# 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: Can we extend the benefits from using relational databases for information retrieval to using graph databases, while being able to express graph related problems easier?*
3. *RQ3: When does 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 7 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. This chapter will highlight one of the latter attempts that revived the idea of expressing bag-of-words ranking functions using SQL. A prototype system that uses these expressions is presented, dubbed OldDog, after the work by Mühleisen et al.. 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, not even for 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 confirms previous findings found in the literature and verifies that relational database 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, it is inconvenient to use unstructured data like text. They proposed extending 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. At the time, that worked well, but scoring was not a feature implemented. Similarly, Macleod [1991] compared the inverted index approach to using the relational 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 structured/formatted data, IR systems deal with unformatted data requiring 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] have proposed 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 experiment. 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:

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

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, subscores need to be calculated for all relevant nodes before they can be aggregated to a final score. 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 database and IR functionalities. In their view, database systems need to be more flexible for scoring and ranking, while IR systems cannot adequately handle structured data and metadata. 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, 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]Latent representations generated by large language models would have been a great example of this kind of metadata had their paper been written today.

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.

2. *Middleware.* In this approach, SQL and IR engines 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-ADTs.* 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.) Even though it is not the preferred option of Chaudhuri et al., this PhD thesis research focuses on the *On-top-of-SQL* approach.

### 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 participation, they used BM25 as a scoring function. In order to reduce the amount of computation necessary for every document-term pair, the BM25 score was precalculated. The disadvantage of this approach is 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 would be 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 translates BM25 queries to run them on a relational engine (particularly X100). However, SRAM is quite an exotic query language, only used by the researchers themselves, and no current (prototype) system offers support for this approach.

### 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 the runtime much faster than Grabs et al.'s original results. Mühleisen et al. specifically focused on the retrieval efficiency of systems and compared the efficiency of inverted indexes with systems built on top of relational engines. 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). Although the same research group developed these two systems, they were completely separate projects.

These results were surprising as the authors took specific care to keep document preprocessing identical for all systems. However, the

[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 each claim to rank their documents using 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, 2013] | 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, 2013] | 0.303 |
| MG4J [Boldi and Vigna, 2005] | 0.299 |
| Terrier [Ounis et al., 2005] | 0.270 |

observed difference in `MAP` of 3% absolute was the largest deviation in the score reported.

### 3.2.7 Reproducibility

The paper by Mühleisen et al. [2014] is not the only one that reports the differences in effectiveness scores for BM25. In the SIGIR 2015 Workshop on Reproducibility, Inexplicability, and Generalizability of Results (RIGOR) [Arguello et al., 2016] and the Open-Source IR Replicability Challenge (OSIRRC) workshop [Clancy et al., 2019b] similar results are observed. See Tables 3.2 and 3.3, respectively.

It is unclear why the results between these systems differ this much; many explanations are possible. Examples include; differences

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 |

in preprocessing,[3] different hyperparameter settings, differences in the definition of the inverse document frequency (*idf*) used, 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 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 crucial to understand why these differences exist and how they arise.

## 3.3 Prototype OldDog

As shown in Table 3.3, one of the workshop's submissions was the prototype system developed for this PhD thesis research [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]. The prototype was based on their work, so we dubbed it *OldDog*. OldDog uses column store database Mon-

[3] But this cannot explain the differences reported in Mühleisen et al., as they ensured preprocessing was the same for all systems.

Table 3.4: Example of tables representing the data in the OldDog system, the dict tables contains all term specific data, the terms table represents all the postings, and the docs table contains all document specific data.

(a) **dict**

| termid | term | df |
|---|---|---|
| 1 | put | 1 |
| 2 | robe | 1 |
| 3 | wizard | 1 |
| 4 | hat | 1 |

(b) **docs**

| docid | term | df |
|---|---|---|
| 1 | doc1 | 8 |

(c) **terms**

| termid | docid | tf |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1 | 1 |
| 3 | 1 | 1 |
| 4 | 1 | 1 |

etDB [Boncz, 2002] for query processing. Mühleisen et al. produced the database tables to represent the data typically found in an inverted index using a custom program running on Hadoop. Instead, we created a Lucene index using the Anserini tool suite by Yang et al. [2017]. From this index, we extracted the data necessary to fill the tables. Anserini takes care of standard document preprocessing. Three tables are constructed. One table contains the data that represents the documents, another one that represents the terms, and one that contains all data that relate terms to documents. To illustrate, for a document named "doc1" with the text "I put on my robe and wizard hat" is shown in Table 3.4.

Using these tables, we can easily express bag-of-word ranking functions in SQL queries. The default ranking function implemented in our implementation of OldDog is the version of BM25 that had been proposed by Robertson et al. [1994], which will be expanded upon in the next section.

