# Graph Databases for Information Retrieval

Chris Kamphuis(✉)

Radboud University, Nijmegen, The Netherlands
chris@cs.ru.nl

**Abstract.** Graph models have been deployed in the context of information retrieval for many years. Computations involving the graph structure are often separated from computations related to the base ranking. In recent years, graph data management has been a topic of interest in database research. We propose to deploy graph database management systems to implement existing and novel graph-based models for information retrieval. For this a unifying mapping from a graph query language to graph based retrieval models needs to be developed; extending standard graph database operations with functionality for keyword search. We also investigate how data structures and algorithms for ranking should change in presence of continuous database updates. We want to investigate how temporal decay can affect ranking when data is continuously updated. Finally, can databases be deployed for efficient two-stage retrieval approaches?

**Keywords:** Graph databases · Information retrieval · Query languages

## 1 Motivation

Many IR systems make use of graph-based models. For social media Clements et al. [5] show this by using random walks over typed social media. In the context of websearch, Metzler et al. [9] and Craswell et al. [6] show the usefulness of using anchor text. Vuurens et al. [12] presented a news tracker for ad-hoc information needs. They used a temporal graph model; implemented as a standalone application over streaming graphs. Entity oriented search has been another topic of interest in the IR community [1,2], where researchers make extensive use of knowledge bases to improve ranking. It would be a natural choice to use graph databases to represent these models, especially if the system allows for data to be updated continuously. The goal of this PhD research is to represent graph-based models in a graph database that also supports keyword search. This would allow graph based methods to be combined with traditional keyword search, without using different system components.

## 2 Background and Related Work

There has been a long history of IR research in combination with relational databases. Recently, Mühleisen et al. [10] implemented such a system, where they achieved efficiency and effectiveness on par with custom made inverted index systems. They argue that using relational databases instead of custom-made inverted indexes has the following advantages for IR.

Firstly, viewing IR as a database application offers a formal framework for complex query operators. This forces IR researchers to be precise on how to deal with more complex query operators and their edge cases. By expressing the ranking logic in query languages, it is not possible to resort to heuristics or shortcuts. This helps explaining the resulting rankings of documents. Secondly, a relational database offers a clean system architecture. Storage management is taken care of by the database engine, separating low level components shared in any data application from IR components. Thirdly, advances in database research offer benefits on systems build using databases. All performance gains in database engines will directly trickle down to the IR system. Fourthly, databases offer additional tools for error analysis. E.g., join together relevance judgement, document representations and the result set to generate scatter plots. Finally, it offers opportunities for rapid prototyping. Many IR researchers might not be interested in data management and query evaluation. This can be taken care of by the database engine, researches can focus on methods for ranking.

In previous work [7] we argued how a graph query languages for IR also yield benefits for reproducible IR research.

Industry has also shown an interest in graph databases in combination with full text IR. Neo4J, the most popular database[1], supports graph analysis in combination with full text analysis using Lucene[2]. Lucene is however embedded in their graph query language, while we envision retrieval to be carried out by the same system that does the graph processing. Busch et al. [4] presented Earlybird, a system that allows continuous updates to be searched real-time. They show how updates can be processed in the context of inverted indexes. A similar approach could be taken in the context of graph databases. Böttcher et al. [3] presented an approach how updates for compressed databases can be implemented efficiently by updating the data without decompressing all data.

## 3 Proposed Research and Methodology

**How can keyword search be integrated with a graph query language?** Many retrieval models make use of graph information, often graph-based models support traditional keyword search. When implementing graph-based models in a graph database, it should be able to integrate graph-based results with keyword search seamlessly.

As shown by Mühleisen et al. [10], keyword search can be expressed as SQL queries over relational databases and executed efficiently. The goal of the proposed research is to extend this idea, and express keyword search in graph databases. This could be approached in two ways. One way would be to try

[1] https://db-engines.com/en/ranking/graph+dbms, Last Accessed: November 2019.

[2] https://neo4j.com/docs/cypher-manual/3.5/schema/index/#schema-index-fulltext-search, Last Accessed: November, 2019.

to express the keyword ranking functions as structured queries in a graph query language. This could for example be achieved by representing documents and terms as properties in the graph e.g. vertices themselves, and translate classic keyword search algorithms to functions over the graph. Especially in the context of entity ranking, where entities are part of a knowledge base, this would make sense. Another approach would be to follow Mühleisen et al., and represent the graph database as an extension on top of the relational database. Using the relational database as the core for a graph database would be a natural fit, as graph traversal algorithms can be expressed using recursive join operations. As keyword search can already be expressed in the relational database, combining keyword search with the additional graph information will computationally be cheap.

**What data structures and algorithms should be used when data is continuously updated?** Often when IR systems are being developed, they are evaluated on static data. When deploying such a system to production, it might not be as effective or efficient when data is constantly changing. For example, index compression algorithms might assign document identifiers to documents according to their contents; delta gaps should be as small as possible. When data is constantly updated, many of these algorithms will not work. Graph data does tend to appear in the context when data constantly is changing. Ideally the data does not have to be re-indexed all the time, the underlying index can just be updated when changes to the data are being presented.

In order for a graph database system to be useful in an IR context, it needs to be efficient in both handling updates to data and when it is used for ranking. In our research we want to investigate which algorithms and data structures can be used to efficiently handle both database transactions and retrieval.

**How should edges/vertices be ranked in the presence of continuous updates?** When graph databases do support continuous updates, temporal graphs could be represented in the graph database. One could add timestamps to vertices to store useful temporal information of the graph (e.g. when the is vertex added). The question then arises how the temporal evolution of the graph should affect the ranking scores of the edges and/or vertices. Examples of use-cases where this would be especially interesting include tweets and comments on videos, where not only their content but also their recency are relevant. Using a temporal graph database would also be an ideal setup to establish a connection between temporal decay and ranking in models for temporal summarizing as described in the Real-Time Summarization track [8].

