UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

DEPARTMENT OF INFORMATION ENGINEERING

MASTER'S DEGREE IN COMPUTER ENGINEERING

Towards data-based search engines for RDF graphs: a reproducibility study

Supervisor:
Prof. Gianmaria Silvello

Co-Supervisor:
Ing. Laura Menotti

Student:
Manuel Barusco
(Student ID: 2053083)

ACADEMIC YEAR: 2022/2023

Graduation Day: October 23, 2023

I have not failed, I've just found 10,000 ways that won't work.

-Thomas A. Edison

Abstract

The RDF framework, thanks to its flexibility and versatility, is one of the most used formats for sharing data and knowledge on the Web. Nowadays a lot of RDF datasets and RDF knowledge repositories are available in the scientific and political fields and can be easily consulted from a lot of open data portals. However, these RDF datasets cannot be fully exploited and accessed due to the absence of advanced search engines that allow users to retrieve the best datasets that suit their needs. These systems solve the Ad-Hoc RDF Datasets retrieval task: answer to a user keyword query with a rank of 10 datasets ordered by relevance. The current systems are not so advanced and are principally based on the datasets metadata, which could be incomplete or not always available, instead of being based on their content. ACORDAR [11] is the first open test collection created to evaluate the systems developed for the Ad-Hoc RDF Datasets retrieval task. This test collection can ensure a boost in the development and improvement of these systems and a possible switch from metadata-based to content-based search systems.
The main focus of this thesis is a reproducibility study on the ACORDAR collection. We are going to actually test how this collection is good, useful and suited for the Ad Hoc RDF datasets retrieval task by reproducing the baseline systems developed by the ACORDAR creators and by discussing all the reproducibility problems encountered during the development of the reproduced systems.

Contents

Chapter 1

Introduction

The enormous availability of data on the Web began an era of unprecedented information abundance. Data comes from various domains and sources (scientific or not) and is available in various formats. Organizations, researchers, and individuals continue to produce, curate, and disseminate data in a lot of forms that are more or less structured: videos, images, audio, text, tables, graphs, and so on. Data is considered the new oil and it is the key element for the development of new Machine Learning systems, new Data Analytics tools, or for Statistical studies on specific fields.

Among the various data representation formats, the Resource Description Framework (RDF) stands out as a cornerstone of the Semantic Web, providing a versatile and expressive means of structuring and interconnecting data through the representation of triples. The RDF model embodies a fundamental shift in data representation, moving beyond traditional tabular structures to capture not only the raw facts but also the intricate relationships between entities. The flexibility of the RDF model made it one of the easiest ways to represent knowledge in a graph structure and to make semantic reasoning on the data: reasons why now the RDF format is one of the most popular formats for spreading data.

Nowadays the availability of RDF datasets and RDF knowledge graphs on various domains is enormous: this is giving rise to an ecosystem of RDF datasets, ranging from domain-specific knowledge graphs and scientific databases to cultural heritage repositories and governmental data hubs. These datasets encode diverse forms of knowledge, enabling applications that span from intelligent virtual assistants and data integration platforms to scholarly research and cultural heritage preservation.

Unfortunately, the enormous availability of RDF data is not fully exploitable due to the lack of advanced search engines that allow users to access the datasets that best suit their needs. These search engines are born to solve the Ad Hoc RDF datasets retrieval problem: answer a user query with a list of datasets in order of relevance. The current systems are not so advanced in tackling this task and a lot of research activities are moving to solve all the lacks of these systems. First of all, the state-of-the-art systems are metadata-centred, thus the datasets search is based

on a simple text search on the datasets metadata instead of relying on the dataset content. The metadata are not always enough for doing an exhaustive search: they are not always available and sometimes they do not reflect the main dataset content or semantics. Furthermore, the current systems do not provide for every returned dataset an exhaustive representation of its content, a part the dataset metadata few other information are visualized (typically simple statistics about the dataset) that are sometimes useless for the user to understand the main content and relevance of the dataset. In conclusion, the relevance of a given dataset to a given query is difficult to assess: the system must understand the data domain indicated by the user and what data the user needs. Probably also the form in which the queries are posed is too simple for the task: the keyword queries can limit the user needs expression and other types of query representation can be used, by expressing domain filters, data types filters, needed entities and so on.
In this scenario, the work done by [11] has lent a useful contribution to solve the presented problems thanks to the release of the ACORDAR collection. This collection, made of a list of RDF datasets and a set of query and relevance judgements, is the first test collection for the Ad Hoc RDF datasets retrieval task and it will boost the development of new and better RDF datasets search engines. Furthermore, the work presents an innovative dashboard for browsing the datasets content that is perfect for assessing the relevance of a given dataset to a query by exploring useful insights, snippets and summaries of the dataset content that could make the user less confused about its quality, structure and about the data that it is representing.
The main focus of this thesis is a reproducibility study on the ACORDAR collection. We are going to actually test how this collection is good, useful and suited for the Ad Hoc RDF datasets retrieval task by reproducing the baseline systems developed by the ACORDAR creators.
The remainder of this thesis is organized as follows. In Chapter 2 we present a background review of all the key elements of the Resource Description Framework, Text Retrieval and Evaluation, and, related works on the Ad Hoc RDF Datasets retrieval problem. In Chapter 3 we dive into the details of the ACORDAR collection and baseline systems, in Chapter 4 we present the developed systems for the reproducibility study and in Chapter 5 we analyze the reproducibility problems met during the systems development. In Chapter 6 we analyze the results of our systems and compare them with the ACORDAR ones, in Chapter 7 we explain some solutions to the reproducibility problems met and in Chapter 8 we conclude our work by discussing the obtained results and possible future work directions on the Ad Hoc RDF datasets retrieval task.

Chapter 2

Background

In this Chapter we discuss and review the main details about the Resource Description Framework (RDF) in Section 2.1 and we define, analyze and, discuss the Ad Hoc RDF Datasets retrieval task in Section 2.2 by analyzing the positive and negative aspects of the actual RDF datasets search engines. Finally in Section 2.3 we discuss some aspects, tools and, concepts about Text Information Retrieval that were fundamental in the development of the first RDF Datasets Search Engines which were developed based on the first Text Search Engines.

2.1 Resource Description Framework (RDF)

The Resource Description Framework (RDF) [13], released as a W3C recommendation in 1999, is the foundation of the Semantic Web. The main focus of the Semantic Web (term coined for the first time by Tim Berners Lee) is the creation of a web of data from around the world that can be linked together and that can be found, browsed, crawled, integrated and in general processed directly and indirectly by machines [24]. This defines a big transition from the Web intended as a Web of interconnected documents to a Web of linked data. This idea was possible thanks to the HTTP protocol that allows the exchange of data between users and machines, RDF that manages the data representation and the URI concept that allows to identify a given resource in a unique way on the whole web.
RDF is a very flexible framework for data representation, indeed it overcomes the limitations of the traditional relational databases that have proven to be effective in capturing structured information but that often falter when it comes to capture the intricate relationships and semantics inherent in real-world data. RDF represents specific domain data by avoiding rigid tabular structures as in the relational model or tree structures such as in XML but it uses a flexible structure based on a directed labelled graph [3]. The nodes of an RDF graph can be of 3 types:

- Entity nodes: an entity is something that can exist or not (for example a concept) in the real world or in the specific data domain of the graph. An entity node has an URI associated.

- Class nodes: these nodes represent entity types. These nodes allow entities to be grouped into various groups. A class node has an URI associated.
- Literal nodes: particular types of nodes that contain a given value, for example, an integer value or a string. A literal node has not a URI associated.
- Blank nodes: nodes that are used to indicate the existence of a thing, without associating to it a URI. They are used for describing multi-component structures, reification and to represent complex/composite attributes. Since they do not have an URI associated they cannot be referenced outside the RDF graph where they have been defined, indeed they have validity only within that graph.

The nodes are connected by arcs that in RDF are called properties and every property has an URI associated. The whole graph is defined with a set of triples (also called statements) in the form of Subject, Predicate (Property) and Object. The subject can be a blank node or a URI identified node, the predicate is the property URI and the object can be a URI identified node, a literal node or a blank node. These triples represent the factual statements of the Knowledge Base and, as mentioned before, this base could be seen as a large, directed, labelled graph called RDF Graph as shown in Figure 2.1. In the Figure each subject/object is represented as a node

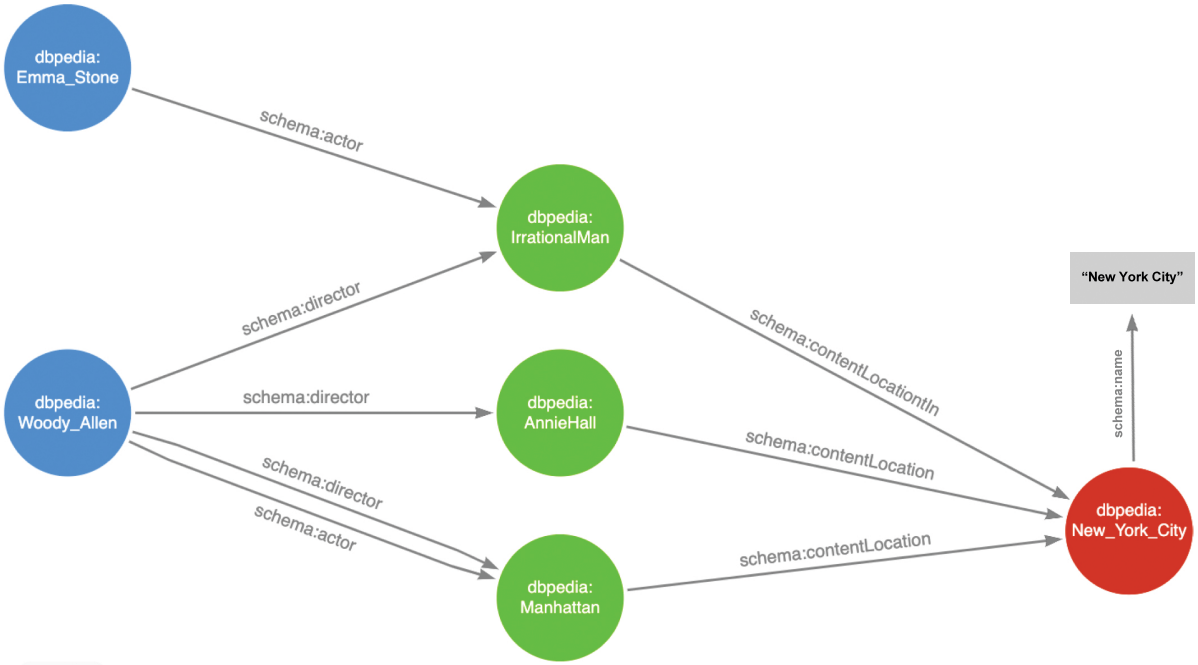


Figure 2.1: Snippet of an RDF graph about movies taken from [18]. URI nodes are represented by circles, literal nodes by rectangles and properties by arcs.

while each predicate is represented as an edge and the URIs are expressed by using the CURIE notation explained below.
In order to control, regulate and describe the structure of the RDF graphs, the W3C introduced

the RDF Schema (RDFS) [14]: this set of classes and properties provides a vocabulary of constructs and constraints that allows the classification and definition of all the information related to RDF data. In particular, it defines the concept of classes, it allows the creation of constraints and so on, like a standard tool for defining the schema of data.
The Uniform Resource Identifier (URI) is a compact sequence of characters that identifies in a unique way an abstract or physical resource in the Semantic Web. This concept is fundamental and it allows us to refer and connect concepts in the web of data and between different RDF datasets. In the traditional relational databases, if you want to connect or refer to concepts in different databases, you have to define and use an ID that must be accepted and known by everyone that uses it [3]. In the Semantic Web, instead of creating an ad-hoc ID for every concept within a database, we assign a global URI identifier to everything that can be referred inside the database, across different databases and in general in the Web. In Figure 2.1 the URI are not expressed in their complete form but by using the CURIE [15] notation. The CURIE notation is based on the scoping concept, where a set of names are created and are valid inside a given scope. The scope is called namespace and is identified by a name. The CURIE notation is composed of two components: a prefix and a reference. The prefix is composed of the namespace name and the reference contains the object name. The usage of namespaces allows to define the same name in different scopes and the namespaces are usually organized as hierarchies. In Figure 2.1 for example, the entity Emma Stone is defined as `dbpedia:Emma_Stone` where dbpedia is the namespace (name of the RDF graph) and Emma Stone is the reference (name of the entity). Every namespace is associated with an URI, in this case, the prefix dbpedia is associated to ”https://dpbedia.org/page/”. The full URI associated with the entity Emma Stone is thus ”https://dpbedia.org/page/Emma_Stone” but it is abbreviated with `dbpedia:Emma_Stone`.
The CURIE notation works thanks to the RDF version 1.1 [16] that allows the creation of the so called RDF datasets (named graphs): the RDF graphs (and thus the set of triples that defines them) can be identified by a URI and can be located and accessed across the web.
There are many ways to serialize, i.e. represent, RDF data in text. We can use the following formats:

- N-triples;
- Turtle/N3: more human readable;
- RDF/XML: best for machine reading;
- JSON-LD: best for WebApp API development.

