Graphs and Information Retrieval

Proefschrift

ter verkrijging van de graad van doctor
aan de Radboud Universiteit Nijmegen
op gezag van de rector magnificus prof. dr. J.H.J.M. van Krieken,
volgens besluit van het college van decanen
in het openbaar te verdedigen

op woensdag 22 maart 2023

om 12:00 uur precies

door

Chris Frans Henri Kamphuis
geboren op 22 maart 1993
te Oldenzaal, Nederland

Promotor:

prof. dr. ir. A.P. (Arjen) de Vries

Manuscriptcommissie:

Person A	(Affiliation)
Person B	(Affiliation)
Person C	(Affiliation)
Person D	(Affiliation)
Person E	(Affiliation)

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).

Printed by a drukkerij met een naam

Typeset using LaTeX

ISBN: 111-11-11111-11-1

Contents

Chapter 1

Introduction

> I *propose* to consider the question, "Can machines think?"
>
> Alan Turing - 1950

I also propose to consider the question, "Can machines think?" Instead of approaching this through a thought experiment as Turing did, nowadays, one can approach this question by asking it to a search engine. When issuing this query to popular web search systems, we get varying results: the first result on Google is a passage generated from the article written by Turing, while the first result on Bing is a passage generated from a website that concludes machines can not think.[1,2]

We use systems that process queries daily when looking for *information.* While Google and Bing are all-purpose web engines that mainly focus on finding and retrieving information from the internet, people also use specialized search systems in their day-to-day lives: Amazon and eBay when we are looking for a product to buy, Scholar and ResearchGate for scientific resources, Youtube and TikTok for Videos, or Facebook and LinkedIn when we are searching for people. It might even be possible that you are reading this text after you found this document through search.

When searching for the query "Can machines think?", searching

[1]However, if a machine can not think, can we trust the result presented by this algorithm?

[2]These results were retrieved in October of 2022

through text documents only might be sufficient for the person who searches. However, more than considering the text is needed when searching today. For example, when one wants to buy a product on Amazon, aspects other than text also need to be considered. Let's say you want to buy an iPhone; information on the price, which edition is the most recent, or which color it has are all essential to determine which one you want. You may also want to consider the rating provided by people that previously bought an iPhone.

If someone searches for people on LinkedIn, they are generally more interested in persons that have connections in common compared to strangers. If you are looking for someone to do a job, it is ideal that a shared connection can vouch for them. In this case, how people relate to each other in their network might indicate *relevance*. Not only the structure of how people relate to each other determines relevance; other examples are their experience, where they work, or reviews of their previous work might matter.

Although it might be possible to encode all this information as written text, often, it is more convenient to save this information in a more structured approach. Where information retrieval researchers research the retrieval of information through text data, the retrieval of structured data is researched by data management researchers. In this thesis, both methods are considered simultaneously: systems that can work with structured and unstructured information are investigated.

1.1 Problem Description and Research Questions

Although information retrieval and data retrieval are research fields investigated by different disciplines, they are closely related, and systems that use both have been researched and developed in the past (In later parts of this thesis, examples are provided). Also, techniques developed in one community might help the other, as things like storing data and quickly retrieving it are essential for information and data retrieval.

In recent years there has been much exciting research in the database community studying graph databases. What these databases exactly are will be described in chapter 2. As these databases are becoming more popular for data retrieval tasks where the data is highly interconnected, they might also benefit similar tasks in the information retrieval field where data is often highly interconnected. This thesis will investigate how these databases, with dedicated graph query languages, can be used for information retrieval tasks. Leading us to the main research question of this thesis: **RQ: How can information retrieval benefit from graph databases and graph query languages?**

Three sub-research questions are defined to guide us in answering the main research question:

1. *RQ1: What are the benefits of using relational databases for information retrieval?*

2. *RQ2: How can we extend the benefits from using relational databases for information retrieval to using graph databases?*

3. *RQ3: Can we show practically an example where information retrieval research benefits from graph data?*

1.2 Thesis Contributions and Structure

- Chapter 2 will describe all necessary background information to understand the other chapters.

- Chapter 3 will present ..., we will discuss content based on the following published works: [Kamphuis et al., 2020, Kamphuis and de Vries, 2019b]

- Chapter 4 will present ..., we will discuss content which has been previously described in the following published works: [Kamphuis and de Vries, 2019a, 2021]

- Chapter 5 will present ..., we will discuss content which has been previously described in the following published work: [Kamphuis et al., 2022]

- Chapter 6 will serve as a conclusion and tries to summarize the content discussed in the book. Here we will reflect on the research question, and discuss what future research is needed.

1.3 Publications

- Kamphuis and de Vries [2019b]
- Kamphuis and de Vries [2019a]
- Kamphuis et al. [2019b]
- Lin et al. [2020a]
- Kamphuis et al. [2020]
- Boers et al. [2020]
- Schoegje et al. [2020]
- Kamphuis [2020]
- Kamphuis and de Vries [2021]
- Kamphuis et al. [2022]

Chapter 2

Related Work

2.1 Information Retrieval

Everything that is needed to process a query like, “Can machines think?”, is subject to research by the field of information retrieval.

2.1.1 Inverted Indexes

2.1.2 Ranking methods

Boolean Retrieval

Vector Space Models

Probabilistic ranking Models

Language Models

Learning to Rank

Vector Space Models revisited

2.1.3 Similarity Search

2.2 Relational Databases

Relational databases are usually used to store structure data.

2.3 Graphs

Instead of using columnar data, it might be more attractive to model your data using graphs.

2.4 Reproducible Science

Chapter 3

IR using Relational Databases

> "Is this new question a worthy one to investigate?" This latter question we investigate without further ado, thereby cutting short an infinite regress.
>
> Alan Turing - 1950

Abstract

There have been many attempts to express information retrieval problems using relational databases. In this chapter, one of the latter attempts that revived the idea of expressing bag-of-words ranking functions using SQL will be highlighted. A prototype system that uses these expressions is presented, dubbed OldDog. This system can be used for rapid IR prototyping and is especially helpful in the context of reproducible information retrieval research. Also, when researchers speak of BM25, it is not always clear which variant they mean since many tweaks to Robertson et al.'s original formulation have been proposed. Does this ambiguity "matter"? We attempt to answer this question with a large-scale reproducibility study of BM25, considering eight variants implemented in the OldDog system. Experiments

on three newswire collections show no significant effectiveness differences between them, including Lucene's often maligned approximation of document length.

3.1 Introduction

Where information retrieval researchers commonly use inverted indexes as data structures, there is also a rich history of researchers using relational databases to represent the data in information retrieval systems. Different approaches in the literature present varying successes. Given this context, we arrive at the first research question:

RQ1: What are the benefits of using relational databases for information retrieval?

In order to answer this question, first, we will look at the history of using database systems for IR. Then, one of the latter attempts of using a relational database for information retrieval will be highlighted. Using this work a prototype system is built, dubbed "OldDog". This system will be used in a reproduction experiment, which compares several variants of BM25 which each other. This reproduction study does not only confirm previous findings found in the literature, it also verifies that relational databases systems are suited for running IR experiments.

3.2 Related work

3.2.1 Boolean retrieval

Perhaps the earliest work on using relational databases for information retrieval is the work by Schek and Pistor [1982]. In their work, the authors recognize that the relational data model is widely accepted as an interface to query structured data. However, in cases of unstructured data, like text, it is inconvenient to use. They proposed an extension for the relational model by allowing Non First Normal Form (NF^2) relations. This extension allows for text queries to be more easily expressed. However, the systems that can be built in this language are boolean retrieval systems. Which at the time worked well, but scoring was not a feature implemented. Similarly, Macleod [1991] compared the relational model with the inverted index model. Macleod showed how queries of the IBM STAIRS system could be expressed using the relational model. These were, however, still boolean queries, so scoring using uncertainty was not considered.

3.2.2 Probabilistic Relational Algebra

Fuhr [1996] recognized that where databases contain formatted data, IR systems deal with unformatted data, which requires uncertain inference. They propose to express this uncertainty using a probabilistic relational algebra [Fuhr and Rölleke, 1997] (PRA). PRA can be considered an extension of standard relational algebra. The basic idea behind PRA is that tuples are assigned weights; the weight represents the probability that the tuple belongs to the relation. These probabilities give two advantages. Uncertain information can be expressed, and tuples representing answers to queries can be ordered by the weights representing the uncertainty. The most certain tuples are ranked at the top. Although these extensions give advantages over boolean retrieval, how to assign these probabilities to, for example, a document-term pair remains a question.

3.2.3 IR on top of a database cluster

Grabs et al. [2004] propose PowerDB-IR, developed to run IR applications on a scalable infrastructure. It should also be able to update the data quickly while retrieving up-to-date results. Grabs et al. achieve this by assigning every document to a category, e.g., sports or news. In their experiments, they use 16 artificial categories. A dedicated node is created for every category, containing tables containing documents, inverted lists, and statistics tables. The system supports both single-category and multi-category searches. For a single query search, the following ranking score value is calculated:

$$\mathrm{RSV}(d,q) = \sum_{t \in q} \mathit{tf}(t,d) \cdot \mathit{idf}(t)^2 \cdot \mathit{tf}(t,q) \tag{3.1}$$

Here $\mathit{tf}(t,d)$ is the term frequency of term t in document d, $\mathit{idf}(t)$ is the inverted document frequency of term t (which is squared in this formula), and $tf(t, q)$ is the term frequency of term t in the query text. Calculating this is straightforward: all statistics necessary are stored on a node. However, when one wants to search on multiple (or all) categories. The cost of this approach is high, but this work may have proposed the first real IR in SQL approach.

3.2.4 Integrating DB + IR

Chaudhuri et al. [2005] also identify the need for systems that integrate both database and IR functionalities. In their view, database systems need to be more flexible for scoring and ranking, while IR systems can not handle structured data and metadata properly. Chaudhuri et al. put together a list of seven requirements that a DB + IR system should be able to support, of which they identify the following three requirements as the most important:

1. *Flexible scoring and ranking.* It should be possible to customize the ranking function for different applications; a news search system probably needs different ranking functions and settings than a web search system.

2. *Optimizability.* Following standard database approaches, queries in a DB+IR system should have a query optimizer that considers the workload and the data characteristics. For example, when only one relevant result is sufficient, the system should be able to abort when a relevant document is found.

3. *Metadata and ontologies.* Other than metadata that describes data sources, other metadata that is used for understanding information demands might be needed. This metadata could be, for example, an ontology or a lexicon used for more effective ranking strategies.[1]

To build a system that can support these requirements, the authors identify four alternatives for designing a DB+IR system:

1. *On-top-of-SQL.* The IR functionalities are built on top of a SQL engine. The disadvantage of this approach is that it is challenging to customize efficient access for both IR and DB functionalities.

[1]Latent representations generated by large language models are a great example of this kind of metadata.

2. *Middleware.* In this approach, a SQL engine and an IR engine are run simultaneously. The two disadvantages of using this approach are that the API needs to talk to two systems, which can have very different design philosophies, and the data needs to be shared between systems, incurring a large overhead and making it harder to combine both functionalities.

3. *IR-via-ADT's.* The third approach is building an IR system using abstract data types. The authors argue that this approach makes the system more customizable than the previous approaches. However, the authors also note that optimization in the case of UDFs is complicated. Also, when programmers need to work with such a system, it has the full complexity of SQL plus the complexity of working with ADTs, making them efficient.

4. *RISC.* The final approach is what the authors prefer; IR functionalities build on top of a relational *storage* engine, as described in an earlier work by them [Chaudhuri and Weikum, 2000]. The DB+IR systems should then be built on top of this engine.

Although the approaches described in this work are interesting, they do not provide prototypes to compare them. (The goal of this paper was to present a theoretical framework for tackling this problem.)

3.2.5 Handwritten plans and Array Databases

Héman et al. [2006] participated in the TREC TeraByte track using the relational engine MonetDB/X100 [Boncz et al., 2005]. They were able to express ranking functions efficiently and effectively in this system. In their submission, they used BM25 as a scoring function. In order to reduce the amount of computing necessary for every document-term pair, the BM25 score was precalculated. The disadvantage of this approach was that the query plans were not generated from SQL but were handwritten. Having to handwrite queries makes this system more challenging to use for IR researchers. Also, because all BM25 scores were precalculated (albeit with some compression), more storage was needed than when only the term frequencies were saved.