**Can a database be used in order to unify different stages in the ranking process?** IR systems often consist of multiple parts that carry out different stages in the retrieval process. An initial ranker calculates an initial ranking score, e.g. BM25, over inverted indexes. After the initial ranking stage, a more effective, but computationally more expensive method re-ranks the top-$k$ retrieved documents. This second stage re-ranker is often completely detached from the initial ranker, e.g. neural approaches and learning-to-rank methods are often implemented in different programming languages than the initial ranker. Data from the inverted index and results from the initial retrieval step, need to be copied in order for the re-ranking algorithm to work. Often re-ranking algorithms work with different data structures than inverted indexes do, so it might be necessary to restructure the data also. These mismatches between retrieval stages can introduce latency in the full retrieval pipeline. In order to avoid this latency, ideally the initial ranking stage and the re-ranking stage can be expressed in the same programming language using the same data structures such that latency introduced by re-ranking is only an effect of the re-ranking algorithm itself.

Recently Raasveldt and Mühleisen [11] presented DuckDB, an analytical embeddable database system. It has been specifically designed for executing analytical queries fast in an embedded environment. This research proposes to use DuckDB as a database backend for fast top-$k$ retrieval by implementing methods as described by Mühleisen et al. [10]. DuckDB has bindings for Python, the *de facto* language used for experiments with neural methods. DuckDB supports extracting the results of the issued queries directly to NumPy arrays, allowing for (neural) re-rankers to quickly start the re-ranking without first having to move the data. We want to investigate whether the results of Mühleisen et al. [10] also hold for DuckDB. If so, latency introduced by moving data might be minimized, and re-ranking systems might be able to re-rank the top-$k$ faster. Allowing more re-ranking methods to be deployed in production.

## 4 Discussion

**Are graphs to correct abstraction level in the context for IR?** One could argue that graphs are not the correct abstraction level in the context of IR. Specifically expressing keyword queries as graph database queries might make things more complex. We would argue that although this might be seen as a disadvantage, it would allow for more complex (graph inherent) structures to be integrated with keyword search more easily.

**How should graphs be constructed for text documents?** It would be possible to construct graphs from text documents in different ways. What would be the right granularity of the graph? Edges could represent terms in a document and link terms together when they appear in the same document. Maybe it would make more sense to express sentences or even documents as edges in the graph. For example in the context of the web, a document level graph could make sense. It would also make sense to change the granularity depending on the search task. When processing the text documents, should preprocessing (stopping, stemming) be integrated in the graph query language or should the graph query language only be used for querying the database retrieval time?

**Acknowledgements.** This work is part of the research program Commit2Data with project number 628.011.001 (SQIREL-GRAPHS), which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

## References

1. Balog, K.: Entity-Oriented Search. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-93935-3
2. Balog, K., Serdyukov, P., De Vries, A.P.: Overview of the trec 2010 entity track. Tech. rep. Norwegian University of Science and Technology Trondheim (2010)
3. Böttcher, S., Bültmann, A., Hartel, R., Schlüßler, J.: Implementing efficient updates in compressed big text databases. In: Decker, H., Lhotská, L., Link, S., Basl, J., Tjoa, A.M. (eds.) DEXA 2013. LNCS, vol. 8056, pp. 189–202. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40173-2_17
4. Busch, M., Gade, K., Larson, B., Lok, P., Luckenbill, S., Lin, J.J.: Earlybird: real-time search at twitter. In: 2012 IEEE 28th International Conference on Data Engineering, pp. 1360–1369 (2012)
5. Clements, M., De Vries, A.P., Reinders, M.J.T.: The task-dependent effect of tags and ratings on social media access. ACM Trans. Inf. Syst. **28**(4), 1–42 (2010). https://doi.org/10.1145/1852102.1852107
6. Craswell, N., Hawking, D., Robertson, S.: Effective site finding using link anchor information. In: Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2001, pp. 250–257. ACM, New York (2001). https://doi.org/10.1145/383952.383999
7. Kamphuis, C., de Vries, A.P.: Reproducible IR needs an (IR) (graph) query language. In: Proceedings of the Open-Source IR Replicability Challenge Co-Located with 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, OSIRRC@SIGIR 2019, Paris, France, 25 July 2019, pp. 17–20 (2019). http://ceur-ws.org/Vol-2409/position03.pdf
8. Lin, J., et al.: Overview of the TREC 2017 real-time summarization track. In: TREC (2017)
9. Metzler, D., Novak, J., Cui, H., Reddy, S.: Building enriched document representations using aggregated anchor text. In: Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2009, pp. 219–226. ACM, New York (2009). https://doi.org/10.1145/1571941.1571981
10. Mühleisen, H., Samar, T., Lin, J., de Vries, A.: Old dogs are great at new tricks: column stores for IR prototyping. In: Proceedings of the 37th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2014, Gold Coast, Australia, pp. 863–866 (2014)
11. Raasveldt, M., Mühleisen, H.: [demo] DuckDB: an embeddable analytical database. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 1981–1984, June 2019. https://doi.org/10.1145/3299869.3320212
12. Vuurens, J.B., de Vries, A.P., Blanco, R., Mika, P.: Online news tracking for adhoc information needs. In: Proceedings of the 2015 International Conference on The Theory of Information Retrieval, ICTIR 2015, pp. 221–230. ACM, New York (2015). https://doi.org/10.1145/2808194.2809474