### 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

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

```
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.

creating the submission for the workshop, we noticed that the effectiveness scores were substantially lower than those of other submissions. The retrieval effectiveness degraded more than we expected a priori, given the results in previous work. 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 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 complicated when using an inverted index. Filtering out the terms with a large document frequency is easy, as all document frequencies are stored in one table. We updated the table removing the terms with large document frequency in only two lines of SQL, 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. For example, we could easily join against a table storing a traditional stopword list.

## 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 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 consists of 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 collection documents containing 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 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 – variations of BM25 have been proposed to deal with this anomaly, which is 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

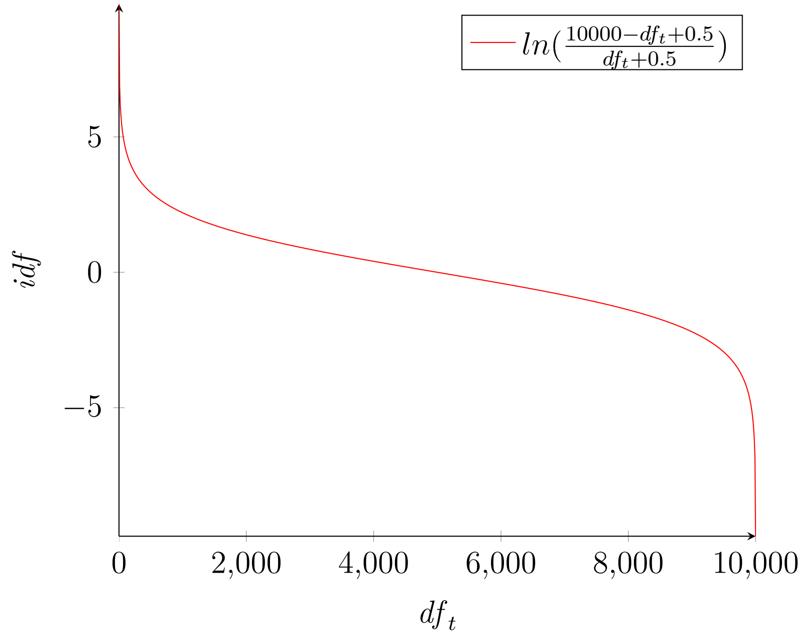


Figure 3.2: Inverse document frequency as used by Robertson et al. [1994]

times versus nine. To achieve this effect of "diminishing return" of additional occurrences, the following convenient formula was chosen to define the *tf* component of the ranking:

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

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, unclear how one should deal with documents being longer than others; the document's author can be verbose, in which case additional term occurrences do not provide extra information. On the other hand, a document can be lengthy 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. As mentioned, the *idf* component of Robertson et al. [1994] is negative when $df_t > \frac{N}{2}$. To avoid negative values in all possible cases, Lucene adds a constant value of 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

$$k_1 \cdot \left(1 - b + b \cdot \left(\frac{L_{d\text{lossy}}}{L_{avg}}\right)\right) \tag{3.6}$$

for each possible length, resulting in fewer computations at query time. Then Equation (3.7) describes BM25 as implemented in Lucene:

$$\sum_{t \in q} \log\left(1 + \frac{N - df_t + 0.5}{df_t + 0.5}\right) \cdot \frac{tf_{td}}{k_1 \cdot \left(1 - b + b \cdot \left(\frac{L_{d\text{lossy}}}{L_{avg}}\right)\right) + tf_{td}} \tag{3.7}$$

### Lucene (accurate)

Equation (3.8) 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.

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

### ATIRE [Trotman et al., 2012]

Equation (3.9) 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.

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

### 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:

$$\frac{(k_1 + 1) \cdot c_{td}}{k_1 + c_{td}} \tag{3.10}$$

with

$$c_{td} = \frac{tf_{td}}{1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)} \tag{3.11}$$

The $c_{td}$ component is further modified by adding a constant $\delta$, boosting the score for longer documents. The authors report using $\delta = 0.5$ for the highest effectiveness. Equation (3.12) presents the final formulation of BM25L:

$$\sum_{t \in q} \log\left(\frac{N+1}{df_t + 0.5}\right) \cdot \frac{(k_1 + 1) \cdot (c_{td} + \delta)}{k_1 + (c_{td} + \delta)} \tag{3.12}$$

### BM25+ [Lv and Zhai, 2011a]

BM25+, as shown in Equation (3.13), 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 $\delta$ to the TF component. The *idf* component is again changed to a variant that disallows negative values.

$$\sum_{t \in q} \log\left(\frac{N+1}{df_t}\right) \cdot \left(\frac{(k_1 + 1) \cdot tf_{td}}{k_1 \cdot \left((1-b) + b \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} + \delta\right) \tag{3.13}$$

### 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 much fewer documents with $t+1$ occurrences versus $t$, it should provide more information than 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:

$$p(1|0,q) = \frac{df_t + 0.5}{N+1} \tag{3.14}$$

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

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

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:

$$G_q^t = \log_2\left(\frac{df_{t+1} + 0.5}{df_t + 1}\right) - \log_2\left(\frac{df_t + 0.5}{N+1}\right) \tag{3.16}$$

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

$$df_t = \begin{cases} |D_{t|c_{td} \geq t-0.5}| & t > 1 \\ df(q) & t = 1 \\ N & t = 0 \end{cases} \tag{3.17}$$

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]):

$$c_{td} = \frac{tf_{td}}{1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)} \tag{3.18}$$

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:

$$k_1' = \arg\min_{k_1} \sum_{t=0}^{T} \left( \frac{G_q^t}{G_q^1} - \frac{(k_1 + 1) \cdot t}{k_1 + t} \right)^2 \tag{3.19}$$

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:

$$\sum_{t \in q} G_q^1 \cdot \frac{(k_1' + 1) \cdot tf_{td}}{k_1' \cdot \left((1 - b) + \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} \tag{3.20}$$

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 3.23, models the non-linear gain of a term occurring multiple times in a document as:

$$1 + \log\left(1 + \log\left(tf_{td}\right)\right) \tag{3.21}$$

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

$$c_{td} = \frac{tf_{td}}{1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)} \tag{3.22}$$

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

$$\sum_{t\in q} \log\left(\frac{N+1}{df_t}\right) \cdot \left(1 + \log\left(1 + \log\left(c_{td} + \delta\right)\right)\right) \tag{3.23}$$

## 3.5 Experiments

This section presents an empirical evaluation of the impact of the different choices of BM25 as described in Section 3.4. 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, we 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

[5] `http://searchivarius.org/blog/accurate-bm25-similarity-lucene` (last accessed – March 3rd, 2023)

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 $\delta$ is set to the value reported as best in the corresponding source publication.

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 3.5 shows the effectiveness scores of the different BM25 variants. The observed differences in effectiveness are small and can be entirely 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 differences due to more mundane settings (such as the choice of stopwords) tend to be larger than the differences we observe here. Although we find no significant improvements over the original [Robertson et al., 1994] formulation, using a variant of BM25 that avoids negative ranking scores might still be worthwhile.

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

Table 3.5: 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 |

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 ($N$):

$$\log\left(\frac{N}{df_t}\right) = \log\left(\frac{N - df_t + df_t}{df_t}\right) \tag{3.24}$$

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

$$= \log\left(\frac{N - df_t}{df_t} + 1\right) \tag{3.26}$$

$$\approx \log\left(\frac{N - df_t + 0.5}{df_t + 0.5} + 1\right) \tag{3.27}$$

Switching our attention from effectiveness to efficiency, Table 3.6 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

Table 3.6: 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. | $158 \pm 25$ | $703 \pm 162$ | $331 \pm 96$ |
| Lucene (default) | $157 \pm 24$ | $699 \pm 154$ | $326 \pm 90$ |
| Lucene (accurate) | $157 \pm 24$ | $701 \pm 156$ | $324 \pm 88$ |
| ATIRE | $157 \pm 24$ | $698 \pm 159$ | $331 \pm 94$ |
| BM25L | $158 \pm 25$ | $697 \pm 160$ | $333 \pm 96$ |
| BM25+ | $158 \pm 25$ | $700 \pm 160$ | $334 \pm 96$ |
| BM25-adpt | $158 \pm 24$ | $700 \pm 157$ | $330 \pm 92$ |
| $\mathrm{TF}_{l\circ\delta\circ p}\times$IDF | $158 \pm 24$ | $698 \pm 158$ | $331 \pm 96$ |

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]. It does not seem to matter which of the multitude of BM25 variants is used. 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

"The reader will have anticipated that I have no very convincing arguments of a positive nature to support my views. If I had I should not have taken such pains to point out the fallacies in contrary views.

Alan Turing - 1950

**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 negligible data transformation cost. 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.

## 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 efficacy of search applications. All these research directions have improved search systems' effectiveness by using more diverse data. Although more diverse data sources are considered, these systems are 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, tree-based methods or graph-based methods.

The previous chapter demonstrates how relational databases can be used for information retrieval problems. Integrating alternative data sources into search systems is easier when using a relational database instead of an inverted index. In the case of information retrieval, graph data can often be used to improve search effectiveness. One of the most famous examples where graphs help information retrieval is the PageRank algorithm [Page et al., 1999]. Although it might be possible to express this data in relational databases, they are not designed with graph structures in mind.

The data management community has shown much interest in graph databases in recent years. Graph databases are different from relational databases in that a relation is not the abstraction for representations, but a graph is. Graphs can be considered a better abstraction for real-world data than relations. Graphs focus much more on concepts (nodes in a graph) and how they relate to each other (edges in a graph), while in a relational database this information is more implicit. Many different graph types exist; one is called the *property graph model*. This type of graph contains nodes and directed edges, which can be labeled; nodes and edges can also have associated key-value pairs. In this chapter, we will work with the property graph model and see how it benefits IR research.

The research goal in this chapter is two-fold; firstly we want to develop a system that can search efficiently and handle graphs. In the previous chapter, we showed that relational databases could search efficiently and help reproducible research. However, these systems do not support graph data structures well. Systems built on inverted indexes use coupled architectures, which can introduce unnecessary overhead.

Secondly, we want to create a system where both first and second-stage retrieval are directly supported. In IR research multi-stage retrieval systems contain components that are not ran in the same ecosystem; introducing overhead in retrieval efficiency. So in this chapter, the prototype system GeeseDB is introduced; it tries to leverage the same techniques for search as relational databases do while also naturally being able to support graphs. This leads to the main research question for this chapter:

**RQ2:** Can we extend the benefits from using relational databases for information retrieval to using graph databases, while being able to express graph-related problems easier?

In order to answer this research question, we will work with the prototype system GeeseDB, but before GeeseDB is discussed, first we will look into work that uses systems that try to combine two systems for two-stage retrieval experiments, and systems built for processing graphs.

## 4.2 Related work

In modern IR it is not uncommon to use coupled architectures. In many situations a ranking model is used that is efficient, but only reasonably effective. BM25 can be considered such a model, it can be calculated cheaply, while more advanced, but less efficient, retrieval models achieve higher effectiveness. In such cases, BM25 can be used to calculate an initial top-$k$ ranking that contains (presumably) all relevant documents. Then, the more expensive model only has to calculate the final ranking scores over the top-$k$ documents.

In this section, first some methods are introduced where such multi-stage approaches are used, then systems that implement these approaches are highlighted. In particular there will be a focus on the coupled-architectures aspect of these systems, and finally some proposed methods will be discussed to deal with cons arising from coupled architectures.

### 4.2.1 Learning To Rank

In learning-to-rank multi-stage retrieval approaches tend to be used, often the models are not efficient enough to compute a relevance score value for all documents in the collection. Deveaud et al. [2014] showed that for the TREC Contextual Suggestion track [Dean-Hall et al., 2014], reranking using user-based features significantly improved retrieval effectiveness over a baseline language modeling approach. These features show that data not present in the document's text can help the retrieval effectiveness of systems; metadata helps to rank.

