Reproducible IR needs an (IR) (Graph) Query Language

Chris Kamphuis
ckamphuis@cs.ru.nl
Radboud University
Nijmegen, The Netherlands

Arjen P. de Vries
arjen@acm.org
Radboud University
Nijmegen, The Netherlands

ABSTRACT

IR research should concentrate on the retrieval model, and not be hindered by its implementation in terms of low-level data structures; and we should embrace graph databases to realize that vision!

Even results from retrieval systems based on the classic Okapi BM25 retrieval model (an approach that *seems* straightforward to implement) have been remarkably difficult to actually reproduce in practice. Pin-pointing the cause of *not* obtaining identical results on an identical collection is surprisingly hard, especially since retrieval systems usually mix the retrieval model itself with the code necessary to make IR efficient. Unlike the database community, our field has never moved away from implementing query processing directly on the file system; historically best attributed to a need for speed on large data sets of heterogeneous documents, never a good match for our database colleagues' solutions.

Nevertheless, this position paper calls for a shift toward a declarative approach to specify retrieval. Not only has the state of the art in database query processing reached a level where doing IR as database queries is not unimaginable, advancing retrieval effectiveness has become dependent on systems that handle complex data models in multiple layers of processing, usually involving machine learning and thereby including the data itself into the retrieval model. If the results of ranking with a straightforward combination of term frequency and document frequency can hardly be reproduced, how will we advance our field when researchers face the implementation of ranking formulas that integrate the data itself, knowledge bases and advanced machine learning methods?

Reproducibility in IR will be much easier to achieve if the retrieval model can be expressed concisely (and precisely!) in terms of operations over the document-term graph, or, probable today, a document-metadata-entity-term-graph. We propose to adopt G-core, a language proposed by the database community to become a standard graph query language, and where necessary extend it for IR purposes. IR researchers and practitioners can represent documents and annotations naturally as graphs. We discuss the advantages of a graph model over previous work using relational databases, and illustrate how the key properties of simple and extended IR models can be expressed naturally.

OSIRRC 2019 co-located with SIGIR 2019, 25 July 2019, Paris, France.

CCS CONCEPTS

• **Information systems** → *Query languages*; *Document representation*; *Evaluation of retrieval results*; *Search engine indexing*.

KEYWORDS

information retrieval, graph query language, reproducibility,

1 INTRODUCTION

The IR community should work towards being a community where reproducibility is the norm. We recognize that such a process can happen gradually, and that weaker forms of reproducibility, like replicability, are better than none. However, when introducing tools for replicability, researchers should keep in mind that replicability is only a temporarily solution in the greater picture when working towards reproducibility.

The definition of reproducibility by the ACM[1]states that, for computational experiments, an independent group can obtain same results using artifacts they developed independently. Exact reproduction is not required, but results obtained should support the claims of the original work. For IR experiments, this implies that evaluation metrics observed in the reproduction study should be within a tolerable margin from those in the published study.

In general, exact reproduction of studies is not a realistic expectation to hold, and illustrated well by two published results from the information retrieval field. First, work carried out by Mühleisen et al. [5] compared multiple systems that all claim to implement the BM25 ranking formula, and found that four different implementations result in four different effectiveness scores, both in MAP and P@5. Given that these systems included Indri [9] and Terrier [6], this came out as quite a surprise; the authors took specific care to keep document pre-processing identical for both systems, but the observed difference in MAP of 3% absolute was the largest deviation in score reported in that study. Similarly, the RIGOR workshop compared different systems to each other, out of which four systems implemented the BM25 ranking model Arguello et al. [2]. Again, the differences in effectiveness amounted to 3% absolute on MAP@1000.

Whether these differences are 'within a tolerable margin' is arguable. Why the differences are so large has not been uncovered in detail; the authors of [2] pointed at tuning parameters, differences in interpretation of the equations, and optimizations. We suspect that implementations might

[1]https://www.acm.org/publications/policies/artifact-review-badging, last accessed July 4*th*, 2019.

(unintentionally) modify the specification of the ranking formula when implementing techniques to obtain faster query processing. Whatever the correct explanation, what we can conclude is that, apparently, widely used experimental IR systems *do not implement the same BM25 ranking formula* (and given the differences between all systems, it is not clear which one implements the ranking formula that was proposed in Robertson and Walker [7]).