These are all possible serializations that can be used to express a given RDF graph in a textual way. Some of these are more human or machine-readable and the serialization choice can be based on taste, experience, tools, conventions and so on: there is not a best serialization.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX foaf: ...

SELECT ...
WHERE {
    #
    ?s ?p ?o .

    FILTER(...) .
}
GROUP BY(...)
HAVING(...)
ORDER BY(...)
```

Query 2.1: Basic structure of a SPARQL query

2.2 SPARQL

SPARQL is a recursive acronym that stands for SPARQL Protocol and RDF Query Language and is the standard RDF Query language [25]. It allows the retrieval of data from the RDF graphs by matching, filtering and grouping the needed triples. Like in the SQL query language for the relational databases the SPARQL query language is based on two main constructs: the SELECT construct which is composed of a set of variables (or expressions) to be returned, and the WHERE construct, which is composed by a set of triple patterns and filters to be matched in the graph. The SPARQL syntax is similar to the syntax used in the RDF Turtle serialization. The other constructs that can be used in the queries are:

- FILTER: allows to filter the triples by specifying a given condition, it is used inside the WHERE construct;
- GROUP BY: allows to group the WHERE matching triples;
- HAVING: allows to filter the GROUP retrieved triple groups by specifying a group condition;
- ORDER BY: allows to order the output data by following a given order.

The conditions that can be specified are boolean conditions that can be specified by using a series of operators: logical, math, boolean, regex and so on. Subqueries and query concatenation by using set operators are also allowed [1]. The Query 2.1 shows the standard structure of a SPARQL query.

2.3 Ad-Hoc RDF Datasets Retrieval Problem

The flexibility of RDF has made it one of the most used data models for representing knowledge and real-world data from different domains, from the political to the scientific field. This caused the growth of the number of available RDF Knowledge graphs, available in a lot of Open Data Portals, that contain a lot of different semantic data.
This gave a lot of research attention to the Ad-Hoc RDF Datasets Retrieval Task: answer a user keyword query with a ranked list of datasets [11]. This task is done by the RDF Datasets Search Engines. The development of these systems and the resolution of this task is yet an undergoing research problem.
The key problem of the task is that it is difficult to capture the query semantic and thus assess the relevance of a given dataset to a given query because the relevance of a dataset can depend on multiple factors such as:

- Does the dataset contain the data searched by the user?
- Does the dataset contain the data needed for the user intended task?

The developed systems are actually solving the task by considering it as a variant of the in-depth studied Ad-Hoc Document Retrieval problem, indeed they are using the same tools and ideas by considering the RDF datasets as textual documents and not as semantic graphs. Indeed the RDF graphs are indexed and searched by using their textual serialization files and not their graph structures [11] [12]. This approach does not consider the semantics of the query or the semantics of the datasets and indeed is a quite ”basic” and not so advanced solution.
These systems are also affected by the following open problems:

- The systems are principally metadata-based [9]. It means that these systems search for the best-suited datasets relying on the metadata associated with the RDF graph such as the description, author, tags and other similar fields. These metadata can be created by human annotators and are not always available. The main problem of this approach is that many times the metadata information is not exhaustive enough to satisfy the user queries and does not reflect the main focus, purpose and content of the datasets. Since the available solutions are metadata-based, they are not based on the content of the dataset. This means that if the user express a given query that match the content of the dataset and not its metadata, the dataset will not be returned in the final rank.
- The actual systems are based on keyword queries. These queries are quite simple and cannot express the complete intention of the users, such as the need of data for a given specific task or the need of a particular type of data.

- The systems are not provided with a dashboard that can give an exhaustive idea of the returned datasets content (and thus their relevance to the queries) by providing useful insights or dataset snippets and summaries.

- Actually there are not a lot of test collections to test the development of the RDF Datasets Search systems.

The Google Dataset Search engine [10] can be considered the first search engine for datasets. It is not restricted only to the RDF datasets but it searches also over CSV and other datasets format. Unfortunately, it is based on the datasets metadata as reported in its website [8].
Another available system is LODAtlas [12], it was built for searching RDF datasets by using keyword queries that match on classes and properties of the RDF graph, furthermore, it also visualizes simple graph level data patterns and insights to facilitate the user in searching the more relevant datasets: these information are useful for assessing the relevance of a given dataset to the query but are not used by the system during the retrieval.

2.4 Document Level Information Retrieval

In the computing and information science domain, Information Retrieval (IR) is the field that is concerned with the management and research of information. As defined by Salton in [17]: *"Information retrieval is a field concerned with the structure, analysis, organization, storage, searching and retrieval of information"*. This field has been extensively studied since the 90s and a lot of tools, frameworks and libraries have been developed. We are going to review the main IR concepts and tools that are at the base of the RDF Datasets Retrieval task and the first RDF datasets search engines.
The Information Retrieval process is composed by the following 4 main phases: Document Acquisition, Indexing, Retrieval and, Evaluation.
The first phase is composed by the process of document and queries acquisitions. In this phase, all the documents where we want to search are acquired and the main user information needs are gathered with a list of queries (longer or shorter strings composed by a series of keywords). Now we are going into the details of the remaining phases.

2.4.1 Indexing

The indexing phase is composed by the Document Analysis and Inverted Index Construction sub-phases. Firstly, all the acquired documents are analyzed and processed mainly with the following three steps:

- Tokenization and Lexical Analysis;

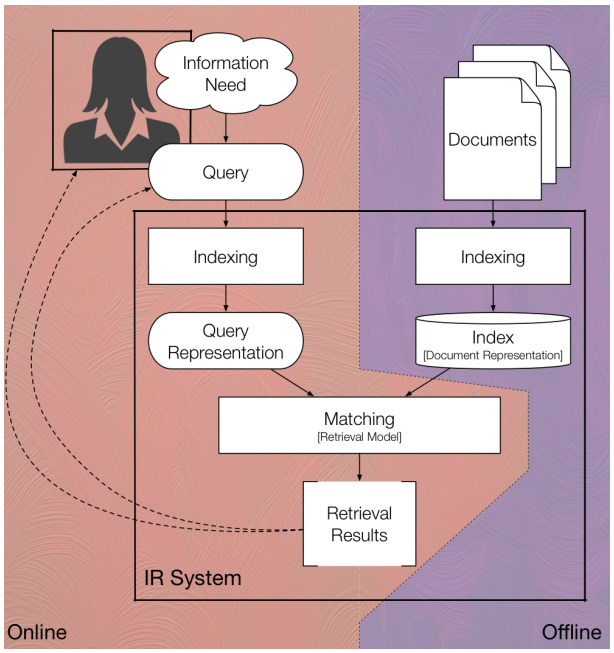


Figure 2.2: Y model of an information retrieval process with its component and phases taken from [6].

- Stop Words removal;
- Stemming.

In the first step, the document is processed in order to decompose it into a series of tokens. The tokens in a text document are basically the terms (words) that compose it. Sometimes this one-to-one relation between words and tokens is not so direct indeed there are different techniques for text tokenization, and sometimes a word can be divided into multiple tokens (in this case named subwords and in this case we talk about subword tokenization) that correspond to the different morphemes or important lexical components that compose it. In the case of textual documents, the tokens can be identified thanks to separator characters like white spaces and punctuation marks but it is important to treat in the correct way digits, words with hyphens (”state-of-the-art” for example) and words with important uppercases letters (”Bank” city in Iraq and the word ”bank”) because the tokens, after parsing them, are converted to lower case. The punctuation after this phase is completely removed.
After this phase, a stop word removal is done. A word is considered a stop word if it has poor information content. In natural language there are a lot of common words or functional words that can appear a lot of times in a given text, these words are for example articles, propositions, conjunctions etc. These words, called in IR ”stop-words”, since they can appear in a lot of documents, have very low discrimination values for retrieval purposes and they are not so important in assessing the relevance of a document to a given query. It is recommended to remove them sparingly in order to maintain flexibility in the retrieval process, maintain the

document content and in order to recognize phrases composed by a lot of stop words (for example 'to be or not be'). The stop words removal can decrease a lot the index size. A lot of stoplists are available, for example, the NLTK stoplist or the Google stoplist.
At this point, it is possible to observe that a lot of tokens differ only in their full inflection form and do not differ from a semantic point of view (for example: open and opened are actually referring to the same action). In order to reduce the syntactic variance of the tokens we can map these to their "root" (also called "stem") form by removing from them all the prefixes and suffixes. This allows us to reduce the index dimension and generalize the indexed information. It is important to be careful also in the stemming phase because sometimes too much information is removed and sometimes the stemmers, when removing affixes and suffixes, can create unknown words (for example we can obtain from "opposing", gerund form of oppose, the stem "oppos" that is not actually a word). A lot of stemmers have been developed during the years such as the Porter stemmer but also the Lovins and Krovetz stemmers.
The final part of the indexing phase is the Inverted Index construction. The Inverted Index is a special data structure that connects a term (token) with the documents in which it appears: it is a map of the form "term" $\rightarrow$ list of documents that contain the term. The inverted index can be augmented by considering the weight of the given token in the given document. The tokens are weighted to reflect their importance in the document, indeed we can notice that not all the tokens in a given document describe its semantics and information with the same effectiveness. The weight assigned could be of two types: binary or frequency-based. The binary weight simply identifies if a token is present or not in the document while the frequency-based weights identify the number of times the token occurs in the document (*Term Frequency - TF*) or in the collection (*Inverse Document Frequency - IDF*). The first type of weights is useful in the Boolean Retrieval model while the second type is useful in the Vector Space Model as explained in the next section.

2.4.2 Retrieval

The retrieval phase exploits all the information and structures obtained during the indexing phase in order to give the user a rank with all the documents closer to its information needs in order of relevance. Specifically, it employs term frequency as a key element and complex statistical models that aim to assign a score to each document: this score quantifies the degree of relevance of the document with respect to the user's query. Historically the first information retrieval model was the Boolean model. In this framework the user's information requirement, expressed through a quite structured query, is linked with the set of documents that satisfy the conditions posed by it. However this model does not rank this set of documents and it employs a binary approach: documents that match the query are assigned a value of 1, which stands for "relevant", while those that fail to match receive a value of 0, which stands for "irrelevant". This

model encounters two primary challenges. Firstly, users are expected to possess familiarity with boolean operations and with a somewhat structured language to navigate the system effectively and obtain the desired information. Secondly, the returned document list lacks of any inherent ordering of documents based on the degree to which a document aligns with the query. The need for creating a ranked list of documents in response to a query, coupled with the requirement to assign varying weights to terms within documents according to their frequencies, has led to the development of alternative retrieval models over time. These include the Vector Space Retrieval model, the Probabilistic Retrieval Model and the Retrieval based on Language Models.

2.5 Vector Space Retrieval Model

In this retrieval model, every document is represented as a vector of dimension t, where t is the number of different terms in the index and d_{ij} is the weight of the term j in the i-th document D_i. So the document D_i is represented as:

$$D_i = (d_{i1}, d_{i2}, ...d_{ij}, ...d_{it}) \tag{2.1}$$

The same representation structure is associated to the query and the terms that compose it.

$$Q = (q_1, q_2, ..., q_t) \tag{2.2}$$

where t is the number of index terms and q_j represents the weight of the j-th term in the query. In the real case, for both the document and the query, t is the number of terms in the entire documents collection. Now documents and queries are represented as vectors with the same dimension t and we can compare these vectors with the cosine similarity function.

$$Cosine(D_i, Q) = \frac{\sum_j (d_{ij} \cdot q_j)}{\sqrt{\sum_j d_{ij}^2 \cdot \sum_j q_j^2}} \tag{2.3}$$

This similarity score is 0 if the two vectors have no common terms or 1 if they are identical. The terms weights (for the document and query vectors) are defined based on the chosen weighting scheme: the most common is the TF-IDF.

2.5.1 TF-IDF

The Term Frequency - Inverse Document Frequency is a weighting scheme based on two components, the Term frequency (TF) that reflects the importance of a term in a specific document, and the Inverse Document Frequency (IDF) that reflects the importance of a term in the collection. Basically, the Term Frequency measures how many times a given term is present

in a given document: the higher this value, the more important the term in the document. The term frequency associated with the term k in the document D_i with $j \in \{1, ..., t\}$ is defined as:

$$tf_{ik} = \frac{f_{ik}}{\sum_j f_{ij}} \tag{2.4}$$

where f_{ij} is the number of occurrences of the term t_j in the document D_i and $\sum_j f_{ij}$ is the sum of the frequencies of all the terms in the document D_i: this is basically the length of the document D_i. Usually, the lengths of the various documents in a collection can be very different one to each other, thus, in order to reduce the importance of terms that appear many times in short documents, it is preferable to use TF in a logarithmic form like:

$$tf_{ik} = \frac{\log(f_{ik} + 1)}{\sum_j \log(f_{ij} + 1)} \tag{2.5}$$

where the +1 ia s correction factor used to avoid the undefined value in case of $f_{ik} = 0$. The Inverse Document Frequency component is defined as:

$$idf_k = \log(\frac{N}{n_k}) \tag{2.6}$$