Macdonald et al. [2012] show examples of many features that are effective for learning to rank, many of these features are non-textual. Some influential features are: click count, click entropy, and the number of displayed results in a session.

Often, in these cases, a decision tree based ensemble like LambdaMART [Burges, 2010] tends to be trained. These models take input that's not typically produced by an inverted index. So, in order to use these models, the output from the inverted index needs to be converted.

### 4.2.2 Dense Retrieval

Where traditionally, search was carried out using inverted indexes, dense retrieval has become more prevalent in recent years. Consider, for example, the work by Gao et al. [2021]; in their work, they proposed `clear`, a model that aims to complement lexical models with a dense retrieval model. In their study, they use BM25 as the lexical model. The BM25 scores are calculated using an inverted index (Anserini [Yang et al., 2017]), while the dense retrieval model is a fine-tuned BERT bi-encoder. This dense model calculates a score using a different maximum inner-product search (MIPS) system (Faiss [Johnson et al., 2019]) to calculate a similarity score for the query document pairs. Then scores are weighted and summed to calculate a new ranking score value.

In a similar study, Luan et al. [2021] compared several methods on the MS Marco document en passage datasets. In their work, one of the two best models on the passage dataset was combining a lexical model and a BERT-based bi-encoder. The scores were combined by retrieving the top-$n$ documents for both the lexical model and the bi-encoder. In order to do this, an inverted index (Anserini) is used to retrieve the documents for the lexical model, while the bi-encoder uses a MIPS system (ScaNN [Guo et al., 2020]).

Then to give a third example, Lin et al. [2020b] do experiments with distilled dense representations. They do an extensive comparison study, and in their work, the best results are reported by either multi-stage or hybrid dense + sparse retrieval methods. The multi-stage retrieval method was BM25 + BERT-large. Although effective, it is a method with a high latency because of the cross-encoder used. A TCT-ColBERT (Tightly-Coupled Teacher - ColBERT) bi-encoder with doc2query-T5 sparse retrieval was the best hybrid method. This sparse retrieval method is a BM25 retrieval method that expands the documents with T5 language model questions [Raffel et al., 2020]. These questions were generated by letting the T5 language model generate questions that the passage might answer. Sparse retrieval is carried out by Anserini, and dense retrieval by Faiss.

### 4.2.3 Knowledge Graphs and Entities

Knowledge graphs have been used a lot as well in IR research. Hasibi et al. [2016b] proposed the Entity Linking incorporated Retrieval (ELR) approach. The idea behind this concept is that when entity annotations are available for queries and documents, they can be incorporated into the ranking model. This study only focuses on entity retrieval, but it generalizes to retrieval. The entities are incorporated by creating a dedicated index for entities on which entity scores can be calculated. These are then merged by the document scores calculated using the regular index. By incorporating entity scores, ELR can increase effectiveness scores for multiple baseline methods.

Dalton et al. [2014] used entity features for query expansion. One of the features was calculated by tagging queries with an entity-linking system. When entities were found, information from the knowledge base linked to them was included in query expansion. Alternative names of the entity were also included in the query expansion. Then this expanded query provided a score for all documents. Another feature they calculate is by ranking the entities in the knowledge base using the original query text. Then the data (e.g., words in the descriptions) included in the knowledge base can be included as terms for query expansion. Multiple other features using entities were also calculated,

Balog [2018] wrote a book on entity-oriented search, giving an excellent outline of methods that use entities for search. I recommend it if you want to read more about entity information for IR.

Basically, in order to include entities, a knowledge graph needs to be queried. This tends to be done by a different systems than the systems used for regular retrieval.

### 4.2.4 Current approaches

Some systems are used a lot for information retrieval research. These systems are typically built as inverted index systems. Recently, retrieval methods using neural networks have become state-of-the-art, mainly using large language models. Some systems will be described, and how they deal with these recent advances will be highlighted.

#### PyTerrier

PyTerrier by Macdonald et al. [2021], is a Python extension of Terrier [Ounis et al., 2005]. Terrier is an open-source search engine system written in Java. It implements state-of-the-art indexing and retrieval techniques on top of an inverted index. It is a system suited for rapidly developing retrieval experiments concerning document collections with many documents. The PyTerrier extension was developed to express complex IR pipelines in Python. Using Python operators directly, it is possible to set up an IR pipeline using PyTerrier in only a few lines of code.

PyTerrier was expanded to support state-of-the-art large language model reranking approaches and dense retrieval methods. As the PyTerrier framework is developed for the Python ecosystem, other (re-)ranking methods that are implemented in Python can readily work with PyTerrier. Because of these features, the PyTerrier system is ideal for modern information retrieval research that often concerns reranking through learned methods, as they are often developed in Python.

Note, however, that PyTerrier uses a coupled architecture, first-stage retrieval is carried out using Terrier in Java (e.g., a bm25 run), and then the resulting top documents are reranked in Python. For this, some overhead is introduced because of data transfer between processes.

#### Pyserini

Pyserini was developed by Lin et al. [2021], which is a Python extension of Anserini [Yang et al., 2017]. Pyserini and Anserini have a lot in common with PyTerrier and Terrier. Just like Terrier, Anserini is developed in Java, but on top of Apache Lucene. Anserini is developed as a system with solid support for reproducible research. Anserini provides reproducible baselines for many retrieval benchmarks that can be run with minimal setup.

Pyserini is developed as a Python wrapper around the Anserini retrieval system. Within the Python ecosystem, it is possible to access the Anserini internals and replicate the same experiments implemented in Anserini. Similarly to PyTerrier, Pyserini also contains features that are not available directly in Anserini; ranking methods that make use of large language models, such as dense retrievers and BERT-based cross encoders. As these methods are generally developed in Python, it is much easier to support them with Pyserini compared to Anserini.

Anserini is included in Pyserini as a JAR file, so when using Pyserini for first-stage retrieval, data needs to be retrieved from JAVA before it can be materialized in Python. The setup is similar to Pyterrier.

### 4.2.5 Proposed approaches

#### Separation of the Logical and Physical Model

As shown in the previous sections, dense and sparse retrieval methods are often implemented using different systems, which introduces the coupled architecture. Lin [2022] recognizes that sparse retrieval models are typically implemented in inverted indexes and dense retrieval methods with approximate nearest neighbor systems. Although these systems are different, the input and the output are the same: the input is a query, and the output is a ranked list of the top-$k$ documents the ranking methods deem the most relevant.

In order to be able to formalize these similarities and distinctions more clearly, Lin proposes a distinction between the logical and physical retrieval models. Logical models consists of encoders that maps documents and queries to a representation, and a comparison function that defines how a ranking score value can be computed from comparing these representations. Physical models define how to create a top-$k$ ranking from a large corpus given a query. This could be applying the logical model to every query-document pair, but through optimizations it might not necessary to compare all documents to the query. For example, dense retrieval methods often make use of HNSW which prunes documents that have a low likelihood of being relevant.

### A Graph Query Language for IR

We [Kamphuis and de Vries, 2019a] proposed to create graph query language for information retrieval problems. In the context of separating logical and physical models, a graph query language can be interpreted as a logical model, while the engine that implements the graph query language can be considered the physical model.

If it is possible to express both first stage and second stage retrieval in such a query language, both stages would be computed by the same physical model, automatically ensuring a coupled architecture is not necessary. Having a graph query language available, ensures that more complicated retrieval strategies can be expressed compared to when a relational query language is used.

So ideally we develop a system with the following two capabilities; the system supports first stage retrieval directly, and graph queries over the output of this first stage retrieval can be processed without a coupled architecture. In order to fulfill these needs, we developed GeeseDB in a follow up study.

## 4.3 GeeseDB

GeeseDB[1] is 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.

In short, GeeseDB aims to provide the following functionalities:

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

- 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 directly running 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, using a graph query language (based on the Cypher query language), making exploring 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 [2018].

## 4.4 Design

At the core of GeeseDB lies the full-text search design presented by Mühleisen et al. [2014]. This work proposes a column-store database for IR prototyping, which uses the database schema described in Figure 4.1, consisting of three database tables. These tables are exactly the same as the tables described in the previous chapter. Using these three tables, the authors show that BM25 can be easily expressed as a SQL

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

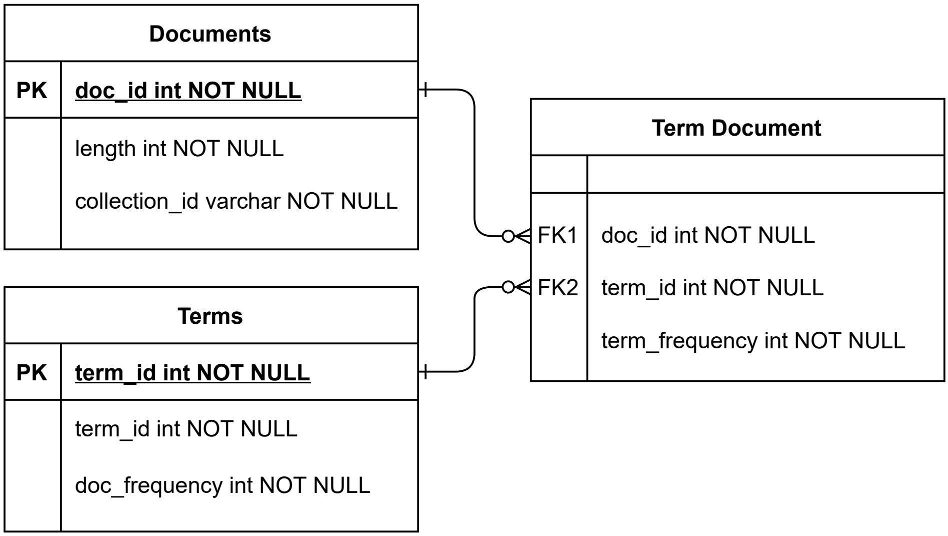


Figure 4.1: Database schema by Mühleisen et al. for full text search in relational databases

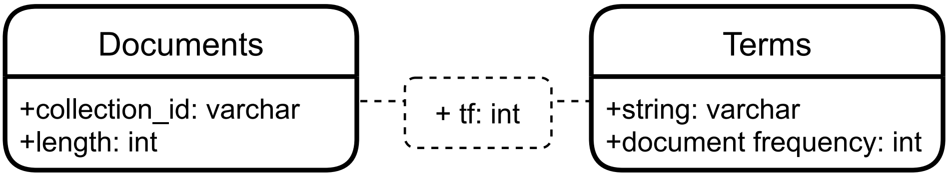


Figure 4.2: Graph schema representing bipartite document-term graph

query with latencies on par with custom-built IR engines. In GeeseDB, we use the 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, documents and terms are considered nodes connected through edges. If a term occurs in a document, an edge exists between that term and the 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

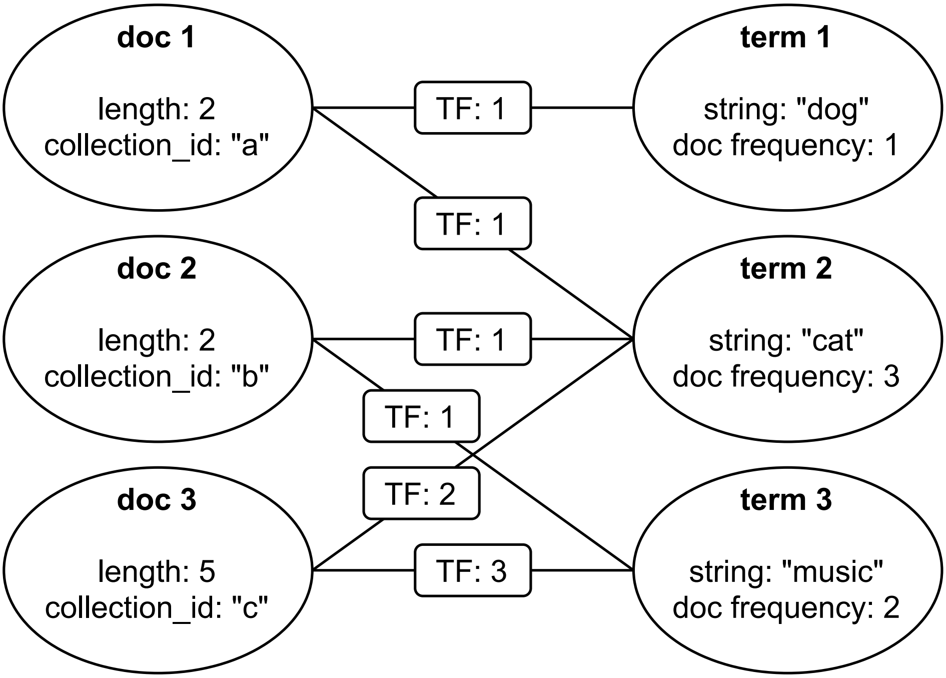


Figure 4.3: Example term-document graph that maps to relational database schema

collection identifier), term nodes contain information relevant to the term (i.e., the term string and the term's document frequency), and 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 to a graph where the edges store the position of a term. If a term appears multiple times in a document, the property graph model will allow multiple edges 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 standard relational tables that represent specific data units (terms, documents), while many-to-many join tables represent edges. So even though we think of the data as graphs, they are represented as relational tables in the backend. When using GeeseDB for search, we at least expect the document-term graph to be present. Of course, new node types can be introduced 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. It aims explicitly 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`. Afterward, it can be used directly. DuckDB has a separate API built around 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 query the data in Pandas DataFrames directly. GeeseDB inherits all these functionalities from DuckDB.

As DuckDB is a database management system, we can execute analytical SQL queries on tables containing our data, including the BM25 rankings described by 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 IR researchers more commonly use it, 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. [1994]. However, supporting other versions of BM25 as described in the previous chapter 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