As the scientific field that aims to identify the mechanisms that can predict relevance computationally, it is disappointing that we cannot even reach conclusive results for classic, highly effective models. So, how can we improve upon the current situation?

This paper takes the position that IR research should use higher level abstractions in the implementation of the retrieval models we study. Prototyping retrieval models using a declarative query language would improve reproducibility, because the query expressions used provide a much better basis to help investigate the differences that would explain observed variations in effectiveness scores. If the reason between such differences can be explained, they can either be fixed or kept depending on the nature of the cause.

More specifically, we propose to adopt a graph query language as prototyping tool to achieve reproducibility in IR experimentation. We first review arguments given in favour of 'a database approach to IR'. We explain how a graph query language would be used for IR tasks and discuss its advantages over other approaches. After sketching our solution direction, we conclude by identifying the challenges that still lie ahead.

2 A GRAPH QUERY LANGUAGE FOR IR

Mühleisen et al. [5] have argued that relational databases implemented as column store would be a suitable choice for creating information retrieval systems, and demonstrated how TF-IDF like ranking functions can be expressed easily and executed efficiently (show-casing a conjunctive BM25 ranking function). They give the following arguments in favour of 'a database approach to IR':

- A query language is a formal framework in which query evaluations have to be formulated precisely. Researchers do not have the option the resort to shortcuts in corner cases;
- data management components are separated from the query evaluation specifications, reducing system complexity and yielding a better structured architecture;
- advances made in the database community (e.g. moving from column-wise query processing to vectorized query processing) will benefit the retrieval engine 'for free';
- error analysis can be carried out on the same database that represents the collection. A researcher can write additional queries for analysis without having to write low-level code to interact with the data;
- a database provides a rapid prototyping tool. IR researchers are primarily interested in questions regarding the ranking components of their problem. They only have to focus on issuing queries, without having to write additional code.

All of these arguments relate directly to improving reproducibility of IR experimentation. However, what the authors did not mention is the increased friction of mapping all the elements of an information retrieval model onto a relational representation, especially when both the documents and the ranking functions have been increasing in complexity in recent years. While it is definitely *possible* to express retrieval models in SQL or a relational algebra variant (assuming it includes aggregation operators), this is not a *convenient* approach, and, in our opinion, error-prone as well.

We embrace the arguments in favour of a database approach to IR, but propose to move along with recent trends in the database community and model documents and queries as graphs, and adopt a graph database model instead of a relational model. Recently, a standard query language over graphs, G-CORE, has been introduced in Angles et al. [1]. The language proposal is supported by database researchers and practitioners united in the Linked Data Benchmark Council (LDBC)[2], and (at least partially) implemented by vendors like Neo4J. In other words, it is the right time to consider the question how to express information retrieval problems in G-CORE, and identify possible extensions that might be necessary for adoption in our community.

2.1 Documents and terms

G-core assumes the property graph data model: graphs are directed, have labels on both nodes and edges, and every node and edge can have associated `<property, value>` pairs.

We can represent both documents and terms as nodes of the graph. Edges are formed between a document node and a term node if the term appears in the document. If a term appears multiple times in a document, G-CORE allows for multiple edges between two nodes to exist. Edges may, optionally, save the position of the term in the document.

G-CORE is a graph query language that is closed over property graphs; when querying a graph, the result is another graph. Expressing a retrieval model like BM25 then corresponds to defining different graphs to determine the different components in the ranking function. To illustrate, consider how the term frequency of term t given a document D is computed from the graph specified using the G-CORE query shown in listing 1.

```
CONSTRUCT (d)<-[:appearsIn]-(t)
  MATCH (d:Document) ON document_graph,
        (t:Term)     ON term_graph
    WHERE t.value = query_t.value
    AND d.id = doc_D.id
```

Listing 1: Term frequency for term document pair

The resulting graph contains two nodes; the document node of document D and the term node of term t. Term frequency t_D is calculated by counting the number of edges connecting these nodes. G-CORE does not offer methods

[2] http://ldbcouncil.org, last accessed June 14th, 2019.

