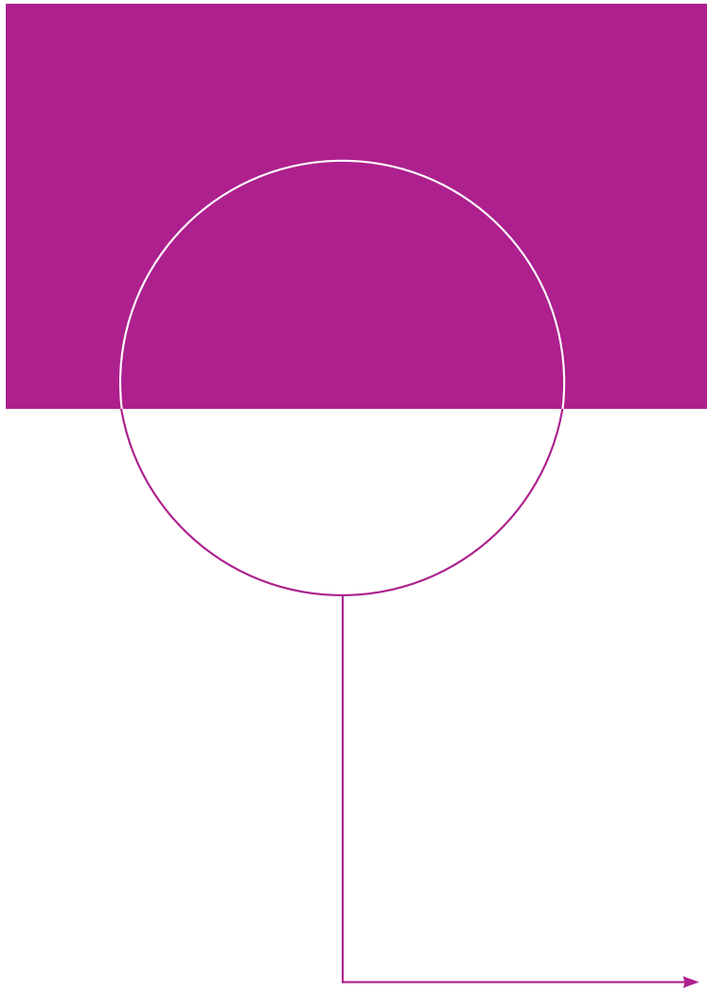

Master's thesis

2020

Anders Grytten Standal

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Computer Science

Anders Grytten Standal

Exploring BM25F for Information Retrieval over Semantic Web Data

June 2020

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Exploring BM25F for Information Retrieval over Semantic Web Data

Anders Grytten Standal

Master of Science in Computer Science
Submission date: June 2020
Supervisor: Trond Aalberg

Norwegian University of Science and Technology
Department of Computer Science

Abstract

An increasing amount of data is extended with semantic markup to form labelled directed graphs, as in the Semantic Web. It provides a richer way to understand relationships between the data points. In such linked data, query languages like SPARQL are used. However, they are often too complicated or impractical for most web search engine users to use in general settings. Instead, information retrieval using keyword search is needed. In information retrieval systems, ranking search results is one of the most vital steps, and creating new ranking methods that are adapted to this graph structure can lead to considerable improvements in precision.

In this thesis we explore different algorithms for ranking search results in information retrieval over semantic web data, and implement and test two of them. Through a series of experiments, we find the optimal values for the tuneable parameters, as well as the best weighting scheme. We then test them against each other, using PageRank and TF-IDF as baselines. We find that one of the two algorithms outperforms all other solutions for our chosen dataset, while the other outperforms only PageRank.

Sammendrag

En økende mengde data er utvidet med semantisk struktur for å forme merkede rettede grafer, slik som i det semantiske web. Det gir en rikere måte å forstå forhold mellom datapunkt på. I slik lenket data er spørringspråk som SPARQL benyttet. Slike spørringspråk er derimot ofte for kompliserte eller for upraktisk for de fleste brukere av websøkemotorer i generelle tilfeller. Det er istedet behov for søk gjennom nøkkelord. I informasjonsgjenfinningssystem er det å rangere søkeresultat en av de viktigste stegene i søkeprosessen, og å lage nye rangeringsmetoder som er tilpasset grafstrukturen kan føre til store forbedringer i presisjon.

I denne masteroppgaven utforsker vi forskjellige algoritmer for å rangere søkeresultater i informasjonsgjenfinning over semantisk web data, og implementerer og tester to av dem. Gjennom en rekke eksperimenter finner vi de optimale verdiene for de innstillbare parameterene, i tillegg til at vi finner den beste vektingsordningen. Vi tester de så mot hverandre, ved å bruke PageRank og TF-IDF som basislinjer. Vi finner ut at en av de to algoritmene utkonkurrerer alle andre løsninger for vårt valgte datasett, mens den andre bare utkonkurrerer PageRank.

Preface

This paper serves as the master's thesis in the subject TDT4900 at the Norwegian University of Science and Technology (NTNU), and marks the end of my studies in the master's degree programme in Computer Science at the department of Computer and Information Science (IDI).

Anders Grytten Standal
Trondheim, June 9, 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Extending the World Wide Web with semantic markup as in the Semantic Web opens up for many possibilities. Hyperlinks between web pages are extended with metadata that enables machines to easily process and better understand the underlying data, while also making it possible for humans to execute more complex queries. The data is usually represented quite differently in linked data. Instead of unstructured, or semi-structured documents, entities are usually more akin to collections of key-value pairs. This type of linked data is becoming more widespread, and can be seen in a wide range of applications. Existing query languages like SPARQL are powerful, but for normal end-users of an information retrieval system, it can be too complicated to use. Other traditional information retrieval methods may not be suitable either, because entity representation of data is generally more strictly structured than web documents.

Since the underlying structure of the data is fundamentally different in linked data, information retrieval over semantic web data has garnered substantial attention over the years. Because of the rich semantic relationship between web pages and entities, there are now many more ways to improve search on the web. The relationships and markup can be exploited to find and return more relevant results, enhance indexing, and perform smarter relevance ranking.

1.1 Background and Motivation

Ranking the retrieved results is one of the most critical steps in an information retrieval system. In search engines that operate on large datasets, such as the world wide web, the set of retrieved results can contain several hundred thousand documents. It is therefore important to present the most relevant results first, as the user is unlikely to even look beyond the first ten results. However, deciding

what are the most relevant results is a complex task. Even in established search engines, it is possible that very relevant results are not even present in the first pages of results.

As the research and development on the Semantic Web has matured and linked data has become more popular, many different technologies for Information Retrieval on semantic datasets have been developed. Many of these techniques exploit the semantic relationships between entities to improve computational cost, recall, and precision. Because of the inherently different structure of entities, in contrast to traditional web pages, using standard ranking methods as they are is not always appropriate or optimal. However, many proposed solutions use tried-and-tested methods such as BM25 as a basis when developing new algorithms for ranking entities. The aim of this thesis is to explore whether entity-ranking methods based on BM25 can help improve performance in keyword search.

1.2 Goals and Research Questions

Goal *Implement and test different ranking algorithms designed to be used over semantic web data, and see which performs better for our test data, and additionally see if they outperform other standard ranking methods.*

The main goal of this thesis is to find better ways of ranking search results in information retrieval over semantic web data. We hope this will be used to improve performance in search applications, where the underlying data is semantically linked.

Specifically, we want to answer the following research questions.

Research question 1 *Can algorithms based on BM25, and adapted for search in semantic web data, outperform standard ranking methods such as TF-IDF and PageRank?*

BM25 is by many considered to be state of the art, so we want to know whether algorithms based on this method will perform significantly better than other tried and tested methods, by comparing their scores in a range of metrics. PageRank is a classic graph algorithm used for ranking, making it interesting to use as a baseline when we are testing over a linked dataset.

Research question 2 *Which of the proposed solutions perform best overall?*

Of the proposed algorithms, we want to know if there are any considerable differences in performance between them. Answering this research question can tell us if any of the proposed algorithms are worth pursuing for further research and development.

Research question 3 *How can we tune the weighting and the parameters to attain the optimal performance for the DBpedia dataset?*

All proposed algorithms have one or more tuneable parameters, in addition to requiring manual weighting of property fields. The authors of the algorithms did not test them on the same data, so we need to find a way to optimize the weighting for our dataset.

To achieve our goals and answer our research questions, the following steps will be taken:

- Implement the algorithms as plugins to the Neo4j graph database platform
- Load and index the dataset into Neo4j
- Create a framework for testing the algorithms using different parameters and weights
- Analyze the results

1.3 Thesis Structure

In chapter 2 we introduce the necessary background theory relevant to this thesis, including a brief overview of the semantic web and information retrieval. In chapter 3 we present related work, which includes several different proposed solutions to the ranking problem. chapter 4 contains the implementation details of the algorithms and our system. In chapter 5 we present the experimental plan and setup, with all the data needed to replicate the experiments. Results of the experiments are presented in chapter 6, along with an analysis of the results. We conclude the thesis in chapter 7.

Chapters 2 and 3 are partly based on the project work [1] done in preparation for this master's thesis.

Chapter 2

Background Theory

In this chapter, we introduce the necessary background theory. We begin with a brief introduction to the semantic web and some of its technologies in section 2.1. In section 2.2 we give a general overview of the field of information retrieval, followed by some more relevant specific concepts in the subsections. Section 2.3 describes how we can evaluate the performance of an information retrieval system, by introducing six different metrics. In section 2.4 we introduce three different open semantic datasets.

2.1 The Semantic Web

Conceived by Tim Berners-Lee, the father of the World Wide Web, at the beginning of the century, the semantic web sought to drastically expand the capabilities of the existing web. The idea behind the semantic web is that it seeks to add meaningful metadata to web pages about the information on them and their relationships with each other. The goal of doing so is to make the data machine-readable. The existing world wide web is designed to be readable for humans, and not to be meaningfully interpreted by machines. Crawlers reading a web page only see what words exist on the page, and what outgoing links are pointing to. In the semantic web, crawlers can understand the relationships between pages and objects.

Although relatively few web pages contain semantic markup, it still enjoys widespread use. Google[1] uses the technology in its knowledge graph to deliver knowledge cards you see when you search for public people, locations, films, etc, when using their search engine [2]. An example of this can be seen on the right-

[1] https://www.google.com/

hand side in figure 2.1. There also exists several large open datasets with semantic data, such as DBpedia[2], Wikidata[3], and SciGraph[4].

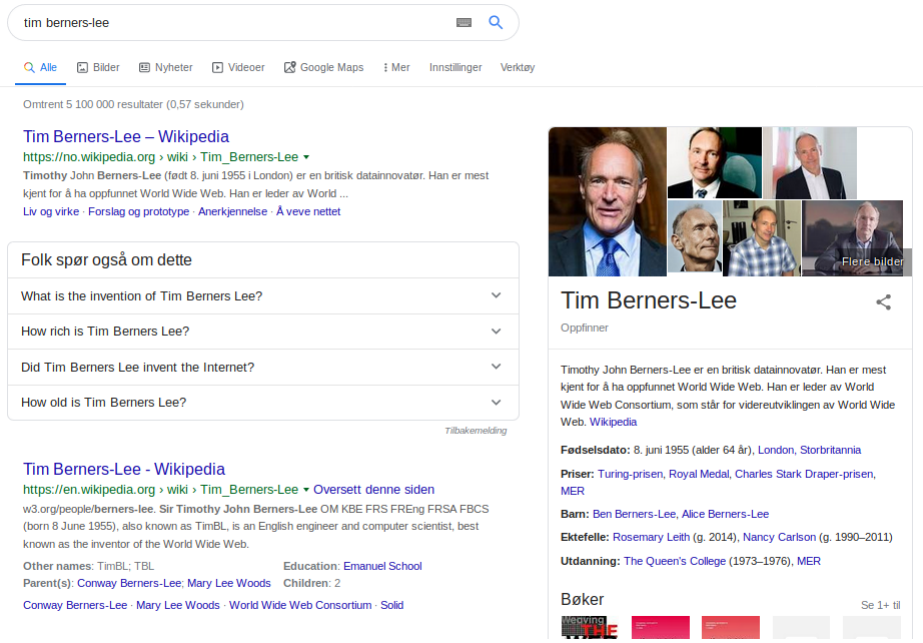

Figure 2.1: Example of an entity presented from Google's knowledge graph

To realize the semantic web, there are a number of standards and technologies that are central, like URI, RDF, RDFS, and OWL. The following subsections briefly explains those standards, as well as providing a description of concepts such as entities and ontologies to provide a basic understanding of how the semantic web works.

2.1.1 Entities and resources

Throughout this paper, we use the term *entity* to describe objects or nodes in the semantic web. Balog [3] defines an entity as something that is uniquely identifiable, and is characterized by its name, type, attributes, and relationships. In other words, an entity can be anything that can be referred to and distinctly identified, such as a product, school, location, person, etc. They can also be more

[2]https://wiki.dbpedia.org/
[3]https://www.wikidata.org/wiki/Wikidata:Main_Page
[4]https://www.springernature.com/gp/researchers/scigraph

abstract concepts, such as distance, emotions, and force. They are often pieces of structured data, in contrast to web pages (referred to as documents hereafter) which contain semi-structured or unstructured data.

The distinction between an entity and a resource is not always clear, as the definition depends heavily on the domain you are working in. A resource is defined by Yu [4] to be the thing denoted in the subject or object part of an RDF statement. In other words, anything that is described with RDF statements.

In this thesis, we interpret these two definitions to mean that entities are the thing or concepts themselves, while resources are the concrete definitions or implementations.

2.1.2 Uniform Resource Identifier

Resources on the semantic web are identified by Uniform Resource Identifiers (URI), which consists of a sequence of characters. A URI can look different depending on the context, as there are several schemes available for expressing it. A URI is a way to globally identify a resource, and can look like a web address, like `http://www.ietf.org/rfc/rfc2396.txt`, while a URI for a person might be their phone number or some other piece of uniquely identifiable information.

URIs are an important building block in the semantic web because they give us a way identify unique objects and concepts. We can for example more easily aggregate information on a person if different sources include a URI for that person. If they just use their name, software agents cannot easily differentiate between people that share the same name.

