if there is a path between two nodes.

This research studies how to improve the RDF QDAG (Query Directed Acyclic Graph), a structure that makes querying faster by removing cycles in the graph. While RDF QDAG is efficient, it has some limits with complex path queries. To solve this, we integrated several indexing techniques such as Tree-Cover Indexing, 2-Hop Labeling, Approximate Transitive Closure, TreeBased Indexing, Generalized Transitive Closure (GTC), and 2-Hop Labeling with Constraints.

We tested these techniques using real RDF data. The results show that our system can answer queries faster and use less memory. This work helps improve how large RDF graphs are managed and makes it easier to run advanced queries for semantic applications.

Keywords: RDF, QDAG, Tree Cover, TREE Based, GTC.

INTRODUCTION

Today, we live in a digital world where a huge amount of data is created every day. Much of this data is connected like a network, and we can represent it using graphs. A graph is a structure made of points (called nodes) and connections (called edges). Graphs are used in many areas like social media, transportation, and biology.

In recent years, one important type of data structure is called the RDF graph (Resource Description Framework). RDF helps organize data as triples (subject, predicate, object), which together form a graph. These graphs are used in the Semantic Web, where computers understand and use data better. Tools like DBpedia and Wikidata are based on RDF[1] [10].

The rise of the Semantic Web and knowledge graphs has revolutionized the way we represent and exploit information. RDF (Resource Description Framework) systems and SPARQL queries have emerged as major tools for structuring and querying data in graph form. The RDF_QDAG triplestore, developed at the LIAS laboratory of the University of Poitiers, offers advanced management of RDF data, with particular attention to query optimization [2].

But as the graphs become very big, it becomes difficult to search or answer questions about the data quickly. One important type of question is a reachability query, where we want to know if there is a path (or link) from one node to another in the graph [7] [8]. These queries are very useful in real-world applications like social networks or medical data.

To solve this problem, we study the system RDF QDAG. This method changes the RDF graph into a simpler form without cycles, which makes searching faster. However, RDF QDAG has limits when it comes to complex path searches.

Journal of Information Systems Engineering and Management
2025, 10(59s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

In this research, we focused on the reachability problem [16]. This problem is very interesting and crucial for answering path-type queries in graphs, but also for solving triplestore optimization problems. As a practical application, we applied reachability techniques to prune RDF QDAG fragments that do not contribute to the construction of final results.

Since the reachability problem is a classic problem in computer science, several techniques exist in the literature. We therefore started by analyzing existing approaches to identify those that are relevant for RDF QDAG. We also established a benchmark to compare these approaches objectively. For this, we collected representative graphs and queries from RDF QDAG. We also developed evaluation metrics to analyze the performance of existing approaches. This work allowed us to detect that six different reachability algorithms:

•Tree Cover Index

•Tree-Based Index

•Two-Hop Labeling with Constraint

•Two-Hop Labeling

•Approximate Transitive

•Generalized Transitive Closure

Each of these approaches provides different strategies to ascertain the existence of a path from one node to another in a graph. Tree-based approaches use hierarchical representations to tackle the search complexity, while labeling approaches are characterized by precomputing and storing reachability information in order to evaluate a query rapidly. By building transitive closure all possible reachable pairs of nodes in advance is the goal, both approximate and generalized versions provide trade-offs in pre-computation time, space, and efficiency of evaluating queries. We assessed each method against RDF QDAG data in a series of benchmark tests to understand their strengths and limitations in the cycle of query optimization and pruning, namely which approaches are most effective depending on the RDF graph structure and size.

This methodological approach is very important because it allows us to reproduce it to address similar problems where reachability plays a central role.

RELATED WORKS

Several approaches exist for processing RDF graphs [4] [5] [6] [8] [9]. There are two main families.

The first is based on relational databases. Graphs are transformed into tables and processed by SQL. These systems benefit from well-known optimizations. However, repeated joins become costly and slow down complex queries. This limits their effectiveness on large graphs.

