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)

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 Drukkerij

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 like Turing did, nowadays one can approach this question by asking it to a search engine. When issuing this query to popular web search engines we get different 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 states machines can not think[1]. We use these systems that process queries every day in our lives to provide us information. Whereas Google and Bing are all purpose web engines that mainly focus on finding and retrieving web data, people also used specialized search systems in their day-to-day lives, examples are: Amazon / EBay for product search, NS for public transport in the Netherlands, Scholar / Zeta Alpha for scientific resources, Youtube / TikTok for Videos, or Facebook / LinkedIn for people. When searching for the query "Can machines think?", the approach of searching through text document only might be sufficient for the user. However in many cases when searching today, only considering text is not sufficient.

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

When one wants to buy a product on Amazon, aspects other than text also need to be considered. Lets say for example you want to buy an IPhone; What is the price, which edition is the most recent, or which color does it have. When someone searches for people on LinkedIn, they are generally more interested in persons that have connections in common compared to complete strangers. If you are looking for someone to do a job, it is ideal that a shared connection can vouch for them.

1.1 Thesis Contributions and Structure

1.2 Publications

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

3.1 Introduction

Where commonly information retrieval researchers use inverted indexes as data structures, there is also history of researchers using relational databases for representing the data in information retrieval systems.

3.1.1 Boolean retrieval

Perhaps the earliest work on using relational databases for information retrieval is the work by Schek and Pistor [45]. In this work the authors recognize that the relational data model is widely accepted as an interface to query structured data, however in cases of unstructured data, like text, it is inconvenient to use. They proposed an extension for the relational model by allowing Non First Normal Form (NF^2)

relations. This allows for text queries to be more easily expressed, however the systems that can be build in this language are basically boolean retrieval systems. Which at the time worked well, but scoring was not a feature implemented.

In as similar fashion, Macleod [37], compared the relational model with the inverted index model. He 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.

3.1.2 Probabilistic Relational Algebra

Fuhr [16] recognized that where databases contain formatted data, IR systems deal with unformatted data and that for this kind of data uncertain inference is required. He proposes to express this uncertainty using a probabilistic relational algebra [17].

3.1.3 BOW retrieval models in SQL

In a more recent work by Mühleisen et al. [39] showed that the common used BM25 ranking function can also be easily expressed using relational tables. Their work specifically focused on the retrieval efficiency of several systems. They argue that instead of using a custom build information retrieval system using an inverted index, researchers could just simply 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.

There was however an interesting observation in the paper that I would like to highlight: All the systems evaluated in this paper implement BM25, there was however a substantial difference between the effectiveness scores produced by these systems, as shown in table 3.1. The only two systems that achieved the exact same effectiveness score were the two database systems. I should however note that both these systems were however developed by the same research group.

These results came out as quite a surprise as the authors took specific care to keep document pre-processing identical for all systems,

Table 3.1: Results presented by Mühleisen et al. [39]; `MAP` and `P@5` on the ClueWeb12 collection are reported for five different systems that run BM25. As shown in the table, only the two database systems achieve the same effectiveness score.

System	MAP	P@5
Indri	0.246	0.304
MonetDB & VectorWise	0.225	0.276
Lucene	0.216	0.265
Terrier	0.215	0.272

Table 3.2: Results from the RIGOR workshop[3], `MAP@1000` on the .GOV2 collection is reported for four different systems that run BM25. As shown in the table, all four implementations report a different effectiveness score.

System	MAP@1000
ATIRE	0.2902
Lucene	0.3029
MG4J	0.2994
Terrier	0.2697

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

3.2 Reproducibility

Not only did we observe the differences in effectiveness scores for BM25 in the paper by Mühleisen et al. [39]. In the SIGIR 2015 Workshop on Reproducibility, Inexplicability, and Generalizability of Results (RIGOR) [3] and the Open-Source IR Replicability Challenge (OSIRRC) workshop [10] similar results are observed. See tables 3.2 and 3.3, respectively.

It is not clear

Table 3.3: Results from the OSIRRC workshop[10], `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 (Lucene)	0.2531	0.3102	0.4240
ATIRE	0.2184	0.3199	0.4211
ielab	0.1826	0.2605	0.3477
Indri	0.2388	0.2995	0.4041
OldDog	0.2434	0.2985	0.4002
Pisa	0.2534	0.3120	0.4221
Terrier	0.2363	0.2977	0.4049

3.3 Variants of BM25

We examined several BM25 variants which will be introduced, how the variant varies from the original formulation as proposed by Robertson et al. is marked in red.