where N is the number of documents in the collection and n_k is the total number of documents in which the term k is present. The IDF term describes the fact that a term which is present in many documents will be less significant for the description of the information content of the documents with respect to a term that appears in fewer documents.
In conclusion, in the document vector, this weighting scheme assigns to every term k in a given document D_i the following weight:

$$d_{i,k} = tf_{ik} \cdot idf_k \tag{2.7}$$

2.6 Probabilistic Retrieval Model

The key idea of this retrieval model is to exploit the probability theory for manipulating the uncertainty inside the whole retrieval process. All these models are based on the so called Probability Ranking Principle [21]:

"If a reference retrieval system response to each user query is a ranking of the documents in the collection in order of decreasing probability of relevance to the user who submitted the request, where the probabilities are estimated as accurately as possible on the basis of whatever data have been made available to the system for this purpose, the overall effectiveness of the system to its user will be the best that is obtainable on the basis of those data"

This Principle actually formulates a new retrieval framework but unfortunately, it does not

tell how to calculate or estimate the probability of relevance. There are several models and algorithms that propose different methods for this probability estimation: the most famous is the BM25.

2.6.1 BM25

The BM25 algorithm is a ranking algorithm (function) used to estimate the relevance of documents to a given search query and it is based on the probabilistic framework. It was introduced by Robertson and Walker in 1994 as an improvement over the previous Okapi BM11 algorithm. Some variations have been proposed during the years but the main structure of the ranking formula is the following. Consider D as a document and $q = q_1 q_2 ... q_n$ as a query, the BM25 Score of the document D for the query q is defined as:

$$score(D, q) = \sum_{i=1}^{n} IDF(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b * \frac{|D|}{avgdl})} \tag{2.8}$$

where:

- $f(q_i, D)$ is the number of times that the query term q_i appears in the document D;
- $|D|$ is the length of the document (number of terms);
- $avgdl$ is the average document length of the indexed documents in the collection;
- k_1 and b are free parameters that can be fixed during the experiments. Usually $k_1 \in [1.2, 2]$ and $b = 0.75$.

The $IDF(q_i)$ is the Inverse Document Frequency weight of the term q_i and in this case it is computed as:

$$IDF(q_i) = \ln\left(\frac{N - n(q_i) + 0 - 5}{n(q_i) + 0.5} + 1\right) \tag{2.9}$$

where:

- N is the total number of documents in the collection;
- $n(q_i)$ is the number of documents in the collection that contains (regardless of the number of occurrences) the term q_i.

The key elements of the BM25 ranking formula are the following:

- **Term Frequency (TF)**: the algorithm uses a slightly changed version for measuring how many times a given term appears in a document. This version takes into account

the saturation effects and prevents overemphasizing heavily repeated terms. The Term Frequency component is the second component in Formula 2.8.

- **Inverse Document Frequency (IDF)**: IDF measures the importance of a term in the entire collection. It assigns higher weights to terms that are rare in the collection and lower weights to terms that are more common.
- **Document Length Normalization**: BM25 includes document length normalization to mitigate the influence of document length on relevance scoring. Longer documents often feature a higher frequency of a given term, introducing potential bias. Document length normalization counteracts this bias by dividing the term frequency by the document length and applying a normalization factor.
- **Query Term Saturation**: BM25 includes a term saturation function (more precisely the logarithm function in the IDF calculation) that mitigates the impact of terms with high frequencies. Very high frequency terms often correspond to less informative terms for retrieval purposes.

All these key points, as reported here [20], are followed by these problems:

- BM25 assumes statistical independence between query terms, which may not hold true in some cases where term dependencies exist;
- the algorithm heavily relies on term frequency and document length, potentially overlooking other important factors like document structure and relevance feedback.

2.7 Language Model based Retrieval

This type of retrieval model can be considered as a variant of the probabilistic model and is one of the most used retrieval models nowadays. The main idea is that a given query is "generated" by a Language Model calculated from the document D that allows to calculate $P(q|D)$: the probability that the query q is generated from the document D language model.
A Language Model is a probabilistic model that given a sequence of words $w_1 w_2 ... w_t$ gives us the probability of the next word w_{t+1}:

$$P(w_{t+1}|w_1 w_2 ... w_t) \tag{2.10}$$

after being tuned on a given text. In this case, the model can be seen as a predictive model, but it can also be considered as a generative model that can assign a probability to a given string:

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i|w_{1:i-1}) \tag{2.11}$$

In the IR case the language model is based on a given document D, this means that it is estimated on it and all its parameters are tuned on its content. Thus the basic idea is to obtain, for every document, a language model from it, and use that model to calculate the probability $P(q|D)$ for a given query q. The higher the probability a given document language model gives to the query q, the higher it will be that document in the final rank. This idea allows us to model the relevance of a document to a given query in a new and interesting way.
The most used language model in the information retrieval field is the n-gram language model and in particular the unigram language model. The N-gram language model approximates the probability in 2.10 in the following way:

$$P(w_t|w_{1:t-1}) \approx P(w_t|w_{t-N+1:t-1}) \tag{2.12}$$

so it assigns a probability to a given word w_t by considering the N-1 words that precede it as a sort of context information in the probability calculation. The higher the N more context information will be considered and the probability will be more precise. This probability can be estimated from a "training" text (in the IR case in a document D) by using the Maximum Likelihood Estimator and in particular the Relative Frequency Estimator:

$$P(w_t|w_{t-N+1:t-1}) = \frac{C(w_{t-N+1:t})}{C(w_{t-N+1:t-1})} \tag{2.13}$$

where $C(w)$ counts the number of times a given string w appears in a given text (in the IR case in a given document). The unigram language model (N=1) approximates the probability in the following way:

$$P(w_t|w_{t-N+1:t-1}) \approx P(w_t) = \frac{C(w_t)}{n} \tag{2.14}$$

where n is the text (document) length. In this case, no context information is used in the probability calculation so the probability will be much less precise and "noisier".
Unfortunately, the Relative Frequency Estimator will generally under-estimate the probability of any unseen word in the document ($C(w_i) = 0$) so a lot of smoothing methods were introduced. The main purpose of smoothing is to assign a non-zero probability to the unseen words (or words combination) and improve the probability estimation in general [26]. All the smoothing methods achieve this by discounting the probabilities of the seen words in the document and then assigning the extra probability mass to the unseen words according to some schema such as the Laplace Smoothing. An interesting variant of the Laplace smoothing is the Bayesian smoothing using Dirichlet priors [26] where the probabilities are calculated as:

$$P_\mu(w|D) = \frac{C(w) + \mu P(w|C)}{n + \mu} \tag{2.15}$$

where:

- $P(w|C)$ is the language model of the collection thus obtained by not only considering the document d but the whole collection C of documents;

- μ is the smoothing parameter, usually it is set to 2000.

2.7.1 FSDM

The Fielded Sequential Dependence Model [27] is a retrieval model for structured documents. It is a variant of the Sequential Dependence Model which is based on the Markov Random Field which allows to consider sequential dependencies for the query and document terms by using three potential functions: one based on unigram and the other two based on bigram language models, either as ordered sequences of terms or as terms that co-occur within a window of pre-defined size. Since it is based on unigram and bigram language models it can be considered as part of the retrieval models based on Language Models.
FSDM overcomes one of the limitations of the SDM model: SDM considers terms matches in different parts of the document as equally important thus disregarding the document structure. Indeed, in the case of unigrams for example, the potential function used in the SDM model is:

$$f_T(q_i, D) = \log P(q_i|\theta_D) = \log \frac{tf_{q_i,D} + \mu \frac{cf_{q_i}}{|C|}}{|D| + \mu} \tag{2.16}$$

where q_i is a query term, D is a document, $tf_{q_i,D}$ is the frequency of term q_i in D, $|D|$ is the document length and μ is the Dirichlet prior usually set to the average document length in the collection. cf_{q_i} is the collection frequency of q_i and $|C|$ is the total number of terms in the collection. In FSDM this single document language model is replaced by a mixture of language models, one for each document field. The potential function for unigram becomes:

$$\tilde{f}_T(q_i, D) = \log \sum_j w_j^T P(q_i|\theta_D^j) = \log \sum_j w_j^T \frac{tf_{q_i,D^j} + \mu_j \frac{cf_{q_i}^j}{|C_j|}}{|D^j| + \mu_j} \tag{2.17}$$

where $j \in \{1, ..., F\}$ where F is the number of document fields, θ_D^j is a language model based on field j and smoothed using its own Dirichlet prior μ_j and w_j are the fields weights with the following constraints: $\sum_j w_j = 1, w_j \geq 0$. tf_{q_i,D^j} is the term frequency of q_i in field j of document D, $cf_{q_i}^j$ is the collection frequency of q_i in field j, $|C_j|$ is the total number of terms in field j across all the documents in the collection and $|D^j|$ is the length of field j in D. The potential functions for ordered and unordered bigrams $q_{i,i+1} = (q_i, q_{i+1})$ in the query are defined

as follows:

$$\tilde{f}_O\left(q_{i,i+1}, D\right) = \log \sum_j w_j^O \frac{tf_{\#1(q_{i,i+1}),D^j} + \mu_j \frac{cf^j_{\#1\left(q_{i,i+1}\right)}}{|C_j|}}{|D^j| + \mu_j} \tag{2.18}$$

$$\tilde{f}_U\left(q_{i,i+1}, D\right) = \log \sum_j w_j^U \frac{tf_{\#uw8(q_{i,i+1}),D^j} + \mu_j \frac{cf^j_{\#uw8\left(q_{i,i+1}\right)}}{|C_j|}}{|D^j| + \mu_j} \tag{2.19}$$

where $tf_{\#1(q_{i,i+1}),D^j}$ is the frequency of exact phrase $q_i q_{i+1}$ in field j of document D, $cf^j_{\#1(q_{i,i+1})}$ is the collection frequency of exact phrase q_i, q_{i+1} in field j, $tf_{\#8(q_{i,i+1}),D^j}$ is the number of times terms q_i and $q_{i,i+1}$ occur together within a window of 8 word positions in field j of document D, regardless of the order of these terms. $cf^j_{\#uw8(q_{i,i+1})}$ is the number of times terms q_i and $q_{i,i+1}$ appear in the collection together within a window of 8 words positions in the field j. The weights w_j are subject to the same constraints mentioned before. The final ranking function is defined as:

$$P(D|Q) \stackrel{\text{rank}}{=} \lambda_T \sum_{q \in Q} \tilde{f}_T(q_i, D) + \lambda_O \sum_{q \in Q} \tilde{f}_O(q_i, q_{i+1}, D) + \lambda_U \sum_{q \in Q} \tilde{f}_U(q_i, q_{i+1} D) \tag{2.20}$$

All the weights should be fixed by using a procedure of parameter tuning and a possible algorithm is explained in the FSDM paper [27].

2.8 Query Boosting

One possible way to control the scores assigned to the documents by a retrieval model is through the boosting weights. First of all, we have to consider the fact that many times the documents are composed of a lot of fields, for example, if we want to create a search engine for books, every book can be considered as a document and every book (document) can have a title, a brief summary, an author and so on. When the user enters the query, this query can match any document field, but sometimes we can consider the fact that a match on the title field is more important than a match in the summary field, especially if the user is searching for a book with a given title or simply because the title field is considered more important. In order to control this behaviour we can set for every field a boost score that will change the global score assigned to a given document if there is a query match in that given field. For example, in the TF-IDF retrieval formula [5], the query boosting change the document D score for a given query $q = q_1 q_2 ... q_m$

as follow:

$$\textit{TF-IDF(q,D)}= \sum_{i=1}^{m} \left(TF(q_i, D) \cdot IDF(q_i) \cdot \sum_{f \in F} boost(q_i, f) \right) \tag{2.21}$$

where:

- F is the set of document fields;
- $boost(q_i, f)$ is the boost factor if there is a match of the term q_i in the field f.

2.9 Evaluation

The evaluation of an IR system is conducted by computing some evaluation metrics that measure the overall effectiveness (and sometimes the efficiency) of the system. The evaluation is fundamental to understand how good the system is and to compare it to other systems or to different versions of the same one. This last aspect is very important: from the previous sections, we can notice that an IR system is composed of a lot of variable parts. For example in the indexing phase, it is possible to choose different stoplists or stemmers and in the retrieval part different retrieval models. Since there is not a universal configuration that always works, a lot of experiments are done to test all the possible variable components and see which configuration is the best by comparing the tested ones. In order to evaluate the system we need a test collection and some evaluation metrics. These elements derive from the standard de facto for the evaluation process: the Cranfield paradigm, which was developed in the 60s by Cyril Cleverdon, a librarian of the College of Aeronautics in Cranfield. The first element, the test collection, is a triple composed by:

- a set of documents;
- a set of topics (surrogate of information needs);
- a set of relevance judgments (binary or graded) also called relevance assessments or qrels that assign for every document and for every query the relevance of that document for the query.

From a logical point of view, in order to have an exhaustive set of qrels we have to evaluate the relevance of every document for every query but this can be quite prohibitive in terms of time and resources since the documents evaluation is made by human annotators. This is especially true if the collection is very big. In order to obtain a set of relevance judgments a method called Pooling can be used. This process selects a subset of documents to evaluate for every query and then these evaluated documents are used to measure the effectiveness of the systems.

The pooling exploits the concept of **run** for the selection of the subset of documents. A run is formally defined as [2]. Let:

- $D = \{d_1, d_2, ..., d_n\}$ be a set of documents;
- $Q = \{q_1, q_2, ..., q_m\}$ be a set of queries (topics).