The same research group [Cornacchia et al., 2008] also ran experiments on the TREC TeraByte track using the array database SRAM

(Sparse Relational Array Mapping). SRAM automatically translate BM25 queries to run them on a relational engine (particularly X100). However, SRAM is quite an exotic query language, only used by some researchers.

3.2.6 Retrieval models only using SQL

In more recent work, Mühleisen et al. [2014] showed that the commonly used BM25 ranking function could also be easily expressed using SQL. This is done similarly to Grabs et al. [2004]. In this work, the MonetDB [Boncz, 2002] and Vectorwise [Zukowski et al., 2012] systems were used, making it much faster. Mühleisen et al. specifically focused on the retrieval efficiency of several systems. They argue that instead of using a custom build IR system using an inverted index, researchers could store their data representations in a column-oriented relational database and formulate the ranking functions using SQL. They show that their implementation of BM25 in SQL is on par in efficiency and effectiveness compared to systems that use an inverted index.[2]

There was an interesting observation in the paper to highlight: All the systems evaluated in this paper implement BM25. However, there was a substantial difference between the effectiveness scores produced by these systems, as shown in table 3.1. The only two systems that achieved the same effectiveness score were the two database systems (MonetDB and Vectorwise). However, these two systems were developed by the same research group.

These results were surprising as the authors took specific care to keep document preprocessing identical for all systems. However, the observed difference in `MAP` of 3% absolute was the largest deviation in the score reported.

3.2.7 Reproducibility

Not only did we observe the differences in effectiveness scores for BM25 in the paper by Mühleisen et al. [2014]. In the SIGIR 2015 Workshop on Reproducibility, Inexplicability, and Generalizability of Results (RIGOR) [Arguello et al., 2016] and the Open-Source IR Replicability

[2]In particular the Vectorwise system.

Table 3.1: Results presented by Mühleisen et al. [2014]. `MAP` and `P@5` on the ClueWeb12 collection are reported for five different systems that run BM25. The table shows that only the two database systems (MonetDB and Vectorwise) achieve the same effectiveness score.

System	`MAP`	`P@5`
Indri [Strohman et al., 2005]	0.246	0.304
MonetDB [Boncz, 2002]	0.225	0.276
Vectorwise [Zukowski et al., 2012]	0.225	0.276
Lucene [Apache Software Foundation]	0.216	0.265
Terrier [Ounis et al., 2005]	0.215	0.272

Table 3.2: Results from the RIGOR workshop [Arguello et al., 2016]. `MAP@1000` on the .GOV2 collection is reported for four different systems that run BM25. The table shows that all four implementations report a different effectiveness score.

System	`MAP@1000`
ATIRE [Trotman et al., 2012]	0.290
Lucene [Apache Software Foundation]	0.303
MG4J [Boldi and Vigna, 2005]	0.299
Terrier [Ounis et al., 2005]	0.270

Challenge (OSIRRC) workshop [Clancy et al., 2019b] similar results are observed. See table 3.2 and table 3.3 respectively.

It is unclear why the results between these systems differ this much; many explanations are possible. Examples include; different preprocessing,[3] different hyperparameter settings, other functions for inverse document frequency (*idf*), or erroneous implementation of the ranking function. Using, for example, non-optimized hyperparameter settings can lead to considerable gaps in differences between effectiveness scores. Yang et al. [2019] showed that in many cases, new ranking methods had been proposed that compared the results of a newly proposed method to a non-fine-tuned version of BM25, making the results look

[3] But not in Mühleisen et al., as they ensured preprocessing was the same for all systems.

Table 3.3: Results from the OSIRRC workshop [Clancy et al., 2019b]. `AP`, `P@30`, and `NDCG@20` on the robust04 collection are reported for seven different systems that run BM25. As shown in the table, all implementations report (again) a different effectiveness score.

System	`AP`	`P@30`	`NDCG@20`
Anserini [Clancy et al., 2019a]	0.253	0.310	0.424
ATIRE [Trotman et al., 2012]	0.218	0.320	0.421
ielab [Scells and Zuccon, 2019]	0.183	0.261	0.348
Indri [Hauff, 2019]	0.239	0.300	0.404
OldDog [Kamphuis and de Vries, 2019b]	0.243	0.299	0.400
Pisa Mallia et al. [2019]	0.253	0.312	0.422
Terrier [Câmara and Macdonald, 2019]	0.236	0.298	0.405

better than they are. The choices for hyperparameters are often left out of papers, while BM25 is the baseline compared against. As BM25 is often used as a baseline, it is important to understand why these difference exists, and how they arise.

3.3 Prototype OldDog

As shown in table 3.3, one of the submissions to the workshop was by us [Kamphuis and de Vries, 2019b]. This prototype is a software project to replicate and extend the database approach to information retrieval presented in Mühleisen et al. [2014]. As the prototype was based on their work, we dub it *OldDog*. OldDog uses column store database MonetDB [Boncz, 2002] for query processing. Mühleisen et al. produced the database tables to represent "postings" using a custom program running on Hadoop. Instead, we relied on the Anserini toolsuite Yang et al. [2017] to create a Lucene index. From this index, we extracted the data necessary to fill the tables. Anserini takes care of standard document preprocessing. The ranking function implemented in OldDog was BM25 as proposed by Robertson et al. [1994].

3.3.1 Docker

For the submission to the OSIRRC workshop [Clancy et al., 2019b], we created a docker image of OldDog.[4] Mühleisen et al. implemented a conjunctive variant of BM25 (all query terms have to be present in a document in order for a document to be considered relevant). When creating the submission for the workshop, we noticed that the effectiveness scores were substantially lower than other submissions. When removing the conjunctive constraint, the effectiveness results increased. So our prototype supports both conjunctive and disjunctive versions of BM25. Our entry in table 3.3 presents the effectiveness scores of the disjunctive variant. The retrieval effectiveness degraded more than we expected a priori, given the results in previous work. The number of relevant documents per topic for this collection was likely relatively low.

3.3.2 Ease-of-Use

Having implemented BM25 in a database system enabled us to carry out some experiments quite easily that are more mundane when using an inverted index. As all document frequencies are stored in one table, filtering out the terms with a large document frequency is easy. In only two lines of SQL, we updated the table removing the terms with large document frequency, as shown in figure 3.1. This approach could be an automatic way to remove stopwords from a collection. This filter was too strict to improve retrieval effectiveness but can easily be fine-tuned.

3.4 Variants of BM25

Having "OldDog" set up, we can quickly run retrieval experiments. As mentioned in the previous section, it is still unclear why the differences between the submissions were this big. Also, many different variants of BM25 that claim to be more effective have been proposed in the literature. A study by Trotman et al. [2014] compared several variants and found that improvements presented in the literature do not add

[4] https://hub.docker.com/r/osirrc2019/olddog

```sql
ALTER TABLE dict
RENAME TO odict;
CREATE table dict
AS SELECT * FROM odict WHERE df <= (
        SELECT 0.1 * COUNT(*) FROM docs
);
```

Figure 3.1: This code updates the `docs` table such that all terms with a document frequency greater than a tenth of the collection size are removed.

up. As we now have a system in which the BM25 formula is written directly in SQL, we can easily swap this version of BM25 with its proposed improvements. By using OldDog, we can ensure the data representation is the same when we compare these variants; the results will only reflect what the effects are applying a different variant of BM25. This way, we can easily confirm the findings of Trotman et al..

Robertson et al. [1994]

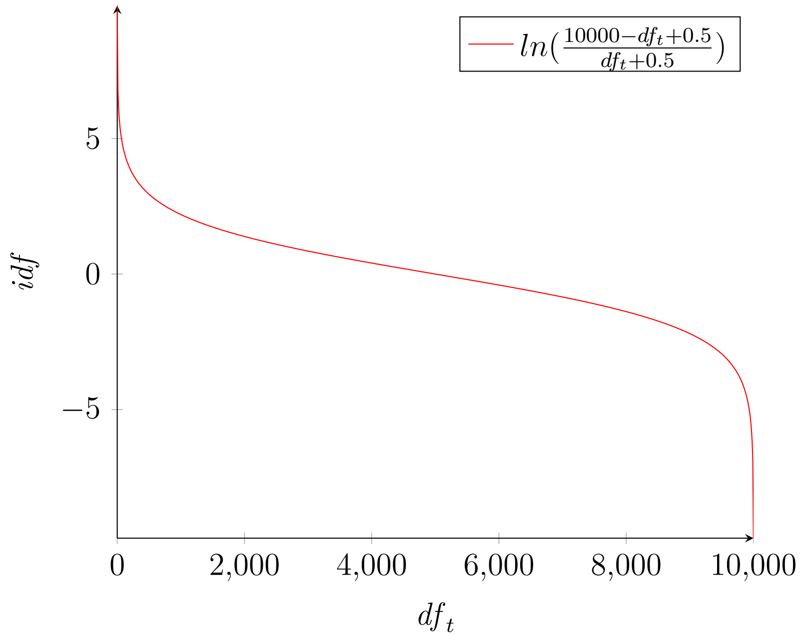
The original formulation of BM25 is constructed from two parts. The first part is derived from the binary independence relevance model [Robertson and Zaragoza, 2009], which results in an approximation of the classical inverse document frequency (*idf*) for a query term t:

$$w_i^{\mathrm{IDF}} = \log\left(\frac{N - df_t + 0.5}{df_t + 0.5}\right) \tag{3.2}$$

where N is the collection size, and df_t are the number of documents in the collection that contain query term t.

There is, however, a negative consequence of using this formula for weighing term importance. Let us say there is a collection with $10,000$ documents; then, it is possible to plot the *idf* for each term as shown in figure 3.2. The figure shows that the *idf* score becomes negative when $df_t > \frac{N}{2}$. This happens for terms that appear in more than half of all documents, e.g.: “the” or “a”. Many systems do not consider these terms when searching by keeping a list of common words that can

Figure 3.2: Inverse document frequency as used by Robertson et al. [1994]



be ignored (stop words). However, when these words are considered, a negative *idf* would decrease the relevance scores of documents with the query term in the document—some variations of BM25 deal with this, which are discussed in the following sections.

The second part of BM25 can be considered as a term frequency weighting *tf*. These two parts are multiplied to get something like the traditional term frequency-inverse document frequency weighting $tf \times w_i^{\mathrm{IDF}}$. However, the *tf* in BM25 is extended: every additional term occurrence does not increase the ranking score value as much as the previous one. For example, a term being present twice in a document versus once provides more information than a term being present ten times versus nine. For this, the following convenient formula as a replacement for *tf* was chosen:

$$\frac{tf}{k + tf} \text{ where } k > 0 \tag{3.3}$$

This approach ensures that the term frequency does not increase linearly. In the final formulation of BM25, k is written as k_1. This is because earlier versions of this ranking formula also had a k_2 and k_3 parameter.