Uniform Resource Locators (URL) are a subset of URIs that also provides a way to access the identifiable resource, by expressing where it can be located, and how [5].

2.1.3 Resource Description Framework

To describe a resource and its relationships in a machine-readable way, a framework called Resource Description Framework (RDF) [6] is used to structure information in expressive statements.

Meaning is encoded in sets of triples, which are written in the following form: `(subject, predicate, object)`. This can be visualised as a graph structure, as shown in figure 2.2. The subject is the resource you want to say something about. The predicate, or property, says something about the relationship between the subject and the object, which can be another resource or a literal value. A triple expresses a single fact about a resource, whether it be the name of a book, or the relationship between two people. For example, if you are defining the birthdate of a person, it could look like this: `(Herman Melville, born, 1 August 1819)`.

Table 2.1: Set of simplified RDF triples (URIs not shown)

Subject	Predicate	Object
Moby Dick	Author	Herman Melville
Herman Melville	Date of birth	1 August 1819
Herman Melville	Spouse	Elizabeth Knapp Melville

The subject and predicate in a triple are expressed as URIs, while the object can be expressed as a URI, a string literal, or a number. A set of RDF statements denote a labelled, directed graph. For example, based on the set of simplified triples shown in table 2.1, we can build the graph shown in figure 2.3. This figure includes URIs taken from Wikidata, to better illustrate how the real data looks. The blue annotations show the cleartext names, for simplicity.

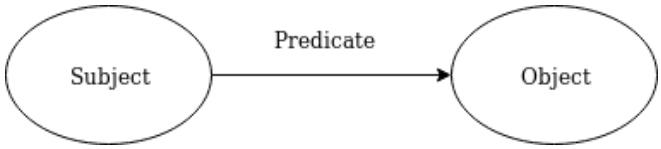

Figure 2.2: RDF statement expressed as a graph structure

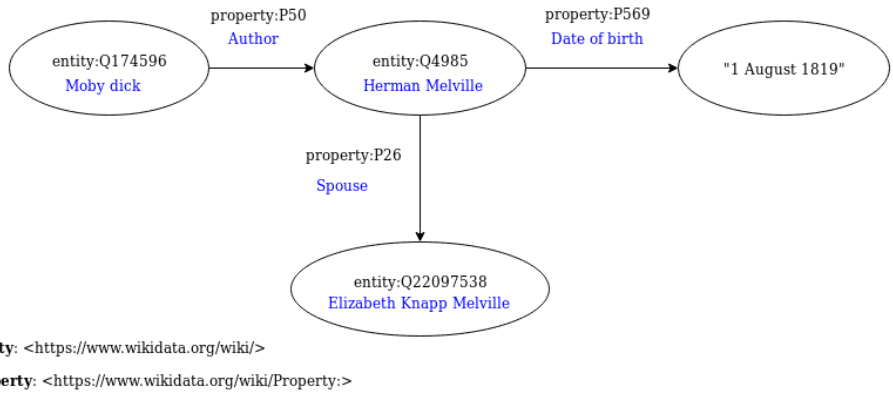

Figure 2.3: RDF graph based on the triples in table 2.1

2.1.4 RDFS and OWL

With RDF, we have a standard way to state facts about real-world objects, which makes information more machine-readable. If we examine the example from figure 2.3, we can imagine many different sources have something to say about the book *Moby Dick*. If everyone is saying something about it using their own definitions, it is suddenly not as easily machine-readable. A shared vocabulary, or a common language, is necessary. RDF Schema (RDFS) is a language we can use to define such a vocabulary.

RDFS consists of a set of terms with we can use to create new classes and properties for a specific domain. Furthermore, all terms are identified by URIs. RDFS contains terms like `rdfs:class` for defining a class, `rdfs:domain` for describing which classes a property can be used with, `rdfs:subClassOf` to define a class to be a subclass on another (`student` can be a subclass of `person`), and so on [4].

Web Ontology Language (OWL) is, like RDFS, a language for creating vocabularies. They have the exact same purpose, and they both provide a set of terms one can use for defining classes and properties. However, OWL can be seen as a natural extension of RDFS, since it provides the means to create a lot more complex connections and vocabularies. For example, with OWL you can put constraints on properties, like how many values a property can take. You can also define two classes to be equal. If you are integrating data from several sources that have used two different classes to describe the same thing, you can define these to be equivalent with the `owl:equivalentClass` property. It also allows you to define two classes to be completely disjoint, so that an entity can never be an instance of both classes. These are just some examples, as OWL contains many more ways to define complex relationships [4].

The vocabularies we create are domain-specific, so in our book example we might for example have standard classes to describe books. In such a vocabulary, we can define classes such as `comic book` and `e-book` that are both subclasses of another class `book`, that can have properties such as `author` and `title`. By having people agree to use this shared language to describe books, it is much easier to create software agents that can process all this information automatically, from different sources.

2.1.5 Ontologies

The book vocabulary we created in the previous section is an example of a very small ontology. An ontology in the context of the semantic web is a collection of information that formally define classes of objects and their relationships in a domain, and represents some area of knowledge [4]. RDFS and OWL are languages used to create these ontologies. The scope of an ontology is tied to

a specific domain, like education, literature, film, or photography. Our book example could be part of a library ontology, providing people with a shared way of describing books and their relationships in a machine-readable way.

In addition to making it easier for software agents to process the data, ontologies have several other benefits. They provide a common understanding about the domain for everyone working on the data, as well as providing a way to reuse the knowledge. Additionally, it makes all assumptions about the domain explicit, and makes the rules about how objects relate to each other clear.

For a more concrete example, say you want to integrate film data from different sources into an RDF model. One of the sources uses the term *star* to describe an actor having a role in a film, while another source uses the more general term *actor*, which can be problematic for automated software agents crawling the data. In this case, an ontology can have a piece of information that says *actor* and *star* means the same thing. You can also extend the ontology to specify that most actors are a subclass of *person*. You can also specify how actors are to be formally identified. This can for example be with the help of a social security number, or other similar pieces of unique identification [7].

2.2 Information Retrieval

Before we introduce solutions to ranking semantic web data, we need to provide a brief overview over the field of information retrieval. Most of the original concepts are important no matter what data representation you are working with.

Information Retrieval (IR) [6] is the field of science focused on the activity of obtaining information, documents, or resources from a structured or semi-structured collection of data. It encompasses building indexes, processing queries, ranking search results, as well as storing, organizing, and representing information items.

IR is a field that has been in continuous development since the 1950's. However, it has historically been a narrow area of interest, primarily gaining attention from experts and librarians. That rapidly changed with the popularity of the world wide web. With billions of documents on the web, the activity of information search has since been at the forefront.

Typically, the way an IR system works on the web, is that you enter one or more keywords into a search engine. This query is then parsed and expanded by the system. The next step involves retrieving all documents that satisfy the query. Typically, this means fetching all the documents that contain some or all of the keywords. The system does this by referring to some index that has been built by crawling all the documents on the web. This index is a structure that might contain the essence of the documents in a way that allows for much faster search. The result might be a list of potentially hundreds of thousands of documents that

match the query. Undoubtedly, the user is not interested in all of them. He or she might have had a specific website in mind when executing the search. These documents therefore then need to be ranked and presented in descending order of relevance. The ranking process tries to identify the documents that have the highest probability of being relevant to the user based on the given query. In the end, the user is presented with a sorted list of documents. The process is illustrated in figure 2.4.

2.2.1 Ranking models

There are many different approaches to ranking. Which one you choose often depends on your use-case, yet some consistently perform better than others in the same domains. A ranking model, or information retrieval model, is like a framework used to produce a ranking function. The choice of model affects how documents and queries are represented, and how documents are ranked. For text-based retrieval, there are three classic models; the boolean model, the vector space model, and the probabilistic model. In this section we will also briefly explain the link analysis ranking model, which is central in the context of linked data.

Boolean model

The boolean model [6] is the simplest retrieval model, and as its name implies, it is based on boolean algebra. Its simplicity is what made it popular, but is also the reason why it is generally outperformed by the other models.

They way it works is that we can pose a query in the form of a boolean expression, using operators like `AND`, `OR`, and `NOT`. The system then retrieves all documents that matches this query. The model only recognizes that terms are present or not in a document. In other words, terms are given a value of 0 or 1 in the index. The simplicity means that a document either matches the query, or it does not. We cannot record how many times a term matches in a document, only that it does. This means we cannot know if a document is more relevant than others, and as a result, we cannot rank these documents. Another downside is that formulating queries using boolean expressions is an arduous task for most users, and most would presumably prefer to express their information needs in keyword queries without boolean operators.

For some domains, the boolean model can still be useful. For expert users, the ability to formulate increasingly complex queries and knowing that the documents returned matches that query exactly, can be very valuable. For lawyers searching in a knowledge base of legal documents, the boolean model might often provide the recall needed.

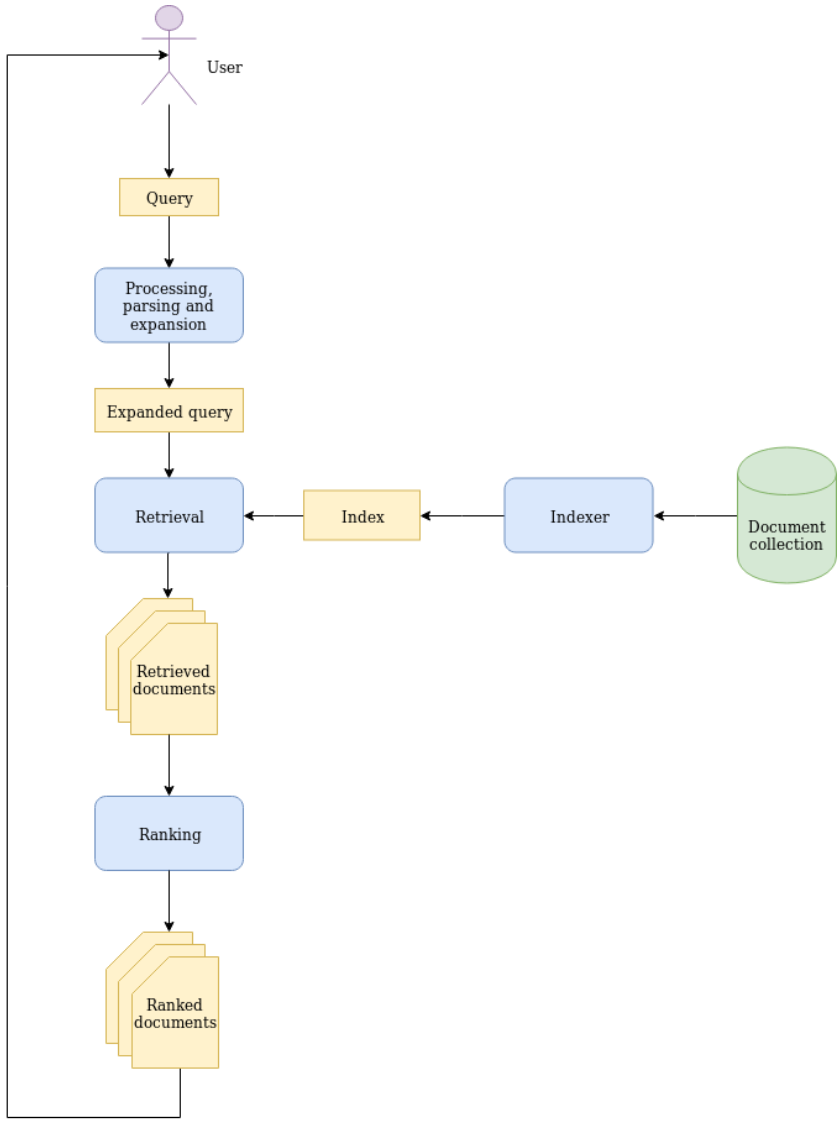

Figure 2.4: The basic components and processes of an IR system

Vector-space model

Since the boolean model is often too limiting for practical use in most domains, some other model is needed. The vector-space model [6] proposes an alternative

solution for producing a ranking function in unstructured text. With this model, partial matching is possible, meaning we can actually rank the returned documents. The vector-space model has proven itself to be simple and fast, yet it still produces good results.

In the vector-space model, the documents and the queries are represented by vectors, where each dimension correspond to a separate term in the corpus. If an index term[5] exists in the document or query, it is given a positive non-zero weight in the vector. These weights are then used to compute the degree of similarity between the documents and the queries.

The weights are usually calculated using TF-IDF (see section 2.2.2), which give higher weights to terms that occur often in a document, but infrequently in the knowledge base as a whole. After calculating the similarity for all documents in a search, this information can then be used to rank the documents in descending order, where the documents that are the closest matches appear at the top.

To calculate the similarity between a document with respect to a given query, we typically calculate the cosine of the angle between the two vectors representing each of them. The function is given in equation (2.1).

$$sim(d, q) = \frac{\vec{d} \bullet \vec{q}}{|\vec{d}| \times |\vec{q}|} \tag{2.1}$$

d represents a document, while q represents the query. The numerator part is the internal product of the document and query vector. The similarity function outputs a value between 0 and 1 (inclusive). If the result is 1, then the document and query are exactly alike, and a value of 0 tells us they have nothing in common.

Probabilistic model

The last of the classic ranking models, the probabilistic ranking model [6], uses a probabilistic framework as its basis. Whereas the vector-space model ranks documents based on similarity to the query, the basic method of the probabilistic model is to rank documents based on the probability of being relevant, based on the user query. There is an underlying assumption in this model that there exists a portion R of the documents that is preferred by the user for a given query. The similarity between a query q and a document d_j is given by the ratio of the probability of the document with representation $\vec{d_j}$ is relevant to the query over the probability that the document is non-relevant to the query, as formulated in

[5] Generally, an index term is any word or keyword that appears in the text of a document [6]

equation 2.2.

$$sim(d_j, q) = \frac{P(R|\vec{d_j}, q)}{P(\bar{R}|\vec{d_j}, q)} \tag{2.2}$$

Link analysis model

With the rise of the world wide web, it began to become clear that just using standard text retrieval models would not be sufficient. The primary reason for this was that the document collections were much larger on the web, and as a result, too many documents were retrieved compared to other systems. There was also a fundamental difference in how the documents were structured. Web pages utilize a link structure; documents contain references to each other. This is something many web search engines were able to take advantage of to modify or improve ranking [6].