The second family is graph-oriented. It retains the native structure and applies specific traversals. These systems are more suited to complex relationships and reachability queries. However, their performance drops when the size of the graph increases significantly. The scalability issue remains.

In the literature, several techniques specifically target reachability queries:

- Tree Cover Index and Tree-Based Index simplify traversals by reducing the structure to a tree. This works well for hierarchical graphs but is less efficient on dense or random graphs.
- Two-Hop Labeling assigns each node two sets of labels. The method is renowned for its speed and accuracy on large graphs.
- Two-Hop Labeling with Constraints further reduces false positives but adds complexity to index construction.
- Approximate Transitive Closure (ATC) seeks a compromise. It speeds up construction but yields approximate results.
- Generalized Transitive Closure (GTC) guarantees accurate results. However, it requires significant memory and computational time.

Journal of Information Systems Engineering and Management
2025, 10(59s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Few studies have tested these techniques directly in an RDF environment like QDAG. Our work fills this gap by providing a simple and targeted comparison.

METHODS

We followed a clear and reproducible process with four steps.

Step 1: Algorithm Selection
We chose six methods: Tree Cover Index, Tree-Based Index, Two-Hop Labeling, Two-Hop Labeling with Constraint, ATC, and GTC. They cover the main approaches.

Step 2: Definition of Criteria
We used five criteria: response time, construction time, memory usage, complexity, and scalability. These criteria allow us to evaluate each method for different needs.

Step 3: Data Preparation
The experiments were carried out on RDF fragments generated by QDAG. We built graphs of different sizes and created both simple and complex queries.

Step 4: Execution and Results Collection
We applied each algorithm to the same datasets with a common protocol. We measured response time, Construction cost, and memory use. The results were shown in tables and graphs.

Algorithms

Tree-Cover Indexing:

This technique constructs interval labels for each node from a spanning tree of the given graph.

A node u can reach node v if the interval assigned to v is contained within the interval assigned to u. This method allows constant-time reachability queries in Directed Acyclic Graphs (DAGs) and can be applied to cyclic graphs through the use of interval inheritance, which propagates interval information along back-edges and cross-edges [3]. Its main advantages include fast query processing and compact index size for DAGs, which are well suited for massive static graphs.

It is not good with dynamic updates and may have many intervals per node for dense or cyclic graphs, resulting in higher complexity and cost of storage. Despite this, it is a seminal approach that has influenced many next-generation reachability indexing schemes and is key to knowing the extent to which efficient querying of graphs can be handled

2-Hop Labeling

This method assigns to each node two label sets that summarize its reachability profile [11]:

• $L_{in}(v)$: the set of nodes that can reach node v,

• $L_{out}(v)$: the set of nodes that are reachable from node v.

If this condition holds, then a path from s to t exists. This indexing approach facilitates efficient query evaluation by enabling direct label lookups instead of requiring full graph traversal at query time. Its key advantages include completeness—queries can be answered using only index data—and applicability to general graphs, not limited to DAGs or tree structures. However, constructing an optimal 2-Hop index is NP-hard, and even approximate solutions can be computationally expensive and memory-intensive, especially on large or dense graphs. Moreover, most implementations assume static graphs, with limited support for dynamic updates such as insertions or deletions. Despite these limitations, 2-Hop Labeling remains a foundational method in reachability indexing, striking a balance between structural generality and query efficiency. It has also served as the basis for more advanced methods such as TFL, PLL, DL, and TOL.

Journal of Information Systems Engineering and Management
2025, 10(59s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Approximate Transitive Closure (ATC):

Approximate Transitive Closure approaches attempt to efficiently approximate reachability between the nodes of a graph without computing the full transitive closure, which is infeasible for large-scale graphs due to its high computational and storage costs. They approximate the set of reachable nodes from each vertex using compact data structures such as min-wise independent permutations as in the IP index [15] or Bloom filtersas in the BFL index [16]. The principle is to ensure that if node t is unreachable from node *s*, the approximation can rule this out with certainty by being contra-positive—if the approximation of *Out(t)* is not a subset of that of *Out(s)*, then *t* is unreachable from *s*. This results in no false negatives while false positives are allowed, with optional graph walking for verification. The advantages of this technique are scalability to extremely large graphs, fast query time, and compact index size. It cannot support exact queries and may return approximate results, and like most index structures, dynamic updates are costly [17].