Robertson et al. [43]

Equation 3.1 shows the original formulation of BM25: N is the number of documents in the collection, df_t is the number of documents containing term t, tf_{td} is the term frequency of term t in document d. Document lengths L_d and L_{avg} are the number of tokens in document d and the average number of tokens in a document in the collection, respectively. Finally, k_1 and b are free parameters that can be optimized per collection.

$$\sum_{t \in q} log\left(\frac{N - df_t + 0.5}{df_t + 0.5}\right) \cdot \frac{tf_{td}}{1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right) + tf_{td}} \tag{3.1}$$

Lucene (default)

Equation 3.2 shows the variant implemented in Lucene (as of version 8), which introduces two main differences highlighted in red. First, since the IDF component of Robertson et al. is negative when $df_t > N/2$,

Lucene adds a constant one before calculating the log value. Second, the document length used in the scoring function is compressed (in a lossy manner) to a one byte value, denoted $L_{d\text{lossy}}$. With only 256 distinct document lengths, Lucene can pre-compute the value of $k_1 \cdot (1-b+b \cdot (L_{d\text{lossy}}/L_{avg}))$ for each possible length, resulting in fewer computations at query time.

$$\sum_{t \in q} \log \left(\mathbf{1+} \frac{N - df_t + 0.5}{df_t + 0.5} \right) \cdot \frac{tf_{td}}{k_1 \cdot \left(1 - b + b \cdot \left(\frac{L_{d\text{lossy}}}{L_{avg}}\right)\right) + tf_{td}} \tag{3.2}$$

Lucene (accurate)

Equation 3.3 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.

$$\sum_{t \in q} \log \left(\mathbf{1+} \frac{N - df_t + 0.5}{df_t + 0.5} \right) \cdot \frac{tf_{td}}{k_1 \cdot \left(1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} \tag{3.3}$$

ATIRE [49]

Equation 3.4 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; this has no effect on the resulting ranked list, as all scores are scaled linearly with this factor.

$$\sum_{t \in q} \log \left(\frac{\mathbf{N}}{\mathbf{df_t}} \right) \cdot \frac{\mathbf{(k_1 + 1)} \cdot tf_{td}}{k_1 \cdot \left(1 - b + b \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} \tag{3.4}$$

BM25L [35]

BM25L, as shown in equation 3.5, 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 $(k_1+1) \cdot c_{td}/(k_1+c_{td})$ with $c_{td} = tf_{td}/(1-b+b \cdot (L_d/L_{avg}))$.

The c_{td} component is further modified by adding a constant δ to it, boosting the score for longer documents. The authors report using $\delta = 0.5$ for highest effectiveness.

$$\sum_{t \in q} \log\left(\frac{\mathbf{N+1}}{\mathbf{df_t + 0.5}}\right) \cdot \frac{(\mathbf{k_1+1}) \cdot (c_{td}+\delta)}{k_1 + (c_{td}+\delta)} \tag{3.5}$$

BM25+ [34]

BM25+, as shown in equation 3.6, encodes a general approach for dealing with the issue that ranking functions unfairly prefer shorter documents over longer ones. The proposal is to add a lower-bound bonus when a term appears at least one time in a document. The difference with BM25L is a constant δ to the TF component. The IDF component is again changed to a variant that disallows negative values.

$$\sum_{t \in q} \log\left(\frac{\mathbf{N+1}}{\mathbf{df_t}}\right) \cdot \left(\frac{(\mathbf{k_1+1}) \cdot tf_{td}}{k_1 \cdot \left((1-b) + b \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} + \delta\right) \tag{3.6}$$

BM25-adpt [33]

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. However, Lv and Zhai argued that this does not necessary have to be the case. If there are much fewer documents that have $t+1$ occurrences versus t, it should provide more information compared to when the number of documents are almost equal. In order to find the optimal term-specific k_1 value, the authors want to maximize the information gain for that particular query term. This is done by first identifying the probability of selecting a document randomly from the collection that contains term q at least once in a document as:

$$p(1|0,q) = \frac{df_t + 0.5}{N+1} \tag{3.7}$$

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

$$p(t+1|t,q) = \frac{df_{t+1} + 0.5}{df_t + 1} \tag{3.8}$$

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:

$$G_q^t = log_2\left(\frac{df_{t+1} + 0.5}{df_t + 1}\right) - log_2\left(\frac{df_t + 0.5}{N+1})\right) \tag{3.9}$$

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

$$df_t = \begin{cases} |D_{t|c_{td} \geq t-0.5}| & t > 1 \\ df(q) & t = 1 \\ N & t = 0 \end{cases} \tag{3.10}$$

In this case $df(q)$ is the "normal" document frequency, and c_{td} is the same as in BM25L (pivoted method for length normalization [46]):

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

This means the following: df_t is equal to number of documents in the collection when $t = 0$, it is equal to the "normal" document frequency when $t = 1$, and otherwise it will be the number of documents that have 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 [9] (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:

$$k_1' = \arg\min_{k_1} \sum_{t=0}^{T} \left(\frac{G_q^t}{G_q^1} - \frac{(k_1+1)\cdot t}{k_1 + t}\right)^2 \tag{3.12}$$