to return properties yet, but Angles et al. [1] show how an extension of the language would implement this; listing 2 expresses this using the current implementation in Neo4J's Cypher [4].

```
CONSTRUCT (d)<-[e:appearsIn]-(t)
  MATCH (d:Document) ON document_graph,
        (t:Term)     ON term_graph
    WHERE t.value = query_t.value
    AND d.id = doc_D.id
RETURN COUNT(e)
```

Listing 2: Extension to return properties

A term's Inverse Document Frequency (IDF) statistic can be determined similarly: construct a graph consisting of all nodes that are directly connected to the term node. The number of document nodes in this graph is equal to the number of documents in the collection containing query term t. Listing 3 shows how this graph can be constructed.

```
CONSTRUCT (d)
  MATCH (d:Document) ON document_graph,
        (t:Term)     ON term_graph
    WHERE t.value = query_t.value
    AND (d)<-[:appearsIn]-(t)
```

Listing 3: Number of documents containing term

2.2 Entities and metadata

For the BM25 example, both the relational and the graph database would be appropriate. However, when the data becomes more (semi-)structured, using a graph model will have clear advantages. Consider for example the TREC news track Soboroff et al. [8], that introduces two retrieval tasks: background linking and entity ranking. Both can be expected to benefit from a richer document representation, where the documents are enriched with their metadata and entity annotations – perfectly modelled as a document graph.

2.2.1 Background linking. For the background linking tasks, the news articles to be retrieved provide background information to help understand a query article of interest. Annotating the documents in the collection with metadata can be expected to help this task. For example, articles can be linked with author information. Articles written by the same author might have a higher chance of being relevant, as authors tend to write articles within a narrow range of topics. As articles can be written by multiple authors it also possible to identify which authors co-authored often. This also might help when determining if an article is suited for background reading. These extra annotations can easily be represented in a graph database, authors are simple nodes that are connected to article nodes if they authored the article. The graph of all articles written by the same author as the query article can be constructed using the query shown in listing 4.

```
CONSTRUCT (d)
  MATCH (d:document)
    WHERE (d)-[:writtenBy]->()<-[:writtenBy]-(query_d)
```

Listing 4: Documents written by same author

2.2.2 Entity ranking. For the entity ranking task, entities that appear in a query article, need to be ranked. The more important an entity is in context of the article, the higher it should be ranked. The set of entities that appear in the document is provided, and, obviously, considering the documents as graphs lets us represent this information directly. For ranking the entities one might want to know the collection statistics of said entities. In order to measure these, we would extend the documents by running an entity tagger. We can easily express entity occurrence in articles as graph queries, and, if deemed useful to improve effectiveness, bring in additional information from a knowledge base, an external graph that we can join as easily, providing a complete and high level specification of the exact way how the knowledge base would be used in the ranking function.

2.3 Helping reproducibility

So far, we have focused on using a graph database for prototyping information retrieval models. The common practice today among IR researchers and practitioners is however to implement their approach to retrieval directly on top of an inverted file index structure. The result is an implementation with many different interwoven individual components: tokenisation, stemming, stop word removal, memory management, query processing, etc. When trying to reproduce a study, it can be quite a challenge to determine which parts of the process are different from the original study.

For a graph database solution, data management is taken care of by the database system. The ranking function is expressed in the graph query language. This setup allows for easy analysis if a particular component fails in the reproduction process. Ideally the original work and the reproduction study both have represented their work with a graph database. If reproduction fails in that case it easy to analyze whether the ranking function represented in the graph query language is the same and whether the document representations are identical. If either one is different it might explain why it was not possible to reproduce results.

Consider again the study [5]. The query expression to capture the retrieval model is easily shared between research groups. The column store approach was executed on two different database engines, MonetDB Boncz [3] and VectorWise Zukowski et al. [10]. Remarkably, only those two systems produced identical effectiveness scores, even though the execution engines are completely different; only the queries representing the IR model are identical. Had the effectiveness scores not been the same, that would show that the systems processed the queries differently. The cause of a difference would be clear: either a bug, or differences in the data. If another group would also represent their document collection on one of these column store databases, and would find different effectiveness scores, the document representations must differ. This might be the case because of text processing. It is immediately clear why differences probably occur when investigating this way.

Even if the original work is not executed in the form of graph queries, our approach is still useful for analysis during a reproduction study. Because of the separation of concerns between data management and the expression of the ranking function, comparing minor modifications in data (pre-)processing and ranking gives the researchers a low-effort opportunity to explore the effects of these changes. If some implementation details are not provided in a scientific publication, the graph database allows for quick analysis to compare different implementations that follow from the left out details.