Algorithms for Path-Constrained Reachability

Path-constrained reachability extends basic reachability to verify whether there exists a path from a node to another node so that the sequence of labels along the path satisfies some given constraint, e.g., in the shape of a regular expression. Such constraints restrict the walk along edges to specific types of relationships, e.g., "friendOf", "follows", or "worksFor".

Algorithms build specialized indexes for testing such queries, which regard label information. For alternation-based constraints, approaches like tree-based indexing and generalized transitive closure (GTC) connect paths with Sufficient Path Label Sets (SPLS), allowing early path pruning that is not in compliance with the label constraints. For concatenation-based constraints, approaches like the RLC index record minimum repeating sequences of labels in a way that enables sound judgment regarding whether a valid path pattern exists or not. The process of evaluation usually consists of either fast index lookups to verify constraint-satisfying paths or guided graph traversal with finite automata. Path-Constrained Reachability Queries [2] [16]

Tree-Based Indexing:

Tree-based indexing extends classical tree-cover methods by incorporating label constraints through Sufficient Path Label Sets (SPLS), which annotate paths with sets of edge labels sufficient to cover a target query constraint [12,13]. This supports efficient evaluation of label-constrained reachability queries by enabling early pruning of paths that cannot satisfy the query constraints. The index classifies edges into types (tree, forward, back, and cross) based on a spanning tree and applies interval labeling and recursive decomposition to handle reachability caused by non-tree edges. These optimizations make both succinct representation and effective computation of SPLS possible during traversal. The key advantages of the method are that it can significantly reduce the search space and improve the performance of expressive label-constrained queries. However, it has drawbacks such as costly preprocessing, lack of support for general path constraints beyond alternation, and limited flexibility for dynamic updates. Still, it plays the crucial role of enabling query evaluation on large edge-labeled graphs with complex structural patterns to be efficient.

Generalized Transitive Closure (GTC):

Generalized Transitive Closure (GTC) augments standard transitive closure with the capability of considering edge label meaning in reachability computation so that it can determine whether or not there exists a path between a pair of nodes that satisfies some label specifications [14].

Rather than performing standard reachability, GTC calculates label-constrained paths in advance and assigns them to Sufficient Path Label Sets (SPLSs) that summarize the minimal collection of label combinations to satisfy alternation-based requirements. Its construction employs a Dijkstra-like algorithm favoring paths with fewer distinct labels and reduces strongly connected components to bipartite forms to maintain label semantics. This enables efficient evaluation of semantically rich queries at low cost without the expense of runtime traversal. Its major advantages are high query performance and expressive, label-based constraints supported, which renders it most desirable for large and very labeled graphs. However, GTC has major flaws like unnecessary preprocessing and

Journal of Information Systems Engineering and Management
2025, 10(59s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

indexing time, poor flexibility in concatenation-based patterns, and limited support for dynamic updates, which hampers its scalability in ever-evolving graph data environments.

2-Hop Labeling with Constraints

This approach is an extension of the typical two-hop labeling method to support reachability queries constrained by injecting edge semantics into the labeling [13]. In the basic 2-hop method, a node receives two sets: Lout of nodes that can be reached by it and Lin of nodes that can reach it. The extension introduces semantic constraints by enriching such labels with path knowledge, e.g., regular expressions, label sequences, or minimum repeat patterns, to ensure that only those paths satisfying a given label constraint are allowed.

EXPERIMENT AND RESULTS

The implementation was carried out on a machine with the following specifications:

•Processor: Intel Core i5 12th Generation

•Graphics Card: NVIDIA GeForce RTX 3060 Ti

•RAM: 32 GB

•Operating System: Windows 10

Evaluation metrics

• *Query Time* : Output Description: The output shows the results of testing 6 different algorithms across 4 query types Linear, Star, Snowflake, and Complex. Each query type is executed as a single full query file, and the table reports:

The execution time (in seconds) for each query type. The average execution time across all query types for each algorithm (see Table 1) .