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:

$$\sum_{t \in q} \mathbf{G_q^1} \cdot \frac{(\mathbf{k_1'} + \mathbf{1}) \cdot tf_{td}}{\mathbf{k_1'} \cdot \left((1-b) + \cdot \left(\frac{L_d}{L_{avg}}\right)\right) + tf_{td}} \tag{3.13}$$

We found that the optimal value of k_1 is actually 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. [50]

TF $l \circ \delta \circ p \times$IDF [44]

TF $l \circ \delta \circ p \times$IDF, as shown in equation 3.14, models the non-linear gain of a term occurring multiple times in a document as $1+log(1+log(tf_{td}))$. To ensure that terms occurring at least once in a document get boosted, the approach adds a fixed component δ, following BM25+. These parts are combined into the TF component using$tf_{td}/(1-b+b \cdot (L_d/L_{avg}))$. The same IDF component as in BM25+ is used.

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

3.4 Experiments

Our experiments were conducted using Anserini (v0.6.0) on Java 11 to create an initial index, and subsequently using relational databases for rapid prototyping, which we dub "OldDog" [24] after Mühleisen et al. [39]; following that work use MonetDB as well. Evaluations with Lucene (default) and Lucene (accurate) were performed directly in Anserini; the latter was based on previously-released code that we updated and incorporated into Anserini.[1] The inverted index was exported from Lucene to OldDog, ensuring that all experiments share exactly the same document processing pipeline (tokenization, stemming, stopword removal, etc.). While exporting the inverted index,

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

we precalculate all k_1 values for BM25- adpt as suggested by Lv and Zhai [33]. As an additional verification step, we implemented both Lucene (default) and Lucene (accurate) in OldDog and compared results to the output from Anserini. We are able to 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. [39]. The term-specific parameter optimization for the adpt variant was already calculated during the index extraction stage, allowing us to upload the optimal (t, k) pairs and directly use the term-specific k values in the SQL query. The advantage of our experimental methodology is that we did not need to implement a single new ranking function from scratch.

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

All experiments were run on a Linux desktop (Fedora 30, Kernel 5.2.18, SELinux enabled) with 4 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.

3.5 Results

Table 3.4 shows the effectiveness scores of the different BM25 variants.

Firstly,

You might have caught that the effectiveness scores of both ATIRE and Lucene (accurate) are exactly the same.

Table 3.4: Effectiveness scores different BM25 variants, all were implement as SQL queries, so the underlying data representations are exactly 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
TF $l \circ \delta \circ p \times$IDF	.2516	.3084	.1932	.4340	.2465	.3647

3.6 Conclusion

We can conclude that

We [25]

Chapter 4

From Tables to Graphs

4.1 Introduction

In recent years there has been a lot of exciting new information retrieval research that makes use of non-text data to improve the effectiveness of search systems. Consider for example dense representations for retrieval [18, 32, 30], knowledge graphs to leverage entity information [21, 5, 11], and non-textual learning-to-rank features [12, 36]. All these research directions have improved the effectiveness of search systems by making use of more diverse data. Despite the fact that search systems consider more diverse sources of data, the usage of this data is often implemented through the use of 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, that facilitates the exploration on how to use non-content data for ranking, and serves the data in a format suitable for reranking with e.g. transformers or tree based methods.

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

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

ranking models quickly.

4.2 GeeseDB

In short, GeeseDB aims to provide the following functionalities:

- GeeseDB is an easy to install, self-contained Python package available through `pip install` with as few as possible dependencies. It contains topics and relevance judgements for several standard IR collections out-of-the-box, allowing researchers to quickly start developing new ranking models.
- First stage (sparse) retrieval is directly supported. In only a few lines of code it is possible to load documents and create BM25 rankings.
- Data is served in a usable format for later retrieval stages. GeeseDB allows to directly run queries on Pandas data frames for efficient data transfer to sequential reranking algorithms.
- Easy data exploration is supported through querying data with SQL, but more interestingly, also using a graph query language (based on the Cypher query language), making the exploration of new research avenues easier. This prototype supports a subset of the graph query language Cypher, similar to the property graph database model query language as described by Angles [2].

4.3 Design

At the core of GeeseDB lies the full text search design presented by Mühleisen et al. [39]. In this work, a column-store database for IR prototyping is proposed, which uses the database schema described in Figure 4.1, consisting of three database tables. (One for all term information, one for all document information, and one that contains the information on how terms relate to documents; the information that is found in a posting list of an inverted index). Using these three tables they show that BM25 can be easily expressed as a SQL query, with latencies that are on par with custom-build IR engines.