The common trait in these approaches that operate on the world wide web, is the usage of *link analysis* to score and rank documents. Link analysis can simply be thought of as the process of evaluating relationships between nodes in a network. It has many uses, even outside of information retrieval, like community detection and market research. However, in the context of information retrieval, it is used to construct ranking measures using link data. Famous approaches in this category include PageRank, which ranks documents based on their number of incoming links and whether those links are from other important documents, and HITS, which ranks documents based on whether they are authoritative on the information they contain, or if they act as hubs for authoritative nodes.

2.2.2 TF-IDF

We have briefly mentioned that terms are often weighted so that we can perform better ranking. It is central in many solutions, especially when many documents are retrieved. Weighting assists us by giving us more precise metrics to help separate relevant documents from less relevant documents. TF-IDF is one of the most popular weighting schemes in IR. It is a combination of term frequency (TF) weighting and inverse document frequency (IDF) weighting.

TF weights

The idea behind term frequency weights is that terms that occur often in a document are likely to be more important than terms that occur less frequently. The simplest way to assign a TF weight to a word in a document is to just set it to the number of occurrences of that word. However, this is not always optimal if documents can be of variable size, since longer documents will then be ranked

higher, even though they are not necessarily more relevant. A way to combat this is to divide the number of term occurrences by the total number of words in the document. You can also compute it logarithmically, with the function $1 + \log f$, where f is the simple term frequency, i.e. number of occurrences [6]. This damping factor helps if the term frequency is very high. For example, if one document has 1000 occurrences of a term, and another document has 2000 occurrences, the difference in relevance is probably negligible.

IDF weights

Some terms occur often in many documents in a collection. If you search a collection with the query "the netherlands", documents containing the term "the" would probably appear at the top of the search results, since many documents include that term more often than the term "netherlands". Intuitively, we do not view "the" as the most important keyword in the query, we probably want to see documents containing "netherlands" near the top instead[6]. That is the idea behind IDF weighting.

Put simply, IDF works by assigning a higher weight to terms that occur rarely across the document set, and lower weights to terms occurring often. IDF for a keyword k_i can be computed with the function $IDF_i = \log \frac{N}{n_i}$, where N is the total number of documents in the set and n_i is the number of documents that contain the keyword k_i [6].

Combining TF and IDF

Combining the two weighting schemes allows for a way to weigh documents that contain a certain term several times higher, but also balance it out if many other documents also mention that term. The final equation can be seen in (2.3).

$$w_{i,j} = (1 + \log f_{i,j}) \times \log \frac{N}{n_i} \tag{2.3}$$

Here, $w_{i,j}$ refers to the weight associated with a term k_i in a document d_j, and $f_{i,j}$ is the frequency of the term in the specified document.

There are several other ways to compute TF-IDF weights, but the principle remains the same: normalize terms based on document length, and accentuate terms that occur in few documents.

[6]Normally, the term "the" would be removed for being a stop-word. This simple example is purely for demonstrative purposes.

2.3 Evaluating performance

Given that information retrieval is a field that is in continuous development where new techniques and methods are created every year to improve search in all sorts of text and media, having standard approaches to evaluating their performance is important. This way, we can more easily compare all the different solutions since we have standard metrics and datasets with an understanding of which documents are relevant given a set of pre-defined queries. In this section we first present TREC and INEX, which are two associations that provide infrastructures to evaluate information retrieval systems. Afterwards, we present a handful of popular metrics to evaluate the performance of information retrieval systems.

2.3.1 TREC

The Text Retrieval Conference (TREC) [8] is an annual series of workshops where the main purpose is to provide the necessary infrastructure for performing evaluation of text retrieval techniques and methods. For each conference, a set of documents and questions are provided by the National Institute of Standards and Technology[7] that the participants run their retrieval systems on, and submit the list of top-ranked results for evaluation. TREC also distributes a piece of software called *trec-eval*, which is used to evaluate information retrieval systems. Given a set of correct answers and the results generated from the information retrieval system you are testing, the system outputs the results of several different evaluation measures, including MAP, R-precision, P@n, and more [9].

2.3.2 INEX

The INitiative for the Evaluation of XML retrieval (INEX) [10], like TREC, was a forum that held yearly workshops where they provided frameworks for evaluating information retrieval systems. However, unlike TREC, INEX focused on structured documents like the ones written with XML, making it a better fit for evaluating semantic web retrieval systems. INEX provided a collection of documents, a set of queries, and the answers (relevant documents) to the information needs associated with the queries. The documents provided varied slightly, but documents from IEEE, Wikipedia, and collections of scanned books have been provided in the past.

[7] https://www.nist.gov/

2.3.3 Precision and recall

Precision describes the fraction of the retrieved results that are relevant. If all the retrieved results are relevant, then precision is at 100 percent, even if not all the total relevant objects are retrieved. This is shown in equation (2.4).

$$precision = \frac{|\text{relevant documents} \cap \text{retrieved documents}|}{|\text{retrieved documents}|} \tag{2.4}$$

Recall describes the fraction of relevant results that have been retrieved over the total number of relevant objects. If all the relevant objects are retrieved, then recall is at 100 percent, but precision is low if many irrelevant objects are retrieved. This is shown in equation (2.5) [6].

$$recall = \frac{|\text{relevant documents} \cap \text{retrieved documents}|}{|\text{relevant documents}|} \tag{2.5}$$

2.3.4 P@n

Precision at n (P@n) [6] is a metric used to calculate the average precision at the top n ranked documents. n is usually set to values like 5 and 10. Calculating the precision for all retrieved documents is usually not interesting from a user-perspective, since most users only look at the first few retrieved results. Therefore, P@n provides a good metric for standard search engines.

2.3.5 MAP

Given a query and its results, the Average Precision (AP) is the average of the precision values for the retrieved relevant documents. Mean Average Precision (MAP) [6] is the mean of all these average precision values for a set of queries and their results. It is a widely used approach for evaluating performance that produces a single number. The function is shown in Equation (2.6).

$$MAP = \frac{1}{N_q} \sum_{i=1}^{N_q} AP_i \tag{2.6}$$

Here, N_q is the total number of queries, and AP_i is the average precision for a query i.

2.3.6 R-precision

R-precision [6] is similar to P@n, in that it calculates the precision at a set position. That position is R in the set of retrieved documents, where R is the

total number of relevant documents for a given query. For example, if in a collection there are 5 relevant documents for a query, and in the first 5 results, 2 are relevant, the R-precision would be $\frac{2}{5}$.

2.3.7 DCG

Discounted Cumulated Gain (DCG) [6] is a metric that uses a graded scale of relevance, instead of a binary one. The main idea behind it is that results can have varying degrees of relevance. You can have several relevant results for a given query, but some may be more relevant than others. If mildly relevant results are ranked higher than highly relevant results, then it carries a penalty. The function for calculating the DCG at a position n, where rel_i is the relevance of the result at the position i is given in equation 2.7.

$$DCG_n = \sum_{i=1}^{n} \frac{rel_i}{\log_2(i+1)} \tag{2.7}$$

2.4 Data

When talking about data representation, we often group them into three categories: unstructured data, semi-structured data, and structured data [3]. Unstructured data is often free-form text, written with full sentences and organized into paragraphs. It is hard to automatically process this data. The content may contain ambiguities, and natural language processing is required. Examples includes emails, blogs, and the majority of web pages. Data starts to become semi-structured by introducing document fields. This often includes fields like *title* that refers to specific parts of the document. Wikipedia pages often contain infoboxes, where key data is listed in a more structured manner. However, not all infoboxes contain the same fields, and not all Wikipedia pages contain infoboxes. This is mostly why we consider them semi-structured. Structured data follows a fixed schema, like in a relational database. Semantic web data is strictly structured, and they are accompanied by ontologies which provide even more structure by defining the relationships and concepts in the data. In the following subsections we present three different semantic datasets, which are defined through sets of triples to form labelled directed graphs.

2.4.1 DBpedia

DBpedia [11] is an open large-scale knowledge base, created by extracting information from Wikipedia[8] and serving it as linked data. The knowledge base contains data in over a hundred languages, and the English version describes over four and a half million entities. Together in all languages, there are over 38 million descriptions of entities [12]. Mappings are created from Wikipedia data to DBpedia's own ontology that consists of over 300 classes. The DBpedia data consists of several billion pieces of information stored as RDF triples. An example of how an entity is described can be seen in table 2.2. Information surrounded by quotation marks are string literals, while the information preceded by a prefix and enclosed with less than and greater than signs are other resources. The DBpedia dataset is often used for testing in the research community, since it is a huge collection of data on many different topics. The dataset is freely available for download as N-triples, or it can be queried online through a SPARQL interface.

Table 2.2: Excerpt from DBpedia: Entity data about Tim Berners-Lee

<dbo:birthDate>	"155-6-8"
<dbo:birthName>	"Timothy John Berners-Lee"
<dbo:birthPlace>	<dbr:England>
	<dbr:London>
<dbp:occupation>	<dbr:Computer_scientist>

2.4.2 Wikidata

Wikidata [13] is a more recent linked data project that also builds upon Wikipedia. It is a large open knowledge base of semantically linked data entities, that anyone can edit and contribute to. It contains descriptions of over 65 million entities, and can be exported in several formats such as RDF and JSON [14]. While DBpedia extracts and adds structure to existing Wikipedia pages, Wikidata focuses on building its knowledge base from the ground-up, and being supplemental to Wikipedia. Wikidata entities are even more strictly structured than entities from DBpedia. The object part of the triples that make up DBpedia entities can contain blocks of text, using predicates such as *abstract* and *comment*. The objects of the Wikidata triples are more often other resources or short and specific literals.

[8] https://www.wikipedia.org/

2.4.3 SciGraph

SciGraph [15] is different kind of dataset. Instead of creating a graph of general Wikipedia data, Springer Nature have created a labelled directed graph connecting entities from the scholarly domain. The entire dataset consists of over 1.5 billion triples, including data about publications, authors, conferences, patents, citations, etc. The data is free to download in several formats, and can also be explored online.

Chapter 3

Related Work

Several ranking systems have been proposed to be used in accordance with semantic web data. They can be broken into several categories, but in this thesis we separate them into two different classes; those that rank documents and those that rank entities. Those that rank documents cannot always be adapted to rank entities, and vice versa. An exception is PageRank, which is an algorithm that exploits the link structure of web pages. Many of the other solutions extend it to also take into account the labeled edges in the graph structure to produce more accurate ranking. We begin this section with an explanation of a couple of algorithm made for ranking web documents. We then go on to explain the solutions constructed to work specifically with entities, until we finally close the section with a longer discussion and comparison of the solutions.

3.1 Document-ranking solutions

Document-ranking solutions are algorithms that are created to rank web documents. Web documents are usually unstructured or semi-structured and often contain both outgoing and incoming links from other web documents.

3.1.1 Vector-space model approach

Vallet *et al.* [16] suggest a method for ranking results in semantic search that is based on the vector-space model. This model does not rank entities themselves, but rather documents that are *annotated* with entities, i.e. documents where the entities are mentioned.

The approach is based on the usage of ontologies. The system they propose can work with virtually any domain ontology, but they provide two ontologies

with some superclasses that must be used. The knowledge base you are working with must be composed from a set of main classes; `DomainConcept` for domain-specific entities such as *Author*, *Book*, and *Library*, `Document` for documents, and `Taxonomy` for class hierarchies that are used for classifying documents and concept classes, and not for instantiating. The second ontology they provide is an annotation ontology, which contains a class `Annotation`. This class has two relational properties; `Instance` and `Document` for relating documents and entities together.

Annotations are automatically assigned a weight using an adaptation of TF-IDF, that reflects how relevant the entity is for the document meaning, based on how frequently the entity occurs in the document. To compute the final similarity between a document and the query, a version of the cosine similarity function is used, altered to compensate for potential shortcomings. This consists of including a normalization factor to compensate for one of the values almost always being too high, and combining the score with the score of a keyword-based algorithm to cope with a potentially incomplete knowledge base, which in the case of the semantic search system employed in the paper, would otherwise result in documents getting a much lower similarity score than they should.

The method was evaluated over a dataset taken from an online newspaper archive, consisting of over 2000 articles, that the authors annotated with almost 3500 annotations. Three reduced ontologies were created, consisting of 143 domain classes. Comparing it to a keyword only search using Lucene[1], it showed significant improvement in precision and recall for some queries. Recall was especially higher when querying for class instances, combined with using class hierarchies and rules.

3.1.2 Pérez-Agüera *et al.*'s BM25F approach

BM25F [18] is an adaptation of the probabilistic BM25 ranking function, developed to perform better with structured documents. BM25 itself can be seen as an extension of the ranking in the vector-space model, as it still relies on term frequency and inverse document frequency, but also incorporating document length normalization [6].

BM25F differs from BM25 by taking into account the structure of the document. This means it uses the different annotated pieces of the document, known as document fields, such as body, title, and anchor text to provide more accurate ranking. For instance, terms in the title of a document usually receive a higher weight than terms in the body.

Pérez-Agüera *et al.* [17] has adapted BM25F to be used in the semantic web by first indexing RDF information along with other field information. As previously

[1] http://lucene.apache.org

mentioned, triples consists of a subject, predicate, and object. The object is often another resource identified by a URI. For example, an entry for the book *Moby Dick* is connected to an entry for its author, *Herman Melville*, most likely using the word *author* as the predicate. Therefore *author* is a descriptive term that can be used to label and index *Herman Melville*, in a field which they have named `inlinks`. They have also created a field called `obj`, where they include information about resources that the current resource is pointing to. *Herman Melville* would then be used to index *Moby Dick*, and vice versa. Other fields they add in the index includes `type`, which are the rdf:type properties that says which classes the resource is an instance of, and `title`, which contain keywords taken from the resource's URI.