Table 1. Query Time Output (s)

Algorithms	Linear	Star	Snowflake	Complex	Average
Tree Cover Index	1.2143	1.2327	1.1951	1.1667	1.2022
2-Hop Labeling	20.5068	19.3679	18.7434	20.1367	19.6887
Approximate Transitive Closure	2.3911	2.1670	2.2202	2.2338	2.2530
Generalized Transitive Closure	10.0094	9.4669	9.3101	10.0822	9.7172
2-Hop Labeling With Contrains	7.9271	7.2156	7.1401	7.6856	7.4921
Tree Based Indexing Reachability	5.6969	5.5733	5.5613	6.0592	5.7227

• *Index Size :*The output table summarizes the index sizes (in megabytes) generated by each indexing algorithm when processing different types of query graphs: Linear, Star, Snowflake, and Complex.

Rows represent the different algorithms tested (e.g., TreeCoverIndex, TwoHopLabeling, etc.). Columns correspond to the query types. Each cell shows the size of the index output file (in MB) produced by running the algorithm on that query type. The last column shows the average index size across all query types for each algorithm (see Table 2).

Table 2. Index Size Output (MB)

Algorithms	Linear	Star	Snowflake	Complex	Average
Tree Cover Index	0.0121	0.0121	0.0121	0.0121	0.0121
2-Hop Labeling	1.5156	1.5156	1.5156	1.5156	1.5156
Approximate Transitive Closure	1.5130	1.5130	1.5130	1.5130	1.5130
Generalized Transitive Closure	2.0950	2.0950	2.0950	2.0950	2.0950

Journal of Information Systems Engineering and Management
2025, 10(59s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

2-Hop Labeling With Contrains	0.2729	0.2729	0.2729	0.2729	0.2729
Tree Based Indexing Reachability	2.1028	2.1028	2.1028	2.1028	2.1028

• *Scalability :* The output shows the results of testing 6 different algorithms across 4 query types Linear, Star, Snowflake, and Complex. Each query type is tested with 3 input sizes labeled Small, Medium, and Large.

For each algorithm and query type, the table reports: The execution time (in seconds) for each input size.

Whether the algorithm is scalable across these sizes, based on how consistent its runtimes

are (less than 15% variation) (see Table 3(a,b,c,d)) .

Table 3 (a). Scalability for Linear Queries (s)

Algorithms	Small	Medium	Large	Scalable
Tree Cover Index	1.526	1.291	1.277	No
2-Hop Labeling	20.151	19.898	20.274	Yes
Approximate Transitive Closure	2.311	2.368	2.268	Yes
Generalized Transitive Closure	10.880	9.744	9.820	Yes
2-Hop Labeling With Contrains	7.242	7.009	7.004	Yes
Tree Based Indexing Reachability	5.823	5.634	5.433	Yes

Table 3 (b). Scalability for Star Queries (s)

Algorithms	Small	Medium	Large	Scalable
Tree Cover Index	1.466	1.276	1.331	Yes
2-Hop Labeling	20.085	20.979	22.229	Yes
Approximate Transitive Closure	2.179	2.276	2.258	Yes
Generalized Transitive Closure	9.261	9.262	9.654	Yes
2-Hop Labeling With Contrains	7.246	7.208	7.252	Yes
Tree Based Indexing Reachability	5.458	5.532	5.532	Yes

Table 3 (c). Scalability for Snowflake Queries (s)

Algorithms	Small	Medium	Large	Scalable
Tree Cover Index	1.428	1.342	1.489	Yes
2-Hop Labeling	21.957	20.496	21.215	Yes
Approximate Transitive Closure	2.267	2.428	2.252	Yes
Generalized Transitive Closure	10.126	10.271	10.001	Yes
2-Hop Labeling With Contrains	7.871	7.169	6.997	Yes
Tree Based Indexing Reachability	5.408	5.422	5.443	Yes