Given a natural number $N \in \mathbb{N}^+$ called *run length*, a **run** is defined as the function:

$$R : Q \to D^N \tag{2.22}$$

$$q \mapsto r_q = (d_1, d_2, ..., d_N) \tag{2.23}$$

such as $\forall q \in Q, \forall j, i \in [1, N] \mid j \neq i \Rightarrow r_q[j] \neq r_q[i]$ where $r_t[j]$ is the j-th element in the vector r_t.
The pooling approach considers a set of runs computed for a specific query and selects the first top-k documents of each run: k is defined as the depth of the pool. Then it creates a set from the union of these documents sets and assigns a relevance judgment for each document to the query by respecting a given relevance score. The relevance score can be binary (0: document not relevant, 1: document relevant) or multi-graded (0: document not relevant, 1: document partially relevant, 2: document relevant) or following another score schema. This process is iterated over all queries. Now that the set of relevance judgments is available we can introduce some evaluation metrics, from the basic to the advanced ones.

2.9.1 Precision and Recall

Precision and Recall are the oldest evaluation metrics introduced for evaluating the IR systems and were introduced in the Cranfield studies. Let's define:

- A: set of relevant documents for a given query q (documents that have a relevance score greater than 0);
- B: set of retrieved documents by the system for the query q.

The two metrics are defined as:

$$Precision(q) = \frac{|A \cap B|}{|B|} \tag{2.24}$$

$$Recall(q) = \frac{|A \cap B|}{|A|} \tag{2.25}$$

The number of relevant documents for the query ($|A|$) is also called Recall Base (RB). Intuitively, the Recall metrics measure how well the system is doing at finding all the relevant documents for a query, and the precision measures how well it is doing at rejecting non-relevant documents

[21]. The metrics just introduced are not taking care of the order of the results in the final ranks, indeed we are considering the output ranks as sets, without considering them as ordered lists. In order to overcome this problem we introduce the rank-based versions: Precision and Recall at Document Cut-Off k:

$$P(k) = \frac{1}{k} \sum_{i=1}^{k} r_i \tag{2.26}$$

$$R(k) = \frac{1}{RB} \sum_{i=1}^{k} r_i \tag{2.27}$$

where r_i is the relevance of the i-th document in the output rank and $r_i \in 0, 1$. Here we are considering a binary relevance for the documents, but we can simply convert a multi-graded relevance score to a binary score by setting a score threshold, for example, if the relevance score is composed of 0,1,2, we can set all the documents with a score of 0 to the binary score of 0 and all the documents with an higher score to the binary score of 1.

2.9.2 Average Precision

The Average Precision (AP) metric gives more importance to the ranks that put relevant documents in the top positions. This is a good aspect to measure because the users want the most important results on top of the output ranks. This metric is measured as:

$$AP(q) = \frac{1}{RB} \sum_{k \in R} P(k) \tag{2.28}$$

where:

- q: is a given query;
- RB: is the Recall Base previously defined;
- R: is the set of rank positions of the relevant retrieved documents in the output rank.

2.9.3 Mean Average Precision

The Mean Average Precision (MAP) metric is based on the AP measure but this metric allows us to make an evaluation of the system on all the queries where the system is tested and not only one. This metric is measured as the mean of all the AP measures obtained from all the test queries:

$$MAP = \frac{\sum_{q \in Q} AP(q)}{|Q|} \tag{2.29}$$

where:

- Q: is the set of all the test collection queries;
- $AP(q)$: is the AP measure of the system on the query q.

2.9.4 Discounted Cumulative Gain

The Discounted Cumulative Gain (DCG) is a measure that takes care of multi-graded relevance scores. In this type of score the document score $r_d \in \{0, 1, 2, 3\}$ or $r_d \in \{0, 1, 2\}$ depending on how the evaluation is fined grained. 0 means that the document is not relevant and so on. The DCG measure takes into account this main aspect: the documents with the highest relevance scores must be in the first rank positions and the documents with lower relevance scores in lower positions. In order to evaluate this aspect, the metric measures the *Discounted Gain (DG)* and uses a *discounting function* to progressively reduce the weight of the documents going from the high rank positions to lower ones: basically the gain is accumulated starting at the top of the rank and may be reduced (discounted) at lower ranks [21]. The DCG is the total gain accumulated at a particular rank p, and is defined as:

$$DCG_p = r_1 + \sum_{i=2}^{p} \frac{r_i}{\log_2 i} \tag{2.30}$$

where r_i is the relevance of the i-th document in the output rank. The $\log_2 i$ at the denominator is the discount or reduction factor that is applied to the gain. There is actually no mathematical justification for the choice of the discount factor, although it controls the reduction from smooth to higher. By varying the base of the logarithm the discount can be made sharper or smoother: the base (b) of the algorithm indicates the patience of the user in scanning the results:

- $b = 2$ represent an impatient user;
- $b = 10$ represent a patient user.

With an impatient user, the discount factor is higher, indicating that the final score will be lower and that the system is not so good if the relevant documents are not returned in the top positions. This metric, unlike the previous ones, returns a score that is not comprised between 0 and 1 but can be higher. For evaluating a given system by using this metric it is possible to set a given rank position p, calculate the DCG_p for every query q and then take the mean of all these scores. This metric is actually not so used, instead its normalized (between 0 and 1) form is preferred. In order to normalize the measure we need to identify the ideal run, i.e. the best rank for the given query, where the relevant documents for the query are inserted in order of relevance from those with the highest scores to those with the lowest. This best rank represents the best possible

rank and gives the maximum DCG values for every position p. We calculate the **Normalized Discounted Cumulative Gain (nDCG)** at position p for a given query q as follows:

$$nDCG_p(q) = \frac{DCG_p(q)}{IDCG_p(q)} \tag{2.31}$$

where:

- DCG_p : Discounted Cumulative Gain at position p for the system for query q;
- $IDCG_p$: Discounted Cumulative Gain at position p for the best run for the query q.

Now the metric value is comprised between 0 and 1. For evaluating the system on all the test queries we can set a given position p, calculate all the $nDCG_p$ for all the queries and take a mean of all the values.

2.9.5 Paired T-Test

Now that we have some evaluation metrics that can be used to evaluate our system and, in particular, to evaluate all the developed versions of our system obtained by changing some components in the whole pipeline, we need something that allows us to understand if the new systems are actually different from the starting one. More precisely, if we take a system A and we obtain the $nDCG_5$ for all the test queries and we do the same for an "improved" version of the system, system B, we obtain two lists of $nDCG_5$ scores, one for every system. Now we want to know if system B is actually better than A. For doing this we can take the average $nDCG_5$ value for the 2 score lists and if the difference between the two means is significant (the value obtained from system B is significantly higher) we can conclude that system B is better. However this approach is not so precise: what does it mean that the difference is significant? We need a more precise method to measure if the two runs are effectively different and, as a result, also the two systems are different. It is here that the paired t-test is used.
The paired t-test is a method used to study whether or not the mean difference between pairs of measurements is zero (or different from zero). In other words, we use the paired t-test to determine if we have enough evidence to say that the observed scores of system B are better just due to chance.
In order to employ the paired t-test to study differences between paired measures (in our case we are considering score pairs), the following assumptions must exist:

- The subjects must be independent. In our case, the measurements of one system must not affect the measurements of the other system.
- The measured differences must have a normal distribution: in our case, we can assume that this is true.

- No outliers in the differences between the two related groups. We can assume it is true since the two systems are not so different.

The hypotheses of the test can be expressed as:

- $H_0 : \mu_D = 0$: the population mean difference is zero, so the two score lists are statistically equal (Null hypothesis);
- $H_1 : \mu_D \neq 0$: the population mean difference is not zero, so the two score lists are not statistically equal.

From our point of view, $\mu_D = 0$ means that the two systems are statistically equal, instead $\mu_D \neq 0$ means that the two systems are statistically different. Let's assume that:

- **x** and **y** are the two $nDCG_5$ score lists;
- x_i or y_i identify the i-th element in the score lists;
- n is the dimension of the two lists.

First of all, we calculate the average difference between the two score lists.

$$\overline{x_D} = \frac{\sum_1^n (x_i - y_i)}{n} \tag{2.32}$$

Then we calculate the standard deviation of the differences:

$$SD_D = \sqrt{\frac{\sum_1^n (diff_i - \overline{X_D})}{n-1}} \tag{2.33}$$

where $diff_i = x_i - y_i$. After that, we have to fix a significance level α, usually $\alpha = 0.05$ gives good results. The general t-statistic is computed as:

$$t = \frac{\overline{x_D} - \mu_D}{SE} \tag{2.34}$$

$$SE = \frac{SD_D}{\sqrt{n}} \tag{2.35}$$

where SE is the standard error. To compute the $p-value$ we use a t-distribution with $n-1$ degrees of freedom, we then calculate the area under the curve in the interval $(-\infty, -|t|) \cup (+|t|, +\infty)$ by using a paired-t table. If our $p - value$ is less than our significance level α we reject the null hypothesis (H_0) and we conclude that the two score lists are different (and thus also the two systems) otherwise we accept the null hypothesis.

Chapter 3

Original Work

The focus of the whole reproducibility study documented in this thesis is the work conducted by Tengteng Lin et al. [11] in producing the ACORDAR collection and baselines. In this work the authors introduce ACORDAR: A Test Collection for Ad Hoc Content-Based RDF Dataset Retrieval and the reference paper describes the following phases and components:

- Datasets collection: all the collection datasets were retrieved from open data portals (most of them are open government data portals). For every dataset, the associated RDF files and metadata were crawled from the portal;
- Datasets browsing dashboard: in order to browse the retrieved datasets a dashboard was developed which can visualize the datasets metadata and some datasets content insights, such as graphs and triple patterns and graphs snippets based on different RDF graph summarization techniques;
- Queries creation;
- Relevance judgments creation;
- Development of the baseline methods used for testing the collection and obtaining the first results.

All the main details of the above points will be discussed in the next sections. In order to download the collection the creators have published a Github repository named ACORDAR [1] that contains:

- the `datasets.json` file: a JSON file with a list of all the datasets information. For every dataset all the metadata and download links for downloading the dataset files are indicated,
- the `all_queries.txt` file: a simple text file with all the queries. This file contains for every line the query id and the query text,

[1] https://github.com/nju-websoft/ACORDAR

- the `qrels.txt` file: text file with all the relevance judgments (expressed by using the TREC convention) obtained,
- a possible collection split into five folds for cross-validation,
- the runs of the baseline methods.

The considered version of the ACORDAR collection is the 1.0 and it is composed of: 31589 datasets (a very good number for a test collection), 493 queries and a set of relevance judgments. All the collection insights provided in this chapter are obtained by using the code inside the ARA (ACORDAR Reproducibility and Results Analysis) repository[2].

3.1 ACORDAR Collection Analysis

In this section, we are going to do an in-depth analysis of the collection to assess its overall quality for the task.

3.1.1 Metadata Analysis

In this subsection, we are going to analyze the metadata information associated to every dataset in the collection. The identified metadata fields found are:

- `dataset_id`: id of the dataset,
- `download`: a list of download URLs for the dataset files,
- `size`: the dataset dimension (considering all the download files),
- `license`: the license of the dataset,
- `author`: the author of the dataset,
- `created`: this field probably contains the date when the dataset was retrieved by the ACORDAR crawler,
- `updated`: this field probably contains the date when the ACORDAR crawler noticed a change in the dataset (metadata or data information) in the source portal,
- `description`: a description of the dataset content,
- `title`: dataset title,
- `tags`: a series of keywords associated to the dataset,

[2]https://github.com/manuelbarusco/ARA

- `version`: dataset version.

Not all the datasets in the collection have the previous fields available, indeed from the table 3.1 we can see the percentage of datasets that have a given field available: Another interesting

Metadata Field	Number of datasets	Percentage
dataset_id	31589	100 %
download	31589	100 %
size	12536	39.68 %
license	21362	67.62 %
author	19375	61.33 %
created	31589	100 %
updated	19265 5	60.98 %
description	27244	86.25 %
title	31589	100 %
tags	21746	68,84 %
version	85	0.27 %

Table 3.1: Availability of datasets metadata

information is the number of download links indicated for every dataset: the datasets have not a fixed number of download links associated. The number of links varies from a minimum of 1 to a maximum of 417. The key metadata fields that are used by the ACORDAR baselines and by the reproduced systems (presented in the next chapters) are: `dataset_id`, `title`, `description`, `author` and `tags`. It is important to note that `title` and `description` (which as can be seen from the table 3.1 are not always available), since they are crawled and probably scraped from the data portals web pages, must be cleaned from HTML tags, control characters, URLs and useless spaces: in general these two metadata fields are very noisy. This aspect must be taken under consideration, especially in the development of semantic search solutions based on the metadata.

3.1.2 Queries Analysis

In the collection 493 queries divided into two groups are reported: synthetic queries and TREC queries.