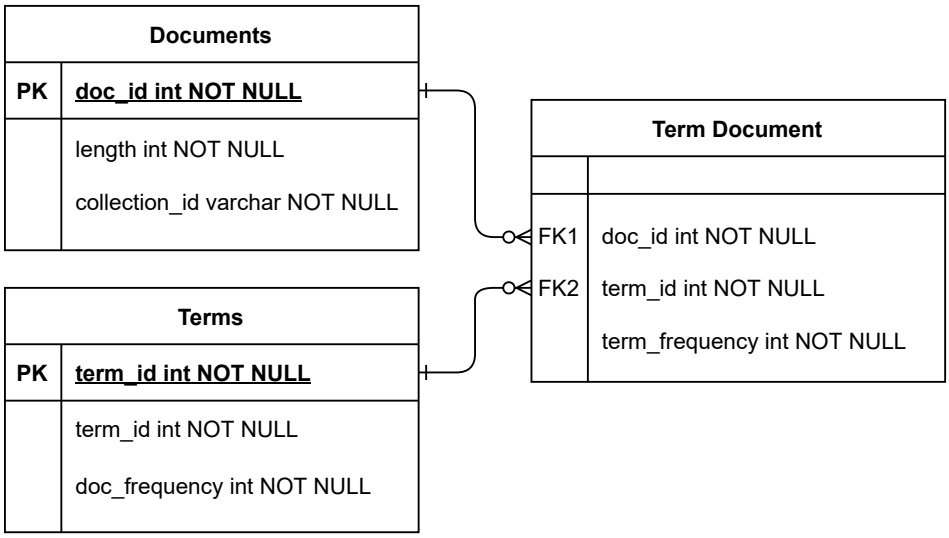

Figure 4.1: Database schema by Mühleisen et al. for full text search in relational databases

In GeeseDB we use the exact same relational schema for full text search. Instead of seeing the document data and term data as tables that relate to each other through a many-to-many join table, it is also possible to consider this schema as a bipartite graph. In this graph both documents and terms are considered as nodes, connected to each other through edges. Basically, if a term occurs in a document there exists an edge between that term and document. GeeseDB uses the data model of property graphs; labeled multigraphs where both edges and nodes can have property-value pairs. The database schema as described in Figure 4.1 would then translate to the property graph schema shown in Figure 4.2. A small example of a graph represented by this schema

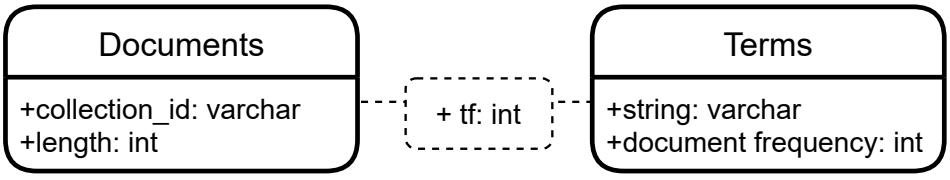

Figure 4.2: Graph schema representing bipartite document-term graph

is shown in Figure 4.3, document nodes contain document specific

information (i.e. document length and the collection identifier), term nodes contain information relevant to the term (i.e. the term string and the term's document frequency), and the the edges between document and terms nodes contain term frequency information (i.e. how often is the term mentioned in the document represented the respective nodes it connects). If one wants to for example also store position data,