```
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.

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 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". Everything that is not 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

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

```
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.

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 quickly set up the graph they want to be 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
```

After installing GeeseDB, we can immediately start using it. 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., 2019]. The goal of this task is: *Given a news story, find other news articles that can provide meaningful context or background information.* These articles can then be recommended to the reader to help them understand the context in which these news articles occur. 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 valuable 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.,

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

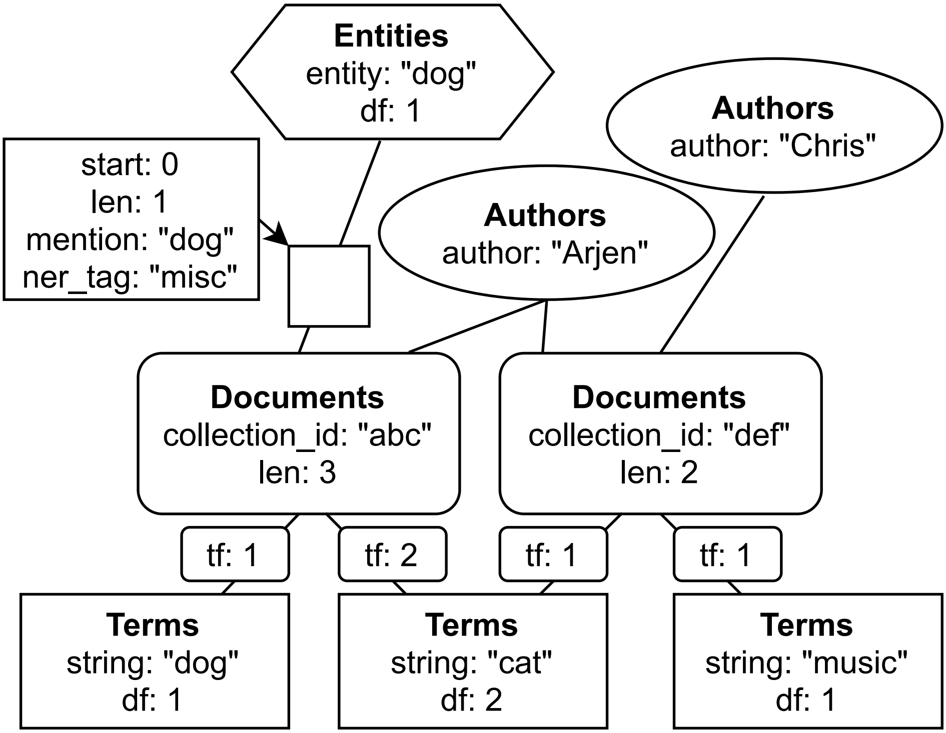


Figure 4.7: Example property graph for the TREC News Track's background linking task. The node types are authors, entities, terms, and documents. Edges connect document nodes to other types of nodes. Both edges and nodes can have properties (following the property graph model). Multiple edges may exist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

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). Figure 4.7 illustrates the data schema that we use for the background linking task using a small example graph.

```python
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. [2014]

```python
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.

### 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 CSV files 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

```
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.

topics; we only have the collection identifiers of the documents for which we need to find relevant background info. In order to search for relevant background reading, queries that represent our information need have to be 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 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 that the Cypher query is more elegant than SQL.

Processing Cypher queries depends on the schema information that must be loaded. 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 to pass to the searcher and create a BM25 ranking. The code that generates the rankings for all topics is presented in Figure 4.12. In only a few lines of Python code, it is easy to create rankings. From this point, writing the content of `hits` to a runfile and evaluating using `trec_eval` is trivial.

Instead of "just" ranking with BM25, using, e.g., the metadata 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 often specialize in specific 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 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 this group of people writes can quickly be done using a graph query, the query that finds these articles is shown in Figure 4.13.

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

For another example, the graph query language is also valuable when considering entities. Journalists write news articles that relate to events concerning, e.g., people, organizations, 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 is worthwhile to consider the entities identified in the article instead. Important entities tend to be mentioned at the beginning of a news article Kamphuis et al. [2019a]; 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. Pyserini [Lin et al., 2021] that can be used to tokenize the text the same way the documents were tokenized. Figure 4.16

```python
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 a processed topic file was used where only the topic identifier and article id are available. 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.

```python
# 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.

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

This chapter described the prototype implementation of GeeseDB, and how we envision graph databases can be used for information retrieval research. The GeeseDB system can be considered the question on our research question: *Can we extend the benefits from using relational databases for information retrieval to using graph databases, while being able to express graph-related problems easier?* As GeeseDB is built on top of a relational engine it automatically inherits the benefits of using relational databases for IR. As GeeseDB can process graph queries, we also are able to express graph related problems.

GeeseDB is still however a prototype system, and more functionalities need to be implemented. In particular, although the architecture is not coupled, it is not as efficient as traditional methods that do use coupled architectures. The overhead introduced introduce in coupled architectures is lower than the retrieval efficiency from methods that are state of the art but work coupled.

Also, not all graph operators are yet supported. In order to make GeeseDB a usable system for IR researchers, more operators need to be implemented and the system needs to be developed to be more robust.

```python
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

# Finding 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 [2012a] 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 | $1.23 \pm 2.09$ | Average time it takes to select candidates and disambiguate per document |
| No embedding reset | $0.26 \pm 1.60$ | 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 | $0.15 \pm 1.31$ | 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 | $0.13 \pm 1.19$ | 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

# MMEAD

**Abstract**

MMEAD, or MS MARCO Entity Annotations and Disambiguations, is a resource for entity links for the MS MARCO datasets. We specify a format for how links for the MS MARCO document and passage collections can be stored and shared. Following this specification, we release entity links to Wikipedia for the document and passage for both MS MARCO collections (v1 and v2). Links have been produced by the REL and BLINK systems. MMEAD is an easy-to-install Python package, allowing users to load the link data and entity embeddings effortlessly. Using the MMEAD data takes only a few lines of code. Finally, we show how MMEAD can be used for IR research that uses entity information. On the MS MARCO v1 passage dataset, we improve recall@1000 and MRR@10 on more complex queries by using this resource. We also provide a demonstration to show that entity expansions can also be used for interactive search applications.

## 6.1 Introduction

The MS MARCO datasets [Bajaj et al., 2016] have become the defacto benchmark for evaluating deep learning methods for Information Retrieval (IR). The TREC deep learning track [Craswell et al., 2021], which has run since 2019, drives its datasets from the MS MARCO passage and document collections. The collections have been used in zero-and few-shot scenarios for diverse retrieval tasks and domains [Thakur et al., 2021, 2022, Xu et al., 2022]. They also serve as primary resources for training deep learning models for downstream IR tasks such as conversational search [Dalton et al., 2021] and search over knowledge graphs [Gerritse et al., 2022], achieving state-of-the-art results.

Purely text-based neural IR models, trained using MS MARCO collections, can generally not reason over complex concepts in the social and physical world [Bosselut et al., 2021, Sciavolino et al., 2021]. In response, the recently proposed neuro-symbolic methods aim to combine neural models and symbolic AI approaches, e.g., by using knowledge graphs, which map concepts to symbols and relations. An essential step in developing neuro-symbolic models is connecting text to represent the world's concepts formally. This step is mainly done using *Entity linking*, an intermediary between text and knowledge graphs, which detects entity mentions in the text and links them to the corresponding entries in a knowledge graph.

Despite the proven effectiveness of neuro-symbolic AI – and for IR models in particular [Tran and Yates, 2022, Gerritse et al., 2022, Chatterjee and Dietz, 2022] – the IR community has made limited efforts to develop these models. A primary hindrance is the annotation of large-scale collections with entities; entity linking methods are computationally expensive. Running them over a large text corpus (e.g., MS MARCO v2 with 12M documents and 140M passages) requires extensive resources. This paper aims to fill this gap by making entity annotations of MS MARCO ranking collections readily available.

With this work, we publish MMEAD,[1] a resource that provides entity links for the MS MARCO document and passage ranking collections. Two state-of-the-art entity linking tools, namely REL [van Hulst et al., 2020, Kamphuis et al., 2022] and BLINK [Wu et al., 2020], are utilized for annotating the corpus. The annotations are stored in a DuckDB database, enabling efficient analytical operations and fast access to the entities. The resource is available as a Python package and can be installed and used from PyPi effortlessly. The resource also includes a visual demo, enabling queries with complex compositional structures about entities.