Using this information about the resources, we can score them using the adapted BM25F seen in equation 3.1. It shows how to obtain the relevance score for a document d with regard to a query q.

$$BM25F_d = \sum_{t \in q \cap d} \frac{tf(t,d)}{k_1 + tf(t,d)} * idf(t) \tag{3.1}$$

As shown, the equation uses both term frequency and inverse document frequency to obtain the final score. k_1 is just a tunable parameter used to control the non-linear growth of the term frequency. The main difference lies in how the term frequency is calculated, which can be seen in equation 3.2.

$$tf(t,d) = \sum_{c \in d} w_c * tf_c(t,d) \tag{3.2}$$

Here, c is each field in the document, w_c represents the weight of each type of field, and $tf_c(t,d)$ is field term frequency of term t in field c. $tf_c(t,d)$ is normalized using, among other things, the average field length for that particular field.

The authors evaluated BM25F by using the INEX evaluation framework with the DBpedia dataset, where they have mapped the Wikipedia documents in INEX to their respective DBpedia entities. The test set contained over 2.2 million document-entities. They used TREC-eval to assess the precision and recall, as well as the quality of the ranking, using measures such as MAP, GMAP, Precision after n documents (P@n), and R-precision, comparing it to BM25 and Lucene. The BM25-based approaches outclasses Lucene, and BM25F was slightly better than BM25 in most cases. However, BM25F did not seem to profit from the semantic data in the fields that did not have as much text.

3.2 Entity-ranking solutions

Entities are often structured differently from normal web documents. They are more strictly structured, consisting often of a collection of triples, whereas web documents often consists of blocks of text. Links are usually labelled, so the entity collection forms a labelled directed graph.

3.2.1 PageRank

PageRank [19][20] is a graph algorithm employed by Google, which is used to rank web pages according to their global importance. Generally speaking, a web page is seen as important if many other pages link to it, or even if just one important page link to it.

The algorithm works by first assigning an initial PageRank value to each node. Every node distributes its PageRank evenly to every node it points to. For example, If a node has an initial value of 0.4 and two outgoing links, it will distribute 0.2 to each of the nodes it is pointing to, as shown in figure 3.1. It also includes a damping factor, usually set to 0.85, to prevent pages with no outgoing links from assimilating the PageRank from nodes around them. If damping was excluded, a random surfer would eventually end up on those pages every time. The equation is formally described in (3.3).

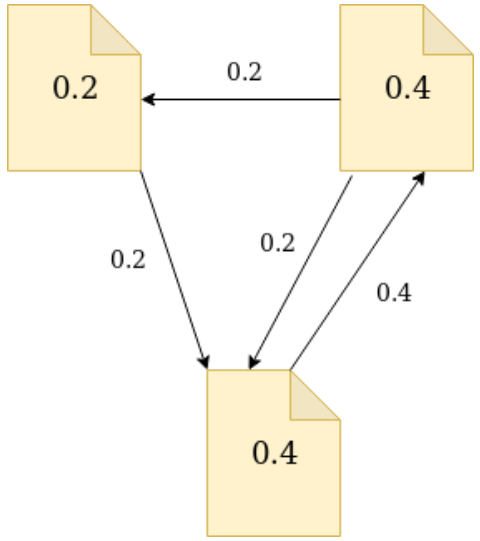

Figure 3.1: Distribution of PageRank (simplified)

$$PR(A) = \frac{1-d}{N} + d\left(\frac{PR(T_1)}{C(T_1)} + ... + \frac{PR(T_n)}{C(T_n)}\right) \tag{3.3}$$

$PR(A)$ Is the PageRank of a node A, while $T_1...T_n$ are nodes pointing to A. $C(T)$ is the T's number of outgoing links, and d is the damping factor.

PageRank has a basis in a random walk model. One can intuitively imagine a web surfer randomly clicking on successive links. The probability that the surfer visits a page is the same as its PageRank. The damping factor corresponds to the probability that the surfer will stop clicking links and request a completely random page.

The algorithm is recursive, and is run until the values converge. This means that the initial values do not matter for the final values, but a smart choice of initial values may reduce the number of iterations needed until convergence.

Although designed to be used for the web, PageRank is suitable for any structure with links. However, it does not take into account the semantic relationships between data if there is any.

3.2.2 ObjectRank

ObjectRank [21] is a ranking algorithm that extends PageRank to perform search in databases modeled as labeled, directed graphs, where there is a natural authority between entities. As opposed to PageRank, ObjectRank performs keyword-specific ranking instead of maintaining a global rank. It also employs a similar random-walk model, with the difference being that random walks start from the entities that contains the keywords. Every object is ranked according to the given keywords, based on the probability that random walkers are found at the entity at that time.

Another way ObjectRank differs from PageRank is that each dataset requires some fine-tuning as to how weights are distributed in the graph. These rules can be easily visualised in an authority transfer schema graph, which shows how much weight an entity type has in regards to other entity types, as shown in Figure 3.2.

One way to think of ObjectRank is to imagine entities distributing authority to other entities. Initially, authority is found at the entities which contain the user-specified keywords. Authority is then distributed from these entities based on the rules in the authority transfer schema graph. The paper illustrates the effects of this approach by using a computer science bibliography as an example. This database contains many scientific papers, authors, conferences, and more as entities, with links between them describing their relationships. The top ranked result for the keyword "OLAP" does not contain the keyword itself, but has been

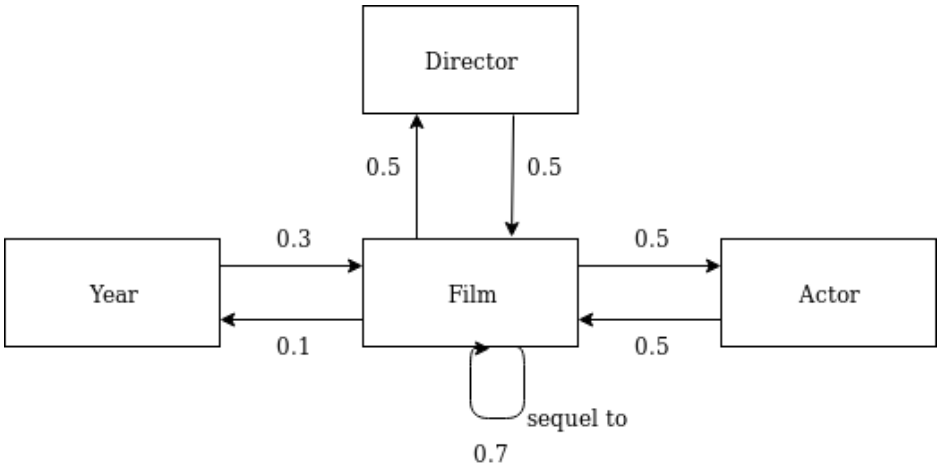

Figure 3.2: Example authority transfer schema from a hypothetical movie graph

cited by many papers that does, or may have been written by authors that have written other important OLAP papers.

ObjectRank was evaluated using two user surveys over two different datasets and by performing experiments to test its computational feasibility. First they used DBLP[2], a large computer science bibliography which contains data on almost five million publications [22], including authors, conferences, and journals. The second survey made use of the IEEE Communications Society[3] publications database, which contain a number of academic papers. In the user surveys, ObjectRank was only evaluated alongside different variations of itself, but in most cases the default algorithm scored best. In the experiments, they conclude that the algorithm is feasible, and demonstrate a number of ways to optimize computation and storage depending on the data and use case.

3.2.3 ReConRank

ReConRank [23] is another extension of PageRank that has been adapted for usage in semantic web data. Its name is a composition of ResourceRank and ContextRank, which are the two methods used to perform the ranking. Put simply, ReConRank works by not only applying a PageRank-like algorithm to resources, but it also applies a similar algorithm to rank their contexts[4]. Not

[2]https://dblp.uni-trier.de/
[3]https://www.comsoc.org/
[4]The paper defines context to be the source or origin of the data.

all data sources are trustworthy or good, so the authors suggest that this can be used as a weight for the ranked resources.

Similar to ObjectRank, ReConRank also only does ranking based on the resultset, instead of maintaining a global rank. This is for efficiency reasons, and to preserve relevancy in the ranking. However, a problem with this approach is that the resulting graph can be quite small with few links. ReConRank solves this by expanding the graph with implied links. For example, contexts are implicitly linked through shared resources, and there are implicit links between resources and their contexts. With this information, resources that occur in more than one context should be ranked higher than those who are just in one. Additionally, if a context is ranked highly, its resources should be ranked higher as well.

It is worth noting that the system that employs ReConRank retrieves not only resources that matches the keyword query, but also adjacent nodes, up to a certain amount of hops, specified by a variable n. When performing the ranking, it removes all resources that do not appear as subjects in the RDF graph. Therefore, the success of the algorithm may not be completely attributed to the ranking itself.

ReConRank was evaluated on a dataset obtained by a crawler on the web following *href* and *rdfs:seeAlso* links, resulting in a dataset of about 15 million triples, with 2.6 million resources and contexts. ReConRank seemed to perform reasonably well on this relatively large dataset, showing a linear increase in computation time. A comprehensive evaluation of ranking quality was not conducted.

3.2.4 Blanco *et al.*'s BM25F approach

An alternative method for scoring RDF resources by adapting BM25F has been conceived by Blanco *et al.* [24]. It is similar to the work of Pérez-Agüera *et al.* in some respects, but they also make a number of different assumptions that dictate how results are calculated, resulting in different rankings.

From a high level, the general approach is the same as in equation 3.1. Calculating the BM25F score for a query Q and document D involves, for each query term, taking the term frequency and dividing it by a tunable parameter $k1$ plus an aggregated normalized weighted term frequency tf_i, and then multiplying it by the term's inverted document frequency.

The normalized weighted term frequency is calculated as seen in equation 3.4.

$$tf_i = \sum_{s=1}^{S} v_s \frac{tf_{si}}{B_s} \tag{3.4}$$

S is all the fields in the document, v_s represent the field weights, while tf_{si} represents the field term frequencies, which is the number of times the term i

appears in the field s. Finally, B_s is the normalization factor, which is calculated by the formula seen in equation 3.5.

$$B_s = \left((1 - b_s) + b_s * \frac{l_s}{avl_s}\right) \tag{3.5}$$

b_s is a tunable parameter between 0 and 1 (inclusive) that is supposed to help control the amount of normalization. l_s is the length of the field s, but the authors argue that indexing the field lengths for all fields is impractical, so they opted to use the size of the document D as the length of all its fields instead. Lastly, avl_s represents the average field length of the given field type.

Finally, once the adapted BM25F score for a query Q and document D has been calculated, it is multiplied by a query-independent factor, such as the document's PageRank score, or the number of links pointing to the document.

The authors evaluated their approach on the Billion Triples Challenge 2009 dataset [25], which contains over 1.14 billion quads, along with a set of queries and relevance assessments. For setting field weights, they manually classified all the property types into three distinct classes: important, unimportant, and neutral. They then assigned the same weight to each property that belonged to the same class. They used MAP as their evaluation metric. They found that their solution improved 50 percent over the BM25 baseline, and tuning parameters and weights resulted in large improvements. Just adjusting BM25's b parameter alone resulted in a 35 percent better MAP score.

3.2.5 Ranking based on resource importance

Bamba *et al.* [26] proposes a solution for ranking resources on the semantic web, inspired by the famous HITS [27] algorithm. On the web, HITS works by assigning two separate scores for each document; an authority score and a hub score. The intuition behind the algorithm is that, broadly, there are two kinds of pages on the web; one kind that is authoritative on the information it contains, and another kind that serves as directories for authoritative pages. Pages of the first kind are known as authorities, and the second kind are known as hubs. Authorities have many incoming links from hubs, and few outgoing links of their own. A page has a high authority score if many pages with high hub scores are pointing to it, and a page has a high hub score if it points to many pages with high authority scores.

Similarly, the algorithm by Bamba *et al.* assigns subjectivity and objectivity scores to each resource, that correspond to the authority and hub scores explained above. A resource with a high subjectivity score is the subject of many RDF triples, while a resource with a high objectivity score is the object of many RDF triples. However, one modification is that when the scores of a vertex are

calculated, the scores of the adjacent vertex is multiplied by the scores of the corresponding link, to ensure unimportant properties do not influence the scores of the resources.

Class weights are also influenced by the weights of its ancestors. If a class `c1` is a subclass of `c2`, then its importance is dependent of `c2`. If `c2` is a subclass of `c3`, then `c1` is also influenced by `c3`, but to a much lesser extent, since `c3`'s distance to `c1` is greater.

There are several other factors which determine the relevance to the user query. The solution uses something called inverse property frequency, which is similar to inverse document frequency in traditional information retrieval. If a property is very common in the dataset, then its corresponding edge is given less weight.

The authors state that a formal evaluation of the method is difficult, and was not performed at the time. However, they note that the time complexity was $O(n + e)R$, where R is the number of results, and that the overall running time is not too significant.

3.2.6 Noc-order approach

Graves, Adali, and Hendler [28] proposes a query-independent solution different from all the aforementioned ones. While the other algorithms are based on the PageRank centrality, the noc-order (NOde Centrality Ordering) approach is based on closeness centrality, which measure the importance of nodes based on their number of shortest paths to all other nodes. The node with the highest score has the shortest distances from all other nodes. The data graph is treated as undirected, since the direction of the semantics is irrelevant for the algorithm to work.

Closeness centrality is usually calculated with the formula seen in equation 3.6, where u is a node, n is the total number of nodes in the graph, and $d(u, v)$ is the distance of the shortest path between nodes u and v.

$$C(u) = \frac{n - 1}{\sum_{v=1}^{n=-1} d(u, v)} \tag{3.6}$$

Finding the shortest path between two nodes is done by using Dijkstra's algorithm [29], which is a popular path-finding algorithm. The distance may be simply measured in the number of edges between the two nodes, or it may be based on some weights on the edges. In this approach it is based on weights, where the weights represent the frequency of occurrence of each predicate: the more common a predicate is, the lower the relevance for the ranking.

In most cases, the noc-order approach will rank general and central concepts higher than the ones that are more specific. For example, while evaluating the

solution on a dataset of facts and information about the different countries in the world, large concepts and organisations such as climate change and WHO were among the top ten, while concepts such as animal husbandry and subsistence farming were amongst the bottom ten.

3.3 Other notable approaches

In addition to all the aforementioned approaches, there are other solutions that perform entity ranking in novel ways, but that are just out of scope for this thesis. We briefly mention some of them here.