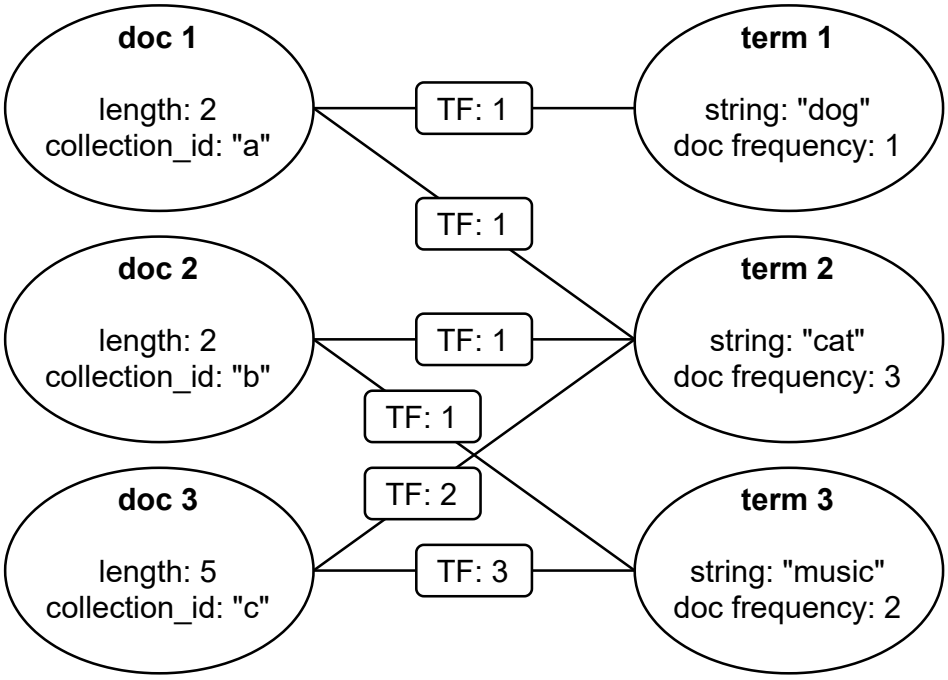

Figure 4.3: Example term-document graph that maps to relational database schema

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

the document-term graph to be present, of course new node types can be introduced in order to explore new search strategies.

4.3.1 Backend

GeeseDB is built on top of DuckDB [41], an in-process SQL OLAP (analytics optimized) database management system. DuckDB is designed to support analytical query workloads, meaning that it specifically aims to process complex long-running queries where a significant portion of the data is accessed, conditions matching the case of IR research. DuckDB has a client Python API which can be installed using `pip`, afterwards it can be used directly. DuckDB has a separated API built around both NumPy and Pandas, providing NumPy/Pandas views over the same underlying data representation, without incurring data transfer (usually referred to as "zero-copy" reading). Pandas DataFrames can be registered as virtual tables, allowing to directly query the data present in Pandas DataFrames. GeeseDB inherits all these functionalities from DuckDB.

As DuckDB is a SQL database management system, we can execute analytical SQL queries on the tables that contain our data, including the BM25 rankings described by Mühleisen et al. [39]. By default, the BM25 implementation provided with GeeseDB implements the disjunctive variant of BM25, instead of the conjunctive variant they used. Although the conjunctive variant of BM25 can be calculated more quickly, we chose to use the disjunctive variant as it is more commonly used by IR researchers and the differences between effectiveness scores are noticeable on smaller collections. For now we only support the original formulation of BM25 by Robertson et al. [43], however support of or adding other versions of BM25 [25] is trivial.

4.4 Graph Query Language

What distinguishes GeeseDB from alternatives, database-backed (OldDog) [24] or native systems (Anserini [54], Terrier [40]) is the graph query language, based on Cypher [15]. For now, GeeseDB implements Cypher's basic graph pattern matching queries for retrieving data. An example of a graph query supported by GeeseDB is presented in Figure 4.4. This query finds all documents written by the same authors

```
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.

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". We plan to support the other keywords of

```
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.

Cypher in the future, as well as directed edges. Everything that is not yet directly supported yet by our implementation can of course still be expressed in SQL, which is fully supported. [2] In order to know how to join nodes to each other if no edge information has been provided, GeeseDB stores information on the schema. This way GeeseDB knows how nodes relate to each other through which edges. GeeseDB has a module for updating the graph schema, allowing researchers to easily set up the graph they want represented in the database.

4.5 Usage

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

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

After installing GeeseDB we can immediately start using it. All examples we show in this paper were run on version v0.0.1 of GeeseDB. However, as GeeseDB is actively being developed we advise readers to use the latest version of GeeseDB, which can be installed when not specifying a package version. It is also possible to install the latest commit by installing the latest version directly from GitHub. As an example, we will show how to use GeeseDB for the background linking task of the TREC News Track [47]. The goal of this task is: *Given a news story, find other news articles that can provide important context or background information.* These articles can then be recommended to the reader to help them understand the context in which these news articles take place. The collection used for this task is the Washington Post V3 collection[3] released for the 2020 edition of TREC. It contains 671.945 news articles published by the Washington Post published between 2012 and 2020, and 50 topics with relevance assessments (topics correspond to collection identifiers of documents for which relevant data has to be found). The articles in this collection contain useful metadata; in particular, we will use authorship information. We extracted 25.703 unique article authors, where it is possible that multiple