We envisage that MMEAD will foster research in neuro-symbolic IR research and can be used to further improve neural retrieval models. In

[1] MMEAD is pronounced as the drink mead.

our experiments, we show significant improvements on recall for neural re-ranking IR models when using MMEAD annotations as bag-of-word expansions for queries and passages. Our experiments reveal that the difference in effectiveness is even greater (concerning both recall and MRR) for complex queries that require further reasoning over entities.

To show the usefulness of our resource, we also present how to enrich interactive search applications. Specifically, we can show these entities' geographical locations by relating the entities found in passages to their Wikidata entries. Plotting these entities on the world map shows that the MS MARCO passages contain entities worldwide. Plus, we can retrieve all passages associated with a geographical location that we present through an interactive demo.

In summary, this paper makes the following contributions:

- We annotate the queries and documents of the MS MARCO passage and document collections and share these annotations. By sharing these annotations, we ease future research in neuro-symbolic retrieval, which extensively uses entity information. We also provide useful metadata such as Wikipedia2Vec Yamada et al. [2016] entity embeddings.

- We provide a Python library that makes our data easy to use. All data is stored in DuckDB tables, which can be loaded and queried quickly. The library is easy to install through PyPi, and the entity annotations are available with only a few lines of code.

- We experimentally show that retrieval effectiveness measured by recall significantly increases when using MMEAD. The improvement is even greater for hard queries, with low retrieval effectiveness using text-only neural IR models.

- We demonstrate how the data can be used in geographical applications. We show for all entities found in the MS MARCO v2 passage collection a static map where they are located when geographical data is available. Plus, through an interactive demo, we can retrieve all passages associated with a geographical location.

MMEAD is publicly available at *github.com/informagi/mmead*.

## 6.2 Background

In this section, we describe systems that are used for creating entity annotations of MS MARCO collections for MMEAD.

### 6.2.1 REL

REL (Radboud Entity Linker) [van Hulst et al., 2020] is a state-of-the-art open-source entity linking tool designed for high throughput and precision. REL links entities to a knowledge graph (Wikipedia) using a three-stage approach: (1) mention detection, (2) candidate selection, and (3) entity disambiguation. We briefly explain these three steps:

1. *Mention Detection.* REL starts the entity linking process by first identifying all text spans that might refer to an entity. In this stage, it is essential that all possible entities in the text are identified, as only the output of this stage can be considered an entity by REL. These spans are identified using a named entity recognition (NER) model based on contextual word embeddings. For our experiments, we use the NER model based on Flair embeddings.

2. *Candidate Selection.* Up to seven candidate entities are considered for every mention found by Flair. Part of these entities is selected according to the prior probability $P(e|m)$ of the mention $m$ being linked to the entity $e$. Precisely, the top-4 ranked entities based on $P(e|m) = \min(1, P_{Wiki}(e|m) + P_{YAGO}(e|m))$ are selected, where $P_{YAGO}(e|m))$ is a uniform probability from the YAGO dictionary [Hoffart et al., 2011] and $P_{Wiki}(e|m)$ is computed based on the summation of hyperlink counts in Wikipedia and the CrossWikis corpus [Spitkovsky and Chang, 2012b]. The remaining three candidate entities are determined according to the similarity of an entity and the context of a mention. For the top-ranked candidates based on $P(e|m)$ probabilities, the context similarity is calculated by $\mathbf{e}^T \sum_{w \in c} \mathbf{w}$. Here $\mathbf{e}$ is the entity embedding for entity $e$, and $\mathbf{w}$ are the word embeddings in context $c$, with a maximum length of 100-word tokens. The entity and word embeddings are jointly learned using Wikipedia2Vec [Yamada et al., 2016].

3. *Entity Disambiguation.* The final stage tries to select the correct entity from the candidate entities and maps it to the corresponding entry in a knowledge graph (Wikipedia). For this, REL assumes latent relation between entities in the text and utilizes the Ment-norm method proposed by Le and Titov [2018].

REL is designed to be a modular system, making it easy to swap, for example, the NER system with another. All necessary scripts to train the REL script are available on GitHub,[2] making it easy to update REL to a more recent Wikipedia dump. Recently a batch extension [Kamphuis et al., 2022] of REL, REBL, was released, which improves the efficiency of REL for large-scale annotations, particularly in the candidate selection and entity disambiguation stages.

### 6.2.2 BLINK

BLINK [Wu et al., 2020] is a BERT-based [Devlin et al., 2018] model for candidate selection and entity disambiguation, which assumes that entity mentions are already given. When utilized in an end-to-end entity linking setup, BLINK achieves similar effectiveness scores as REL. Below we describe the three steps of mention detection, candidate selection, and entity disambiguation for end-to-end entity linking using BLINK.

1. *Mention Detection.* The mention detection stage can be done using a NER model. Like REL, we utilized Flair NER [Akbik et al., 2019] for mention detection.

2. *Candidate Selection.* BLINK considers ten candidates for each mention. The candidates are selected through a bi-encoder (similar to Humeau et al. [2019]) that embeds mention contexts and entity descriptions. Both the mention and entity are encoded to two single vectors using the `[CLS]` token of BERT. The similarity score is then calculated using the dot-product of the two vectors representing the mention context and the entity.

3. *Entity Disambiguation.* For entity disambiguation, BLINK employs a cross-encoder to re-rank the top 10 candidates selected by

[2] `https://github.com/informagi/rel`

the candidate selection stage. The cross-encoder used is similar to the work by Humeau et al. [2019], which employs a cross-attention mechanism between the mention context and entity descriptions. The input is the concatenation of the mention text and the candidate entity description.

### 6.2.3 DuckDB

DuckDB [Raasveldt and Mühleisen, 2019] is an in-process column-oriented database management system. It is designed with requirements that are beneficial for the MMEAD resource:

1. *Efficient analytics.* DuckDB is designed for analytical (OLAP) workloads, while many database systems are optimized for transactional queries (OLTP). DuckDB is especially suitable for cases where analytics are more important than transactions. As we release a resource, transactions (after loading the data) are unnecessary, making an analytics database more useful than a transactional-focused one.

2. *Embeddability.* DuckDB runs in-process, which means no database server is run, and all data processing happens in-process. This allows the database to be installed from PyPi without any additional steps.

3. *Efficient transfer.* Because DuckDB runs in-process, it can transfer data from and to the database more easily, as the address space is shared. In particular, DuckDB uses an API built around NumPy and Pandas, which makes data (almost) immediately available for further data analysis within Python.

DuckDB also supports the JSON and parquet file formats, making data loading especially fast when data is provided in such formats.

## 6.3 MMEAD

MMEAD provides links for MS MARCO collections v1 and v2 using BLINK and REL entity linkers; for REL, we use its batch entity linking extension, REBL [Kamphuis et al., 2022]. The knowledge graphs used for the REL and BLINK entity linker are Wikipedia dumps 2019-07 and 2019-08, respectively. Both dumps are publicly available from the linking systems' Github pages.

### 6.3.1 Goals

MMEAD design and creation is based on the following goals:

- *Easy-to-use.* It should be easy to load and use the linked entities in experiments. In only a few lines of code, it should be possible to load entities and use them for analysis. Additional information should also be directly available, like where entities appear in the text or latent representations.
- *High-quality entity links.* With MMEAD, we want to release high-quality entity links for the MS MARCO collection, so applying (neuro-)symbolic models and reasoning over entities becomes reliable.
- *Extendibility.* It should be easy to link the collections with a different entity linking system and publish them in the same format as MMEAD. This way, we can integrate links produced by other entity linking systems and make them automatically available through the MMEAD framework.
- *Useful metadata.* Additional data that can help with experiments should be provided; this includes mapping entities to their respective identifiers and latent representations.

### 6.3.2 Design

**Easy-to-use.** To create an easy-to-use package, we make the MMEAD data publicly available as JSONL files, and this is the same format in which the MS MARCO v2 collection is available. Each line of JSON contains entity links for one of the documents or passages in the collections; see Figure 6.1. The corresponding document can be identified through the JSON field that represents the document/passage identifier: `docid` for documents and `pid` for passages. Then for every section of a document, a separate JSON field is available to access the entities in that section. For passages, there is only one section containing the entity annotation of the passage, while for MS MARCO v2 documents, we link not only the `body` of the document but also the `header` and the `title`.

All essential information about the entity mentions and linked entities are stored in the JSON objects. Specifically, the following metadata is made available: `entity_id`, `start_pos`, `end_pos`, `entity`, and `details`. The field `entity_id` stores the identifier that refers to the entry in the knowledge graph (Wikipedia, in our case). The `start_pos` and `end_pos` fields store the start and end position of the text span that refers to the linked entity (i.e., entity mention). The field `entity` stores the text representation of the entity from the knowledge graph. We chose to store this field for human readability. The `details` field is a JSON object that stores linker-specific information; examples include the entity type available from the NER module and the certainty of the NER module for the identified mention.

**High-quality entity links.** MMEAD provides entity links produced by state-of-the-art entity linking systems. For this paper, we provide links from REL for both MS Marco v1 and v2 passages and docs, and links from BLINK for MS Marco v1 passages. Both these systems have high precision, ensuring that identified mentions and their corresponding entities are likely correct. The knowledge graphs used by the entity linkers are the same as those used in the original studies; this way, extensive research has been done to confirm the precision of the linking systems.