Then lastly, a second component is added that can correct for documents longer than others. It is, however, not clear how one should deal with documents being longer than others; the author of a document can be verbose, in which case additional term occurrences do not provide any more information. On the other hand, a lengthy document can be because more relevant information is provided, and the document is more relevant than its shorter counterpart. For these reasons, the following soft-length normalization is introduced:

$$(1-b) + b \times \left(\frac{L_d}{L_{avg}}\right) \text{ with } 0 <= b <= 1 \tag{3.4}$$

When setting $b = 1$, full-length normalization is used, while if $b = 0$, none is used. Combining these parts, including the correction for term frequency and the length normalization, we get BM25 as initially proposed by Robertson et al. [1994]:

$$\sum_{t \in q} \log\left(\frac{N - df_t + 0.5}{df_t + 0.5}\right) \cdot \frac{tf_{td}}{k_1 \cdot \left(1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} \tag{3.5}$$

Lucene (default)

The variant implemented in Lucene (as of version 8) introduces two main differences. First, since the *idf* component of Robertson et al. [1994] is negative when:

$$df_t > \frac{N}{2} \tag{3.6}$$

To avoid negative values in all possible cases, Lucene adds a constant one before calculating the *log* value. Second, the document length used in the scoring function is compressed (in a lossy manner) to a one-byte value, denoted $L_{d\text{lossy}}$. With only 256 distinct document lengths, Lucene can pre-compute the value of

$$k_1 \cdot \left(1 - b + b \cdot \left(\frac{L_{d\text{lossy}}}{L_{avg}}\right)\right) \tag{3.7}$$

for each possible length, resulting in fewer computations at query time. Then equation (3.8) describes BM25 as implemented in Lucene:

$$\sum_{t \in q} \log\left(1 + \frac{N - df_t + 0.5}{df_t + 0.5}\right) \cdot \frac{tf_{td}}{k_1 \cdot \left(1 - b + b \cdot \left(\frac{L_{\mathrm{dlossy}}}{L_{avg}}\right)\right) + tf_{td}} \tag{3.8}$$

Lucene (accurate)

Equation (3.9) represents our attempt to measure the impact of Lucene's lossy document length encoding. We implemented a variant that uses exact document lengths but is otherwise identical to the Lucene default.

$$\sum_{t \in q} \log\left(1 + \frac{N - df_t + 0.5}{df_t + 0.5}\right) \cdot \frac{tf_{td}}{k_1 \cdot \left(1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} \tag{3.9}$$

ATIRE [Trotman et al., 2012]

Equation (3.10) shows BM25 as implemented by ATIRE; it implements the *idf* component of BM25 as $\log(N/df_t)$, which also avoids negative values. The TF component is multiplied by $k_1 + 1$ to make it look more like the classic RSJ weight [Robertson and Spärck Jones, 1976]; this does not affect the resulting ranked list, as all scores are scaled linearly with this factor.

$$\sum_{t \in q} \log\left(\frac{N}{df_t}\right) \cdot \frac{(k_1 + 1) \cdot tf_{td}}{k_1 \cdot \left(1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} \tag{3.10}$$

BM25L [Lv and Zhai, 2011c]

BM25L builds on the observation that BM25 penalizes longer documents too much compared to shorter ones. The *idf* component differs to avoid negative values. The TF component is reformulated as follows:

$$\frac{(k_1 + 1) \cdot c_{td}}{k_1 + c_{td}} \tag{3.11}$$

with

$$c_{td} = \frac{tf_{td}}{1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)} \tag{3.12}$$

The c_{td} component is further modified by adding a constant δ, boosting the score for longer documents. The authors report using $\delta = 0.5$ for the highest effectiveness. Equation (3.13) presents the final formulation of BM25L:

$$\sum_{t \in q} \log\left(\frac{N+1}{df_t + 0.5}\right) \cdot \frac{(k_1 + 1) \cdot (c_{td} + \delta)}{k_1 + (c_{td} + \delta)} \tag{3.13}$$

BM25+ [Lv and Zhai, 2011a]

BM25+, as shown in equation (3.14), encodes a general approach for dealing with the issue that ranking functions unfairly prefer shorter documents over longer ones. Lv and Zhai propose adding a lower-bound bonus when a term appears at least once in a document. The difference with BM25L is a constant δ to the TF component. The *idf* component is again changed to a variant that disallows negative values.

$$\sum_{t \in q} \log\left(\frac{N+1}{df_t}\right) \cdot \left(\frac{(k_1 + 1) \cdot tf_{td}}{k_1 \cdot \left((1-b) + b \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} + \delta\right) \tag{3.14}$$

BM25-adpt [Lv and Zhai, 2011b]

BM25-adpt is an approach that varies k_1 per term (i.e., uses term specific k_1 values). In the original formulation of BM25, k_1 can be considered a hyperparameter that regulates the increase of score for additional occurrences of a term; k_1 ensures that every additional occurrence gets discounted as it provides less information than its previous. However, Lv and Zhai argued that this does not necessarily have to be the case. If there are fewer documents with $t+1$ occurrences versus t, it should provide more information compared to when the number of documents is almost the same. In order to find the optimal term-specific k_1 value, the authors want to maximize the information gain for that particular query term. First, they identify the probability of selecting a document randomly from the collection that contains the term q at least once in a document as:

$$p(1|0, q) = \frac{df_t + 0.5}{N + 1} \tag{3.15}$$

The probability of a term occurring one more time is defined as:

$$p(t + 1|t, q) = \frac{df_{t+1} + 0.5}{df_t + 1} \tag{3.16}$$

In both these formulas, 1 and 0.5 are added for smoothing to avoid zero probabilities. Then the information gain from t to $t+1$ occurrences is computed as, subtracting the initial probability:

$$G_q^t = \log_2\left(\frac{df_{t+1} + 0.5}{df_t + 1}\right) - \log_2\left(\frac{df_t + 0.5}{N + 1}\right) \tag{3.17}$$

Here df_t is not defined as a standard document frequency but based on the length normalized term frequency:

$$df_t = \begin{cases} |D_{t|c_{td} \geq t-0.5}| & t > 1 \\ df(q) & t = 1 \\ N & t = 0 \end{cases} \tag{3.18}$$

In this case $df(q)$ is the "normal" document frequency, and c_{td} is the same as in BM25L (pivoted method for length normalization Singhal et al. [1996]):

$$c_{td} = \frac{tf_{td}}{1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)} \tag{3.19}$$

This means the following: df_t is equal to the number of documents in the collection when $t = 0$, and it is equal to the "normal" document frequency when $t = 1$. Otherwise, it will be the number of documents with at least t occurrences of the term (rounded up) using the pivoted method c_{td}.

Then, the information gain is calculated for $t \in \{0, \cdots, T\}$,until $G_q^t > G_q^{t+1}$. This threshold is chosen as a heuristic: When t becomes large, the estimated information gain can be very noisy. So T is chosen as the smallest value that breaks the worst burstiness rule [Church and Gale, 1995] (the information gain starts decreasing). The optimal value for k_1 is then determined by finding the value for k_1 that minimizes the following equation:

$$k_1' = \arg\min_{k_1} \sum_{t=0}^{T} \left(\frac{G_q^t}{G_q^1} - \frac{(k_1 + 1) \cdot t}{k_1 + t} \right)^2 \tag{3.20}$$

Essentially, this gives a value for k_1 that maximizes information gain for that specific term; k_1 and G_q^1 are then plugged into the BM25-adpt formula:

$$\sum_{t \in q} G_q^1 \cdot \frac{(k_1' + 1) \cdot tf_{td}}{k_1' \cdot \left((1 - b) + \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} \tag{3.21}$$

We found that the optimal value of k_1 is not defined for about 90% of the terms. A unique optimal value for k_1 only exists when $t > 1$ while calculating G_q^t. For many terms, especially those with a low df, $G_q^t > G_q^{t+1}$ occurs before $t > 1$. In these cases, picking different values for k_1 has virtually no effect on retrieval effectiveness. For undefined values, we set k_1 to 0.001, the same as Trotman et al. [2014].

TF $l \circ \delta \circ p \times$IDF [Rousseau and Vazirgiannis, 2013]

TF$l \circ \delta \circ p \times$IDF, as shown in equation 3.24, models the non-linear gain of a term occurring multiple times in a document as:

$$1 + \log\left(1 + \log\left(tf_{td}\right)\right) \tag{3.22}$$

To ensure terms occurring at least once in a document get boosted, the approach adds a fixed component δ, following BM25+. These parts are combined into the TF component using the pivoted method for length normalization [Singhal et al., 1996]:

$$c_{td} = \frac{tf_{td}}{1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)} \tag{3.23}$$

The same IDF component as in BM25+ is used, which gives us TF$l \circ \delta \circ p \times$IDF:

$$\sum_{t \in q} \log\left(\frac{N + 1}{df_t}\right) \cdot \left(1 + \log\left(1 + \log\left(c_{td} + \delta\right)\right)\right) \tag{3.24}$$

3.5 Experiments

Our experiments were conducted using Anserini (v0.6.0) on Java 11 to create an initial index, and subsequently using relational databases for rapid prototyping, using “OldDog” [Kamphuis and de Vries, 2019b] after Mühleisen et al. [2014]; following that work use MonetDB as well. Evaluations with Lucene (default) and Lucene (accurate) were performed directly in Anserini; the latter was based on previously-released code that we updated and incorporated into Anserini.[5] The inverted index was exported from Lucene to OldDog, ensuring that all experiments share the same document processing pipeline (e.g., tokenization, stemming, stopword removal). While exporting the inverted index, we precalculate all k_1 values for BM25-adpt as suggested by Lv and Zhai [2011b]. As an additional verification step, we implemented both Lucene (default) and Lucene (accurate) in OldDog and compared the results to the output from Anserini. We can confirm that the results are the same, setting aside unavoidable differences related to floating point precision. All BM25 variants are then implemented in OldDog as minor variations upon the original SQL query provided in Mühleisen et al.. The term-specific parameter optimization for the *adpt* variant was already calculated during the index extraction stage, allowing us to upload the optimal (t, k) pairs and directly use the term-specific k values in the SQL query. The advantage of our experimental methodology is that we did not need to implement a single new ranking function from scratch.

The experiments use three TREC newswire test collections: TREC Disks 4 and 5, excluding Congressional Record, with topics and relevance judgments from the TREC 2004 Robust Track (Robust04); the New York Times Annotated Corpus, with topics and relevance judgments from the TREC 2017 Common Core Track (Core17); the TREC Washington Post Corpus, with topics and relevance judgments from the TREC 2018 Common Core Track (Core18). Following standard experimental practice, we assess ranked list output in terms of average precision (`AP`) and precision at rank 30 (`P@30`). The parameters shared by all models are set to $k_1 = 0.9$ and $b = 0.4$, Anserini's defaults. The parameter δ is set to the value reported as best in the corresponding source publication.

[5] http://searchivarius.org/blog/accurate-bm25-similarity-lucene

Table 3.4: Effectiveness scores different BM25 variants. All were implemented as SQL queries, so the underlying data representations are the same.

	Robust04		Core17		Core18	
	`AP`	`P@30`	`AP`	`P@30`	`AP`	`P@30`
Robertson et al.	.2526	.3086	.2094	.4327	.2465	**.3647**
Lucene (default)	.2531	.3102	.2087	.4293	.2495	.3567
Lucene (accurate)	.2533	.3104	.2094	.4327	.2495	.3593
ATIRE	.2533	.3104	.2094	.4327	.2495	.3593
BM25L	.2542	.3092	.1975	.4253	**.2501**	.3607
BM25+	.2526	.3071	.1931	.4260	.2447	.3513
BM25-adpt	**.2571**	**.3135**	**.2112**	.4133	.2480	.3533
$\text{TF}_{l\circ\delta\circ p} \times \text{IDF}$	.2516	.3084	.1932	**.4340**	.2465	.3647

All experiments were run on a Linux desktop (Fedora 30, Kernel 5.2.18, SELinux enabled) with four cores (Intel Xeon CPU E3-1226 v3 @ 3.30 GHz) and 16 GB of main memory; the MonetDB 11.33.11 server was compiled from source using the `--enable-optimize` flag.

3.6 Results

Table 3.4 shows the effectiveness scores of the different BM25 variants. The observed differences in effectiveness are small and can be fully attributed to variations in the scoring function; our methodology fixes all other parts of the indexing pipeline (e.g., tag cleanup, tokenization, and stopwords). Both an ANOVA and Tukey's HSD show no significant differences between any variant on all test collections. These results confirm the findings of Trotman et al. [2014]: effectiveness differences are unlikely an effect of the choice of the BM25 variant. Across the IR literature, we find that differences due to more mundane settings (such as the choice of stopwords) are often larger than the differences we observe here. Although we find no significant improvements over the original [Robertson et al., 1994] formulation, it might still be worthwhile to use a variant of BM25 that avoids negative ranking scores.