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

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

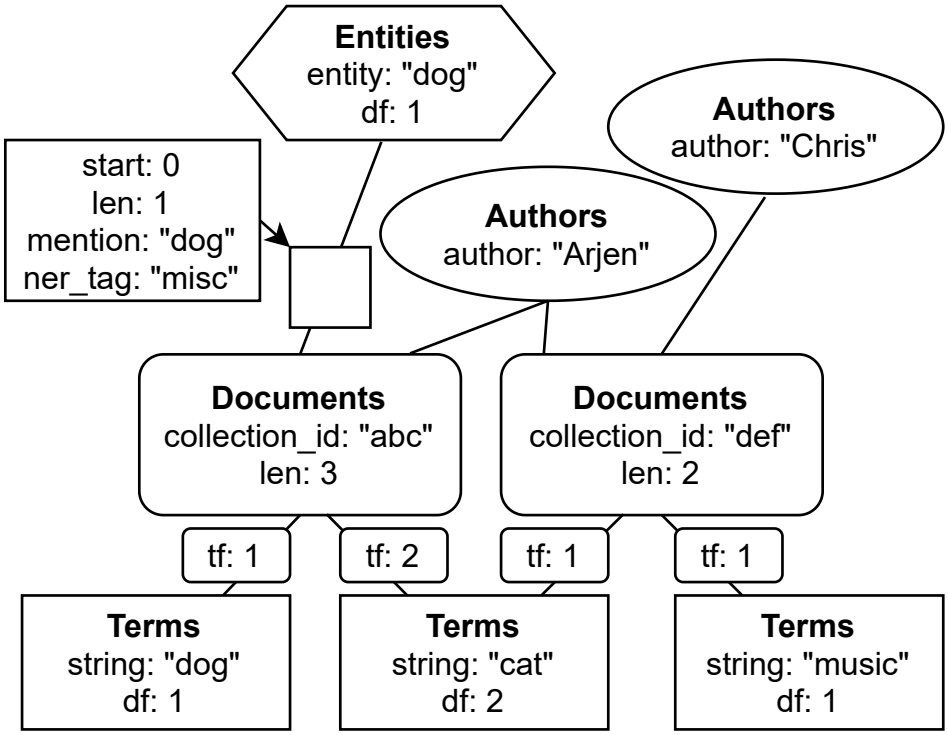

Figure 4.7: Example property graph for the TREC News Track's background linking task. The node types are authors, entities, terms and documents. Edges connect document nodes to other types of nodes. Both edges and nodes can have properties (following the property graph model). Multiple edges may exist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

authors co-wrote a news article. We also annotate documents with entity information which was obtained by using the Radboud Entity Linker [51]. In total 31.622.419 references to 541.729 unique entities were found, the links also contain mention and location information, as well as the `ner_tag` found by the linker's entity recognition module (The `ner_tag` is part of a link, as the entity linker can assign different tags to the same entity).[4] Figure 4.7 illustrates the data schema that we use for the background linking task.

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

```
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. [39]

```
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.

4.5.1 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 [29]. GeeseDB also has functionalities to create the CSV files used here from the CIFF format. Authorship information and entity links can be loaded similarly. After loading the data we can quickly create a BM25 ranking for ad hoc search in the Washington Post collection as shown in Figure 4.9.

For the background linking task however, we do not have regular topics; we only have the collection identifiers of the documents we need

```
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.

to find relevant background info for. In order to search for relevant background reading, queries that represent our information need 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 shown in Figure 4.10. Instead of using Cypher it would also be possible to use SQL, as shown in Figure 4.11; however this example shows again the Cypher query is more elegant.

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

create rankings. From this point it is quite trivial to write the content of `hits` to a runfile, and evaluate using `trec_eval`.

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

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

To give another example; the graph query language is also useful when considering entities. When journalists write news articles, the articles relate to events concerning e.g. people, organisations, or countries. In other words, the basis of news articles lay the entities as they are often the subject of news. So, instead of using the most informative terms in a news article, it could be useful to consider the entities identified in the article instead. Important entities tend to be mentioned in the beginning of a news article [26]; 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 [54] (Lucene) index. Anserini has an easy to use Python extension, Pyserini [28],

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

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

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

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