The synthetic queries (251) were obtained from 9 human annotators who were presented with 30 random datasets and, for every dataset, the annotators were asked to produce a keyword query for which that dataset was relevant. Thanks to the developed browsing dashboard and a mechanism that ensures the data-oriented nature of the query (explained in the paper [11]), all the queries are actually pointing not only to the datasets metadata but are also focused on the datasets content.

The TREC Queries (249) were imported from the TREC English Test Collection [19], these

queries are about general topics and are thus considered suited for the task and of great potential for finding relevant datasets: usually these queries are more general than the previous ones.
From a qualitative point of view, the number of queries is very good since the minimum is usually 250. With 493 queries, derived from 9 human annotators and from TREC queries, the right level of diversification is guaranteed to reflect the users needs.

3.1.3 Relevance Judgments Analysis

Obtaining a complete relevance judgments set would be infeasible due to the large number of datasets and queries. In order to reduce the number of required relevance judgments a pooling technique was used: all the datasets metadata and content were indexed by using an inverted index and all the queries were searched by using 4 retrieval models: TF-IDF, BM25, FSDM and LMD. The indexing and retrieval strategies used in the pooling were used also in the development of the baseline methods explained in the next sections. For each query and for each model a rank of 10 datasets was returned and the union of these ranks generate 15038 query-dataset pairs to be evaluated. The same 9 human annotators involved in the synthetic queries creation were asked to give a relevance score to all the retrieved pairs. The score is comprised between 0 and 2: 0 not relevant, 1 partially relevant and 2 relevant. A mechanism for ensuring a good and data-based evaluation was used and it is documented in the paper [11].
From a qualitative point of view, the qrels creation was not so good. Despite the relevance scores were multi graded the pool depth used in the pooling phase was too small (only 10). A typical pool depth, used in a lot of TREC collections, is instead 100. Fixed the pool depth, the maximum size of the pool of query-dataset pairs is:

$$max\_size = n_{runs} \cdot n_{queries} \cdot depth \tag{3.1}$$

where:

- n_{run}: number of runs conducted, in ACORDAR one for every retrieval model: 4,
- $n_{queries}$: number of queries, 493 in ACORDAR,
- $depth$: number of datasets retrieved in the final ranks (in ACORDAR only 10).

This problem can be seen from a quantitative point of view from the fact that, on average, the number of judged datasets for each query is 21.65, ranging from a minimum of 10 to a maximum of 36 datasets. Usually, in the TREC collections, for each query there are 300-700 judged documents. This problem is very important because a developed system tested on this collection can return for a given query a rank of datasets that may not have a relevance score associated: this will ruin the calculation of some evaluation measures.
Another problem is the distribution of the relevance scores in the qrels, indeed there is 65.1% of

query dataset pairs with a 0 score (dataset not relevant for the query), 22.1% with a 1 score and 12.8% with a 2 score. There is an imbalance in the score distribution that can cause problems during the systems development and evaluation.

3.1.4 General Purpose ontologies

After downloading the collection (the collection download will be entirely documented in Chapter 4) and after inspecting accurately the download links in the datasets list provided, one can see that some datasets contain some big and general-purpose ontologies. These ontologies contain a lot of information from different fields: intuitively this makes these ontologies (and thus the datasets that contain their files) relevant to a lot of queries due to the very wide range of domains that these ontologies can cover. The main detected ontology of this type is DBpedia, Wikidata is present only once with a custom version and FOAF can be considered as a general ontology but not so big: it was however briefly analyzed.
The FOAF ontology, after an in-depth analysis of the large number of datasets that contain it, is used to model individual and organisational relationships in the other RDF files contained in the datasets. It is actually imported from the other RDF files in the dataset as an auxiliary ontology. This aspect is also reflected in the qrels, where all the datasets that contain this ontology are relevant only to specific queries where the other RDF files are suited for it: the FOAF ontology files are thus not affecting the qrels.
The DBPedia ontology is present in some datasets in the ACORDAR collection and in particular, four different versions of it are present. The table 3.2 reports all the datasets that contain the DBpedia ontology files, for every dataset the queries to which is considered relevant and other information about its content are shown.

Dataset ID	**File**	**Qrels**	**Comment**
13461	dbpedia-3.5.1.owl	Not present	• DBPedia file: general • RDF files: about Berlin history
14530	dbpedia-2014.owl	• 1056 : ”English and Spanish terminology”, score: 1 • 1070 : ”Spanish Terminology”, score: 1	• DBPedia file: general • RDF files: about spanish terminology

Dataset ID	File	Qrels	Comment
14336	dbpedia-3.6.owl	• 218: ”Greek, philosophy, stoicism”, score: 0	• DBPedia file: general • RDF files: about Linux operating system
13369	dbpedia-3.6.owl	• 218: ”Greek, philosophy, stoicism”, score: 0	• DBPedia file: general • RDF files: about Linux operating system
15414	dbpedia-3.6.owl	• 1027: ”Canada Core Subject Thesaurus, English, French”, score: 0	• DBPedia file: general • RDF files: about Paris history
13338	dbpedia-3.6.owl	Not present	• DBPedia file: general • RDF files: about Paris history
14084	dbpedia-2016-04.owl	• 1167 : ”wikidata alignment”, score: 1	• DBPedia file: general • RDF files: about Berlin history

Table 3.2: Datasets that contain DBPedia Ontology files.

From the table 3.2 and by inspecting the content of the datasets it is possible to see that the datasets that contain the DBpedia Ontology use it as an external ontology for referring to general and standard concepts such as general classes, properties and so on: it is used mainly for its schema level information. From the qrels, we can see that these datasets are considered relevant to the queries only if the other RDF files contained in the dataset are relevant to the query.
Nevertheless, it is easy to understand that the presence of these general-purpose ontologies can give problems to a search engine because these datasets can be considered relevant to a lot of

queries. To confirm this fact another analysis was conducted on the ACORDAR baseline runs, in particular on the BM25 run based on the datasets content. In this run for every query the output ranks were analyzed to see if a dataset with the DBpedia ontology inside was returned in the top 5 positions. The results obtained are shown in the table 3.3 where for every dataset that contains the DBpedia ontology the queries for which the dataset appears in the top 5 positions of the output ranks are reported.

Dataset ID	**Query**	**Position**
13461	1127: ”cases for city planning 2013”	4
14530	55: ”Non-commercial Satellite Launches”	1
	1053: ”Australia Public Record Office wiki”	3
	1125: ”person name in National Library of Russia”	3
	44: ”International Military Equipment Sales”	4
	174: ”Native American casino”	4
	1115: ”monument type”	1
	21: ”Satellite Launch Contracts”	5
	176: ”Nobel prize winners”	5
	1013: ”music genre”	5
	1034: ”British National Bibliography, book and its author, topic, name, language”	4
	1156: ”building structure”	5
	1184: ”monument classification”	1
14084	55: ”Non-commercial Satellite Launches”	2
	1053: ”Australia Public Record Office wiki”	4
	1125: ”person name in National Library of Russia”	1
	174: ”Native American casino”	3
	1034: ”British National Bibliography, book and its author, topic, name, language”	2

Table 3.3: Datasets that contain the DBPedia Ontology that are returned in the top 5 positions in the ACORDAR BM25 run ranks based on the datasets content.

From the table above it is possible to see that the DBPedia RDF files sometimes confuse the baseline search engine developed by ACORDAR: it considers the datasets that contain these files as relevant for the wrong queries. The main example is the query ”Non-commercial Satellite Launches” for which the datasets 14530 and 14084 are returned in very high positions because their DBPedia file matches the keyword query. This match boosts a lot the dataset score despite the content of the other dataset RDF files is not matching the query. The problem is that the DBPedia RDF files are used as external ontologies and the other RDF files, that

are actually defining the main content of the dataset, are not considered with a higher level of importance. This is a problem that must be tackled in some way because it can ruin the system performances: the main RDF files that define the dataset content must be highlighted in some way or the DBPedia files must be removed from the datasets content during the indexing phase (and returned only after the search if the user needs that dataset, and must therefore be returned entirely).

3.1.5 Database Dump Datasets

Another important aspect that arises from an accurate analysis of the collection after the download phase is the presence of a lot of `rows.rdf` files inside the datasets. These files are downloaded from different portals and are a particular type of RDF files because they are not real RDF datasets or ontologies but are RDF files that contain a dump of tabular datasets such as relational databases. A lot of open portals make the user select the favourite format to download the dataset, but, if the dataset is for example a CSV table, the dataset dump in RDF will reflect this tabular structure and thus the RDF file is not a real ontology or a knowledge repository but instead a list of items. In this kind of RDF files are present a lot of entities (one for each row of the original table), with as many properties as the attributes specified in the original tabular view. One example is the dataset 8006. This dataset contains a list of closed discrimination cases managed at the Seattle Office for Civil Rights in 2018. From the qrels this dataset is relevant for the following queries:

- 1237 : ”Discrimination Case”, score: 2.
- 1121 : ”Closed cases in September 2018”, score: 2.
- 1142 : ”Housing cases in July 2018”, score: 2.

and by looking at the `rows.rdf` file contained inside the dataset we notice the presence of the structure just described. By inspecting the download request of the file [3] one can note that a tabular view of the dataset is requested and it is also possible to view the tabular structure that the RDF file is representing.
In this case converting to an RDF file probably is not the best solution for searching this type of datasets and other solutions based on schema level information of CSV or SQL structures are preferred [4]. The presence of this kind of datasets in the collection is however important because a lot of RDF files are obtained in this way from the open data portals so it is important to consider also this possible aspect.

[3] https://data.seattle.gov/api/views/qsyq-cvfs/

3.1.6 Duplicated Datasets Analysis

The last aspect that arises from the collection download and recovering phases (documented in the following chapter) and from an accurate analysis of the `datasets.json` list is the fact that some datasets are duplicated. The first thing that one can notice is that some datasets download the same files and sometimes these datasets have also the same metadata associated. In the following two subsections, we are going to analyze this aspect in detail because it is present in a strong way inside the ACORDAR collection and must be taken into consideration.

3.1.6.1 Duplicated Datasets

We are going to consider the following definition: "two or more datasets are considered duplicated if they share the same title, description, author, tags and links fields in the `datasets.json` list". All the steps of this analysis are completely documented in the `duplicated_datasets.ipynb` notebook in the ARA repository: at the end of this analysis 80 groups of duplicated datasets are found: in every group (composed by a minimum of 2 to a maximum of 6 datasets) the datasets inside share the same metadata fields considered in the definition. After this analysis, all the datasets inside a given group can be "merged" into one dataset and 118 datasets could be potentially removed from the collection. These datasets differ only on the dataset id (obviously) and on the other minor metadata fields such as `updated_at` and `created_at`: there was probably a problem in the collection creation and in particular in the crawling phase where a procedure of collection updating was not implemented and this is more clear in the following analyses.
Another important aspect of these duplicated datasets is that they are actually affecting also the collection qrels. Indeed, by analyzing the collection qrels, one can notice that the datasets inside a given group of duplicated datasets sometimes are reported in the qrels with different relevance scores in the same query. In particular in the following list are reported which duplicated datasets have different relevance scores for a given query:

Query ID	Dataset	Score
1237	596	2
	1012	None
1122	1937	None
	3328	None
	3802	2
	7794	None
	9541	None
1138	3313	0
	7353	1
90	3332	None

Query ID	Dataset	Score
	4803	None
	6463	0
153	10445	0
	1080	None
	11413	0
1070	13530	None
	15050	0
1034	13530	None
	15050	0
99	13530	None
	15050	0
54	13530	None
	15050	0
104	13579	0
	14924	1
1056	13611	0
	15054	1
1062	13765	0
	15344	None
79	13765	0
	15344	None
1044	13765	0
	15344	None
252	13907	1
	15270	0
1125	13907	0
	15270	None

Table 3.4: Duplicated Datasets with different relevance scores in the qrels.

The ”None” score indicates that there is no relevance score indicated for the dataset in the qrels. In the baseline runs submitted by ACORDAR it is possible to see that these duplicated datasets are not treated (for example merged into one single dataset) and are considered as different in all the runs. For example in the following run:

- Run name: BM25[m]
- Query: 1090

- Output Rank: [13568, 15014, 13582, 14491, 13290, 13622, 15092, 14921, 13577, 14050]
- Datasets Scores: [13.94, 10.43, 5.21, 5.21, 4.77, 4.46, 4.46, 3.47, 3.47, 1.60]

are returned the datasets: [13622, 15092] that are considered as duplicated but, as indicated in the rank, they are returned next to each other in the final rank and also with the same score.

3.1.6.2 Datasets with the same download links

A complementary analysis that can be conduced after the previous one is done by considering the datasets that share the same download links: these datasets are actually downloading the same files and it is interesting to see if the metadata associated to these datasets are different or not, meaning that we are considering different aspects of the same files in different datasets.