Table 3.5: Average retrieval time per query in ms: Anserini (top) and OldDog (bottom)

	Robust04	Core17	Core18
Lucene (default)	52	111	120
Lucene (accurate)	55	115	123
Robertson et al.	158 ± 25	703 ± 162	331 ± 96
Lucene (default)	157 ± 24	699 ± 154	326 ± 90
Lucene (accurate)	157 ± 24	701 ± 156	324 ± 88
ATIRE	157 ± 24	698 ± 159	331 ± 94
BM25L	158 ± 25	697 ± 160	333 ± 96
BM25+	158 ± 25	700 ± 160	334 ± 96
BM25-adpt	158 ± 24	700 ± 157	330 ± 92
$\text{TF}_{l \circ \delta \circ p} \times \text{IDF}$	158 ± 24	698 ± 158	331 ± 96

You might have caught that the effectiveness scores of ATIRE and Lucene (accurate) are the same. This is not a mistake. As explained, the $k_1 + 1$ in ATIRE scales the scores linearly and does not affect the ranking. So the only difference that can change the effectiveness scores is the different *idf* functions. However, these are practically the same, especially when a collection has a large number of documents (N):

$$\log\left(\frac{N}{df_t}\right) = \log\left(\frac{N - df_t + df_t}{df_t}\right) \tag{3.25}$$

$$= \log\left(\frac{N - df_t}{df_t} + \frac{df_t}{df_t}\right) \tag{3.26}$$

$$= \log\left(\frac{N - df_t}{df_t} + 1\right) \tag{3.27}$$

$$\approx \log\left(\frac{N - df_t + 0.5}{df_t + 0.5} + 1\right) \tag{3.28}$$

Table 3.5 presents the average retrieval time per query in milliseconds (without standard deviation for Anserini, which does not report time per query). MonetDB uses all cores for inter- and intra-query parallelism, while Anserini is single-threaded.

Comparing Lucene (default) and Lucene (accurate), we find negligible differences in effectiveness. However, the differences in retrieval time are also negligible, which calls into question the motivation behind the original length approximation. Currently, the similarity function and, thus, the document length encoding are defined at index time. Storing exact document lengths would allow for different ranking functions to be swapped at query time more effortlessly, as no information would be discarded at index time. Accurate document lengths might additionally benefit downstream modules that depend on Lucene. We suggest that Lucene might benefit from storing exact document lengths.

3.7 Conclusion

In summary, the previous sections describe a double reproducibility study. The study methodologically validated the usefulness of databases for IR prototyping and performed a large-scale study of BM25 to confirm the findings of Trotman et al. [2014]. Returning to our original motivating question regarding the multitude of BM25 variants: "Does it matter?", we can conclude that the answer appears to be "no, it does not". Furthermore, to return to our research question, we can conclude that using relational databases for information retrieval is beneficial. Because data processing and storage are separated in relational databases, comparing different ranking functions in a relational system is much easier compared to a system that uses an inverted index. The work by Mühleisen et al. [2014] also confirmed that relational databases could be as efficient as inverted indexes in retrieval tasks. In short, databases have use cases in which they are easier to work with, while it is possible to have efficient systems.

Chapter 4

From Tables to Graphs

Abstract

This chapter introduces GeeseDB. GeeseDB is a Python toolkit for solving information retrieval research problems that leverage graphs as data structures. It aims to simplify information retrieval research by allowing researchers to formulate graph queries through a graph query language quickly. GeeseDB is built on top of DuckDB, an embedded column-store relational database for analytical workloads. GeeseDB is available as an easy-to-install Python package. In only a few lines of code, users can create a first-stage retrieval ranking using BM25. Queries read and write Numpy arrays and Pandas dataframes at zero or negligible data transformation cost (dependent on a base datatype). Therefore, the results of a first-stage ranker expressed in GeeseDB can be used in various stages in the ranking process, enabling all the power of Python machine learning libraries with minimal overhead. Also, because data representation and processing are strictly separated, GeeseDB forms an ideal basis for reproducible IR research.

4.1 Introduction

In recent years there has been a lot of exciting new information retrieval research that uses non-text data to improve the effectiveness of search systems. All these research directions have improved search systems' effectiveness by using more diverse data. Although search systems consider more diverse data sources, the usage of this data is often implemented through a coupled architecture. In particular, first-stage retrieval is often carried out with different software compared to later retrieval stages, where these novel reranking techniques tend to be used. In our view, researchers could benefit from a system where retrieval stages are more tightly coupled, which facilitates the exploration of how to use non-content data for ranking, and serves the data in a format suitable for reranking with, e.g., transformers or tree-based methods.

4.2 Related work

Consider for example dense representations for retrieval Gao et al. [2021], Luan et al. [2021], Lin et al. [2020b], knowledge graphs to leverage entity information Hasibi et al. [2016b], Balog [2018], Dalton et al. [2014], and non-textual learning-to-rank features Deveaud et al. [2014], Macdonald et al. [2012].

4.2.1 Current solutions

PyTerrier was developed by Macdonald et al. Macdonald et al. [2021], which is a Python extension of Terrier Ounis et al. [2005]. Pyserini was developed by Lin et al. Lin et al. [2021], which is a Python extension of Anserini Yang et al. [2017].

4.3 GeeseDB

In order to fulfill these needs we propose GeeseDB,[1] a prototype Python toolkit for information retrieval that leverages graphs as data structures, allowing metadata and graph oriented data to be easily included in the ranking pipeline. The toolkit is designed to quickly set up first stage retrieval, and make it easy for researchers to explore new ranking models quickly.

In short, GeeseDB aims to provide the following functionalities:

[1] `https://github.com/informagi/geesedb`

- GeeseDB is an easy to install, self-contained Python package available through `pip install` with as few as possible dependencies. It contains topics and relevance judgements for several standard IR collections out-of-the-box, allowing researchers to quickly start developing new ranking models.

- First stage (sparse) retrieval is directly supported. In only a few lines of code it is possible to load documents and create BM25 rankings.

- Data is served in a usable format for later retrieval stages. GeeseDB allows to directly run queries on Pandas data frames for efficient data transfer to sequential reranking algorithms.

- Easy data exploration is supported through querying data with SQL, but more interestingly, also using a graph query language (based on the Cypher query language), making the exploration of new research avenues easier. This prototype supports a subset of the graph query language Cypher, similar to the property graph database model query language as described by Angles Angles [2018].

4.4 Design

At the core of GeeseDB lies the full text search design presented by Mühleisen et al. Mühleisen et al. [2014]. In this work, a column-store database for IR prototyping is proposed, which uses the database schema described in Figure 4.1, consisting of three database tables. (One for all term information, one for all document information, and one that contains the information on how terms relate to documents; the information that is found in a posting list of an inverted index). Using these three tables they show that BM25 can be easily expressed as a SQL query, with latencies that are on par with custom-build IR engines. In GeeseDB we use the exact same relational schema for full text search. Instead of seeing the document data and term data as tables that relate to each other through a many-to-many join table, it is also possible to consider this schema as a bipartite graph. In this graph both documents and terms are considered as nodes, connected to each other through edges. Basically, if a term

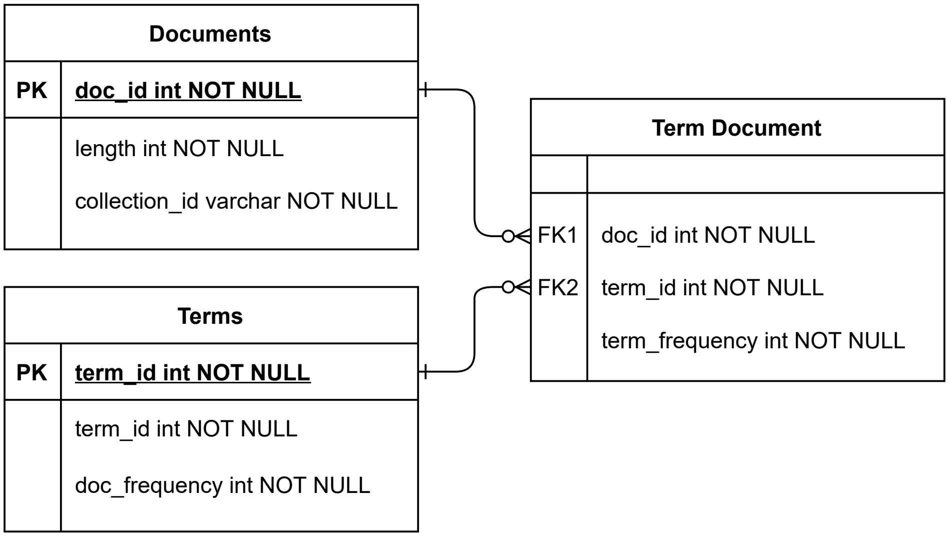


Figure 4.1: Database schema by Mühleisen et al. for full text search in relational databases

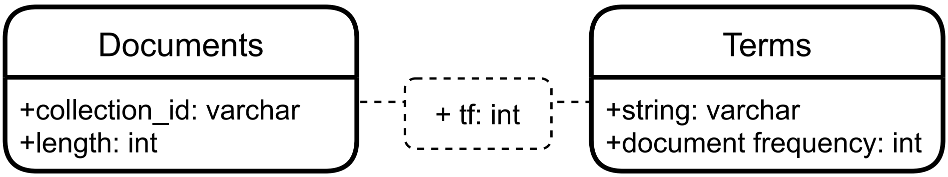


Figure 4.2: Graph schema representing bipartite document-term graph

occurs in a document there exists an edge between that term and document. GeeseDB uses the data model of property graphs; labeled multigraphs where both edges and nodes can have property-value pairs. The database schema as described in Figure 4.1 would then translate to the property graph schema shown in Figure 4.2. A small example of a graph represented by this schema is shown in Figure 4.3, document nodes contain document specific information (i.e. document length and the collection identifier), term nodes contain information relevant to the term (i.e. the term string and the term's document frequency), and the the edges between document and terms nodes contain term frequency information (i.e. how often is the term mentioned in the document represented the respective nodes it connects). If one wants to for example also store position data, this graph can easily be changed

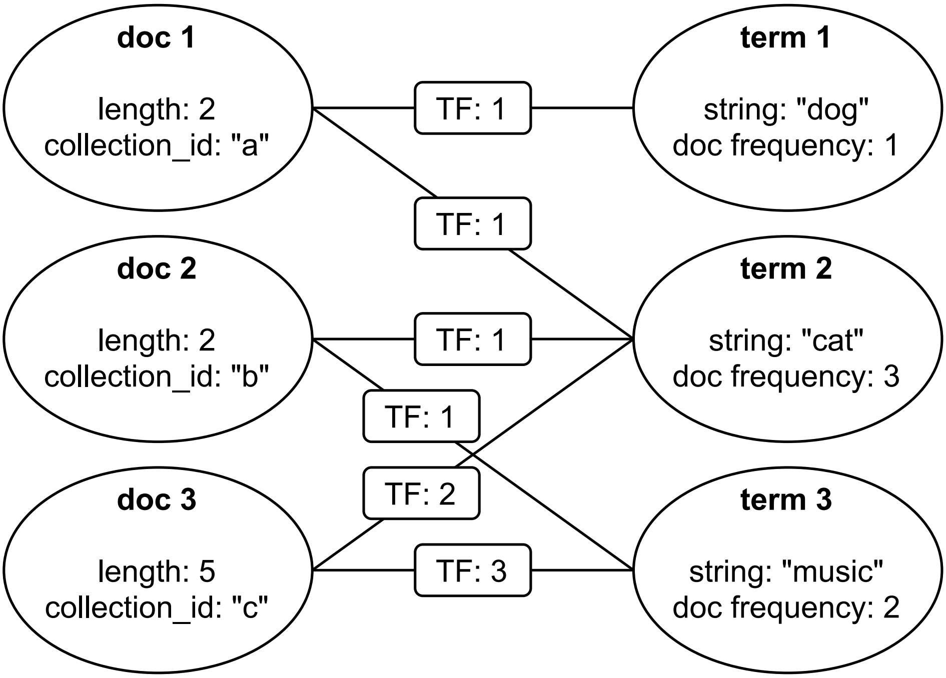


Figure 4.3: Example term-document graph that maps to relational database schema

to a graph where the edges store the position of a term. If a term would appear multiple times in a document, the property graph model would allow for multiple edges to exist between two nodes. The graph schema that we described by Figure 4.2 maps one-to-one to the relational database schema described by Figure 4.1, so nodes are represented by normal relational tables that represent specific data units (terms, documents), while edges are represented by many-to-many join tables. So even though we think of the data as graphs, in the backend they are represented as relational tables. When using GeeseDB for search we at least expect the document-term graph to be present, of course new node types can be introduced in order to explore new search strategies.