```

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

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

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

```
# 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.

that can be used to tokenize the text in the same way as the documents were tokenized. Figure 4.16 shows the Python code where we extract the mentions, process them such that they become a usable query for GeeseDB, and then BM25 ranking is created with this query.

4.6 Conclusion

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

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

analyzer = Analyzer(get_lucene_analyzer())

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

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

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

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

Chapter 5

Applications

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 [20, 19, 53, 22, 6, 42, 8], natural language processing [31, 14], and recommendation [55]. 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 [38], TAGME [13], Nordlys [23], GENRE [7], and REL [52].

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) [4].

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:

(i) 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.

(ii) 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 [48] 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).

(iii) 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 [27].

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 [4]. 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] 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 [52] uses Flair [1] 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

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

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

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

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 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.

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

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. [52] 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.

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

[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.

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.

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

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

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

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 [52] on MS MARCO v2 [4] 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 in mentions that are similar, we expect that it might be worthwhile to implement a locality sensitive hashing algorithm to decrease the number of comparisons needed in this stage. However, we would need to run additional experiments to ensure the effectiveness of the model does not suffer.

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

The candidate selection stage considers the context of a mention. This context has to be constructed from the source document. As a result, we load the source data a second time during candidate selection. Alternatively, we may output mention context in the mention detection

stage, which could then speed up the remaining. However, this would significantly increase the size of the mention detection output. More experiments are needed to strike the right balance here.

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

Chapter 6

Conclusion

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

Bibliography

[1] Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., and Vollgraf, R. 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)* (2019), pp. 54–59.

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

[3] Arguello, J., Crane, M., Diaz, F., Lin, J., and Trotman, A. Report on the SIGIR 2015 Workshop on Reproducibility, Inexplicability, and Generalizability of Results (RIGOR). *SIGIR Forum 49*, 2 (jan 2016), 107–116.

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

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

[6] Balog, K., Ramampiaro, H., Takhirov, N., and Nørvåg, K. Multi-Step Classification Approaches to Cumulative Citation Recommendation. In *Proceedings of the 10th Conference on Open Research Areas in Information Retrieval* (2013), OAIR '13, p. 121–128.

[7] Cao, N. D., Izacard, G., Riedel, S., and Petroni, F. Autoregressive Entity Retrieval. In *International Conference on Learning Representations* (2021).

[8] Chatterjee, S., and Dietz, L. 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* (2022), SIGIR '22, p. 1466–1477.

[9] Church, K. W., and Gale, W. A. Poisson mixtures. *Natural Language Engineering 1*, 2 (1995), 163–190.

[10] Clancy, R., Ferro, N., Hauff, C., Lin, J., Sakai, T., and Wu, Z. Z. 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* (New York, NY, USA, 2019), SIGIR'19, Association for Computing Machinery, p. 1432–1434.

[11] Dalton, J., Dietz, L., and Allan, J. Entity Query Feature Expansion Using Knowledge Base Links. In *Proceedings of the 37th International ACM SIGIR Conference on Research; Development in Information Retrieval* (New York, NY, USA, 2014), SIGIR '14, Association for Computing Machinery, p. 365–374.

[12] Deveaud, R., Albakour, M.-D., Macdonald, C., and Ounis, I. 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* (New York, NY, USA, 2014), CIKM '14, Association for Computing Machinery, p. 1827–1830.

[13] Ferragina, P., and Scaiella, U. 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* (2010), CIKM '10, p. 1625–1628.

[14] Ferrucci, D. Introduction to "This is Watson". *IBM Journal of Research and Development 56* (05 2012), 1:1–1:15.

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

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

[17] Fuhr, N., and Rölleke, T. A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst. 15*, 1 (jan 1997), 32–66.

[18] Gao, L., Dai, Z., Chen, T., Fan, Z., Van Durme, B., and Callan, J. Complement Lexical Retrieval Model with Semantic Residual Embeddings. In *Advances in Information Retrieval* (Cham, 2021), ECIR '21, Springer International Publishing, pp. 146–160.

[19] Gerritse, E. J., Hasibi, F., and de Vries, A. P. Graph-Embedding Empowered Entity Retrieval. In *Proceedings of the 42nd European Conference on Information Retrieval* (2020), pp. 97–110.