By considering only the download links and by removing the 80 groups of datasets found in the previous analysis a list of 360 couples of datasets are found, where the datasets in the couples have the same download links but have other different metadata fields. The problem is that it is difficult to quantify how these fields differ from each other from a semantic point of view.
A first possible analysis can be made on the tags field, in this case 48 of the previous datasets couples presents datasets that differ a lot in the tags: the two datasets tags are simply considered as two sets of keywords and are compared by using the Jaccard Similarity (distance):

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.2}$$

where A and B are the two sets, in our case the two sets of tags obtained from the two datasets. The distance value can range from 0 to 1, 0 means that the two sets are totally different (they do not share any value) and 1 means that the two sets are completely equal. A value of 0.7 was chosen. However this analysis is not complete from a semantic point of view: the tags are compared directly as strings and not as semantic keywords, thus the tags can be different in the string form but they could have the same meaning for example.
A second analysis can be made on the title and description fields. In this case, the analysis is made by concatenating the title and description of every dataset in the couple and the strings obtained are then split by considering the space as separator. The two strings obtained from the two datasets are thus considered as two sets composed essentially by their words (roughly obtained from the space split). In order to measure a difference between these two strings that can allow some sort of tolerance, the Jaccard distance is used. In this analysis, if the Jaccard distance is less than 0.7 the two datasets are considered very different in description and title. From this analysis, only 7 couples of datasets are considered very different.
Unfortunately from the tags and title + description analysis and by inspecting the retrieved datasets couples, it emerges that the differences in these datasets are not about content but are about small additions and changes in the metadata fields. This is in line with what was said

in the previous section: probably in the crawling system a mechanism of datasets merging and updating was not implemented, so, if there was a difference in the tags or other metadata fields of the dataset, instead of updating it in the json list, a new version of the dataset was registered with the new info. This is a big problem since now the number of duplicated datasets from a semantic point of view is very big and this decreases the quality of the collection.

3.2 ACORDAR Developed systems

In order to test how suited the ACORDAR collection was for the task of Ad Hoc Datasets Search, 4 retrieval models were tested on the collection and they defined a sort of baseline methods and results for the task. The results obtained from the ACORDAR runs based on these models are reported in the Table 3.5.

Model	**NDCG@5**	**NDCG@10**	**MAP@5**	**MAP@10**
TF-IDF	0.5088	0.5452	0.2871	0.3976
TF-IDF [m]	0.4743	0.5019	0.2676	0.3685
TF-IDF [d]	0.1910	0.1963	0.0998	0.1199
BM25F	0.5538	0.5877	0.3198	0.4358
BM25F [m]	0.5045	0.5250	0.2859	0.3838
BM25F [d]	0.2163	0.2196	0.1385	0.1550
FSDM	0.5932	0.6151	0.3592	0.4602
FSDM [m]	0.4853	0.4958	0.2770	0.3516
FSDM [d]	0.2497	0.2606	0.1478	0.1758
LMD	0.5465	0.5805	0.3266	0.4324
LMD [m]	0.4363	0.4573	0.2543	0.3325
LMD [d]	0.2398	0.2523	0.1415	0.1672

Table 3.5: Original results.

As can be seen, three search configurations were developed (based on datasets metadata, data and metadata + data) and 4 retrieval models were used (BM25, FSDM, LMD and TF-IDF). The main purpose of this thesis is to obtain these values by developing a system that reproduces in the most similar way all the steps that can be inferred by the ACORDAR paper and repositories.

Chapter 4

Developed System

In this Chapter, we dive into the details of the developed system built for reproducing the ACORDAR baselines. We analyze all the implementation details of every phase from the download to the searching phase. In particular, the download and parsing phases are conducted by the Python and Java codes contained in the ADE (Acordar Download and Extraction) Repository[1] and the indexing and searching phases are conducted by the Java code contained in the ARDFS (Acordar RDF Search) repository[2] by using the Lucene Library. Each repository contains a `README.md` file that describes how to execute the code and lists all the command line arguments needed to start the scripts.

4.1 Download

The download phase comprises three subphases: download, download retry and download checking.

4.1.1 Download

The download subphase is conducted by the **`downloader.py`** script. The main goal of this script is to read the datasets list in the `datasets.json` file provided in the ACORDAR collection and download all the datasets in a given directory (the directory path is provided as a command line argument to the script). For each dataset, the script creates a directory named "dataset-dataset_id" which includes:

- all the files downloaded from the download links associated with the dataset,
- a new file named **`dataset_metadata.json`** that contains the following dataset metadata: dataset ID, title, description, author and tags. These fields are cleaned from control

[1] https://github.com/manuelbarusco/ADE/tree/main
[2] https://github.com/manuelbarusco/ARDFS/tree/main

characters and HTML tags. This file also contains a list of successful download mappings (URL → file name in the directory), a list of failed links for the dataset download and a summary of the download info (total number of links associated to the dataset and the number of successfully downloaded links).

All the errors encountered by the script during the download are reported in a log file. Every encountered error is reported by indicating the ID of the dataset, the download link associated to the dataset that caused the error and the error message. For every link, the name of the associated file is assigned based on the HTTP header "Content-Disposition" if available or from the URL string. The script has a resume mechanism for stopping and resuming the collection download: the download of the whole collection took more or less 36 hours.

4.1.2 Download Retry

The download retry subphase is conducted by the **download_retry.py** script. The main purpose of this script is to retry the download for all the error links reported in the log file of the previous subphase. Indeed it is possible that some download errors have been caused by a server overload or maintenance or for a temporal problem (for example a given resource is not available at the moment). If a given error link works, the script will download the link file in the correct dataset folder and it will update all the download-related information in the `dataset_metadata.json` file in the dataset. If a given error link still does not work, it will log the error with the same format adopted in the **downloader.py** script in a log file.

4.1.3 Download Checking

The download checking subphase is conducted by the **check_datasets.py** script. The main purpose of this script is to clean up the downloaded file names from the URL noise and to report in a log file all the files that do not have a valid RDF extension. In the log file it reports all the non-RDF files names and their MIME types inferred by the python Magic library[3].

4.2 File Recovering

The main purpose of this phase is to curate the downloaded files. In particular, the steps of this phase are:

1. run the **simple_file_recover.py**: this script tries to assign a possible RDF extension to all the files that has an empty one by using the Mime Type reported in the download checking phase and by inspecting their content.

[3]https://pypi.org/project/python-magic/

2. manual recovering: at this stage all the files that have not one extension yet are checked, starting from the files reported in the `simple_file_recover.py` log file and in the download checking phase. All the archives are extracted and their content is checked by assigning an extension to all the files that have an empty one (or a not RDF extension).

3. BOM Checking: the **`bom_checker.py`** script checks if a RDF file contains a UTF8 BOM sequence at the beginning. This convention sometimes gives problems to the RDFLib parsing phase so we want to remove it.

After this phase, the collection is ready to be parsed. It is worth noting that at the end of this phase, all the files that have not an RDF extension (for example HTML files) or still have an empty extension, are not deleted but are simply not parsed and excluded from further processings.

4.3 Parsing

The main focus of the parsing phase is to extract from all the datasets RDF files the needed data to reproduce the ACORDAR baselines. In particular, we have to extract for every dataset all the entities, literals, properties and classes from all its RDF files. To get closer to the ACORDAR baselines, three possible parsing strategies have been developed. This is due to the fact that the parsing strategy used by ACORDAR is not explained in a very clear way neither in the paper, nor in the Github repositories (ACORDAR and ACORDAR 2.0[4]). In the following sections all the parsing strategies are analyzed, the order in which the various parsing strategies are explained coincides with the order in which the various strategies have been developed over time. All the parsing strategies have been developed by considering the limited available resources for the experiments.

4.3.1 Basic Parsing

This parsing strategy was the first developed and it is composed by 3 parsers:

1. **`JenaAcordarExtractor.java`**: this parser, written in Java, is a stream parser that uses Apache Jena in order to parse the RDF files. The key value of this parser is that it parses all the RDF files in a stream way, triple per triple, allowing to get good performances. Because of the low resources available for the experiment, this parser is limited to 300 MB RDF files to not break through the Java Heap Memory limits. The problem with Jena is that it strictly manages the syntax errors in the RDF files: a space in the URI is enough to stop the parser. If this parser encounters a syntax error, it blocks the parse. Fortunately, this means that, until the first error, it is possible to extract some information from the file.

[4]https://github.com/nju-websoft/ACORDAR-2

2. **rdflib_extractor.py**: this parser is based on the Python RDFLib library. This parser was developed to parse all the RDF files with syntax errors that were blocking the previous parser. Indeed, after some experiments, one can see that the RDFLib library is less strict than Apache Jena and it allows to parse files that have some little errors (such as space in the URIs). The main problem of this parser is that every RDF file must be completely loaded into the RAM before being queried with SPARQL to extract the needed information. Because of the low resources available for the experiment, this parser is limited to 100 MB RDF files.

3. **lightrdf_large_extractor.py**: this parser, written in Python, is based on the LightRDF library. This library, written in Rust, is very efficient and allows to parse very big RDF files in a stream fashion and in a very efficient way also in a low-resource environment. This parser parses all the RDF files up to 4 GB.

Every parser listed above reports in a separate log file all the problematic RDF files (files that are causing errors in the parsing). Every parsing error is reported with the following information: dataset ID, error file and error message thrown by the parser.
The execution flow of this parsing strategy is the following: the collection is parsed with the `JenaAcordarExtractor.java` parser, than the `rdflib_extractor.py` parses all the problematic RDF files from the previous step and then all the big RDF files not yet parsed are parsed by the `lightrdf_large_extractor.py` script.
Both the `JenaAcordarExtractor.java` and the `rdflib_extractor.py` parser output the extracted information in a json file named `dataset_content_jena.json` or `dataset_content_rdflib.json` with 4 lists (one for entities, literals, classes and properties). Instead, the `lightrdf_large_extractor.py`, due to the dimension of the files that it parses and the low RAM resources available, reports all the extracted information in separate text files: one for classes, entities, properties and literals. In this case, the classes, entities, literals and properties are reported once per line.
The above parsers extract and insert in the output lists the classes, properties and entities URIs and the literal values.
The Basic Parsing strategy was the first implemented and it was developed before the ACORDAR 2.0 repository release: all the information available for the development was in the paper. The main focus of this attempt was to parse the largest number of files and obtain the first results from which to start the next improvements.

4.3.2 Labels Parsing

This parsing strategy was developed to study which information the ACORDAR baselines were extracting from the RDF files. Indeed, from the ACORDAR 2.0 repository code (and from the

paper) it is not so clear if they are extracting, for example for an entity, the URI or a ”label”. This label concept is not well defined and it is difficult to understand if they are referring to the `rdfs:label` or other types of labels. In order to tackle this problem the labels parsing strategy was developed.
This parsing strategy is conducted by the **`rdflib_labels_extractor.py`**. This parser, written in Python and based on the RDFLib library, mines the datasets files in a quite different way with respect to the previous parsing strategy. Indeed, since this parser is based on the RDFLib library, it is quite slow because it has to load the whole RDF graph in the RAM before doing the SPARQL queries. In order to speed up the parser, for each dataset, the RDF files are considered in the following parsing order: if multiple files are present with the same name and different extensions (for example in the dataset 15414 there are the files: paris.nt, paris.n3 and paris.rdf), these files are ordered by dimension and only the biggest file is mined. If the biggest file has syntax problems the parser goes for the second heaviest file and so on. The limit size for the RDF files is always 100 MB. This parsing strategy can be considered as an attempt to do some sort of triple deduplication since multiple files with the same name and different extensions may contain the same information and thus the same triples. The development of this parsing strategy tries to generalize the label concept by not considering only the `rdfs:label` property. In fact, after analyzing some RDF files in the collection, we notice that some entities, classes and properties have a name or a label associated by using some custom properties defined in that ontology.
The queries used to extract the labels for classes, entities and properties are the following:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT DISTINCT ?p ?hdp
WHERE {
    ?s ?p ?o .

    OPTIONAL {
        ?p ?lp ?hdp .
        FILTER isLiteral(?hdp) .
        FILTER(REGEX(str(?lp), "(name|description|label|title)"
           , "i"))
    }

}
```