**Extendibility.** We ensure extendibility by clearly describing the format in which the entity links are provided. If another system shares its links in the same format, the MMEAD python library can work with the data directly. As there is a `details` field per entity annotation for linker-specific information, it is always possible to include additional information in new data sources. In the case of REL, there are specific instructions on updating REL to newer versions of Wikipedia, making it possible to easily release links to newer versions of Wikipedia if needed.

**Useful metadata.** Next to the entity links, we also provide additional useful metadata. Specifically, we release Wikipedia2Vec [Yamada et al., 2016] embeddings (300d and 500d feature vectors). REL uses the 300d Wikipedia2Vec feature vectors for candidate selection. These feature vectors consist of word embeddings *and* entity embeddings mapped into the same high-dimensional feature space. These embeddings can be used directly for information retrieval research [Gerritse et al., 2020, 2022]. We also release a mapping of entities to their identifiers. The entity descriptions can change in different versions of Wikipedia, but their identifiers remain constant. The identifier can also be used to find the corresponding entity in other knowledge graphs, e.g., Wikidata.

### 6.3.3 An Example

A passage from the MSMARCO v1 passage ranking collection is shown below.[3]

> *The Manhattan Project and its atomic bomb helped bring an end to World War II. Its legacy of peaceful uses of atomic energy continues to have an impact on history and science.*

There are a couple of text spans in this text that can be considered as entities: "the Manhattan Project", "World War II", and "atomic energy." REL identifies two of these entities: the *Manhattan Project* and *World War II*. The output of the system is converted to our JSON specification, which results in the JSON object presented in Figure 6.1. The values of the `tag` field shows that Flair is more certain about "World War II" being an entity than the "Manhattan Project."

Table 6.1 shows the number of entities found in the collections by the REL system. Blink found 21,968,356 entity links for the v1 passage collection. For 11,177,904 entities the two linking systems produced exactly the same output.

[3]This is the second passage from the collection.

Table 6.1: Number of Entities linked by REL, we show the total number of entities found and how many entities this are per passage.

| | passages | docs |
|---|---|---|
| MS MARCO v1 | 18,561,221 (2.10) | 145,725,732 (45.34) |
| MS MARCO v2 | 233,254,024 (1.69) | 661,183,287 (55.28) |

## 6.4 How To Use

MMEAD comes with easy-to-use python code, allowing users to work with MMEAD effortlessly. To start, MMEAD can be installed from PyPi using pip:

```
$ pip install mmead
```

After installation, the entity links can be loaded into a DuckDB [Raasveldt and Mühleisen, 2019] database with only a couple of lines of code, as shown in Figure 6.2. When running this code for the first time, it will take some time, as all the data needs to be downloaded and loaded into the DuckDB database. After loading the data for the first time, everything will be automatically stored on disk. Then, loading the data for later usage will only take a couple of seconds.

Once the data is loaded, it is ready to use. We provide a simple interface to access the data. The code shown in figure 6.3 loads the entity links available for a document in the MSMARCO v1 passage ranking collection. When using this function the data is provided in JSON format, making it easy to access the annotations.

We also provide word and entity embeddings generated by Wikipedia2Vec [Yamada et al., 2016] based on the Wikipedia dump 2019-07. These embeddings are stored in DuckDB tables and are available as Numpy arrays after loading. Figure 6.4 shows how embeddings are loaded using MMEAD. The example demonstrates that the entity embedding of *Montreal* and the word embedding of "monteral" are closer to each other than the word embeddings of two words "Montreal" and "grean" based on dot-product as a similarity function. The dimension of the embedding vectors (300 or 500) can be specified in the code.

The mapping between the official Wikipedia identifiers and entity text representations is extracted from the 2019-07 Wikipedia dump. If entity annotations from another version of Wikipedia are available, the MMEAD mappings can be used to match entities between the dumps. Needless to say, emerging entities in newer versions of Wikipedia cannot be mapped to the version that is available in MMEAD. However, existing entities in MMEAD can be mapped to newer versions of Wikipedia. Figure 6.5 shows how entity identifiers can be matched to their text and the other way around.

As DuckDB is used as a database engine for MMEAD, it is possible to directly access the underlying tables and issue structured queries. Figure 6.6 shows an example, where a connection to the database is created, and the identifiers of passages containing the entity *Nijmegen* are retrieved. of the passage where the city of Nijmegen was identified as an entity.

All data can be downloaded directly as well, and links to the data are provided on our github page.[4]

## 6.5 Entity Expansion with MMEAD

To demonstrate the usefulness of MMEAD for (neural) retrieval models, we conduct some experiments, extending existing neural models with MMEAD annotations. These experiments serve as a demonstrative application, and the full potential of this resource is to be further explored in (neuro-)symbolic IR models, as shown in Gerritse et al. [2022], Tran and Yates [2022],

### 6.5.1 Methods

**BM25 expansion.** We experiment with three retrieval methods to show the benefits of entity annotation for passage ranking: one baseline method and two methods that use query entity expansion [Dahlia Shehata, 2022]:

**a BM25 – No Expansion.** As a baseline method we used BM25 as implemented in Anserini Kamphuis et al. [2020] using hyper-parameters $k_1 = 0.82$ and $b = 0.68$, shown to be optimal for the MS MARCO dataset. MS MARCO was indexed normally, and no expansion was considered for the queries or the passages.

[4] `github.com/informagi/mmead`

Table 6.2: Results on MS MARCO v1 passage collection, using only the queries that have entity annotations. Bolded numbers are the highest achieved effectiveness. Scores with a dagger (†), are significantly better compared to BM25 with no expansion (run *a*), following a paired t-test with bonferroni correction. For MRR we have not calculated significance scores due to its ordinal scale. [Fuhr, 2018]

| | | R@1000 | | |
|---|---|---|---|---|
| | | dev | hard | harder |
| a. | BM25 – No Expansion | 0.9111 | 0.7855 | 0.7444 |
| b. | BM25 – Entity Text | 0.9183 | 0.8240† | 0.7951† |
| c. | BM25 – Entity Hash | 0.9105 | 0.7980 | 0.7576 |
| d. | RRF – No Expansion + Entity Text | **0.9338**† | **0.8436**† | **0.8124**† |
| e. | RRF – No Expansion + Entity Hash | 0.9250† | 0.8260† | 0.7921† |
| f. | RRF – Entity Text & Hash | 0.9231 | 0.8260† | 0.7982† |
| g. | RRF – No Expansion + Entity Text & Hash | 0.9313† | 0.8370† | 0.8043† |

**b BM25 – Entity Text Expansion.** In this method, passages and queries are expanded with the text representation from their annotated entities. Once the passages and queries have been expanded with entities, we run BM25 with the same hyper-parameter settings described in **a**.

**c BM25 – Entity Hash Expansion.** Instead of using the text representation of entities as an expansion, we consider expanding the passages and queries by the MD5 hash of the entity text. The use of MD5 hashing is to provide a consistent representation of the multi-word terms and avoid partial or incorrect matching between queries and non-relevant passages; e.g., passages that contain the word "united", do not benefit if the query contains the "United States" as an entity. Again after expansion, we run BM25 with the same hyper-parameter settings described in **a**.

In these experiments, the identified entities are deduplicated. As a demonstration of the proposed text expansion method, Figure 6.7 shows how the query expansion is performed using explicit and hashed forms. The added entities provide a more precise context and help eliminate ambiguous terms. Figure 6.8 shows the expansion methods on the relevant passage for this query. The relevant passage can be found through our expansion technique. The linking system recognizes that both the query and passage contain a reference to the entity *Sacagawea*, while they are not spelled the same in the query and the passage.

**Reciprocal Rank Fusion.** As a second series of experiments, we applied Reciprocal Rank Fusion (RRF) [Cormack et al., 2009] to the runs described above. RRF is a fusion technique that can combine rankings produced by different systems. RRF creates a new ranking by only considering the rank of a document in the input. Given a set of documents $D$ and a set of rankings $R$, RRF can be computed as:

$$RRF(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)} \tag{6.1}$$

Here $k$ is a hyperparameter that can be optimized, but we used the default value $k = 60$.

This provides us with four new rankings; the RRF of the pairwise combinations of the three rankings described above and the RRF of all three of these runs:

**d. RRF – No Expansion + Entity Text.** RRF fusion of runs **a** and **b**. The run with no expansions and the run with entity text expansions are considered.

**e. RRF – No Expansion + Entity Hash.** RRF fusion of runs **a** and **c**. The run with no expansions and the run with entity hash expansions are considered.

**f. RRF – Entity Text + Entity Hash.** RRF fusion of runs **b** and **c**. The run with entity text expansions and the run with entity hash expansions are considered.

**g. RRF – No Expansion + Entity Text + Entity Hash.** RRF fusion of runs **a**, **b** and **c**. All three runs are considered.

### 6.5.2 Experimental setup

In our experiments, we use MMEAD as a resource to expand queries and passages with entities. The experiments are performed using MS

MARCO v1 passage ranking collection, where queries containing at least one entity annotation are used. We do not expect meaningful differences for queries without any linked entities, as query expansion with entity is not possible.

As we expect the found entities to provide additional semantic information about the queries and passages, we conduct further testing on the obstinate query sets of the MS MARCO Chameleons [Arabzadeh et al., 2021], which consist of challenging queries from the original MS MARCO passage dataset. In general, ranking methods show poor effectiveness in finding relevant matches for these queries, so our testing focuses on the bottom 50% of the worst-performing queries from the subsets of Veiled Chameleon (Hard), Pygmy Chameleon (Harder), and Lesser Chameleon (Hardest), which represent increasing levels of difficulty.

This gives us four query sets on which we evaluate; (1) all queries that contain entity annotations (*dev* – 1984 queries), (2) all queries in the hard subset that contain entity annotations (*hard* – 680 queries), (3) the queries in the harder subset that contain entity annotations (*harder* – 493 queries), and lastly (4) all queries in the hardest subset that have entity annotations (*hardest* – 322 queries).