Swoogle [30] is yet another system that adopts PageRank's way of ranking documents. While PageRank uses a random surfer model, the creators of Swoogle argues that this is not a an appropriate model for the semantic web. Instead they use a *rational random surfing model*, that considers the fact that links can be of different types. Swoogle only considers inter-ontology relations, for example imports and extensions, meaning it is not adequate for single-ontology datasets.

DBpediaRanker [31] is a system for performing RDF ranking in DBpedia. It computes the similarity between the query and the entities by querying external sources of information, like search engines such as Google and Bing.

Vercoustre *et al.* [32] have created an approach for ranking entities in Wikipedia, where one of the functions for calculating the score is based on the notions of ancestors, common ancestors, and shorter paths between Wikipedia categories. This is used to define a distance between a set of categories associated with a target entity page, and a set of categories associated with a set of example entities. The system maintains a global score by combining this scoring with a PageRank-like function, along with the score obtained by the Zettair[5] system.

Harth *et al.* [33] use what they call naming authority to help with entity ranking. A naming authority is the data source that has the power to define identifiers (URIs) of a particular structure. If a source a owns an identifier used by another source b, then it benefits the ranking of a. A naming authority graph is created from the dataset, which PageRank scores are then derived from.

3.4 Discussion

Almost all of the algorithms presented in this chapter are primarily based on some link analysis ranking model. This is of course expected, since they operate over linked data. Many of them have combined the link analysis approach with other classic information retrieval models in the pursuit of better ranking, such

[5] http://www.seg.rmit.edu.au/zettair/

as the probabilistic ranking model. A high-level overview of the algorithms is shown in table 3.1.

Another thing almost all of the algorithms have in common is that they are based on the same type of centrality algorithm. Centrality algorithms are used to measure the importance of a node in a network. They can for example determine their importance by their degree, which is the number of connections a node has; their closeness, which determines the importance by the number of nodes they can easily reach; betweenness, which measures the number of shortest paths that passes through a node; and finally PageRank, which estimates a node's importance based on the other nodes linking to it [34]. The algorithms presented in this paper, except for Noc-order, are all based on the PageRank approach. Intuitively, this makes sense for most applications, especially on the web, as we think of a web page to be popular if many other web pages are linking to it. On the other hand, we can imagine applications where some of the other approaches may be better suited. For example, degree centrality might be better suited for search in a social network.

All the algorithms are query-dependent, except for PageRank and Noc-order. Being query-dependent means the scores calculated for each document is reliant on the given query. Query-independent algorithms maintain a global rank, and the query merely acts as a filter.

Many of the algorithms can be used as-is, but some of them require some manual tweaking and tuning to be able to perform ranking efficiently. ObjectRank, for example, requires that the user or some domain-expert decides the amount of authority that is going to flow between nodes linked by some property. The vector-space model approach requires the classes in the ontology to be subclasses of a set of specific top-level classes. The annotation should also be done manually to get the best results, but the authors also provide a method for automatic annotation. The BM25F approach requires specific field data to be indexed, but not much more than that. Parameter-tuning is of course possible in algorithms like PageRank, where the damping factor can be adjusted if you desire more randomness in the random walk. A more educated guess of initial PageRank values can also help speed up convergence.

Although not critical when looking at the ranking algorithms in isolation, it is worth noting that some of them are presented as parts of a bigger system. As such, they might be optimized for different contexts. The system that implements the vector-space model approach, for example, retrieves documents by RDQL queries, which can fundamentally determine the type of documents retrieved. The ranking is then tested using these documents, which might have been different when used in a system that uses keyword-based queries. PageRank, ObjectRank, and ReConRank are all developed for keyword queries.

Lastly, one of the more important differences is what kind of unit the algo-

Table 3.1: High-level overview of the proposed solutions

Solution	**Model**	**Query-dependence**	**Query**	**Tuning**	**Unit of retrieval**
PageRank	Link analysis, probabilistic	Independent	Keyword	Optional	Entities
Vector-space approach	Vector-space	Dependent	RDQL	Required	Documents
Pérez-Agüera *et al.*	Probabilistic	Dependent	Keyword	Minimal indexing configuration required	Documents
Blanco *et al.*	Probabilistic	Dependent	Keyword	Optional	Entities
ObjectRank	Link analysis, probabilistic	Dependent	Keyword	Required	Entities
ReConRank	Link analysis, probabilistic	Dependent	Keyword	Optional	Entities
Resource importance	Link analysis	Dependent	RDQL	Optional	Entities
Noc-order	Link-analysis	Independent	Not specified	Optional	Entities

rithms work with. The vector-space approach, and the BM25F approach are all created to work with documents. PageRank was originally created to work with documents as well, but since it does not rely on the contents of the documents to perform the ranking, it can easily be adapted to work with any linked structure. BM25F and the vector-space approach rely on the textual content and structure of the document to work properly. The other proposed solutions are created specifically for ranking entities, often in the form of RDF resources.

Chapter 4

Model and Implementation

This chapter describes the architectural model and how the system is conceptualized and implemented. We implement and test the BM25F algorithms by Pérez-Agüera *et al.* and Blanco *et al.*, which are detailed in sections 3.1.2 and 3.2.4. We begin by describing Neo4j more closely in section 4.1, including details about the graph API that is heavily used in the implementation. In section 4.2 we describe the main components and architecture of the system, while section 4.3 and 4.4 describe how the data was imported and indexed. Finally, section 4.5 describes how the algorithms themselves are implemented.

The aim of this chapter is to show how our environment used for testing was set up. Information regarding the setup for the experiments is detailed in chapter 5.

4.1 Neo4j

Neo4j is an ACID-compliant database management system for storing, querying, and processing graph data. It comes packaged with its own querying language, called Cypher, which is created to be more efficient and intuitive when writing queries for graph data [35]. Our rationalization for using Neo4j as our base for writing and testing the ranking algorithms, is that it provides a lot of the functionality we need out of the box. As of version 3.5, Neo4j supports both full-text indexing of nodes and relationships, as well as full-text search [36]. Additionally, through the graph algorithms plugin, you can instantly run PageRank and other centrality algorithms on your data [34]. Neo4j also makes it simple to extend the query language by providing an API for writing custom plugins.

4.1.1 Neo4j Graph API

The Java graph API provided by Neo4j contains a collection of classes and methods that greatly simplifies the process of writing custom plugins for the database engine. For example, it contains methods for manipulating nodes, relationships, and paths, while still allowing for executing Cypher queries and handling the results [37].

4.2 Architecture

The architecture of the system is shown in figure 4.1. All the main components are contained within Neo4j, which provides a user interface where we can write and execute queries. From here we can call our custom procedures, which is in fact the ranking algorithms we have written using the graph API and loaded into the database engine as plugins. When these custom procedures are called, they consult the full-text index we created for our data, in order to find the nodes to return as our result, through the Graph API. These are then fetched from the underlying database, and our procedures rank them according to our specifications. These are then presented to the user through the graphical user interface.

The plugin containing the implemented algorithms are written so that you can call them from the Neo4j user interface, with the search terms as arguments. The classes that holds the algorithms are wrapped in a search class that calls the full text search Cypher query, and uses the results as input to the ranking methods. The main steps of each algorithm class in the plugin is the following.

1. Trim and escape query
2. Execute query against database
3. Return results of query to ranking function
4. Rank each entity according to specification
5. Return a ranked list of entities to user

The graph data is indexed using Neo4j's full-text indexer, which is powered by Lucene. The indexer indexes nodes and relationships by string properties. The strings are tokenized and split into terms, and stop words are removed [38].

Default full-text search over the indexes in Neo4J returns a ranked and sorted list of results. The rank, or score, is calculated using an implementation of TF-IDF. Search returns approximate results, as one would expect from an information retrieval system. For example, if you provide several keywords to the search

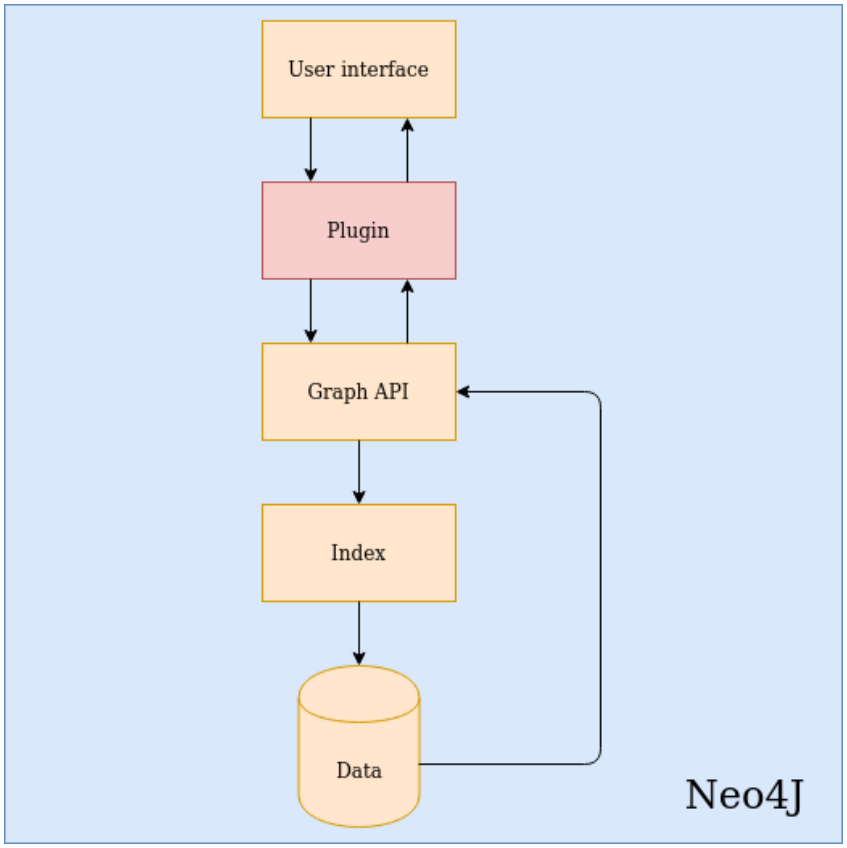

Figure 4.1: General architecture of system

function, it will also return nodes where only one or more keywords match. Since search is also powered by Lucene, you can construct more complex queries by for example defining which properties you want to search in, or by using AND and OR operators [38].

4.3 Importing

Importing the DBpedia dataset into Neo4j was accomplished using Neosemantics [39][40], which is a plugin that lets you easily use RDF in Neo4j. For importing RDF data, it first requires that an index be created on property *uri*. Then you can import individual files in a variety of formats, including Turtle, N-triples,

JSON, and XML. The DBpedia data was downloaded in the N-triples format, so we used the `semantics.importRDF` command specifying that format, as well as file location. Neosemantics then builds the Neo4j graph out of the triples in the file, converting datatype properties into node properties, and object properties into relationships. Neosemantics also shortens namespaces to make relationships more human-readable, while still having them be unique, ending up with prefixes such as `ns0` and `ns1`.

4.4 Indexing

The indexing of the DBpedia dataset is done according to the experimental setup of [41]. The set contains over 40,000 different property keys, and we identified and indexed the 1000 most common properties. These properties include popular fields like "title", "name", and "label". The indexing was carried out with the help of the built-in procedure `db.index.fulltext.createNodeIndex` [42]. The full-text indexes are powered by Lucene, and allows for partial matching, in contrast to regular Neo4j indexes. The procedure also allows you to choose what kind of analyzer you wish to use. We use the standard English analyzer, for stemming and tokenizing English words, because we are working with the English version of DBpedia.

4.5 Implementation of Algorithms

This section provides details about the implementation of the ranking algorithms. All algorithms were implemented as plugins, written in Java, using the Neo4j graph API. The algorithms were primarily chosen for being interesting approaches to entity-ranking, and by building upon the same base method which is known for generally performing well in normal data. The algorithms expand upon BM25F in different ways, which makes for interesting grounds for comparison.

4.5.1 Pérez-Agüera *et al.*

Although the BM25F approach is originally meant to rank documents, we have successfully adapted it for use in our graph system. In the paper, the authors mention that as part of their system, they indexed the underlying data differently in order to better accommodate the solution. However, we feel that changing the indexing for this algorithm means that it is not evenly compared to the others, since the retrieved nodes it ranks may be different. For the sake of a fair comparison, we decided to not implement this indexing, and instead let all the algorithms work on the same set of retrieved nodes.

Our implementation starts off by calculating the average lengths of each property type. The graph API does not contain any methods for getting all properties in the graph, so we wrote a Cypher query that fetches all property keys, that is executed as a transaction from the code. However, calculating the average property length for all forty thousand property types proved to be too time-consuming. We chose to instead calculate an approximation for each type by fetching the first 100 nodes that contain the given type and using those values to get an average. This reduced the runtime significantly, but it still took over 33 hours to calculate for all fields on consumer-grade hardware. Fortunately, we only had to run this calculation once, since they did not change between queries. The average field lengths are later used to calculate the normalized term frequencies for each field.

The BM25F score for each node is calculated according to equations 3.1 and 3.2. The weighting is done in the term frequency method, where the term frequency is implemented as the sum of the normalized term frequency of each field of a node, multiplied by the given weight of the field type.

4.5.2 Blanco *et al.*

The implementation of this algorithm was done according to the approach and formulas outlined in section 3.2.4. The inverse document frequency was calculated differently than from the approach of Pérez-Agüera *et al.*, using a probabilistic approach. We could reuse the average field lengths we calculated for Pérez-Agüera *et al.*, reducing run-time.

In calculating B_s, we used the size of the given node as the length of the field we calculated the normalization factor for, as the authors did. We defined the size of a node as the sum of the length of all its fields (ie. its literals).

The final step of the algorithm involves multiplying the BM25F score with a query-independent feature. We decided to use PageRank for several reasons. First of all, it is a heavily-researched and popular feature, that has proven to be effective in commercial search engines. Second of all, it is intuitive, especially for a linked dataset such as DBpedia. Lastly, it is easily available in Neo4J, simply by installing the Graph Algorithms plugin. We ran the PageRank algorithm over the data and wrote the results to the database, inserting it as a field in each node. This way we could easily use it for the final step of the BM25F algorithm.