4.4.1 Backend

GeeseDB is built on top of DuckDB Raasveldt and Mühleisen [2019], an in-process SQL OLAP (analytics optimized) database management system. DuckDB is designed to support analytical query workloads, meaning that it specifically aims to process complex long-running queries where a significant portion of the data is accessed, conditions matching the case of IR research. DuckDB has a client Python API which can be installed using `pip`, afterwards it can be used directly. DuckDB has a separated API built around both NumPy and Pandas, providing NumPy/Pandas views over the same underlying data representation, without incurring data transfer (usually referred to as "zero-copy" reading). Pandas DataFrames can be registered as virtual tables, allowing to directly query the data present in Pandas DataFrames. GeeseDB inherits all these functionalities from DuckDB.

As DuckDB is a SQL database management system, we can execute analytical SQL queries on the tables that contain our data, including the BM25 rankings described by Mühleisen et al. Mühleisen et al. [2014]. By default, the BM25 implementation provided with GeeseDB implements the disjunctive variant of BM25, instead of the conjunctive variant they used. Although the conjunctive variant of BM25 can be calculated more quickly, we chose to use the disjunctive variant as it is more commonly used by IR researchers and the differences between effectiveness scores are noticeable on smaller collections. For now we only support the original formulation of BM25 by Robertson et al. Robertson et al. [1994], however support of or adding other versions

```
MATCH (d:docs)-[]-(:authors)-[]-(d2:docs)
WHERE d.collection_id = "96ab542e"
RETURN DISTINCT d2.collection_id
```

Figure 4.4: An example cypher query that finds all documents that were written by the same author that wrote the document with the `collecion_id` “96ab542e”

```
SELECT DISTINCT d2.collection_id
FROM docs AS d2
JOIN doc_author AS da2 ON (d2.collection_id = da2.doc)
JOIN authors AS a2 ON (da2.author = a2.author)
JOIN doc_author AS da3 ON (a2.author = da3.author)
JOIN docs AS d ON (d.collection_id = da3.doc)
WHERE d.collection_id = '96ab542e'
```

Figure 4.5: SQL query that corresponds to the graph query described in Figure 4.4.

of BM25 Kamphuis et al. [2020] is trivial.

4.5 Graph Query Language

What distinguishes GeeseDB from alternatives, database-backed (OldDog) Kamphuis and de Vries [2019b] or native systems (Anserini Yang et al. [2017], Terrier Ounis et al. [2005]) is the graph query language, based on Cypher Francis et al. [2018]. For now, GeeseDB implements Cypher's basic graph pattern matching queries for retrieving data. An example of a graph query supported by GeeseDB is presented in Figure 4.4. This query finds all documents written by the same authors as those who wrote document “96ab542e”. For comparison, Figure 4.5 illustrates the same query represented in SQL; much more complex than the Cypher version, due to the join conditions that have to be made explicit. In order to connect the “docs” table with the “authors” table 2 joins are needed, reconnecting the “docs” table again introduces two more joins.

At the moment of writing, GeeseDB supports the following Cypher

```
MATCH (d:docs {d.collection_id: "96ab542e"})
RETURN d.len
```

Figure 4.6: Graph query where the length of document with `collection_id` is returned.

keywords: `MATCH`, `RETURN`, `WHERE`, `AND`, `DISTINCT`, `ORDER BY`, `SKIP`, and `LIMIT`. Instead of using `WHERE` to filter data, it is also possible to use graph matching, as shown in Figure 4.6; the query returns the length of document "96ab542e". We plan to support the other keywords of Cypher in the future, as well as directed edges. Everything that is not yet directly supported yet by our implementation can of course still be expressed in SQL, which is fully supported.[2] In order to know how to join nodes to each other if no edge information has been provided, GeeseDB stores information on the schema. This way GeeseDB knows how nodes relate to each other through which edges. GeeseDB has a module for updating the graph schema, allowing researchers to easily set up the graph they want represented in the database.

4.6 Usage

GeeseDB comes as an easy-to-install Python package that can be installed using pip, the standard package installer for Python:

```
$ pip install geesedb==0.0.1
```

After installing GeeseDB we can immediately start using it. All examples we show in this paper were run on version v0.0.1 of GeeseDB. However, as GeeseDB is actively being developed we advise readers to use the latest version of GeeseDB, which can be installed when not specifying a package version. It is also possible to install the latest commit by installing the latest version directly from GitHub. As an example, we will show how to use GeeseDB for the background linking task of the TREC News Track Soboroff et al. [2018]. The goal of this task is: *Given a news story, find other news articles that can*

[2] GeeseDB supports the graph queries by translating them to their corresponding SQL queries, both nodes and edges are after all just tables in the backend.

provide important context or background information. These articles can then be recommended to the reader to help them understand the context in which these news articles take place. The collection used for this task is the Washington Post V3 collection[3] released for the 2020 edition of TREC. It contains 671, 945 news articles published by the Washington Post published between 2012 and 2020, and 50 topics with relevance assessments (topics correspond to collection identifiers of documents for which relevant data has to be found). The articles in this collection contain useful metadata; in particular, we will use authorship information. We extracted 25, 703 unique article authors, where it is possible that multiple authors co-wrote a news article. We also annotate documents with entity information which was obtained by using the Radboud Entity Linker van Hulst et al. [2020]. In total 31, 622, 419 references to 541, 729 unique entities were found, the links also contain mention and location information, as well as the `ner_tag` found by the linker's entity recognition module (The `ner_tag` is part of a link, as the entity linker can assign different tags to the same entity).[4] Figure 4.7 illustrates the data schema that we use for the background linking task.

4.6.1 Indexing and Search

In order to start, a database containing at least the document and term information needs to be created. Figure 4.8 shows how the data can be easily loaded using CSV files.

Instead of loading the data from CSV files it is also possible to load the text data directly using the CIFF format for data exchange Lin et al. [2020a]. GeeseDB also has functionalities to create the CSV files used here from the CIFF format. Authorship information and entity links can be loaded similarly. After loading the data we can quickly create a BM25 ranking for ad hoc search in the Washington Post collection as shown in Figure 4.9.

For the background linking task however, we do not have regular topics; we only have the collection identifiers of the documents we need to find relevant background info for. In order to search for relevant background reading, queries that represent our information need to be

[3] `https://trec.nist.gov/data/wapost/`

[4] The annotated data will be made publicly available.

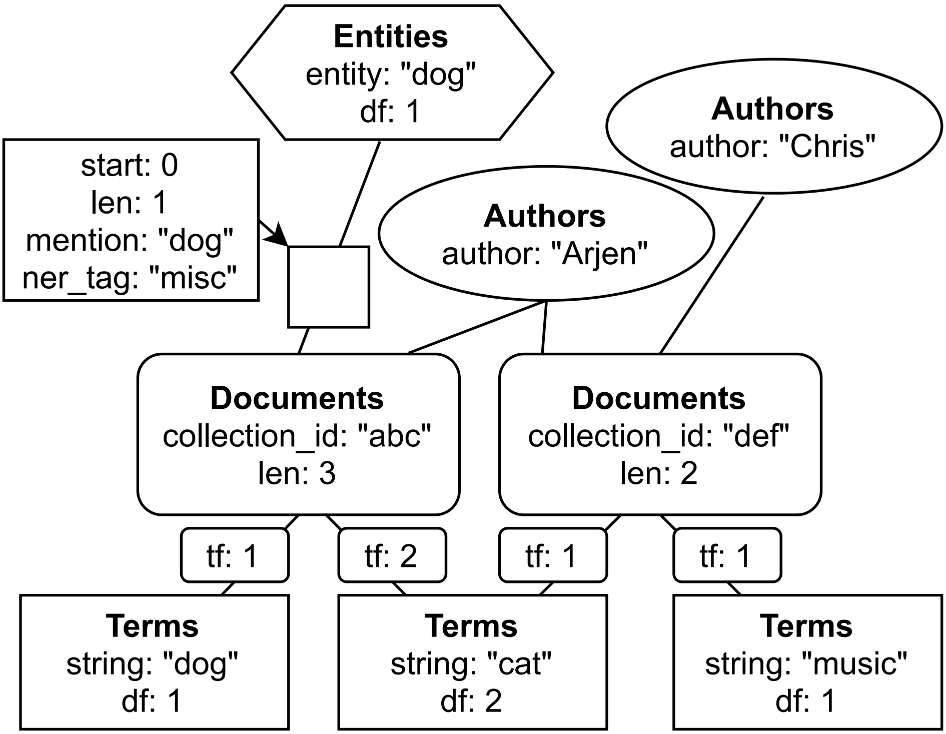


Figure 4.7: Example property graph for the TREC News Track's background linking task. The node types are authors, entities, terms and documents. Edges connect document nodes to other types of nodes. Both edges and nodes can have properties (following the property graph model). Multiple edges may exist between one entity node and one document node, as one entity can be linked multiple times to one document.

```
from geesedb.index import FullTextFromCSV

index = FullTextFromCSV(
        database='/path/to/database',
        docs_file='/path/to/docs.csv',
        term_dict_file='/path/to/term_dict.csv',
        term_doc_file='/path/to/term_doc.csv'
)
index.load_data()
```

Figure 4.8: Load text data from the WashingtonPost collection formatted as csv files in the format as described by Mühleisen et al. Mühleisen et al. [2014]

```
from geesedb.search import Searcher

searcher = Searcher(
        database='/path/to/database',
        n=10
)
hits = searcher.search_topic('obama and trump')
```

Figure 4.9: Example on how to create a BM25 ranking for the query "obama and trump" that returns the top 10 documents.

```
MATCH (d:docs {collection_id: ?})-[]-(t:term_dict)
RETURN string
ORDER BY tf*log(671945/df)
DESC
LIMIT 5
```

Figure 4.10: Prepared Cypher statement that finds the top-5 TF-IDF terms in a document.

```
SELECT term_dict.string
FROM term_dict
JOIN term_doc ON
        (term_dict.term_id = term_doc.term_id)
JOIN docs ON
        (docs.doc_id = term_doc.doc_id)
WHERE docs.collection_id = ?
ORDER BY term_doc.tf * log(671945/term_dict.df
DESC
LIMIT 5;
```

Figure 4.11: Prepared SQL statement that finds the top-5 TF-IDF terms in a document.

constructed. A common approach is to use the top-k TF-IDF terms of the source article. These can easily be found using the Cypher statement shown in Figure 4.10. Instead of using Cypher it would also be possible to use SQL, as shown in Figure 4.11; however this example shows again the Cypher query is more elegant.

Processing Cypher queries depends on the schema information that needs to be loaded as well. We have a supporting class for this, and the schema data used in this paper will be available via GitHub. Using the terms found with Cypher, we can construct queries that we can pass to the searcher, and create a BM25 ranking. The code that generates the rankings for all topics is presented in Figure 4.12. As you can see, with only a limited number of lines of Python code it is quite easy to create rankings. From this point it is quite trivial to write the content of `hits` to a runfile, and evaluate using `trec_eval`.

```
from geesedb.search import Searcher
from geesedb.connection import get_connection
from geesedb.resources import
    ↪ get_topics_backgroundlinking
from geesedb.interpreter import Translator

db_path = '/path/to/database'
        searcher = Searcher(
        database=db_path,
        n=1000
)

translator = Translator(db_path)
c_query = """cypher TFIDF query"""

query = translator.translate(c_query)
cursor = get_connection(db_path).cursor
topics = get_topics_backgroundlinking(
        '/path/to/topics'
)
for topic_no, collection_id in topics:
        cursor.execute(query, [collection_id])
        topic = ' '.join(cursor.fetchall()[0])
        hits = searcher.search_topic(topic)
```

Figure 4.12: Create a BM25 ranking for all background linking topics using the top-5 TFIDF terms. Note that in this case a processed topic file was used that only contains the topic identifier and the topic article id. The topic file in this format is provided on our GitHub.

```
MATCH (d:docs)-[]-(:authors)-[]-(:docs)-[]-(:authors)-[]-(d2:docs {collection_id: ?})
RETURN DISTINCT d.collection_id
```

Figure 4.13: Cypher query to find documents written by co-authors of the authors of the topic article.

Instead of "just" ranking with BM25, using e.g. the metadata in order to adapt the ranking is straightforward. In the case of background linking, it makes sense to consider authorship information when recommending articles that might be suitable as background reading. As journalists are often specialized in certain news topics (e.g. politics, foreign affairs, tech), the stories they write often share context. Also, when journalists collaborate on stories they write together on topics they specialize in as well. As authorship information is available to us, we can decide to use the information whether an article is written by the authors of the topic article, or by someone they have collaborated with in the past. Finding the articles that are written by this group of people can easily be done using a graph query, the query that finds these articles is shown in Figure 4.13.