The experiments are evaluated using Mean Reciprocal Rank (MRR) at rank ten and Recall (R) at rank one thousand. MRR@10 is the official metric for the MS MARCO passage ranking task, while R@1000 gives an upper limit on how well reranking systems perform. The Anserini [Yang et al., 2017] toolkit was used to generate our experiments.

### 6.5.3 Results

Table 6.2 presents the results for our experiments. If we first look at lines **a-c** in the results table, we can examine the results of our expansion methods compared to the baseline run. Looking at R@1000, we can see that more relevant passages are found using entity expansion for the *dev* collection and its harder subsets. We find more documents only for the harder subsets when using hash-based expansion. However, this effect is smaller compared to using entity text expansion. There is, however, to increase in MRR@10 when using this expansion method. So, entity expansions help when evaluating using R@1000, especially when the queries are more complex. The difference in recall effectiveness becomes larger the more complex the queries get. MRR@10 only gets better when using entity text expansion.

The reciprocal rank fusion methods are presented in lines **d-g**. When using these methods, the R@1000 increases even more. Again, the subsets that contain the more complex queries tend to benefit more. Regarding R@1000 effectiveness, the best RRF method uses a ranking from the normal, not expanded index, with the index that has been expanded with the entity text. So again, entity text expansion helps recall more than using hash expansion. Although the RRF methods improve recall, MRR@10 does not benefit from RRF compared to using only one of the expansion techniques.

## 6.6 Beyond Quantitative Results

In the previous section, we demonstrated the value of MMEAD in quantitative evaluations, where we leverage entities to improve retrieval effectiveness in standard benchmark datasets. Beyond these quantitative results, MMEAD can also help enrich interactive search applications in various ways. This section describes a few such examples.

Entity links to Wikidata provide an entrée into the broader world of open-linked data, which enables integration with other existing resources. This allows us to build interesting "mashups" or support search beyond simple keyword queries. As a simple example, we can take the entities referenced in MS MARCO, look up the coordinates for geographic entities, and plot them on a map. Figure 6.9 shows a world map with all entities found in the MS MARCO v2 passage collection mapped onto it (each shown with a red dot). The results are as expected, as the red dots' density largely mirrors worldwide population density, although (also as expected) we observe more representation from entities in North America, Europe, and other better-developed parts of the world.

Figure 6.9 is a static visualization, but we can take the same underlying data and principles to create interesting interactive demonstrations. Geo-based search is an obvious idea, where users can specify a geographic region—either by dragging a box in an interactive interface to encompass a region of interest, or specifying a geographic entity. For example, the user might ask “Show me content about tourist sites in Paris” and receive passages about the Eiffel Tower in which Paris is not mentioned explicitly. Simple reasoning based on geographic containment relationships on open-linked data resources would be sufficient for answering this query. While it is true that pretrained transformers might implicitly contain this information, they can never offer the same degree of fine-grained control provided by explicit entity linking.

As a simple demonstration, we have taken MMEAD, reformatted the entity links into RDF, and ingested the results into the QLever SPARQL engine [Bast and Buchhold, 2017].[5] By combining MMEAD with RDF data from Wikidata and OpenStreetMap, we can issue SPARQL queries such as “Show me all passages in MS MARCO about France”.

The query is shown in figure 6.10, which gives us 122,316 entities found in the collection that have a connection with France (most of them are located there). Then we can automatically show the entities on a map, as presented in figure 6.11, we only show the first 1000 entities found.

Not all found entities are located in France, that is because there are some entities that are related to France (France is mentioned in their Wikidata), but are located somewhere else. One of the blue dots in Germany is for example the source of the river *Moselle*. This river starts in Germany by splitting of from the *Rhine*, and then goes through France.

## 6.7 Conclusion and Future Work

This research presents the resource MMEAD, or MS MARCO Entity Annotations and Disambiguations. MMEAD contains entity annotations for the passages and documents in MS MARCO v1 and v2. These annotations make entity-oriented research on the MS MARCO collection easier. Links have been provided using the REL and BLINK entity linking systems. Using DuckDB, the data can quickly be queried, making the resource easy to use. Through a demonstration, we show that our resource can be used to enrich interactive search applications.

[5] `https://github.com/ad-freiburg/qlever`

In particular, we present an interactive demo where all entities related to geographical locations can be found and mapped to their location on a map. We experimentally show that MMEAD improves recall effectiveness significantly when using entities for query and passage expansion. When using reciprocal rank fusion, the effectiveness difference becomes even more prominent. With MMEAD, we support the ease of information retrieval research that combines deep learning and entity information.

In the future, we would like annotations from a more diverse group of linking systems. Using the MMEAD format, releasing entity links for collections other than MS MARCO is also possible. We already showed that using entity links improves recall when using the found entities for query expansion. What the effects are when training, e.g., DPR methods that include the entity links is yet to be investigated, an exciting research opportunity in our eyes.

```
{
        "passage": [
        {
                "entity_id": 19603,
                "start_pos": 4,
                "end_pos": 21,
                "entity": "Manhattan Project",
                "details": {
                        "tag": "ORG",
                        "md_score": 0.613243
                }
        },
        {
                "entity_id": 32927,
                "start_pos": 65,
                "end_pos": 77,
                "entity": "World War II",
                "details": {
                        "tag": "MISC",
                        "md_score": 0.991474
                }
        }
        ],
        "pid": 1
}
```


Figure 6.1: Example of MMEAD annotations for a MS MARCO passage in JSON format. The field `tag` depicts the type of the entity and `md_score` shows the certainty of the mention detection component in identifying the text span as a mention.

```
>>> from mmead import get_links
>>> links = get_links('v1', 'passage', linker='rel')
```

Figure 6.2: Example on how to load MMEAD entity links for the MSMARCO v1 passage collection.

```
>>> links.load_links_from_docid(123)
{"passage":[{"entity_id":"7954681", ... }
```

Figure 6.3: Example on how to load the entity links for a document. For formatting reasons, we do not show the full output.

```
>>> from mmead import get_embeddings
>>> e = get_embeddings(300, verbose=False)
>>> montreal_word = \
e.load_word_embedding("Montreal")
>>> montreal_entity = \
e.load_entity_embedding("Montreal")
>>> green_word = \
e.load_word_embedding("green")

>>> montreal_word @ montreal_entity
31.83191792
>>> montreal_word @ green_word
5.55568354

>>> toronto_word = \
e.load_word_embedding("Toronto")
>>> toronto_word
array([-1.497e-01, -7.765e-01, -1.000e-02, ...])
>>> montreal_word @ toronto_word
21.62585146
```

Figure 6.4: Example code for loading the word and entity embeddings. It show the dot-product between "Montreal" word and entity embeddings is higher compared to the dot-product of embedding vectors for word Montreal and a random word. The word embeddings of Montreal and Toronto, two cities in Canada are more simlilar.

```
>>> from mmead import get_mappings
>>> m = get_mappings(verbose=False)
>>> m.get_id_from_entity('Montreal')
7954681
>>> m.get_entity_from_id(7954681)
'Montreal'
```

Figure 6.5: Entity names and identifiers are accessible in MMEAD. Given an entity text, we can directly find its corresponding identifier and vice versa.

```
>>> from mmead import load_links
>>> cursor = load_links(
...     'msmarco_v1_passage_links',
...     verbose=False
... )
>>> cursor.execute("""
...     SELECT pid
...     FROM msmarco_v1_passage_links_rel
...     WHERE entity='Nijmegen'
... """)
[(771129,), (1273612,), (1418035,), ... ]
```

Figure 6.6: All data is stored in DuckDB tables, it is possible to directly access the tables and issue queries. In this example, we extract the identifiers of passages that contain the city of Nijmegen.

**a.** did sacajawea cross the pacific ocean with lewis and clark

**b.** The same text as shown in **a.** + *Sacagawea Clark Pacific Ocean C. S. Lewis*

**c.** The same text as shown in **a.** + `860324 a97fed 3e3b0e 3fe907`

Figure 6.7: Example of queries for the three different experiments; (**a**) non-expanded passage, (**b**) Entity Text Expansion, and (**c**) Entity Hash Expansion. Expansions are written in italic. The MD5 hashes shown in (**c**) are shortened in this example for formatting.

**a.** Introduction. Sacagawea, as everyone knows, was the young Indian woman who, along with her baby, traveled with Lewis and Clark to the Pacific Ocean and back.She was a great help to the expedition and many organizations are preparing celebrations to commemorate the 200-year anniversary of the endeavor.y: M. R. Hansen. Sacagawea, as everyone knows, was the young Indian woman who, along with her baby, traveled with Lewis and Clark to the Pacific Ocean and back.

**b.** The same text as shown in **a.** + *Indian Ocean James Hansen Sacagawea India Oceania William Clark Meriwether Lewis Pacific Ocean*

**c.** The same text as shown in **a.** + `fe6fc8 860324 aa84e6 7847ef 3e3b0e 7d31e0 2d8836 e58bef`

Figure 6.8: The relevant passage for the query presented in figure 6.7; (**a**) the non-expanded passage, (**b**) the passage with entity text expansion, and (**c**) the passage with entity hash expansion. Expansions are written in italic. The MD5 hashes shown in (**c**) are shortened in this example for formatting.