Query 4.2: Labels Parsing: SPARQL query for the properties extraction

As can be seen from the previous queries, if a label is available it is returned, otherwise only the URI (of the entity, class or property) is returned. Thus the lists returned by the parser in the

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?s ?hds ?c ?hdc
WHERE {
    ?s a ?c .

    FILTER(!isBlank(?s)) .

    OPTIONAL {
        ?s ?p ?hds .
        FILTER isLiteral(?hds) .
        FILTER(REGEX(str(?p), "(name|description|label|title)",
            "i"))
    }

    OPTIONAL {
        ?c ?pc ?hdc .
        FILTER isLiteral(?hdc) .
        FILTER(REGEX(str(?pc), "(name|description|label|title)"
           , "i"))
    }
}
```

Query 4.1: Labels Parsing: SPARQL query for the classes and entities extraction

output json file (the file structure is the same as before) can contain both labels and URIs for classes, entities and properties. The literals list contains the literals values as before. All the errors encountered by the parser are reported in a log file.

4.3.3 Stream Parsing with Triple Deduplication

This parsing strategy was developed after analyzing in an exhaustive way the indexing code provided in the ACORDAR 2.0 repository. Furthermore, another project developed by the ACORDAR authors was deeply reviewed: CADDIE, a Content-Based Ad Hoc RDF Dataset Retrieval system[5]. The CADDIE repository contains the same indexing code present in the ACORDAR 2.0 repository but with more details that allow us to understand how to parse the RDF files and which information is actually indexed.
After the code analysis, it emerges that the ACORDAR systems parse the triples in a pure stream manner one after the other. Thus, in the previous parsing strategies, the `rdflib_extractor.py` parser or the labels parser were not in line with this idea and could probably move the system away from the baseline. Furthermore, during the parsing, the blank nodes are discarded.
After this in-depth analysis also the triple deduplication mechanism cited in the paper began to

[5]https://github.com/nju-websoft/CADDIE

emerge. The SQL table in which all the RDF triples are stored by the ACORDAR baselines allows the triples deduplication but, beyond that, another system is considered: the MSG (Minimum Spanning Graph) Labeling. This labelling mechanism assigns a code to every triple in a given RDF graph by considering the Minimum Spanning Graphs that compose it. First of all, the mechanism identifies all the MSGs in the whole RDF graph and then a code is assigned to each of them: every triple that belongs to a given MSG inherits the code. This triple deduplication mechanism is used when a dataset contains multiple RDF files because all those files together define the whole RDF graph associated with the dataset. During the indexing all the datasets triples are retrieved and indexed by considering their MSG codes: if the triple is assigned to an MSG code that was already seen during the indexing it is discarded. The whole RDF graph is thus indexed by MSGs: if the triples of a given MSG have already been indexed, they are not considered anymore.
An easier but still effective triple deduplication method was developed and implemented in the `JenaAcordarExtractor.java` (a new version of the parser was developed) and `lightrdf_extractor.py` parsers. These parsers read all the triples from all the RDF files associated to a dataset and insert them in a set: every triple is a tuple composed by the 3 URIs of subject, predicate and object (or the literal value). This allows us to deduplicate the triples in a fast but effective way.
In order to follow all these aspects, a new parsing strategy was developed and it comprises:

1. **`JenaAcordarExtractorDeduplication.java`**: evolution of the `JenaAcordarExtractor.java` parser: now it comprises the triple deduplication mechanism and it discards the blank nodes.

2. **`lightrdf_extractor_deduplication.py`**: this parser, written in Python and based on the LightRDF library, parses all the RDF files up to 1 GB that raise any errors to the previous parser. It comprises triple deduplication and it discards the blank nodes. To parse the big RDF files (bigger than 1 GB) as well, this parser reads and does triple deduplication only for the first 500.000 triples.

As before, every parser reports in separated log files all the problem files (files that are causing errors in the parsing). Every parsing error is reported with the following information: dataset ID, error file and error message thrown by the parser.
The execution flow of this parsing strategy is the following: the collection is parsed with the first parser and then the second parser parses all the problematic and big files for the first one. The output files are the same json files as before.

4.3.3.1 Triple Deduplication and Labels

Since, as can be seen from the results shown in the next chapters, the stream parsing with triple deduplication was better than the previous strategies, a small variant was developed in

order to include once again the labels concept in the index. This variant is contained in the **JenaAcordarExtractorDeduplicationLabels.java** and **lightrdf_extractor_deduplication_labels.py** codes. During the triples deduplication (same as before) if some triple is formed by: subject URI, label-property and literal, the literal is considered as a label for the URI subject, so a map URI → label is constructed. The label-property is every property that contains the keyword ”label” or ”name” in its URI. The constructed map is then considered during the output json file creation, where classes, entities and properties are saved using their URIs or labels previously retrieved. The parsers are the same as before with only this implementation addition. Also the execution flow, the output format, and the errors reports remain the same.

4.4 Indexing

The indexing phase is conducted by the various versions of the **DatasetIndexer.java** code. All the versions are written in Java and are based on the Lucene library for creating the index. Three distinct versions of the `DatasetIndexer` exist, each tailored to a specific parsing strategy. This differentiation is essential because each parsing strategy generates distinct output files: this ensures clarity and prevents confusion during testing.
All the `DatasetIndexer` versions scan the datasets directory and index the datasets content and metadata by reading the json and text files produced by the parsers for the dataset content and the `dataset_metadata.json` file for the dataset metadata. The `DatasetIndexer` allocates one Lucene Document for each dataset, and each document contains four metadata fields (title, description, author and tags) and four data fields (entities, literals, classes and properties). This implementation detail is very clear from the paper [11] and from the ACORDAR 2.0 repository. Both the data and metadata fields are indexed with the following options (found in the ACORDAR 2.0 repository indexing code):