Depending on the number of documents found by this query, different rescoring strategies can be decided upon. If the set of documents written by the authors or their co-authors is large, perhaps it is possible to only consider these documents, but if the set is small, a score boost might be more appropriate. Figure 4.14 shows an example on how to only consider documents found with the query in Figure 4.13, in this particular case we ensure that at least 2000 documents are found before filtering.

To give another example; the graph query language is also useful when considering entities. When journalists write news articles, the articles relate to events concerning e.g. people, organisations, or countries. In other words, the basis of news articles lay the entities as they are often the subject of news. So, instead of using the most informative terms in a news article, it could be useful to consider the entities identified in the article instead. Important entities tend to be mentioned in the beginning of a news article Kamphuis et al. [2019a];

```
# import and first lines the same as previous example

author_c_query = """cypher authorship query"""
author_query = t.translate(author_c_query)

cursor = get_connection(db_path).cursor
topics = get_topics_backgroundlinking(
        '/path/to/topics'
)
for topic_no, collection_id in topics:
cursor.execute(query, [collection_id])
topic = ' '.join(cursor.fetchall()[0])
hits = searcher.search_topic(topic)

cursor.execute(author_query, [collection_id])
docs_authors = {
        e[0] for e in cursor.fetchall()
}
if len(docs_authors) > 2000:
        hits = hits[hits.collection_id.isin(docs_authors)]
```

Figure 4.14: Find documents written by all authors that collaborated with the authors of the topic article, if there are more than 2000 documents found only consider these documents as background reading candidates.

```
MATCH (d:docs {collection_id: ?})-[]-(e:entities)
RETURN mention
ORDER BY start
LIMIT 5
```

Figure 4.15: Retrieve the first five entities mentioned in the topic article; and return the terms used to mention the entity.

Figure 4.15 shows the Cypher query to retrieve the text mentions of the first five mentioned entities.

Before it is possible to search using the text describing the first five entity mentions, the text needs to be processed. The term data loaded in GeeseDB was already processed, as it was data loaded from CSV files built from a CIFF file created from an Anserini Yang et al. [2017] (Lucene) index. Anserini has an easy to use Python extension, Pyserini Lin et al. [2021], that can be used to tokenize the text in the same way as the documents were tokenized. Figure 4.16 shows the Python code where we extract the mentions, process them such that they become a usable query for GeeseDB, and then BM25 ranking is created with this query.

4.7 Conclusion

```
from geesedb.search import Searcher
from geesedb.connection import get_connection
from geesedb.resources import get_topics_backgroundlinking
from geesedb.interpreter import Translator
from pyserini.analysis import Analyzer, get_lucene_analyzer

db_path = '/path/to/database'
        searcher = Searcher(
        database=db_path,
        n=1000
)

analyzer = Analyzer(get_lucene_analyzer())

translator = Translator(db_path)
c_query = """cypher entity query"""
query = translator.translate(c_query)

cursor = get_connection(db_path).cursor
topics = get_topics_backgroundlinking(
        '/path/to/topics'
)

for topic_no, collection_id in topics:
cursor.execute(query, [collection_id])
topic = ' '.join([e[0] for e in cursor.fetchall()])
topic = ' '.join(analyzer.analyze(topic))
hits = searcher.search_topic(topic)
```

Figure 4.16: Create a BM25 ranking for all background linking topics using the mention text of the first five linked entities in the source article.

Chapter 5

A Graph of Entities

Abstract

This is the abstract

5.1 Introduction

5.2 Entity Linking

Entity linking concerns the task of automatically identifying entity mentions in the text and linking them to the corresponding entities in a knowledge-base (KB). It fulfils a key role in knowledge-grounded understanding of text and has been proven effective for diverse tasks in information retrieval Gerritse et al. [2022, 2020], Xiong et al. [2017], Hasibi et al. [2016a], Balog et al. [2013], Reinanda et al. [2015], Chatterjee and Dietz [2022], natural language processing Lin et al. [2012], Ferrucci [2012], and recommendation Yang et al. [2018]. Utilizing entity annotations in these downstream tasks depends upon the annotation of text corpora with a method for entity linking. Due to the complexity of entity linking systems, this process is often performed by a third-party entity linking toolkit, examples including DBpedia Spotlight Mendes et al. [2011], TAGME Ferragina and Scaiella [2010], Nordlys Hasibi et al. [2017], GENRE De Cao et al. [2021], and REL van Hulst et al. [2020].

A caveat in existing entity linking toolkits is that they have not been designed for batch processing large numbers of documents. Existing entity linking toolkits are primarily optimized to annotate individual documents, one at a time. This severely restricts utilization of state-of-the-art entity linking tools such as REL and GENRE, that employ neural approaches and require GPUs for fast operation. Annotating millions of documents incurs significant computational overhead, to the extent that annotation of a large text corpus becomes practically infeasible using modest computational power resources. Batch entity linking is however necessary to build today's data-hungry machine learning models, considering large text corpora like the new MS MARCO v2 (12M Web documents) Bajaj et al. [2016].

5.3 REL

This paper describes our experience with optimizing the Radboud Entity Linking (REL) toolkit for batch processing large corpora. REL annotates individual documents efficiently, requiring only modest computational resources, while performing competitively when compared to the state-of-the-art methods on effectiveness. It considers entity linking as a modular problem consisting of three stages:

5.3.1 Mention Detection

The goal of this step is to identify all possible text spans in a document that might refer to an entity. If a text span that refers to an entity is not identified properly in this stage, the system will not be able to correctly link the entity in later stages.

5.3.2 Candidate Selection

For every detected mention, REL considers up to $k_1 + k_2 (= 7)$ candidate entities. $k_1 (= 4)$ candidate entities are selected based on their prior occurrence probability $p(e|m)$ (for entity e given mention m). These priors are pre-calculated from Wikipedia hyperlinks and the CrossWiki Spitkovsky and Chang [2012] corpus. The other $k_2 (= 3)$ entities are chosen based on the similarity of their embeddings to the contextual embedding of the mention (considering a context of maximum 200 word tokens).

5.3.3 Entity Disambiguation

The goal of this final step is to map the mention to the correct entity in a knowledge base. The candidate entities for each mention are obtained from the previous stage and REL implements the Ment-norm method proposed by Le and Titov Le and Titov [2018].

This paper explains the challenges of batch processing in REL and presents the approaches we found to overcome these challenges. We show that our updated REL toolkit, REBL, improves REL efficiency 9.5 times, decreasing the processing time per document (excluding mention detection) on a sample of 5000 MS MARCO documents from 1.23 seconds to 0.13 seconds. We demonstrate that REBL enables the annotation of a large corpus like MS MARCO v2, given modest computational resources. We discuss potential improvements that can be made in order to further improve efficiency of batch entity linking. The REBL code and toolkit are available publicly at `https://github.com/informagi/REBL`.

5.4 From REL to REBL

The objective that led to this paper was to link the MS MARCO v2 collection Bajaj et al. [2016]. This collection contains 11,959,635 documents split into 60 compressed files, totaling roughly 33GB in size. Decompressed, these files are in JSON line format (where every line represents a JSON document). Documents have five fields: *url*, *title*, *headings*, *body*, and *docid*. For our experiments we wanted to link the title, headings, and body of the documents. We use the 2019-07 Wikipedia dump to link to, which is one of the two dumps REL was initially developed on. It is, however, straightforward to take another dump of Wikipedia and develop another REL instance.

In order to ease linking this size of data, we separated the GPU heavy mention detection stage from the CPU heavy candidate selection and entity disambiguation stages; the modified code can be found on GitHub.[1] The inputs for mention detection are the compressed MS MARCO v2 document files, and its output consists of the mentions found and their location in the document, in Apache Parquet format.[2]

[1] `https://github.com/informagi/REBL`

[2] `https://github.com/apache/parquet-format`

These files together with the source text are the input for the subsequent phases (candidate selection and entity disambiguation). The final output consists of Parquet files containing spans of text and their linked entities. In the following, we discuss what is changed for mention detection, candidate selection, and entity disambiguation steps to make REL more suited to link the MS MARCO v2 collection.

5.4.1 Mention Detection

REL van Hulst et al. [2020] uses Flair Akbik et al. [2019] for mention detection, a state-of-the-art named entity recognition system. Flair uses the `segtok`[3] package to segment an (Indo-European) document in sentences, internally represented as `Sentence` objects. These sentences are split into words / symbols represented as `Token` objects. When creating these representations however, it is not possible to recreate the source text properly, as Flair removes multiple whitespace characters when occurring after each other. REL corrects for this to preserve the correct span data with regard to its location in the source text, which is an inefficient process. We set out to construct the underlying data structures ourselves for REBL. To do this, we used the `syntok`[4] package, a follow-up version of `segtok`. The author of both packages claims that the `syntok` package segments sentences better than `segtok`.

When constructing the sentences from the token objects, we ran into another issue originated from data handling procedure in Flair: Flair removes various zero width Unicode characters from the source text: zero width space (`U+200B`), zero width non-joiner (`U+200C`), variation selector-16 (`U+FE0F0`), and zero-width no-break space (`U+FEFF`). These characters occur rarely, but in a collection as big and diverse as MS MARCO v2, these characters are found in some documents. When encountering these characters, the token objects were constructed such that the span and offset of the token still referred to that of the source text.

For the case of the zero width space, we updated the `syntok` package; although zero width space is not considered a whitespace character according to the Unicode standard, it should be considered a character that separates two words. For the other Unicode characters

[3] `https://github.com/fnl/segtok`
[4] `https://github.com/fnl/syntok`

removed by Flair, we manually update the span in the `Token` objects created by Flair such that they refer correctly to the positions in the source text. Now, when Flair identifies a series of tokens as a possible mention, we can directly identify the location in the source text from the `Token` objects.

Flair supports named entity recognition in batches; this way multiple batches of text can be sent to the GPU for faster inference time. Because REL had been designed to tag one document at a time, it did not use this functionality. REBL exploits this feature, allowing the user to specify the number of documents to be tagged simultaneously.

5.4.2 Candidate Selection and Entity Disambiguation

REL makes use of a $p(e|m)$ prior, where e is an entity, and m is a mention. These priors are saved in a (SQLite) database, and up to 100 priors per mention are considered. Data conversion between client and the representation stored in the database incurred however a large serialization cost. We updated this to a format that is faster to load, with the additional benefit of a considerably decreased database size.[5] We experimented with data storage in the DuckDB column oriented database as an alternative, but found that SQLite was (still) more efficient as key-value store, at least in DuckDB's current state of development.

We found that the entity disambiguation stage took much longer than reported in the original REL paper. This difference is explained by the length of the documents to be linked. The documents evaluated by Van Hulst et al. van Hulst et al. [2020] were on average 323 tokens long with an average of 42 mentions to consider. The number of tokens in an MS MARCO v2 document is on average 1800, with 84 possible mentions per document.[6] Per mention, 100 tokens to the left, and 100 tokens to the right are considered as the context for the disambiguation model. The larger documents result in a larger memory consumption per context and per document, with higher processing costs as a consequence.

[5]The table that represents the priors shrank from 9.6GB to 2.2GB.

[6]These figures are calculated over the body field; we also tagged the shorter title and headers fields.

We improve the efficiency of the entity disambiguation step such that it could be run in a manageable time. REL recreates database cursors for every transaction. We rewrote the REL database code such that one database cursor is created for the entity disambiguation module. Within a document, the same queries were issued to the database multiple times. This happens for example when a mention occurs multiple times within a document. By caching the output of these queries, we were able to significantly lower the number of database calls needed. We cached all database calls per every segment in the collection, as we ran the process for every segment separately.

The default setting of REL is to keep embeddings on the GPU after they are loaded. This, however, slowed down disambiguation when many documents are being processed consecutively, because operations like normalization were carried out over all embeddings on the GPU. By clearing these embeddings as soon as a document is processed, a significant speed up has been achieved.

Finally, after retrieving the embeddings from the database, REL puts them in a python list. We rewrote the REL code such that the binary data is directly loaded from NumPy, a data format that Pytorch operates on.