We decided not to do any special indexing for this algorithm either. We mainly wanted to focus on the strengths of the actual algorithms, and since they are both based on BM25, we decided it was optimal that the indexing remained unchanged.

4.5.3 PageRank

PageRank is already adapted and implemented for Neo4J through the existing graph algorithms plugin. In this implementation of PageRank, it runs a set number of iterations, instead of stopping when values converge, using a damping factor of 0.85 [43].

Chapter 5

Experiments

In this chapter, we present the experimental plan, detailing the setup and software used to conduct the experiments.

The goal of the experiments is primarily to be able to answer our research questions, regarding how our chosen ranking algorithms stack up against standard ranking methods, and how we can fine-tune them to perform better with our chosen dataset.

5.1 Plan

To test the algorithms with regard to our research questions, several sets of experiments were needed. Both BM25F methods required manual property weighting and parameter tuning. For property weighting, the following three approaches were planned.

1. No weighting

We wanted to establish a baseline, to see how the algorithms performed with no weighting and default parameter values.

2. Name and title weighting

The most simple kind of weighting, where we only considered two property types. These are very common, appearing in most entities. We tested different weights for these two properties, starting at 2, which essentially means it was twice as important as all other properties.

3. Blanco *et al.* weighting

A weighting scheme inspired by the weighting done in Blanco *et al.* It involves weighting many more properties, that the authors have considered important. These include more properties similar to `name`, as well as other popular properties such as `label`. Here we also tested different weights, starting at 2.

Regarding parameter tuning, we tested several values for both k and b_s. Because of the considerable time needed to run a single experiment, we were not able to test every combination of the parameters with the different values for each weighting scheme. Instead, we tested each part separately, and combined the best approaches. Our approach was optimize for the SemSearch ES queries, since they are shorter keyword queries that one would expect to see in a general web search engine.

5.2 Setup

This section details all the weights, parameters, software, data, and metrics needed to replicate the experiments. The focus of the experiments is not on the computational performance, but rather the end results. In other words, we want to optimize the performance for the metrics used.

5.2.1 Weights

For the experiments testing the different weights, we tested for nine values each (2 to 9). We believe this is more than sufficient, as the weighting acts as a multiplier. In other words, a weight of 3 for a property indicates that it is three times as important as the other properties. A weight of 1 is the same as no weighting.

For the name and title weighting, we simply weighted the following two properties.

- `title`, `name`

For the weighting scheme inspired by Blanco *et al.*, we weighted the following set of properties.

- `label`, `title`, `name`, `nickname`, `fullname`, `othername`, `birthname`, `surname`, `lastname`, `firstname`, `description`

5.2.2 Parameters

For the parameters k and b_s, the default values are at 0.5. Our plan was to test all values between 0.1 and 0.9, with increments of 0.1. These tests were done separately, which means we tested the range of different values for one parameter

while the other was at its default. The reason for this, as stated earlier, is that it is computationally infeasible to test all combinations of weights and parameters.

5.2.3 Data

Our test data is the DBpedia dataset, version 3.7 (described in section 2.4.1). It is a very practical choice for testing for several reasons. It contains information about many different entities across many different topics. People, places, things, and events are among the entities described in the data. Secondly, it is large enough to be statistically useful, with over four and a half million entities. Thirdly, the data is very popular, so finding ways to generally improve ranking on it can be beneficial to many different applications and researchers. Finally, we have a set of queries and query relevances created by Balog and Neumayer [41], which means we can conveniently test our algorithms using a ground truth, without needing to conduct more time-consuming user studies.

The entity search test collection by Balog and Neumayer provides a set of queries and corresponding relevance rankings, which can be used to test new information retrieval methods. The queries range from simple keyword queries, to whole questions. The set contains 485 queries, taken from several different sources. These include:

- INEX-LD: The INEX 2012 Linked Data Track

Queries from this source are mostly keyword queries, ranging from very general (*"indian food"*) to longer, more specific answer-seeking queries (*"John Turturro 1991 Coen Brothers film"*).

- INEX-XER: The INEX 2009 Entity Ranking Track

Queries from this source are seeking sets of entities. Examples include *"films shot in Venice"* and *"Nobel Prize in Literature winners who were also poets"*.

- QALD-2: The Question Answering over Linked Data Challenge

Contains natural language queries posed as questions, or sentences starting with "give me all...". For example *"What is the second highest mountain on Earth?"* and *"Give me all people that were born in Vienna and died in Berlin"*.

- SemSearch ES: The entity search task of the 2010 and 2011 Semantic Search Challenge

The queries here are mostly short keyword queries, usually just a couple of terms. Examples include *"austin powers"* and *"sedona hiking trails"*. Queries like these are somewhat general and can have many relevant results.

- SemSearch LS: The list search task of the 2011 Semantic Search Challenge

Similar to INEX-XER, the queries from SemSearch LS targets sets of entities. A few of them are stated as questions (*"what books did paul of tarsus write?"*), but most of them are keyword-style queries (*"Apollo astronauts who walked on the Moon"*).

- TREC Entity: The TREC 2009 Entity Track

The smallest set of queries, with only 20 entries. The queries here mostly focuses on relationships between entities, like *"Carriers that Blackberry makes phones for"*. All queries here also seek out sets of results, instead of single answers.

The relevance rankings for most of the sources are binary, either a resource is relevant to a given query, or it is not. The data from the Semantic Search Challenges are the exceptions, as they include one additional level of relevance: 1 for 'fair', and 2 for 'excellent'.

5.2.4 Software

The following is the list of software used to implement the algorithms and run the experiments.

- Neo4j Desktop 1.2.4
- Neo4j 3.5.15
- Neosemantics 3.5.0.4
- Graph Algorithms 3.5.4
- Java 8
- Graph API 3.5

5.2.5 Metrics

The metrics we are using to compare the different algorithms are MAP, P@n, R-precision, and DCG, as described in section 2.3. These metrics were chosen for being highly popular, and being generally simple to reason about. DCG is only really applicable when you have a graded scale of relevance. In other words, when the relevance is non-binary. Such is the case with the queries from SemSearch ES and SemSearch LS, where queries are either not relevant (0), relevant (1), or very relevant (2).

Chapter 6

Results and Discussion

In this chapter, we present the results of the experiments. We begin with an introduction to how the algorithms are evaluated in section 6.1. In section 6.2 we present the results of trying different weighting schemes, and different values for the weights. In section 6.3 we present the results of tuning the parameters k and b_s. In section 6.4 we present our final results. Finally, in section 6.5, we provide an analysis of our findings, in addition to the short analyses in the preceding sections.

6.1 Evaluation

The algorithms were evaluated over the data, using the set of queries and query relevances detailed in section 2.4.1. We used four different metrics for measuring the performance of the algorithms: MAP, P@N, R-precision, and DCG where applicable. We used 40 as a cut-off point for MAP. For each query in the text file supplied by Balog and Neumayer [41], we executed the query against the Neo4j database, and ran each of the ranking algorithms over the results. We then calculated the metrics based on the ranked results and the contents of the query relevances file. We did two levels of P@N: P@5 and P@10. This was because real users are unlikely to look past the first page of results in a standard information retrieval system, and therefore we feel that these levels are most relevant. We calculated the average for all these metrics for each source of queries (INEX, semsearch, TREC, etc). The results of one run, for one algorithm, could then look like what is shown in table 6.1.

Plotting the results of one run of Pérez-Agüera *et al.*'s BM25F yields the results shown in figure 6.1. Similar results are obtained by running Blanco *et al.*'s algorithm. The graph shows that `semsearch_es` has the best results for

Table 6.1: Example results for one algorithm for one run of the evaluation over the DBpedia dataset (values rounded to three decimal places for readability)

Source	MAP	P@5	P@10	R-precision	DCG
inex_ld	0.217	0.134	0.089	0.054	
inex_xer	0.129	0.084	0.060	0.033	
qald2	0.048	0.030	0.024	0.015	
semsearch_es	0.382	0.267	0.176	0.202	2.232
semsearch_ls	0.109	0.051	0.044	0.030	0.369
trec_entity	0.021	0.012	0.006	0.008	

each metric. We believe the main reason for this is because the queries from that source are all simple keyword queries, consisting of about two or three terms per query. This is in contrast to the other sources, where many of the queries consists of eight or more terms. Additionally, many of those other queries are framed as questions or commands, which the query relevance file takes into consideration. Since the algorithms we are testing are only trying to answer simple keyword queries, we lean towards optimizing them more for sources like `semsearch_es` and `inex_ld`.

6.2 Property weighting

Deciding how to weight the properties is a considerable task. It has to be done manually for each dataset, and this particular dataset contains over 40,000 unique property types. We experimented with a few different weighting schemes, also including no weighting.

6.2.1 No weighting

In table 6.2 we show the results of running both algorithms with no weighting, and using default values for parameters. The best values for each columns has been highlighted. Unsurprisingly, all the best results are achieved with the `semsearch_es` queries, since they consist of short keyword queries. The algorithm by Pérez-Agüera *et al.* outperforms Blanco *et al.* on almost all points, except for P@5 for `inex_ld`, where Blanco *et al.* performs 109 percent better. On the `semsearch_es` dataset, Pérez-Agüera *et al.* outperforms Blanco *et al.* by an average of 375 percent. A side-by-side comparison of the just these results can be seen in figure 6.2.

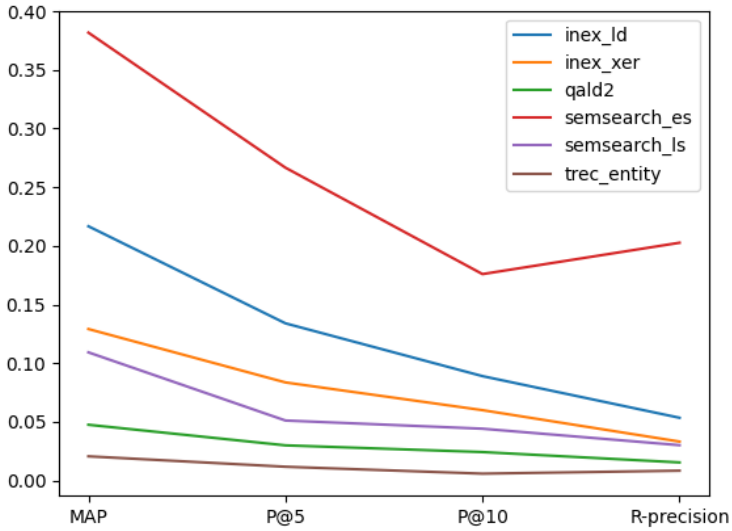

Figure 6.1: Results for running Pérez-Agüera *et al.*'s BM25F with default parameters and no term weighting

The results of running the algorithms with no weighting and various values for k and b_s can be seen in section 6.3.

6.2.2 Name and title weighting

A simple approach to property weighting in traditional web documents would be to weigh the title higher than other fields, since it often carries more precise information about the content than other fields. An entity often has a title field if it describes a thing, or a name field if it describes a living being. Our assumption is then that just weighting these two fields higher than everything else should cause the algorithms to perform better. Figure 6.3 shows the results of adjusting the weights for these two fields, for the `semsearch_es` queries (weight of 1 is equivalent to no weighting). Disappointingly, the impact is minimal. For Pérez-Agüera *et al.*'s algorithm, a weight of 2 slightly increases performance, but the effect is too insignificant to be of much real practical value. Increasing the weight shows no real improvement either, for *MAP*, *P@10*, and *R-precision* it flattens out, while slightly decreasing again for *P@5*. Blanco *et al.*'s shows no significant

Table 6.2: Results of running both algorithms with no weighting, using default parameter values (Best values for each column highlighted in bold)

Source	Pérez-Agüera *et al.*					Blanco *et al.*				
	MAP	P@5	P@10	R-precision	DCG	MAP	P@5	P@10	R-precision	DCG
inex_ld	0.217	0.134	0.089	0.053		0.039	0.280	0.021	0.007	
inex_xer	0.129	0.084	0.060	0.033		0.039	0.033	0.029	0.022	
qald2	0.048	0.030	0.024	0.015		0.026	0.007	0.006	0.013	
semsearch_es	**0.382**	**0.267**	**0.176**	**0.203**	**2.233**	**0.105**	**0.057**	**0.040**	**0.040**	**0.376**
semsearch_ls	0.109	0.051	0.044	0.030	0.369	0.057	0.014	0.014	0.015	0.109
trec_entity	0.021	0.012	0.006	0.008		0.010	0.012	0.006	0.006	

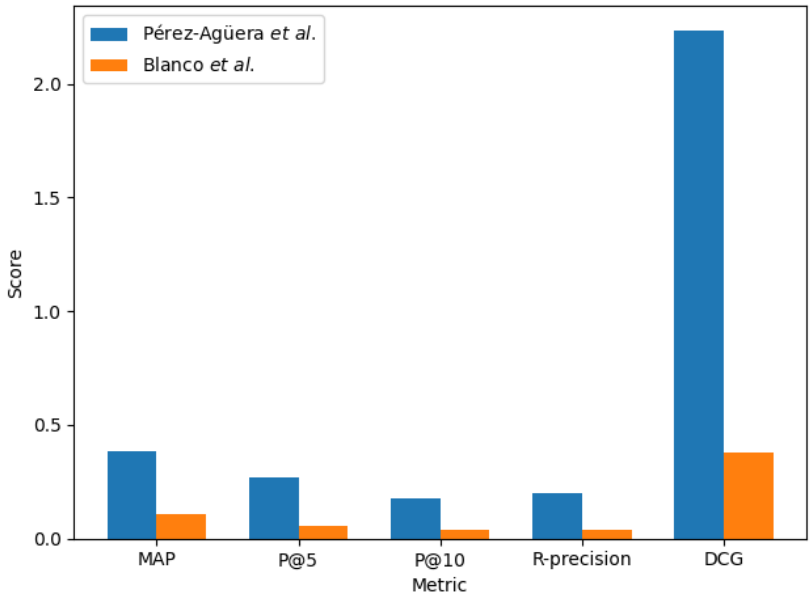

Figure 6.2: Comparison of the two algorithms over the `semsearch_es` queries, using no weighting and default parameter values

results either, the difference in performance between the different weights is so small. A reason for these results might simply be that the query terms seldom match the contents of the title and name fields, since they are two fields out of 40,000. If the query terms match other fields like description or labels instead, it would explain why the results seem generally unaffected.