Table 3 (d). Scalability for Complex Queries (s)

Algorithms	Small	Medium	Large	Scalable
Tree Cover Index	1.498	1.432	1.813	No

Journal of Information Systems Engineering and Management
2025, 10(59s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

2-Hop Labeling	20.729	19.942	20.427	Yes
Approximate Transitive Closure	2.216	2.180	2.277	Yes
Generalized Transitive Closure	9.656	9.645	9.281	Yes
2-Hop Labeling With Contrains	7.097	7.141	7.051	Yes
Tree Based Indexing Reachability	5.524	5.455	5.586	Yes

• *Index Construction Time :* The table.4 reports the index construction time (in seconds) for six different algorithms across four types of query graph structures: Linear, Star, Snowflake, and Complex.

Each cell shows the time taken by an algorithm to build its index when given the input query file corresponding to a specific query type. The Average column provides the mean construction time for each algorithm over all query types. Lower times indicate faster index construction performance.

Table 4. Index Construction Time Output (s)

Algorithms	Linear	Star	Snowflack	Complex	Average
Tree Cover Index	1.2015	1.2480	1.4614	1.3076	1.3046
2-Hop Labeling	20.3359	19.8517	20.7754	20.2210	20.2960
Approximate Transitive Closure	2.5881	2.5048	2.2073	2.3855	2.4214
Generalized Transitive Closure	10.1109	10.1948	9.9391	10.0056	10.0626
2-Hop Labeling With Contrains	7.0810	7.4425	7.5847	7.7797	7.4720
Tree Based Indexing Reachability	5.6092	5.7582	5.9000	5.8202	5.7719

DISCUSSION

The results show that no algorithm is universally optimal. Two-hop labeling methods offer a good compromise between response time and index size. Transitive closure methods remain relevant for cases requiring completeness. Tree-based indexes are simple and efficient for small graphs but lose performance on large structures.

Thus, each method's relevance depends on the context of use (See Table 5). Integrating an adaptive approach, dynamically selecting the algorithm based on the size and structure of the RDF graph, could be an avenue for future optimization.

Table 5. Strengths and Limitations of the Tested Algorithms

Algorithm	Main strengths	Identified limitations
Tree Cover Index	Simple to implement, efficient on hierarchical graphs	Not suitable for large graphs
Tree-Based Index	Good fragment organization, low memory cost	Limited scalability
Two-Hop Labeling	Fast response time, suitable for large graphs	Higher index size
Two-Hop Labeling (Constraint)	Reduced false positives, good trade-off	Increased construction complexity
Approx. Transitive Closure	Faster construction than a full closure	Approximation may produce errors
Generalized Transitive Closure	Guaranteed completeness, always correct answers	Very costly in memory and computation

CONCLUSION

In this work, we studied the problem of reachability in large RDF graph environments, focusing on improving the execution performance of the RDF QDAG system. Our main objective was to better understand reachability indexing techniques and to test how well existing methods perform through a detailed benchmarking study.

Journal of Information Systems Engineering and Management
2025, 10(59s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

We built a benchmarking framework and evaluated six main approaches: Tree Cover Index, Tree-Based Index, Two-Hop Labeling, Two-Hop Labeling with Constraint, Approximate Transitive Closure, and Generalized Transitive Closure. Each technique was tested on RDF QDAG fragments and compared using several criteria, such as query execution time, index size, construction time, scalability, complexity, support for dynamic queries, index structure, and compatibility with path constraints and edge-labeled graphs.
The results showed that every method has its own strengths:

- **Query Time**: Two-Hop Labeling with Constraint gave the fastest query responses, which makes it suitable for real-time applications.
- **Index Size**: Tree Cover Index produced the smallest index structures, useful in memory-limited settings.
- **Scalability**: Generalized Transitive Closure worked well on large and complex RDF graphs, though it required more time to build the index.
- **Path-Constrained Queries**: Two-Hop Labeling with Constraint and Generalized Transitive Closure handled edge labels and structural patterns effectively.