[20] Gerritse, E. J., Hasibi, F., and de Vries, A. P. Entity-Aware Transformers for Entity Search. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2022), SIGIR '22, pp. 1455–1465.

[21] Hasibi, F., Balog, K., and Bratsberg, S. E. Exploiting Entity Linking in Queries for Entity Retrieval. In *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval* (New York, NY, USA, 2016), ICTIR '16, Association for Computing Machinery, p. 209–218.

[22] Hasibi, F., Balog, K., and Bratsberg, S. E. Exploiting Entity Linking in Queries for Entity Retrieval. In *Proceedings*

of the 2016 ACM International Conference on the Theory of Information Retrieval (2016), ICTIR '16, p. 209–218.

[23] Hasibi, F., Balog, K., Garigliotti, D., and Zhang, S. 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* (2017), SIGIR '17, p. 1289–1292.

[24] Kamphuis, C., and de Vries, A. P. 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* (Aachen, 2019), CEUR-WS.org, pp. 47–49.

[25] Kamphuis, C., de Vries, A. P., Boytsov, L., and Lin, J. Which BM25 Do You Mean? A Large-Scale Reproducibility Study of Scoring Variants. In *Advances in Information Retrieval* (Cham, 2020), J. M. Jose, E. Yilmaz, J. Magalhães, P. Castells, N. Ferro, M. J. Silva, and F. Martins, Eds., ECIR '20, Springer International Publishing, pp. 28–34.

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

[27] Le, P., and Titov, I. 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)* (July 2018), pp. 1595–1604.

[28] Lin, J., Ma, X., Lin, S.-C., Yang, J.-H., Pradeep, R., and Nogueira, R. Pyserini: An easy-to-use python toolkit to support replicable ir research with sparse and dense representations. *arXiv preprint arXiv:2102.10073* (2021).

[29] Lin, J., Mackenzie, J., Kamphuis, C., Macdonald, C., Mallia, A., Siedlaczek, M., Trotman, A., and de Vries,

A. 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* (New York, NY, USA, 2020), SIGIR '20, Association for Computing Machinery, p. 2149–2152.

[30] Lin, S., Yang, J., and Lin, J. Distilling Dense Representations for Ranking using Tightly-Coupled Teachers. *CoRR abs/2010.11386* (2020).

[31] Lin, T., Mausam, and Etzioni, O. Entity linking at web scale. In *Proceedings of the Joint Workshop on Automatic Knowledge Base Construction and Web-scale Knowledge Extraction (AKBC-WEKEX)* (June 2012), pp. 84–88.

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

[33] Lv, Y., and Zhai, C. Adaptive Term Frequency Normalization for BM25. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management* (New York, NY, USA, 2011), CIKM '11, Association for Computing Machinery, p. 1985–1988.

[34] Lv, Y., and Zhai, C. Lower-Bounding Term Frequency Normalization. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management* (New York, NY, USA, 2011), CIKM '11, Association for Computing Machinery, p. 7–16.

[35] Lv, Y., and Zhai, C. When Documents Are Very Long, BM25 Fails! In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 2011), SIGIR '11, Association for Computing Machinery, p. 1103–1104.

[36] Macdonald, C., Santos, R. L., and Ounis, I. On the Usefulness of Query Features for Learning to Rank. In *Proceedings of the 21st ACM International Conference on Information and*

Knowledge Management (New York, NY, USA, 2012), CIKM '12, Association for Computing Machinery, p. 2559–2562.

[37] Macleod, I. A. Text Retrieval and the Relational Model. *Journal of the American Society for Information Science 42*, 3 (1991), 155–165.

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

[39] Mühleisen, H., Samar, T., Lin, J., and de Vries, A. 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* (New York, NY, USA, 2014), SIGIR '14, Association for Computing Machinery, p. 863–866.

[40] Ounis, I., Amati, G., Plachouras, V., He, B., Macdonald, C., and Johnson, D. Terrier Information Retrieval Platform. In *Advances in Information Retrieval* (Berlin, Heidelberg, 2005), D. E. Losada and J. M. Fernández-Luna, Eds., Springer Berlin Heidelberg, pp. 517–519.

[41] Raasveldt, M., and Mühleisen, H. DuckDB: An Embeddable Analytical Database. In *Proceedings of the 2019 International Conference on Management of Data* (New York, NY, USA, 2019), SIGMOD '19, Association for Computing Machinery, p. 1981–1984.

[42] Reinanda, R., Meij, E., and de Rijke, M. Mining, Ranking and Recommending Entity Aspects. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2015), SIGIR '15, p. 263–272.

[43] Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M., and Gatford, M. Okapi at TREC-3. In *TREC* (1994).

[44] Rousseau, F., and Vazirgiannis, M. 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* (New York, NY, USA, 2013), SIGIR '13, Association for Computing Machinery, p. 917–920.

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

[46] Singhal, A., Buckley, C., and Mitra, M. Pivoted document length normalization. In *Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (New York, NY, USA, 1996), SIGIR '96, Association for Computing Machinery, p. 21–29.

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

[48] Spitkovsky, V. I., and Chang, A. X. A Cross-Lingual Dictionary for English Wikipedia Concepts. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)* (2012), European Language Resources Association (ELRA), pp. 3168–3175.

[49] Trotman, A., Jia, X., and Crane, M. Towards an Efficient and Effective Search Engine. In *OSIR@ SIGIR* (2012), pp. 40–47.

[50] Trotman, A., Puurula, A., and Burgess, B. Improvements to BM25 and Language Models Examined. In *Proceedings of the 2014 Australasian Document Computing Symposium* (New York, NY, USA, 2014), ADCS '14, Association for Computing Machinery, p. 58–65.

[51] van Hulst, J. M., Hasibi, F., Dercksen, K., Balog, K., and de Vries, A. P. 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* (New York, NY, USA, 2020), SIGIR '20, Association for Computing Machinery, p. 2197–2200.

[52] van Hulst, J. M., Hasibi, F., Dercksen, K., Balog, K., and de Vries, A. P. 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* (2020), p. 2197–2200.

[53] Xiong, C., Callan, J., and Liu, T.-Y. Word-Entity Duet Representations for Document Ranking. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2017), SIGIR '17, p. 763–772.

[54] Yang, P., Fang, H., and Lin, J. 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* (New York, NY, USA, 2017), SIGIR '17, Association for Computing Machinery, p. 1253–1256.

[55] Yang, Y., Irsoy, O., and Rahman, K. S. 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)* (2018), pp. 777–786.

Summary

Samenvatting

Acknowlegements

Research Data Management

Curriculum Vitae