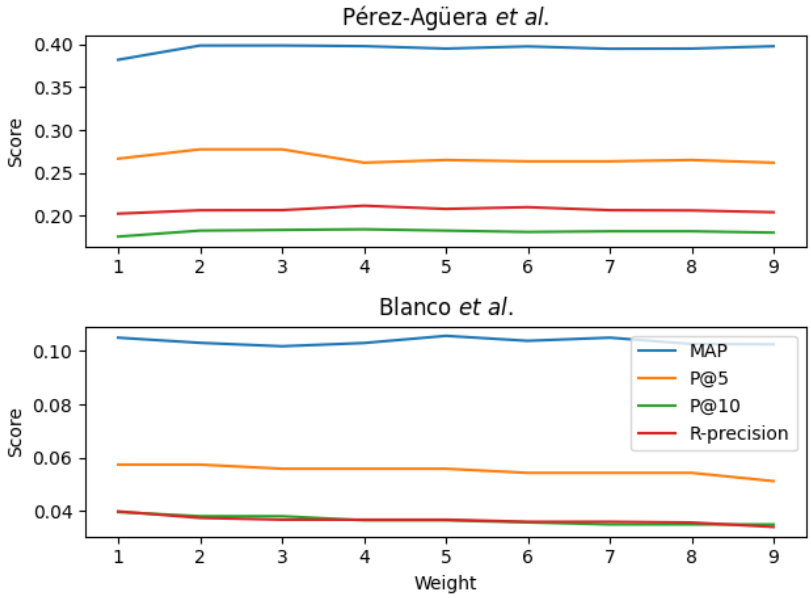

Figure 6.3: Results of adjusting weight of properties *title* and *name* for the `semsearch_es` queries

6.2.3 Blanco *et al.* weighting

Blanco *et al.* [24] identify properties they deem important. These include properties like `label`, `title`, `family-name`, and `description`. For one run we used the DBpedia equivalents of these, in addition to closely related properties like `nickname` and `officialname`.The intuition is that these fields carry more meaning than fields like `language, imagesize, birthdate`, etc. They also cover a broader spectrum of identifiers, so that query terms are more likely to appear in our weighted fields. The `semsearch_es` results are shown in figure 6.4.

From the figure, we see that very little has changed from figure 6.3, where we only weighted the fields *title* and *name*. We have placed the same values in a table to make it easier to see the differences, shown in table 6.3. The best values of each column is highlighted. In the top subtable, we show the results for Pérez-Agüera *et al.*'s algorithm. We see that for name and title weighting, it peaks at around a weight of 3, while it peaks at around 2 for the other weighting scheme. Comparing the best values for each side, we see that title and name weighting actually performs slightly better. For Blanco *et al.*'s algorithm, we see that there

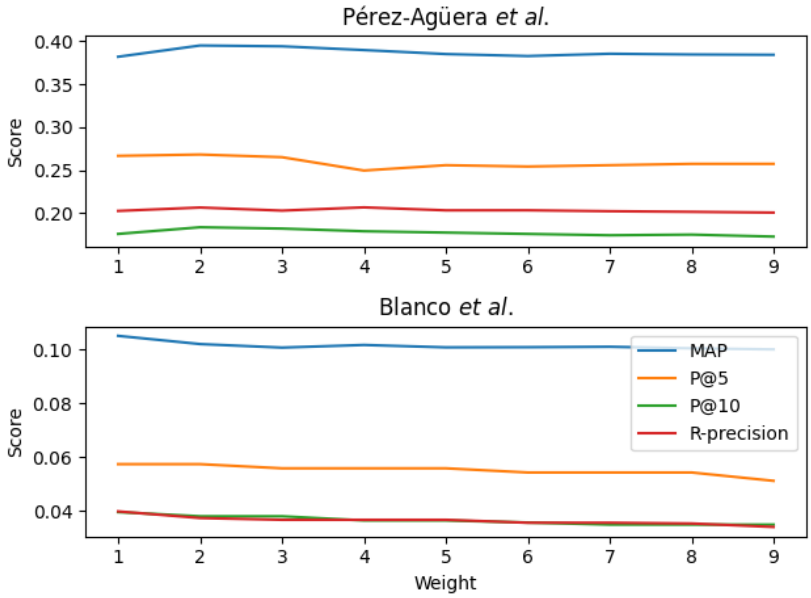

Figure 6.4: Results of adjusting weight of several different properties for the `semsearch_es` queries

is virtually no difference. For both weighting schemes, it actually performs best with no weighting at all. It is possible that the PageRank part of the algorithm almost overrides any weighting we try to incorporate. As for Pérez-Agüera *et al.*, it is possible that assigning the same weight to many fields makes it less accurate. Just weighting two fields may not cover a lot of bases, but is more precise when a query term matches.

6.3 Tuning parameters

The next step was to tune the parameters to our data. We first tried to find the optimal values for k, which is present in both algorithms. The default value suggested by both papers was 0.5. We tested values between 0.1 and 0.9, with increment 0.1. Figures 6.5 and 6.6 shows the results on the `semsearch_es` queries for Pérez-Agüera *et al.*'s BM25F algorithm and Blanco *et al.*'s BM25F algorithm, respectively. We see from the figures that both approaches prefer higher values for these particular keyword queries. Blanco *et al.*'s approach in particular perform

Table 6.3: Results of different weights for name and title (left) versus weights of properties laid out in section 6.2.3 (right) over the `semsearch_es` queries for both algorithms (best values for each column emphasised in bold)

Weight	Pérez-Agüera *et al.*									
	Title and name weighting					Blanco *et al.* weighting				
	MAP	P@5	P@10	R-precision	DCG	MAP	P@5	P@10	R-precision	DCG
1	0.382	0.267	0.176	0.203	2.233	0.382	0.267	**0.186**	0.203	2.233
2	**0.398**	**0.278**	0.183	0.207	2.377	**0.395**	**0.268**	0.184	**0.207**	**2.353**
3	**0.398**	**0.278**	**0.184**	0.207	**2.406**	0.394	0.265	0.182	0.203	2.342
4	**0.398**	0.262	**0.184**	**0.212**	2.376	0.390	0.250	0.179	**0.207**	2.284
5	0.395	0.265	0.183	0.208	2.349	0.385	0.256	0.178	0.203	2.257
6	0.397	0.264	0.181	0.210	2.324	0.383	0.254	0.176	0.203	2.223
7	0.395	0.264	0.182	0.207	2.322	0.385	0.256	0.174	0.203	2.242
8	0.395	0.265	0.182	0.206	2.305	0.384	0.257	0.175	0.202	2.227
9	**0.398**	0.262	0.181	0.204	2.303	0.384	0.257	0.172	0.201	2.213
Weight	Blanco *et al.*									
	Title and name weighting					Blanco *et al.* weighting				
	MAP	P@5	P@10	R-precision	DCG	MAP	P@5	P@10	R-precision	DCG
1	0.105	**0.057**	**0.040**	**0.040**	**0.040**	**0.105**	**0.057**	**0.040**	**0.040**	**0.040**
2	0.103	**0.057**	0.038	0.037	0.037	0.102	**0.057**	0.038	0.037	0.037
3	0.102	0.056	0.038	0.037	0.037	0.101	0.056	0.038	0.037	0.037
4	0.103	0.056	0.036	0.037	0.037	0.102	0.056	0.036	0.037	0.037
5	**0.106**	0.056	0.036	0.037	0.037	0.101	0.056	0.036	0.037	0.037
6	0.104	0.054	0.036	0.036	0.036	0.101	0.054	0.036	0.036	0.036
7	0.105	0.054	0.035	0.036	0.036	0.101	0.054	0.035	0.036	0.036
8	0.103	0.054	0.035	0.036	0.036	0.100	0.054	0.035	0.035	0.035
9	0.103	0.051	0.035	0.034	0.034	0.100	0.051	0.035	0.034	0.034

best at the highest k value 0.9. However, the effects are minimal. The difference between the highest and lowest MAP score is only about 0.020.

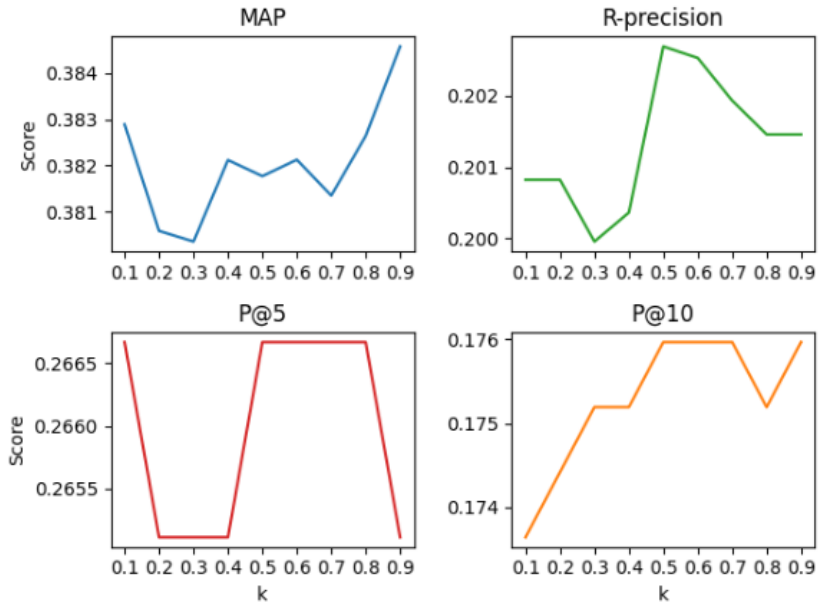

Figure 6.5: Results of adjusting k for Pérez-Agüera *et al.*'s BM25F

In addition to k, Blanco *et al.*'s approach also has a tunable parameter b_s, which controls the amount of normalization. Here we also experimented with the same range of values as we did with k, and again we focus on the `semsearch_es` dataset. The results are shown in figure 6.7. Here we also see that higher values are preferred, and the same trend can be seen for most of the other query sources.

6.4 Final results

With the optimal weighting and parameters discovered in previous sections, we obtain our final results by combining them. The values are shown in table 6.4. Both algorithms achieved their best results with a high k and b_s. The optimal weighting for Pérez-Agüera *et al.* was a title and name weight of 3, while Blanco *et al.* performed best with no weighting at all. The results of these runs, for all query sets, are shown in table 6.5. We also included Lucene and PageRank scores, as a baseline. The Lucene scores are an adaptation of the standard TF-IDF, as

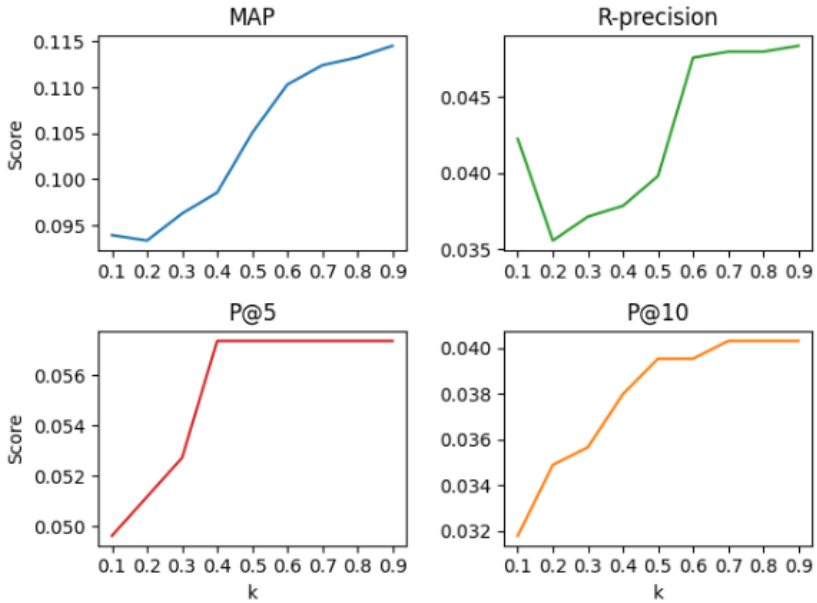

Figure 6.6: Results of adjusting k for Blanco *et al.*'s BM25F

described in section 4.2. The PageRank scores are provided by Neo4j, which ran with default parameters.

From the table, we see that Pérez-Agüera *et al.* surpasses all other ranking algorithms on almost every metric for every query set. The exception is for the `trec_entity` dataset. This set consists of long (6-10 terms) queries that focuses more on specific relationships between entities, like *'Carriers that Blackberry makes phones for'*. This set is also the smallest, at 20 queries.

An interesting find is the general poor performance of PageRank. It has been used with great success in commercial web search engines, but is struggling to keep up with the other algorithms for this particular dataset.