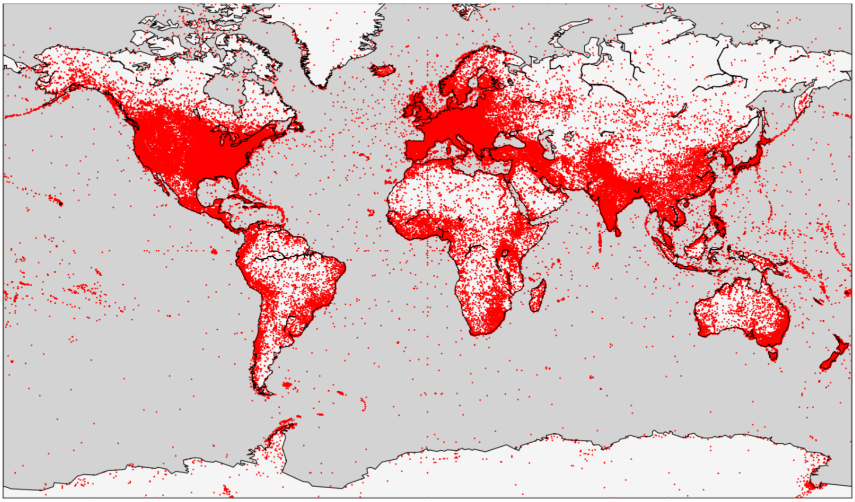

Figure 6.9: Locations of entities found in the MS MARCO v2 passage collection.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ex: <http://example.org/>
PREFIX schema: <https://schema.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX passage: <http://example.org/passage>
PREFIX geo: <http://www.opengis.net/ont/geosparql#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>
SELECT ?pid ?content ?entity ?label ?coord
WHERE {
        ?pid rdf:type passage: .
        ?pid schema:description ?content .
        ?pid passage:has ?entity .
        FILTER (regex(?entity, "wikidata", "i"))
        ?entity rdfs:label ?label .
        ?entity wdt:P625 ?coord .
        ?entity wdt:P17 wd:Q142 .
        FILTER (LANG(?label) = "en")
}
```

Figure 6.10: SPARQL query that produces all entities in the passages of the MS MARCO v2 collection that are related to the country of France.

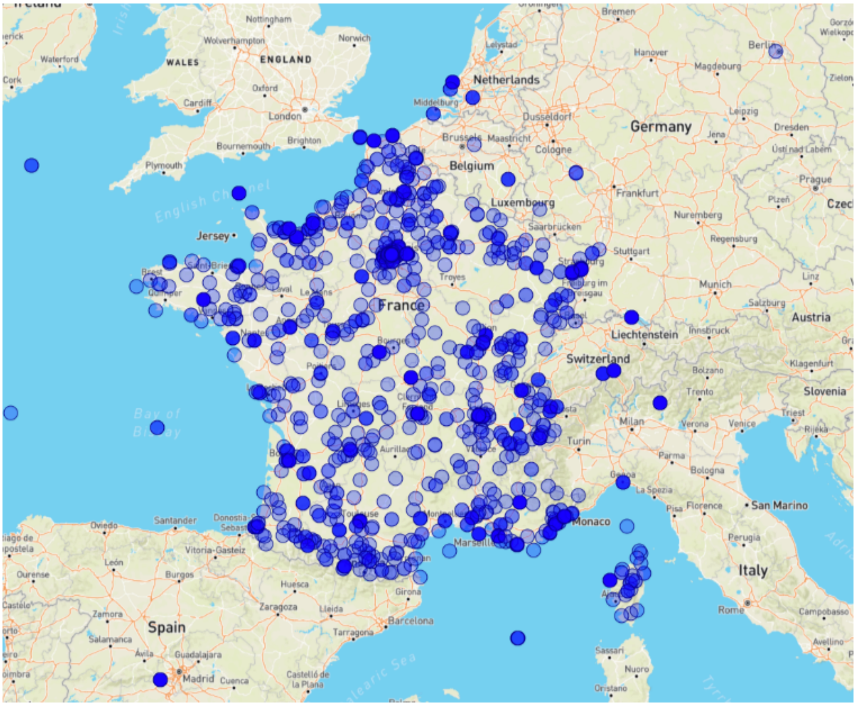


Figure 6.11: First 100 entities found in that are connected to France. Entities are represented with a blue dot on the map.

# Chapter 7

# 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, 2013. URL `https://lucene.apache.org/core/4_3_0/`.

N. Arabzadeh, B. Mitra, and E. Bagheri. MS MARCO Chameleons: Challenging the MS MARCO Leaderboard with Extremely Obstinate Queries. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, page 4426–4435, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384469. URL `https://doi.org/10.1145/3459637.3482011`.

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, France, 2013. Le centre de hautes études internationales d'informatique documentaire. ISBN 9782905450098.

H. Bast and B. Buchhold. QLever: A Query Engine for Efficient SPARQL+Text Search. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 647–656, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349185.

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. Bosselut, R. Le Bras, and Y. Choi. Dynamic neuro-symbolic knowledge graph construction for zero-shot commonsense question answering. In *Proceedings of the AAAI conference on Artificial Intelligence*, volume 35, pages 4923–4931, 2021.

C. J. C. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical report, Microsoft Research, 2010. URL `http://research.microsoft.com/en-us/um/people/cburges/tech_reports/MSR-TR-2010-82.pdf`.

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.

G. V. Cormack, C. L. A. Clarke, and S. Buettcher. Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods. In

*Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 758–759, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605584836. URL `https://doi.org/10.1145/1571941.1572114`.

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.

N. Craswell, B. Mitra, E. Yilmaz, D. Campos, E. M. Voorhees, and I. Soboroff. TREC Deep Learning Track: Reusable Test Collections in the Large Data Regime. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 2369–2375, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380379. URL `https://doi.org/10.1145/3404835.3463249`.

Dahlia Shehata. Information Retrieval with Entity Linking. Master's thesis, 2022. URL `http://hdl.handle.net/10012/18557`.

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.

J. Dalton, C. Xiong, and J. Callan. CAsT 2020: The Conversational Assistance Track Overview. In *The Twenty-Ninth Text REtrieval Conference (TREC 2020) Proceedings*, Gaithersburg, Maryland, USA, 2021. [Sl]: NIST.

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`.

A. Dean-Hall, C. L.A. Clarke, J. Kamps, P. Thomas, N. Simone, and E. Voorhees. Overview of the TREC 2013 Contextual Suggestion Track. In *Proceedings of The Twenty-Second Text REtrieval Conference (TREC 2013) Proceedings*, TREC '13, Gaithersburg, Maryland, USA, 2014. National Institute for Standards and Technology (NIST).

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.

J. Devlin, M.-W. Chang, K. Lee, and K. N. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2018. URL `https://arxiv.org/abs/1810.04805`.

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. Some Common Mistakes In IR Evaluation, And How They Can Be Avoided. *SIGIR Forum*, 51(3):32–41, feb 2018. ISSN 0163-5840. URL `https://doi.org/10.1145/3190580.3190586`.

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.

R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, and S. Kumar. Accelerating Large-Scale Inference with Anisotropic Vector Quantization. In *International Conference on Machine Learning*, 2020. URL `https://arxiv.org/abs/1908.10396`.

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. P. 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.

J. Hoffart, M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum. Robust Disambiguation of Named Entities in Text. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 782–792, Edinburgh, Scotland, UK., July 2011. Association for Computational Linguistics. URL https://aclanthology.org/D11-1072.

S. Humeau, K. Shuster, M.-A. Lachaux, and J. Weston. Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *Proceedings of the 7th International Conference on Learning Representations*, ICLR'19, New Orleans, LA, USA, may 2019. URL https://openreview.net/pdf?id=SkxgnnNFvH.

J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2019.

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. A Proposed Conceptual Framework for a Representational Approach to Information Retrieval. *SIGIR Forum*, 55(2), mar 2022. ISSN 0163-5840. URL `https://doi.org/10.1145/3527546.3527552`.

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. P. 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.

L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report 1999-66, Stanford InfoLab, November 1999. URL `http://ilpubs.stanford.edu:8090/422/`. Previous number = SIDL-WP-1999-0120.

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.

C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL `http://jmlr.org/papers/v21/20-074.html`.

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. 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 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, 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`.

C. Sciavolino, Z. Zhong, J. Lee, and D. Chen. Simple Entity-Centric Questions Challenge Dense Retrievers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6138–6148, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. URL `https://aclanthology.org/2021.emnlp-main.496`.

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, 2019. 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 2012a. European Language Resources Association (ELRA). URL `http://www.lrec-conf.org/proceedings/lrec2012/pdf/266_Paper.pdf`.

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 2012b. 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*, number 6 in ICIA'05, pages 2–6. Washington, DC., 2005. URL `http://ciir.cs.umass.edu/pubfiles/ir-407.pdf`.

N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL `https://openreview.net/forum?id=wCu6T5xFjeJ`.

N. Thakur, N. Reimers, and J. Lin. Domain adaptation for memory-efficient dense retrieval. *arXiv preprint arXiv:2205.11498*, 2022.

H. D. Tran and A. Yates. Dense Retrieval with Entity Views. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, page 1955–1964, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392365. URL `https://doi.org/10.1145/3511808.3557285`.

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.

L. Wu, F. Petroni, M. Josifoski, S. Riedel, and L. Zettlemoyer. Zero-shot Entity Linking with Dense Entity Retrieval. In *EMNLP*, 2020.

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.

C. Xu, D. Guo, N. Duan, and J. McAuley. Laprador: Unsupervised pretrained dense retriever for zero-shot text retrieval. *arXiv preprint arXiv:2203.06169*, 2022.

I. Yamada, H. Shindo, H. Takeda, and Y. Takefuji. Joint Learning of the Embedding of Words and Entities for Named Entity Disambiguation. In *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, pages 250–259, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. URL `https://aclanthology.org/K16-1025`.

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