3 FUTURE DIRECTIONS

Using a graph database with a graph query language instead of a column store database with a structured query language, offers new research opportunities. Firstly, we need empirical research to determine whether graph databases have matured sufficiently to execute the information retrieval queries as efficient as their alternatives. Secondly, how should keyword search be integrated in a graph query language? In the previous section a suggestion on how this can be accomplished was made, but different representations may be explored, both from an efficiency and an effectiveness perspective.

With respect to expressiveness of the language, the XQuery Fulltext vs. NEXI debate from the XML IR era should be revisited. Should we add operators that express clearly how the graphs are constructed from the underlying documents, including stemming and text normalization? With the increased use of neural methods, topic models, and word embeddings, the question remains to what extent the construction and/or their application should also be captured in declarative queries. Creation of a topic model can be viewed as an aggregation operation, just like we express the computation of IDF in context of a collection graph. The nearest neighbour search operations in high dimensions for using word embeddings can be expressed declaratively, and even the efficient query processing strategies necessary (e.g. citebond02). Future research will have to point out the need for expressing these (often merely pre-processing) steps in the same graph query language, or are more naturally addressed as user-defined functions that primarily call external machine learning libraries.

Finally, it is an open question how to rank edges and/or nodes when taking into account the graph structure itself. When the data is represented as a graph, graph specific ranking components like pagerank might easily be taken advantage of. Here, a specific property of interest is the notion of path expressions offered by the G-CORE query language.

Graph structures often appear in the context of social media. In this context the graph constantly changes. New nodes and edges are created all the time, and nodes and edges might also leave the network. It will be useful to consider the aspect that the graph structure constantly changes. So, it is finally of interest to investigate which data structures and algorithms are efficient in presence of continuous updates. Finally, how can continuous updates of the graph be sampled efficiently? It might be hard do determine when we need to check which parts of the graph need to be updated when.

4 CONCLUSION AND RECOMMENDATIONS

In this position paper we propose that IR practitioners should represent their data as graphs managed in a graph database system. We argue that this helps when trying to reproduce studies, and will simultaneously make your own research more reproducible for others.

ACKNOWLEDGMENTS

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] Renzo Angles, Marcelo Arenas, Pablo Barcelo, Peter Boncz, George Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan Sequeda, Oskar van Rest, and Hannes Voigt. 2018. G-CORE: A Core for Future Graph Query Languages. In *SIGMOD (SIGMOD '18)*. ACM, New York, NY, USA, 1421–1432. https://doi.org/10.1145/3183713.3190654

[2] Jaime Arguello, Fernando Diaz, Jimmy Lin, and Andrew Trotman. 2015. Sigir 2015 workshop on reproducibility, inexplicability, and generalizability of results (rigor). In *SIGIR*. ACM, 1147–1148.

[3] Peter Boncz. 2002. *Monet: A next-generation DBMS kernel for query-intensive applications*. Ph.D. Dissertation. Universiteit van Amsterdam.

[4] Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An evolving query language for property graphs. In *SIGMOD*. ACM, 1433–1445.

[5] Hannes Mühleisen, Thaer Samar, Jimmy Lin, and Arjen De Vries. 2014. Old dogs are great at new tricks: Column stores for IR prototyping. In *SIGIR*. ACM, 863–866.

[6] Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Christina Lioma. 2006. Terrier: A high performance and scalable information retrieval platform. In *Proceedings of the OSIR Workshop*. 18–25.

[7] Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *SIGIR*. Springer, 232–241.

[8] Ian Soboroff, Shudong Huang, and Donna Harman. 2018. TREC 2018 News Track Overview. In *The Twenty-Seventh Text REtrieval Conference (TREC 2018) Proceedings*.

[9] Trevor Strohman, Donald Metzler, Howard Turtle, and W. Bruce Croft. 2005. *Indri: A language-model based search engine for complex queries (extended version)*. IR 407. University of Massachusetts.

[10] Marcin Zukowski, Mark Van de Wiel, and Peter A Boncz. 2012. Vectorwise: A Vectorized Analytical DBMS.. In *ICDE*. 1349–1350.