5.5 Effects on Execution

In the mention detection stage, we improved tokenization and applied batching. In the MS MARCO v2 collection, $411,906$ documents have tokens that were automatically removed by Flair, which are 3.4% of all documents in the collection. The MS MARCO v1 collection does not have documents that contained these characters; the documents in that version of the collection are (probably) sanitized before publishing. Batching documents in the mention detection stage decreased the average time for finding all named entities. We used batches of size 10, as the documents are relatively large. The optimal batch size will depend on the available GPU memory.

A few documents in the MS MARCO v2 collection could not be linked. This happened only in extraordinary cases, where linking with entities did not make sense in the first place; an example being a document consisting of numbers only.[7] Here, the `syntok` package created one long `Sentence` object from this file that could not fit in GPU memory.

Table 5.1 shows the improvements we made to the candidate selection and entity disambiguation step, and describes how much time is saved in REBL. The code improvements to create the database cursor only once and to load the data directly from NumPy had no noticeable effect on the overall run time of entity disambiguation and are not reported in this table. Note that the large standard deviations are primarily due to the differences in processing costs between long and short documents.

5.6 MMEAD

5.7 Results

5.8 Conclusion

We introduced REBL, an extension for the Radboud Entity Linker. We utilize REL's modular design to separate the GPU heavy mention detection stage from the CPU heavy candidate selection and entity disambiguation stages, as many researchers have dedicated GPU and CPU machines. The mention detection module has been made more robust and reliable, using a better segmenter and preserving location metadata correctly. The candidate selection step and entity disambiguation step were updated to improve their runtime, especially for longer documents.

Although it is now possible to run REL van Hulst et al. [2020] on MS MARCO v2 Bajaj et al. [2016] in a (for us) somewhat reasonable time, we identified further improvements to implement, that we work on actively.

Found mentions are compared to all other mentions during the candidate selection step, the complexity of this step is $O(n^2)$, with n being the number of mentions found in a document, which is especially problematic for longer documents. As we are only interested

[7]The source document was a price list in PDF format.

Table 5.1: Efficiency improvements for Candidate Selection and Entity Disambiguation. Improvements are calculated over a sample of 5000 documents using a machine with an Intel Xeon Silver 4214 CPU @ 2.20GHz using 2 cores, that has 187GB RAM memory, and a GeForce RTX 2080 Ti (11GB) GPU. Improvements are cumulative; the times shown include the previous improvement as well.

Improvement	Seconds	Explanation
Old Candidate Selection + Entity Disambiguation	1.23 ± 2.09	Average time it takes to select candidates and disambiguate per document
No embedding reset	0.26 ± 1.60	The default setting of REL was to keep embeddings in GPU memory after they were loaded, by clearing them from GPU memory after every document a speed up was achieved.
Cache database calls	0.15 ± 1.31	When an entity occurs within a document, there is a high probability of it occurring multiple times. By caching the calls, we increase the memory usage but are able to lower the time needed for candidate selection + entity disambiguation.
Representation change candidates	0.13 ± 1.19	By representing the candidates better in the database, we were able to save on conversion time lowering the time needed for candidate selection.

in mentions that are similar, we expect that it might be worthwhile to implement a locality sensitive hashing algorithm to decrease the number of comparisons needed in this stage. However, we would need to run additional experiments to ensure the effectiveness of the model does not suffer.

REBL now implements a two step approach that writes intermediate results to the file system in Parquet format. A streaming variant would be preferable. We have also kept SQLite as database backend, but will consider specialized key-value stores to speed up candidate selection and entity disambiguation. We will revisit DuckDB upon progress in the implementation of zero-cost positional joins.

The candidate selection stage considers the context of a mention. This context has to be constructed from the source document. As a result, we load the source data a second time during candidate selection. Alternatively, we may output mention context in the mention detection stage, which could then speed up the remaining. However, this would significantly increase the size of the mention detection output. More experiments are needed to strike the right balance here.

Overall, it has become clear that a data processing oriented perspective on entity linking is necessary for efficient solutions. Having made explicit quite a few implicit design choices, re-evaluating these might lead to more effective entity linking as well.

Chapter 6

Conclusion

> We can only see a short distance ahead, but we can see plenty there that needs to be done.
>
> — Alan Turing - 1950

Bibliography

A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter, and R. Vollgraf. FLAIR: An Easy-to-Use Framework for State-of-the-Art NLP. In *Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pages 54–59, 2019.

R. Angles. The Property Graph Database Model. In *Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management*, AMW '18, Aachen, 2018. CEUR-WS.org.

Apache Software Foundation. Lucene. URL `https://lucene.apache.org/core/4_3_0/`.

J. Arguello, M. Crane, F. Diaz, J. Lin, and A. Trotman. Report on the SIGIR 2015 Workshop on Reproducibility, Inexplicability, and Generalizability of Results (RIGOR). *SIGIR Forum*, 49(2):107–116, jan 2016. ISSN 0163-5840. doi: 10.1145/2888422.2888439.

P. Bajaj, D. Campos, N. Craswell, L. Deng, J. Gao, X. Liu, R. Majumder, B. M. Andrew McNamara, T. Nguyen, M. Rosenberg, X. Song, A. Stoica, S. Tiwary, and T. Wang. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. In *InCoCo@NIPS*, 2016.

K. Balog. *Entity-oriented search.* Springer Nature, Gewerbestrasse 11, 6330 Cham, Switzerland, 2018.

K. Balog, H. Ramampiaro, N. Takhirov, and K. Nørvåg. Multi-Step Classification Approaches to Cumulative Citation Recommendation. In *Proceedings of the 10th Conference on Open Research Areas in Information Retrieval*, OAIR '13, page 121–128, Paris, FRA, 2013. LE CENTRE DE HAUTES ETUDES INTERNATIONALES D'INFORMATIQUE DOCUMENTAIRE. ISBN 9782905450098.

P. Boers, C. Kamphuis, and A. P. de Vries. Radboud University at TREC 2020. In *NIST Special Publication 1266: The Twenty-Ninth Text REtrieval Conference Proceedings (TREC 2020)*, TREC'20, Gaithersburg, Maryland, 2020. [Sl]: NIST. URL `https://trec.nist.gov/pubs/trec29/papers/RUIR.N.pdf`.

P. Boldi and S. Vigna. MG4J at TREC 2005. In *The Fourteenth Text REtrieval Conference (TREC 2005) Proceedings*, number SP 500-266 in Special Papers. NIST, 2005. URL `http://mg4j.di.unimi.it/`.

P. A. Boncz. *A Next-Generation DBMS Kernel For Query-Intensive Applications*. PhD thesis, University of Amsterdam, May 2002.

P. A. Boncz, M. Zukowski, and N. Nes. MonetDB/X100: Hyper-Pipelining Query Execution. In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research*, CIDR'05, pages 225–237. www.cidrdb.org, 2005.

A. Câmara and C. Macdonald. Dockerising Terrier for The Open-Source IR Replicability Challenge (OSIRRC 2019). 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, July 25, 2019*, pages 26–30, Aachen, 2019. CEUR-WS.org. URL `https://ceur-ws.org/Vol-2409/docker02.pdf`.

S. Chatterjee and L. Dietz. BERT-ER: Query-Specific BERT Entity Representations for Entity Ranking. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 1466–1477, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450387323. doi: 10.1145/3477495.3531944.

S. Chaudhuri and G. Weikum. Rethinking Database System Architecture: Towards a Self-Tuning RISC-Style Database System. In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, page 1–10, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc. ISBN 1558607153.

S. Chaudhuri, R. Ramakrishnan, and G. Weikum. Integrating DB and IR Technologies: What is the Sound of One Hand Clapping? In *Proceedings of the Second Biennial Conference on Innovative Data Systems Research*, CIDR'05, 2005.

K. W. Church and W. A. Gale. Poisson mixtures. *Natural Language Engineering*, 1(2):163–190, 1995. doi: 10.1017/S1351324900000139.

R. Clancy, Z. Akkalyoncu Yilmaz, Z. Z. Wu, and J. Lin. University of Waterloo Docker Images for OSIRRC at SIGIR 2019. 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, July 25, 2019*, page 36, Aachen, 2019a. CEUR-WS.org. URL `https://ceur-ws.org/Vol-2409/docker04.pdf`.

R. Clancy, N. Ferro, C. Hauff, J. Lin, T. Sakai, and Z. Z. Wu. The SIGIR 2019 Open-Source IR Replicability Challenge (OSIRRC 2019). In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 1432–1434, New York, NY, USA, 2019b. Association for Computing Machinery. ISBN 9781450361729. doi: 10.1145/3331184.3331647.

R. Cornacchia, S. Héman, M. Zukowski, A. P. Vries, and P. Boncz. Flexible and Efficient IR Using Array Databases. *The VLDB Journal*, 17(1): 151–168, jan 2008. ISSN 1066-8888. doi: 10.1007/s00778-007-0071-0.

J. Dalton, L. Dietz, and J. Allan. Entity Query Feature Expansion Using Knowledge Base Links. In *Proceedings of the 37th International ACM SIGIR Conference on Research; Development in Information Retrieval*, SIGIR '14, page 365–374, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450322577. doi: 10.1145/2600428.2609628.

N. De Cao, G. Izacard, S. Riedel, and F. Petroni. Autoregressive Entity Retrieval. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=5k8F6UU39V`.

R. Deveaud, M.-D. Albakour, C. Macdonald, and I. Ounis. On the Importance of Venue-Dependent Features for Learning to Rank Contextual Suggestions. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, page 1827–1830, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450325981. doi: 10.1145/2661829.2661956.

P. Ferragina and U. Scaiella. TAGME: On-the-Fly Annotation of Short Text Fragments (by Wikipedia Entities). In *Proceedings of the 19th ACM*

International Conference on Information and Knowledge Management, CIKM '10, page 1625–1628, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450300995. doi: 10.1145/1871437.1871689.

D. Ferrucci. Introduction to "This is Watson". *IBM Journal of Research and Development*, 56:1:1–1:15, 05 2012. doi: 10.1147/JRD.2012.2184356.

N. Francis, A. Green, P. Guagliardo, L. Libkin, T. Lindaaker, V. Marsault, S. Plantikow, M. Rydberg, P. Selmer, and A. Taylor. Cypher: An Evolving Query Language for Property Graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1433–1445, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450347037. doi: 10.1145/3183713.3190657.

N. Fuhr. Models for Integrated Information Retrieval and Database Systems. *IEEE Data Engineering Bulletin*, 19(1):3–13, 1996.

N. Fuhr and T. Rölleke. A Probabilistic Relational Algebra for the Integration of Information Retrieval and Database Systems. *ACM Trans. Inf. Syst.*, 15(1):32–66, jan 1997. ISSN 1046-8188. doi: 10.1145/239041.239045.

L. Gao, Z. Dai, T. Chen, Z. Fan, B. Van Durme, and J. Callan. Complement Lexical Retrieval Model with Semantic Residual Embeddings. In *Advances in Information Retrieval*, ECIR '21, pages 146–160, Cham, 2021. Springer International Publishing. ISBN 978-3-030-72113-8.

E. J. Gerritse, F. Hasibi, and A. P. de Vries. Graph-Embedding Empowered Entity Retrieval. In *Advances in Information Retrieval: 42nd European Conference on IR Research, ECIR 2020, Lisbon, Portugal, April 14–17, 2020, Proceedings, Part I*, page 97–110, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-45438-8. doi: 10.1007/978-3-030-45439-5_7.

E. J. Gerritse, F. Hasibi, and A. P. de Vries. Entity-Aware Transformers for Entity Search. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 1455–1465, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450387323. doi: 10.1145/3477495.3531971.

T. Grabs, K. Böhm, and H.-J. Schek. PowerDB-IR – Scalable Information Retrieval and Storage with a Cluster of Databases. *Knowl. Inf. Syst.*, 6 (4):465–505, jul 2004. ISSN 0219-1377.

F. Hasibi, K. Balog, and S. E. Bratsberg. Exploiting Entity Linking in Queries for Entity Retrieval. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, page 209–218, New York, NY, USA, 2016a. Association for Computing Machinery. ISBN 9781450344975. doi: 10.1145/2970398.2970406.