The key conclusion is that there is no single best solution. The choice of method depends on the application's priorities—whether it requires speed, small memory use, or support for complex queries.
This study combines theoretical analysis with extensive experiments and provides a reproducible framework for evaluating reachability methods in RDF graphs. We believe it offers a solid base for future work in semantic graph processing, data integration, and scalable knowledge representation.

REFRENCES

[1] C. Opara, Yingke Chen, and Bo.wei. "Look Before You Leap: Detecting Phishing Web Pages by Exploiting Raw URL And HTML Characteristics." shaq Zouaghi, Amin Mesmoudi, Jorge Galicia, Ladjel Bellatreche, and Taoufik Aguili. Gofast: Graph-based optimization for efficient and scalable query processing over large rdf graphs. Information Systems, 99:101738, 2021.
[2] Chao Zhang, Angela Bonifati, and M. Tamer Özsu. An overview of reachability indexes on graphs. In Companion of the 2023 International Conference on Management of Data (SIGMOD), pages 61–68. Association for Computing Machinery, 2023.
[3] Rakesh Agrawal, Alexander Borgida, and H. V. Jagadish. Efficient management of transitive relationships in large data and knowledge bases. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 253–262, 1989.
[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms. MIT Press, 3rd edition, 2009.
[5] Reinhard Diestel. Graph Theory. Springer, 5th edition, 2017.
[6] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. Scientific American, 284(5):34–43, 2001.
[7] Hui Wang, Xin Wang, Menglu Ma, and Yiheng You. Rpqbench: A benchmark for regular path queries on graph data. In Proceedings of WISE 2024. Springer, 2025.
[8] Mingdao Li, Peng Peng, Zheyuan Hu, Lei Zou, and Zheng Qin. Variable-length path query evaluation based on worst-case optimal joins. In Proceedings of the 2024 IEEE 40th International Conference on Data Engineering (ICDE). IEEE, 2024.
[9] Houssameddine Yousfi, Amin Mesmoudi, Allel Hadjali, Houcine Matallah, and Seif-Eddine Benkabou. Srdf qdag: An efficient end-to-end rdf data management when graph exploration meets spatial processing. Computer Science and Information Systems, 20(4), 2023.
[10] Leslie G. Valiant. A bridging model for parallel computation. Communications of the ACM, 33(8):103–111, 1990.
[11] Edith Cohen, Eran Halperin, Haim Kaplan, and Uri Zwick. Reachability and distance queries via 2-hop labels. SIAM Journal on Computing, 32(5):1338–1355, 2003.
[12] Ruoming Jin, Hui Hong, Haixun Wang, Ning Ruan, and Yang Xiang. Computing labelconstraint reachability in graph databases. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 123–134, 2010.
[13] Yangjun Chen and Gagandeep Singh. Graph indexing for efficient evaluation of labelconstrained reachability queries. ACM Transactions on Database Systems, 46(2), 2021.
[14] Lei Zou, Kian-Lee Tan, Jeffrey Xu Yu, and Dongxiang Zhang. Efficient processing of labelconstraint reachability queries in large graphs. Information Systems, 40:47–66, 2014.
[15] Hao Wei, Jeffrey Xu Yu, Can Lu, and Ruoming Jin. Reachability querying: An independent permutation labeling approach. Proc. VLDB Endowment, 7(12):1191–1202, 2014.
[16] Jiefeng Su, Qin Zhu, Hao Wei, and Jeffrey Xu Yu. Reachability querying: Can it be even faster? IEEE Transactions on Knowledge and Data Engineering, 29(3):683–697, 2017.

Journal of Information Systems Engineering and Management
2025, 10(59s)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

[17] James Cheng, Yi Ke, and Hong Cheng. Efficient processing of reachability queries with label constraints. In Proceedings of the 22nd ACM international conference on Information & Knowledge Management, pages 1229–1238, 2013.