6.5 Analysis

The query set that consistently achieves the best scores are the `semsearch_es` queries. These are short keyword queries that refer to a particular entity. Examples of queries in this set include *"american embassy nairobi"* and *"martin luther king"*. Examples of relevant results for the latter include resources *Mar-*

Table 6.4: Parameters and weighting used for final results

Pérez-Agüera *et al.*	**Blanco *et al.***
k: 0.9	k: 0.9
Title/name weight: 3	No weighting
	b_s: 0.9

Table 6.5: Final results of running the algorithms with the parameters and weighting from table 6.4, compared to Lucene's TF-IDF and PageRank (Best scores for each row highlighted in bold)

Source	**metric**	**Pérez-Agüera *et al.***	**Blanco *et al.***	**Lucene**	**PageRank**
`inex_ld`	MAP	**0.223**	0.044	0.145	0.030
	P@5	**0.130**	0.032	0.074	0.020
	P@10	**0.092**	0.022	0.049	0.014
	R-p	**0.060**	0.010	0.025	0.004
`inex_xer`	MAP	**0.115**	0.040	0.053	0.042
	P@5	**0.076**	0.033	0.036	0.033
	P@10	**0.055**	0.300	0.049	0.029
	R-p	0.025	0.022	**0.030**	0.020
`qald2`	MAP	0.045	0.030	**0.052**	0.021
	P@5	**0.031**	0.009	0.026	0.006
	P@10	**0.024**	0.006	0.020	0.006
	R-p	**0.013**	**0.013**	**0.013**	0.005
`semsearch_es`	MAP	**0.402**	0.119	0.305	0.073
	P@5	**0.278**	0.056	0.163	0.037
	P@10	**0.184**	0.041	0.111	0.028
	R-p	**0.213**	0.048	0.146	0.030
	DCG	**2.395**	0.427	1.385	0.223
`semsearch_ls`	MAP	**0.127**	0.070	0.076	0.048
	P@5	**0.060**	0.014	**0.060**	0.009
	P@10	**0.051**	0.019	0.037	0.012
	R-p	**0.035**	0.017	0.034	0.008
	DCG	**0.396**	0.110	0.345	0.095
`trec_entity`	MAP	0.018	0.008	**0.021**	0.007
	P@5	0.000	**0.012**	**0.012**	0.000
	P@10	**0.012**	0.006	**0.012**	0.006
	R-p	0.000	0.006	**0.009**	0.006

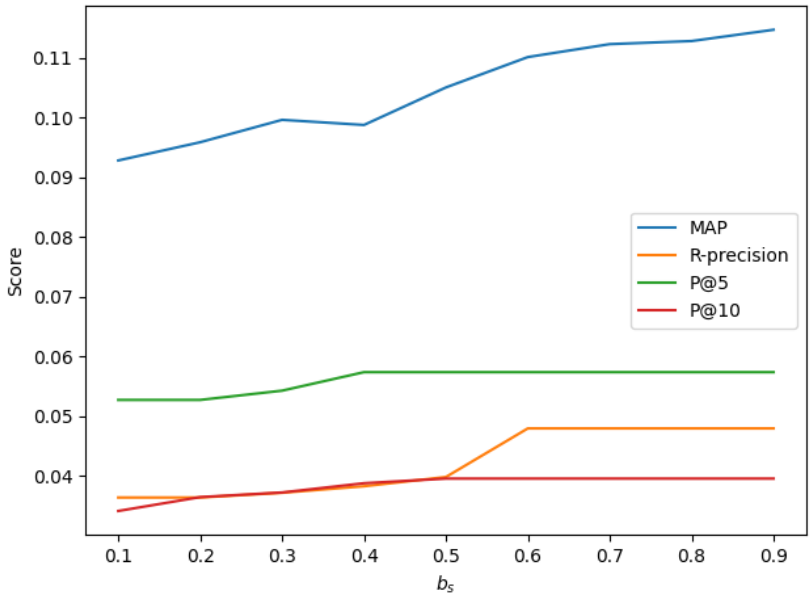

Figure 6.7: Results of adjusting b_s for Blanco *et al.*'s BM25F

tin_Luther_King_III, *Martin_Luther_King,_Sr*, *Martin_Luther_King_High_School*, as well as other institutions and landmarks named after Martin Luther King. Examining the results of PageRank for this query, reveals why it performs poorly. The top results include the city of Seattle at first place with the highest PageRank score, followed by George VI of the UK. In fact, there is not a single relevant entity in the top 50 results. The reason is that Seattle and the other entries are highly connected entities, with lots of other entities referencing them. Very few of them contain the keyword *"luther"*, but many have both *"martin"* and *"king"* since they are very common keywords.

Lucene performs much better, where the first four results are deemed relevant. They include all of the keywords in the query, and they are repeated many times in the resource pages. It also seems that the keyword *"luther"* carries more weight, since it occurs in less entities than the two other keywords, which helps lift the relevant results to the top.

Blanco *et al.* performs a bit worse. Examining the results, we see the influence from the PageRank scores, which is included in the calculation. It has entities like Seattle and George VI at the top as well, but it also has Martin Luther King Jr. in the top 10. This may be a result of also incorporating term frequency and

inverse document frequency in the algorithm. It brings up more entities with the less common keyword, *"luther"*, which exists primarily in relevant results.

Pérez-Agüera *et al.* averages the best results overall. The question is, even without weighting, why does this method outperform the others? If Blanco *et al.*'s detriment is the incorporated PageRank score, it still also outperforms Lucene. One reason may be that it includes field length normalization in addition to term frequency and inverse document frequency. BM25 is also widely accepted as state-of-the-art in ranking methods, and is known for producing good results for general collections [6].

As we have briefly mentioned earlier, the `semsearch_es` queries consistently achieve higher scores, followed by `inex_ld`. We believe the main reason for this is because the queries are short, often general, keyword queries. This is in contrast to some of the other query sets where the queries are posed as questions, or seeking specific connections between entities. None of the tested solutions incorporate any natural language processing, they are simply looking at the individual query terms that make up the query. It is still interesting to compare results from these queries, since we can see how the algorithms compare in other settings. A user of a web search engine might form their queries like questions or commands anyway, so we want to achieve better results for all queries.

Chapter 7

Conclusion

In this thesis, we have explored different solutions to the problem of ranking entities in an information retrieval system. We have implemented two of them, both based on the highly popular BM25F method, which is widely considered state of the art. We implemented them as plugins in the graph database Neo4j, making use of their graph API. We tested the algorithms on the DBpedia dataset, which is a semantically connected linked open dataset, using data from Wikipedia. We compared them to two other popular ranking methods, Lucene's TF-IDF and PageRank. We found that the BM25F approach of Pérez-Agüera *et al.* generally outperformed all the other methods. PageRank generally scored the lowest. It promotes the most "important" entities, which in this case are not always the most relevant. Lucene's performance was adequate, but on the average performed slightly worse than Pérez-Agüera *et al.* However, it is conceptually simple while still being effective. Blanco *et al.* generally performed poorly. One of the differences from Pérez-Agüera *et al.* is that they incorporate the PageRank scores of the entities when calculating the rank. Since PageRank performed poorly in this dataset, it may be that it had a significant negative impact on the final scores.

Our first research question in section 1.2 was if algorithms based on BM25F, made for use in linked data, could outperform standard ranking solutions like TF-IDF and PageRank. Our findings show that Blanco *et al.* outperformed PageRank, and Pérez-Agüera *et al.* outperformed both of them, as well as Lucene's TF-IDF. It also answers the second research question, which was which of the two BM25F solutions was performed better for this type of data.

Our final research question was how we could tune the parameters of the two BM25F algorithms to obtain the optimal performance. We found that, in general, the higher the value of k and b_s, the better the algorithms performed.

The final results were obtained by using the value 0.9 for both k and b_s. However, while these choices of parameters increased performance, the improvement was not drastic. For example, the difference in average MAP score for Blanco *et al.* was about 0.02 between the best and the worst results. We also found that Pérez-Agüera *et al.* performed best with weighting just the properties *title* and *name* 3, while Blanco *et al.* performed best with no weighting at all.

Bibliography

[1] A. G. Standal, "Ranking technology in information retrieval over semantic web data", 2019.

[2] A. Singhal. (May 16, 2012). Introducing the knowledge graph: Things, not strings, [Online]. Available: `https://googleblog.blogspot.com/2012/05/introducing-knowledge-graph-things-not.html`.

[3] K. Balog, *Entity-Oriented Search*, 1st. Springer Publishing Company, Incorporated, 2018, ISBN: 3319939335, 9783319939339.

[4] L. Yu, *A developer's guide to the semantic Web*, 2nd. Springer Science & Business Media, 2014, ISBN: 3662437953.

[5] T. Berners-Lee, R. Fielding, and L. Masinter. (2005). Uniform resource identifier (uri): Generic syntax, [Online]. Available: `https://tools.ietf.org/html/rfc3986` (visited on 04/10/2019).

[6] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval: The Concepts and Technology Behind Search*, 2nd. USA: Addison-Wesley Publishing Company, 2008, ISBN: 9780321416919.

[7] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web", May 17, 2001.

[8] TREC. (Apr. 9, 2019). Overview, [Online]. Available: `https://trec.nist.gov/overview.html`.

[9] C. Macdonald, I. Soboroff, and B. Gamari, *Trec-eval*, `https://github.com/usnistgov/trec_eval`, 2019. (visited on 10/29/2019).

[10] G. Kazai, "Initiative for the evaluation of xml retrieval", in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 1531–1537, ISBN: 978-0-387-39940-9.

[11] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. Van Kleef, S. Auer, *et al.*, "Dbpedia - a large-scale, multilingual knowledge base extracted from wikipedia", *Semantic Web*, vol. 6, no. 2, 2015.

[12] DBpedia. (2019). About dbpedia, [Online]. Available: `https://wiki.dbpedia.org/about`.

[13] D. Vrandečić and M. Krötzsch, "Wikidata: A free collaborative knowledge base", *Communications of the ACM*, vol. 57, pp. 78–85, 2014. [Online]. Available: `http://cacm.acm.org/magazines/2014/10/178785-wikidata/fulltext`.

[14] Wikidata. (). Statistics, [Online]. Available: `https://www.wikidata.org/wiki/Special:Statistics`.

[15] C. Shepherd. (2017). Springer nature scigraph: Pioneering semantic platform with linked open data, [Online]. Available: `https://www.digital-science.com/blog/news/springer-nature-scigraph-pioneering-semantic-platform-linked-open-data/` (visited on 03/06/2020).

[16] D. Vallet, M. Fernández, and P. Castells, "An ontology-based information retrieval model", in *European Semantic Web Conference*, Springer, 2005.

[17] J. R. Pérez-Agüera, J. Arroyo, J. Greenberg, J. P. Iglesias, and V. Fresno, "Using bm25f for semantic search", in *Proceedings of the 3rd international semantic search workshop*, ACM, 2010.

[18] H. Zaragoza, N. Craswell, M. J. Taylor, S. Saria, and S. E. Robertson, "Microsoft cambridge at trec-13: Web and hard tracks", in *TREC*, vol. 4, 2004.

[19] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web", Jan. 29, 1998.

[20] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine", *Computer networks and ISDN systems*, vol. 30, no. 1-7, pp. 107–117, 1998.

[21] A. Balmin, V. Hristidis, and Y. Papakonstantinou, "Objectrank: Authority-based keyword search in databases",

[22] DBLP. (Oct. 24, 2019). Record in dblp, [Online]. Available: `https://dblp.uni-trier.de/statistics/recordsindblp`.

[23] A. Hogan, S. Decker, and A. Harth, "Reconrank: A scalable ranking method for semantic web data with context", 2006.

[24] R. Blanco, P. Mika, and S. Vigna, "Effective and efficient entity search in rdf data", in *International Semantic Web Conference*, Springer, 2011, pp. 83–97.

[25] T. L. (2016). Billion triples challenge, [Online]. Available: `https://github.com/timrdf/DataFAQs/wiki/Billion-Triples-Challenge`.

[26] B. Bamba and S. Mukherjea, "Utilizing resource importance for ranking semantic web query results", 2004.

[27] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment", 1999.

[28] A. Graves, S. Adali, and J. Hendler, "A method to rank nodes in an rdf graph", in *Proceedings of the 2007 International Conference on Posters and Demonstrations-Volume 401*, CEUR-WS. org, 2008, pp. 84–85.

[29] E. W. Dijkstra, "A note on two problems in connexion with graphs", *Numer. Math.*, pp. 269–271, 1959. DOI: 10.1007/BF01386390. [Online]. Available: http://dx.doi.org/10.1007/BF01386390.

[30] L. Ding, T. Finin, A. Joshi, R. Pan, R. S. Cost, Y. Peng, P. Reddivari, V. Doshi, and J. Sachs, "Swoogle: A search and metadata engine for the semantic web", in *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, ser. CIKM '04, Washington, D.C., USA: ACM, 2004, pp. 652–659, ISBN: 1-58113-874-1. DOI: 10.1145/1031171.1031289. [Online]. Available: http://doi.acm.org/10.1145/1031171.1031289.

[31] R. Mirizzi, A. Ragone, T. Di Noia, and E. Di Sciascio, "Ranking the linked data: The case of dbpedia", in *International Conference on Web Engineering*, Springer, 2010, pp. 337–354.

[32] A.-M. Vercoustre, J. A. Thom, and J. Pehcevski, "Entity ranking in wikipedia", in *Proceedings of the 2008 ACM Symposium on Applied Computing*, ser. SAC '08, Fortaleza, Ceara, Brazil: ACM, 2008, pp. 1101–1106, ISBN: 978-1-59593-753-7. DOI: 10.1145/1363686.1363943. [Online]. Available: http://doi.acm.org/10.1145/1363686.1363943.

[33] A. Harth, S. Kinsella, and S. Decker, "Using naming authority to rank data and ontologies for web search", in *International Semantic Web Conference*, Springer, 2009, pp. 277–292.

[34] M. Needham and A. E. Hodler, *Graph Algorithms: Practical Examples in Apache Spark and Neo4j*. O'Reilly Media, 2019, ISBN: 9781492047650.

[35] Neo4J. (). Neo4j product, [Online]. Available: https://neo4j.com/product/.

[36] ——, (). Release notes: Neo4j 3.5.0, [Online]. Available: https://neo4j.com/release-notes/neo4j-3-5-0/.

[37] ——, (). Neo4j 4.0.1 api, [Online]. Available: https://neo4j.com/docs/java-reference/4.0/javadocs/index.html.

[38] ——, (). Indexes, [Online]. Available: https://neo4j.com/docs/cypher-manual/3.5/schema/index/.

[39] J. Barrasa, E. Arkan, R. Piris, A. Santurbano, M. Needham, M. Hunger, K. Yankov, S. Araujo, J. Calder, and C. Willemsen, *Neosemantics*, `https://github.com/neo4j-labs/neosemantics`, 2020.

[40] Neo4J. (). Importing rdf data, [Online]. Available: `https://neo4j.com/docs/labs/nsmntx/current/import/`.

[41] K. Balog and R. Neumayer, "A test collection for entity search in dbpedia", in *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, 2013, pp. 737–740.

[42] Neo4J. (). Indexes for full-text search, [Online]. Available: `https://neo4j.com/docs/cypher-manual/current/administration/indexes-for-full-text-search/`.

[43] M. Needham and A. Hodler. (2019). Graph algorithms in neo4j: Pagerank, [Online]. Available: `https://neo4j.com/blog/graph-algorithms-neo4j-pagerank/` (visited on 08/05/2020).