F. Hasibi, K. Balog, and S. E. Bratsberg. Exploiting Entity Linking in Queries for Entity Retrieval. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ICTIR '16, page 209–218, New York, NY, USA, 2016b. Association for Computing Machinery. ISBN 9781450344975. doi: 10.1145/2970398.2970406.

F. Hasibi, K. Balog, D. Garigliotti, and S. Zhang. Nordlys: A Toolkit for Entity-Oriented and Semantic Search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1289–1292, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350228. doi: 10.1145/3077136.3084149.

C. Hauff. Dockerizing Indri for OSIRRC 2019. 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, July 25, 2019*, pages 44–46, Aachen, 2019. CEUR-WS.org. URL `https://ceur-ws.org/Vol-2409/docker06.pdf`.

S. Héman, M. Zukowski, A. de Vries, and P. Boncz. MonetDB/X100 at the 2006 TREC TeraByte Track. In *NIST Special Publication: SP 500-272. The Fifteenth Text REtrieval Conference (TREC 2006) Proceedings*, TREC'06, Gaithersburg, Maryland, USA, 2006. [Sl]: NIST. URL `https://trec.nist.gov/pubs/trec15/papers/cwi-heman.tera.final.pdf`.

C. Kamphuis. Graph Databases for Information Retrieval. In *Advances in Information Retrieval*, pages 608–612, Cham, 2020. Springer International Publishing. ISBN 978-3-030-45442-5.

C. Kamphuis and A. P. de Vries. 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, July 25, 2019*, pages 17–20, Aachen, 2019a. CEUR-WS.org. URL `http://ceur-ws.org/Vol-2409/position03.pdf`.

C. Kamphuis and A. P. de Vries. The OldDog Docker Image for OSIRRC at SIGIR 2019. 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, July 25, 2019*, pages 47–49, Aachen, 2019b. CEUR-WS.org. URL `http://ceur-ws.org/Vol-2409/docker07.pdf`.

C. Kamphuis and A. P. de Vries. GeeseDB: A Python Graph Engine for Exploration and Search. In *Proceedings of the 2nd International Conference on Design of Experimental Search & Information REtrieval Systems*, DESIRES '21, pages 10–18, Aachen, 2021. CEUR-WS.org. URL `http://ceur-ws.org/Vol-2950/paper-11.pdf`.

C. Kamphuis, F. Hasibi, A. P. de Vries, and T. Crijns. Radboud University at TREC 2019. In *Proceedings of The Twenty-Eight Text REtrieval Conference*, TREC '19, Gaithersburg, Maryland, USA, 2019a. National Institute for Standards and Technology (NIST).

C. Kamphuis, F. Hasibi, A. P. de Vries, and T. Crijns. Radboud University at TREC 2019. In *NIST Special Publication 1250: The Twenty-Eighth Text REtrieval Conference Proceedings (TREC 2019)*, TREC'19, Gaithersburg, Maryland, 2019b. [Sl]: NIST. URL `https://trec.nist.gov/pubs/trec28/papers/RUIR.N.C.pdf`.

C. Kamphuis, A. P. de Vries, L. Boytsov, and J. Lin. Which BM25 Do You Mean? A Large-Scale Reproducibility Study of Scoring Variants. In *Advances in Information Retrieval*, ECIR '20, pages 28–34, Cham, 2020. Springer International Publishing. ISBN 978-3-030-45442-5.

C. Kamphuis, F. Hasibi, J. Lin, and A. P. de Vries. REBL: Entity Linking at Scale. In *Proceedings of the 3rd International Conference on Design of Experimental Search & Information REtrieval Systems*, DESIRES '22, Aachen, 2022. CEUR-WS.org. URL `https://desires.dei.unipd.it/2022/papers/paper-08.pdf`.

P. Le and I. Titov. Improving Entity Linking by Modeling Latent Relations between Mentions. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1595–1604, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1148. URL `https://aclanthology.org/P18-1148`.

J. Lin, J. Mackenzie, C. Kamphuis, C. Macdonald, A. Mallia, M. Siedlaczek, A. Trotman, and A. P. de Vries. Supporting Interoperability Between Open-Source Search Engines with the Common Index File Format. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 2149–2152, New York, NY, USA, 2020a. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401404.

J. Lin, X. Ma, S.-C. Lin, J.-H. Yang, R. Pradeep, and R. Nogueira. Pyserini: A Python Toolkit for Reproducible Information Retrieval Research with Sparse and Dense Representations. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 2356–2362, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380379. doi: 10.1145/3404835.3463238.

S. Lin, J. Yang, and J. Lin. Distilling Dense Representations for Ranking using Tightly-Coupled Teachers. *CoRR*, abs/2010.11386, 2020b. URL `https://arxiv.org/abs/2010.11386`.

T. Lin, Mausam, and O. Etzioni. Entity Linking at Web Scale. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction (AKBC-WEKEX)*, pages 84–88, June 2012.

Y. Luan, J. Eisenstein, K. Toutanova, and M. Collins. Sparse, Dense, and Attentional Representations for Text Retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345, 2021.

Y. Lv and C. Zhai. Lower-Bounding Term Frequency Normalization. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, page 7–16, New York, NY, USA, 2011a. Association for Computing Machinery. ISBN 9781450307178. doi: 10.1145/2063576.2063584.

Y. Lv and C. Zhai. Adaptive Term Frequency Normalization for BM25. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, page 1985–1988, New York, NY, USA, 2011b. Association for Computing Machinery. ISBN 9781450307178. doi: 10.1145/2063576.2063871.

Y. Lv and C. Zhai. When Documents Are Very Long, BM25 Fails! In *Proceedings of the 34th International ACM SIGIR Conference on Research* *and Development in Information Retrieval*, SIGIR '11, page 1103–1104, New York, NY, USA, 2011c. Association for Computing Machinery. ISBN 9781450307574. doi: 10.1145/2009916.2010070.

C. Macdonald, R. L. Santos, and I. Ounis. On the Usefulness of Query Features for Learning to Rank. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, page 2559–2562, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450311564. doi: 10.1145/2396761.2398691.

C. Macdonald, N. Tonellotto, S. MacAvaney, and I. Ounis. PyTerrier: Declarative Experimentation in Python from BM25 to Dense Retrieval. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, page 4526–4533, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384469. doi: 10.1145/3459637.3482013.

I. A. Macleod. Text Retrieval and the Relational Model. *Journal of the American Society for Information Science*, 42(3):155–165, 1991. doi: https://doi.org/10.1002/(SICI)1097-4571(199104)42:3<155::AID-ASI1>3.0.CO;2-H.

A. Mallia, M. Siedlaczek, J. Mackenzie, and T. Suel. PISA: Performant Indexes and Search for Academia. 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, July 25, 2019*, pages 50–56, Aachen, 2019. CEUR-WS.org. URL `https://ceur-ws.org/Vol-2409/docker08.pdf`.

P. N. Mendes, M. Jakob, A. García-Silva, and C. Bizer. DBpedia Spotlight: Shedding Light on the Web of Documents. In *Proceedings of the 7th International Conference on Semantic Systems*, I-Semantics '11, page 1–8, 2011.

H. Mühleisen, T. Samar, J. Lin, and A. de Vries. Old Dogs Are Great at New Tricks: Column Stores for IR Prototyping. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval*, SIGIR '14, page 863–866, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450322577. doi: 10.1145/2600428.2609460.

I. Ounis, G. Amati, V. Plachouras, B. He, C. Macdonald, and D. Johnson. Terrier Information Retrieval Platform. In *Advances in Information Retrieval*, pages 517–519, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. ISBN 978-3-540-31865-1.

M. Raasveldt and H. Mühleisen. DuckDB: An Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 1981–1984, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450356435. doi: 10.1145/3299869.3320212.

R. Reinanda, E. Meij, and M. de Rijke. Mining, Ranking and Recommending Entity Aspects. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, page 263–272, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450336215. doi: 10.1145/2766462.2767724.

S. Robertson and H. Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, apr 2009. ISSN 1554-0669. doi: 10.1561/1500000019.

S. E. Robertson and K. Spärck Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976. doi: https://doi.org/10.1002/asi.4630270302.

S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Overview of the third text Retrieval conference*, TREC-3, Gaithersburg, Maryland, USA, 1994. [Sl]: NIST. URL `https://trec.nist.gov/pubs/trec3/papers/city.ps.gz`.

F. Rousseau and M. Vazirgiannis. Composition of TF Normalizations: New Insights on Scoring Functions for Ad Hoc IR. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '13, page 917–920, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320344. doi: 10.1145/2484028.2484121.

H. Scells and G. Zuccon. ielab at the Open-Source IR Replicability Challenge 2019. 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, July 25, 2019*, pages 57–61, Aachen, 2019. CEUR-WS.org. URL `http://ceur-ws.org/Vol-2409/docker09.pdf`.

H.-J. Schek and P. Pistor. Data Structures for an Integrated Data Base Management and Information Retrieval System. In *Proceedings of the 8th International Conference on Very Large Data Bases*, VLDB '82, page 197–207, San Francisco, CA, USA, 1982. Morgan Kaufmann Publishers Inc. ISBN 0934613141.

T. Schoegje, C. Kamphuis, K. Dercksen, D. Hiemstra, T. Pieters, and A. P. de Vries. Exploring task-based query expansion at the TREC-COVID track. *CoRR*, abs/2010.12674, 2020. URL https://arxiv.org/abs/2010.12674.

A. Singhal, C. Buckley, and M. Mitra. Pivoted document length normalization. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '96, page 21–29, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917928. doi: 10.1145/243199.243206.

I. Soboroff, S. Huang, and D. Harman. TREC 2018 News Track Overview. In *Proceedings of The Twenty-Seventh Text REtrieval Conference*, TREC '18, Gaithersburg, Maryland, USA, 2018. National Institute for Standards and Technology (NIST).

V. I. Spitkovsky and A. X. Chang. A Cross-Lingual Dictionary for English Wikipedia Concepts. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, pages 3168–3175, Istanbul, Turkey, May 2012. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2012/pdf/266_Paper.pdf.

T. Strohman, D. Metzler, H. Turtle, and W. B. Croft. Indri: A language model-based search engine for complex queries. In *Proceedings of the International Conference on Intelligent Analysis*, volume 2 of *ICIA'05*, pages 2–6. Washington, DC., 2005. URL http://ciir.cs.umass.edu/pubfiles/ir-407.pdf.

A. Trotman, X. Jia, and M. Crane. Towards an Efficient and Effective Search Engine. In *Proceedings of the SIGIR 2012 Workshop on Open Source Information Retrieval*, OSIR@ SIGIR'12, pages 40–47, 2012.

A. Trotman, A. Puurula, and B. Burgess. Improvements to BM25 and Language Models Examined. In *Proceedings of the 2014 Australasian Document Computing Symposium*, ADCS '14, page 58–65, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450330008. doi: 10.1145/2682862.2682863.

J. M. van Hulst, F. Hasibi, K. Dercksen, K. Balog, and A. P. de Vries. REL: An Entity Linker Standing on the Shoulders of Giants. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 2197–2200, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401416.

C. Xiong, J. Callan, and T.-Y. Liu. Word-Entity Duet Representations for Document Ranking. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 763–772, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350228. doi: 10.1145/3077136.3080768.

P. Yang, H. Fang, and J. Lin. Anserini: Enabling the Use of Lucene for Information Retrieval Research. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 1253–1256, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350228. doi: 10.1145/3077136.3080721.

W. Yang, K. Lu, P. Yang, and J. Lin. Critically Examining the "Neural Hype": Weak Baselines and the Additivity of Effectiveness Gains from Neural Ranking Models. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 1129–1132, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361729. doi: 10.1145/3331184.3331340.

Y. Yang, O. Irsoy, and K. S. Rahman. Collective Entity Disambiguation with Structured Gradient Tree Boosting. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 777–786, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1071. URL `https://aclanthology.org/N18-1071`.

M. Zukowski, M. van de Wiel, and P. Boncz. Vectorwise: A Vectorized Analytical DBMS. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, page 1349–1350, USA, 2012.

IEEE Computer Society. ISBN 9780769547473. doi: 10.1109/ICDE.2012.148.

Summary

Samenvatting

Acknowlegements

Research Data Management

Curriculum Vitæ