```
type.setStored(true);
type.setTokenized(true);
type.setStoreTermVectors(true);
type.setStoreTermVectorPositions(true);
type.setIndexOptions(IndexOptions.DOCS_AND_FREQS_AND_POSITIONS)
```

Query 4.3: Metadata Fields indexing options

A lightweight version of the index without Positions Vectors storing and without field storing can be used if someone wants to use only the BM25, LMD and TF-IDF similarities; for the FSDM ranking these indexing options are needed. Furthermore from the ACORDAR 2.0 repository it is evident that a Lucene StandardAnalyzer is used with an NLTK stoplist (provided in the repository).

4.5 Searching

The searching phase is done by the **DatasetSearcher.java** and the **FSDMRanker.java**. The DatasetSearcher, written in Java and based on the Lucene library, searches over the previously created index. This searcher reads all the queries provided by ACORDAR in the all_queries.txt file and searches for every query in the index. The searcher uses the TF-IDF, BM25 and LMD Lucene built-in similarities and it performs the search by considering three possible search configurations (as reported in the ACORDAR paper [11]):

- **Metadata configuration [m]**: the query is searched only in the dataset metadata fields,
- **Data configuration [d]**: the query is searched only in the dataset data fields,
- **Combined configuration [m+d]**: the query is searched in all the dataset fields (metadata and data fields).

The search is always performed using boosting weights. In particular, for every configuration ([m], [d] and [m+d]), the ACORDAR 2.0 repository provides a set of weights which takes into account the fields considered for the given search configuration. These boosting weights have been fixed, as reported in the paper [11], with a parameter tuning process based on 5-Fold Validation. The used folds are provided in the repository ACORDAR repository. The searcher returns one different run file for every similarity and searching configuration used. For example for the BM25 similarity the following run files are returned:

- BM25Boost[m].txt: BM25 Similarity on the metadata fields with boosting,
- BM25Boost[d].txt: BM25 Similarity on the data fields with boosting,
- BM25Boost[m+d].txt: BM25 Similarity on all the fields with boosting.

In order to implement also the FSDM ranking criteria the **FSDMRanker.java** was developed. This ranker, written in Java, was developed by following the FSDM implementation provided in the ACORDAR 2.0 repository, the code was reviewed and modified a little bit in order to fit the whole pipeline but the main operation has been maintained. Rather than implementing a similarity based on the FSDM criteria, this ranker is actually a Re-Ranker. It searches the datasets by considering two subphases: in the first subphase it retrieves 10 datasets by using a BM25 similarity and then it re-orders these 10 datasets by using the FSDM criteria. The FSDMRanker reads all the queries provided in the all_queries.txt file and searches them in the index by considering the [m], [d] and [m+d] possible search configurations and by using the boosting weights provided. However, one important thing to note in this FSDM implementation is that the FSDM weights provided do not respect the weights rule present in the FSDM paper

[27]:

$$\sum_{j} w_j = 1 \tag{4.1}$$

where $j = 1, ..., F$ where F is the number of fields. Indeed, if we take a look at the FSDM weights for the [m+d] search configuration, we have for the fields: ”title”, ”description”, ”author”, ”tags”, ”entity”, ”literal”, ”class”, ”property” the following weights in order: 1.0, 0.1, 0.5, 0.9, 0.1, 0.1, 0.4, 0.6. This problem can be noticed also in the boosting weights of the other search configurations.

Chapter 5

Reproducibility Problems

In this Chapter, we analyze all the reproducibility problems encountered during the development of the solutions. In particular, there are two areas where the problems are present: the download and parsing phases. We analyze the problems of each of these phases separately.

5.1 Download

After the downloading phase, the following statistics are obtained:

- Total number of tried links: 34283;
- Number of links successfully downloaded: 28982;
- Number of error links: 5301;
- Number of datasets: 31589;
- Number of full downloaded datasets: 26637;
- Number of partially downloaded datasets: 86;
- Number of empty downloaded datasets: 4866.

We consider a dataset:

- fully downloaded: if all the links associated to the dataset are correctly downloaded;
- partially downloaded: if not all the links associated are correctly downloaded;
- empty downloaded: no link was correctly downloaded for the dataset.

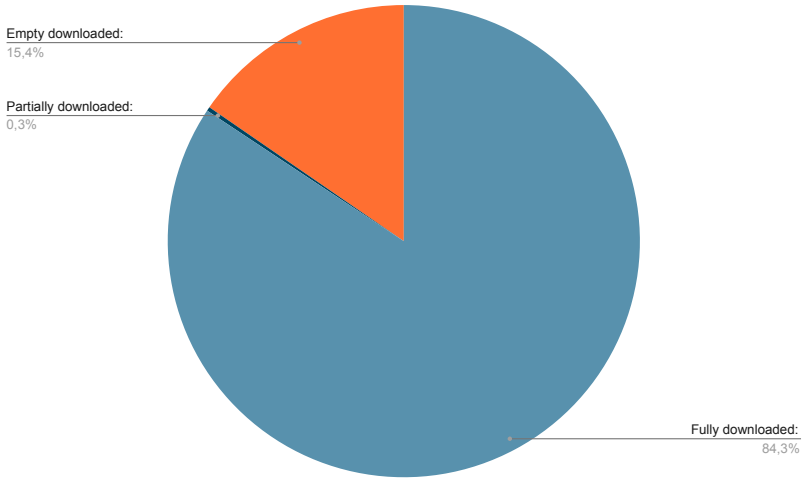


Figure 5.1: Empty, partially and fully downloaded datasets pie chart

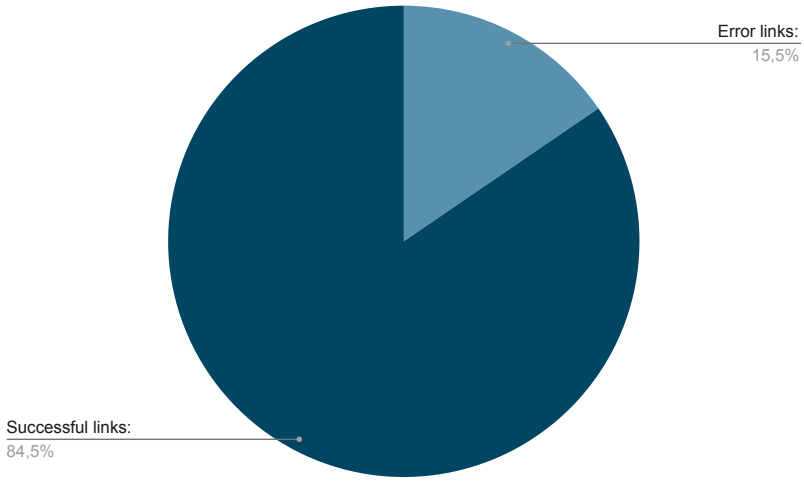


Figure 5.2: Error and successful download links pie chart

As can be seen, there are a lot of empty datasets, so datasets with only the metadata available for them and no possibility to extract data information because there are no RDF files (and in general files) associated to them. This can be a problem for the reproducibility study because we are actually testing the developed systems in an incomplete version of the ACORDAR collection. The same applies to the partial datasets. In the ACORDAR paper [11] the 31589 datasets reported in the `datasets.json` list are considered without download problems, so the ACORDAR experiments were conducted in a collection with 31589 full downloaded datasets. The error links encountered during the download phase throw different errors, in particular, the following HTTP errors are reported:

- 504 Server: Gateway Timeout;
- 500 Server: Internal Server Error;

- 503 Server: Service Unavailable;
- 502 Server: Bad Gateway;
- 404 Client: Not Found;
- 406 Client: Not Acceptable;
- 403 Client: Forbidden;
- 410 Client: Gone;
- 400 Client: Bad Request.

The main problem is that we are downloading all the datasets by using a list of URLs that can be accurately maintained or not over time. Indeed a given link can be valid one day and not the day after due to changes in the URL, in the server configuration, in the permissions or in the access mechanism and this is a clear problem highlighted in the HTTP errors encountered. There are a lot of factors that can cause problems with this mechanism for the collection download and it is quite a big problem because the datasets metadata are always present but the datasets content provided by the downloaded RDF files is not always available: this is a problem for a collection that will push the development of RDF datasets search systems based on the datasets content. Apart from retrying the download for the error links, there are no other solutions that can be tried to overcome the problem.
Furthermore, another aspect that must be considered with this download mechanism is that the download content associated to a given link can vary over time: this means that a given download link can download a given version of the dataset today but probably another different version tomorrow so the collection is not ”frozen” to a given ”version”. This is obviously another reproducibility problem because we do not know which datasets versions were used in the ACORDAR tests.

5.2 Parsing

During the parsing phase, two main reproducibility problems arise:

- Syntax errors during the parsing of the RDF Files.
- Unknown parsing strategy used by the ACORDAR baselines.

5.2.1 Syntax Errors during parsing

During the execution of every parsing strategy some syntax errors were encountered: the downloaded RDF files present a variety of syntax errors such as spaces in the URIs, triple definition rules not respected or external ontologies not correctly imported.
In the ACORDAR paper [11] Apache Jena is suggested for parsing the files and no problem regarding the parsing was reported but, during the development and testing of the solutions, some RDF files were not parsed due to syntax errors. In order to solve the problem RDFLib was introduced in some parsing strategies because it is less strict and allows the parsing of RDF files with spaces in the URIs and other small syntax errors that do not compromise the semantic integrity of the RDF file. Also LightRDF is less strict than Jena and it was added mainly because it is very efficient (it is based on the RIO engine[1] that is written in RUST) and allows parsing big RDF files without using a lot of resources. All the details about how many files were parsed by every parsing strategy are provided in the next chapter.
In the end, all the RDF files that have syntax errors were not considered because it was impossible to parse them in a complete way: the variety of errors is too big to develop a parsing strategy that can always work. Some solutions based on reading the RDF files as text files and parsing them by using Regex or other libraries of text mining and processing were developed but they do not provide a solid solution: it was difficult to obtain a list of classes, entities, literals and properties with correct information inside.
These syntax errors are a problem for the reproducibility because ACORDAR is not reporting any problem in the parsing so one should assume that there were not problems of this type and all the RDF files were parsed correctly. Despite this, as can be seen from the parsing statistics reported in the next chapter a lot of RDF files have been parsed correctly so it is not a big problem.
In the upcoming chapter, we will provide information about the number of correctly parsed RDF files and other useful statistics for analyzing the problem better.

5.2.2 Unknown ACORDAR parsing strategy

The parsing strategy used by the ACORDAR baseline systems is actually not known. In order to develop the most faithful solution possible to the ACORDAR baselines the indication provided in the paper and the code found in the repositories were followed, to clean some doubts also the CADDIE repository was analyzed. All the information that was inferred from the code and the paper is summarized in the following points:

- the RDF files are parsed triple per triple. This is confirmed by the indexing code in the ACORDAR 2.0 and CADDIE repositories, where the content of every dataset is retrieved

[1] https://github.com/oxigraph/rio

from a relational database (structure not provided by ACORDAR but retrieved from the CADDIE repository and inferred from the indexing code) where all the triples are stored. Every triple is associated to a dataset and every triple element (subject, predicate and object) is associated to a URI and a label (not well defined).

- there is a triple deduplication phase to remove from the datasets all the possible duplicated triples. This may occur if a given triple is present in different RDF files associated to the dataset (for example in the dataset 15414 there are the files: paris.nt, paris.n3 and paris.rdf, these files can be complementary to each other but can also contain some overlapping triples). This triple deduplication mechanism is used only if the dataset has more than one RDF file associated. This phase is done by using a relational database (it deletes the duplicates in a straight way) and, from what can be inferred from the repositories, by using a code that is assigned to the triples (named MSG) by considering the Minimum Spanning Graph. The labeler is not mentioned in the ACORDAR paper and repository but it is present in the CADDIE repository.

- it is not clear how the classes, entities, literals and, properties are indexed, in the ACORDAR and CADDIE indexing code a URI but especially a label concept is introduced but it is not clear what it is referring to, to the `rdfs:label` or to another kind of label.

- every dataset is indexed by considering 8 fields: 4 for the metadata and 4 for the content, used also for implementing the different search configurations.

Since the ACORDAR parsing strategy is not well defined, neither in the paper nor in the repository code, it can only be guessed through several attempts and solutions based on different ideas.

Chapter 6

Results Analysis

In this Chapter, we analyze the results of all the attempts conducted to reproduce the ACORDAR baselines and try to reach the results declared in the paper. After explaining how metadata-based results were used to fix the indexing and searching parameters, the results obtained by using the different parsing strategies on the data and combined search configurations are shown. The collection is parsed after the manual recovering phase previously described and it contains after that stage 28476 RDF files and 31589 datasets. All the experiments were conducted with the following hardware resources:

- Intel Core i5 4690k;
- 8 GB RAM + 5 GB Swap;
- 500GB HDD + 256 GB SSD.

and the following software and libraries versions:

- Ubuntu 22.04;
- Java 19 (Java Heap Memory: 5 GB);
- Apache Lucene 9.0;
- Apache Jena 4.7.0;
- RDFLib 6.3.2;
- LightRDF 0.3.1.

All the metrics shown in this chapter are obtained by using the `trec_eval` tool[1], the standard tool used by the TREC community for evaluating an ad hoc retrieval run given the run file and the qrels file.

[1]https://github.com/usnistgov/trec_eval

6.1 Metadata Results

In order to fix the indexing parameters (such as the Analyzer and Stop list used by the ACORDAR baselines) the metadata search configuration and the ACORDAR original results based on the metadata were considered. Since the metadata are completely available for all the datasets the runs based only on them can be used as a benchmark to tune these parameters without problems or other affecting factors such as missing RDF files.
In the tables 6.1 and 6.2 the results obtained by all the retrieval models on the runs based only on the metadata are reported. For every retrieval model the baseline and reproduced values of nDCG@5, nDCG@10, MAP@5 and MAP@10 are reported, along with the difference between the reproduced and the original values: if the difference is negative it means that we are under the baseline value, otherwise we are above it. This table structure will be used also in the rest of the chapter to show the other results.
Table 6.1 contains the results obtained by using the Lucene Standard Analyzer and Lucene standard stoplist while table 6.2 contains the results obtained by expanding the Lucene stoplist with the NLTK stoplist (the configurations used are also reported in the tables descriptions). From the results, it is clear that the baseline system uses a Lucene Standard Analyzer combined with a Lucene plus NLTK Stoplist (provided in the ACORDAR 2.0 repository), and no stemmer is used. The reproduced results obtained with this configuration are indeed very close to the ACORDAR baseline in the TF-IDF similarity and are even better in the BM25, FSDM, and LMD similarities. To confirm this fact we conducted a paired t-test for each retrieval model on the nDCG@5 measure and saw no statistical difference between our results using BM25F and TF-IDF and the original ones by considering a p-value of 0.05. Instead, there is a statistical difference in the FSDM and LMD results, where the reproduced values are better.
This indexing configuration will be used for indexing the content of the datasets for the data and combined search configurations. The results obtained in those search configurations will be shown in the next section. This indexing set-up is confirmed also by the code provided in the ACORDAR 2.0 repository which is quite clear on this implementation aspect.

6.2 Data and Combined Results

After fixing the indexing and searching parameters it is possible to test all the parsing strategies developed on the runs based on the data and combined search configurations. For every parsing strategy, two tables (one for the data and one for the combined search configuration) with all the results are provided together with some statistics about how many syntax errors have been encountered during the parsing. In particular, the number of full datasets (datasets with all the original RDF files correctly parsed), partial datasets (datasets with not all the original RDF files parsed), empty datasets (datasets without an RDF parsed file) and the total number of RDF

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.4743	0.5019	0.2676	0.3685
	Reproduced	0.4533†	0.4706	0.2559	0.3394
	Difference	-0.0210	-0.0313	-0.0117	-0.0291
BM25F	Original	0.5045	0.5250	0.2859	0.3838
	Reproduced	0.4980	0.5140	0.2842	0.3742
	Difference	-0.0065	-0.0110	-0.0017	-0.0096
FSDM	Original	0.4853	0.4958	0.2770	0.3516
	Reproduced	0.5043	0.5232	0.2830	0.3795
	Difference	0.0190	0.0274	0.0060	0.0279
LMD	Original	0.4363	0.4573	0.2543	0.3325
	Reproduced	0.4563†	0.4704	0.2698	0.3444
	Difference	0.0200	0.0131	0.0155	0.0119

Table 6.1: Metadata search configuration results. Indexing configuration: Lucene Standard Analyzer, Lucene Stoplist, No Stemmer and Query Boosting. In the table are reported the original and reproduced results. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.4743	0.5019	0.2676	0.3685
	Reproduced	0.4663	0.4855	0.2634	0.3519
	Difference	-0.0080	-0.0164	-0.0042	-0.0166
BM25F	Original	0.5045	0.5250	0.2859	0.3838
	Reproduced	0.5099	0.5268	0.2899	0.3859
	Difference	0.0054	0.0018	0.0040	0.0021
FSDM	Original	0.4853	0.4958	0.2770	0.3516
	Reproduced	0.5242†	0.5415	0.2986	0.3981
	Difference	0.0389	0.0457	0.0216	0.0465
LMD	Original	0.4363	0.4573	0.2543	0.3325
	Reproduced	0.4532†	0.4708	0.3457	0.3462
	Difference	0.0169	0.0135	0.0914	0.0137

Table 6.2: Metadata search configuration results. Indexing configuration: Lucene Standard Analyzer, NLTK + Lucene Stoplist, No Stemmer and Query Boosting. In the table are reported the original and reproduced results. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

files correctly parsed will be provided. The full, partial and empty dataset definitions take into account how many files were downloaded for the dataset (see definitions considered in Chapter 5) and how many RDF files were parsed without problems. For example, a partially downloaded dataset that has no syntax errors in parsing the RDF files remains partial, and a fully downloaded dataset if it has syntax errors during parsing becomes a partial dataset.

6.2.1 Basic Parsing

After the Basic Parsing strategy, the following statistics about the parsed RDF files are retrieved:

- Total number of parsed RDF files by Jena: 27402;
- Increment of parsed RDF files with RDFLib: 674;
- Increment of parsed RDF files with LightRDF: 4;
- Total number of parsed RDF files (Jena + RDFLib + LightRDF): 28075, (98% of the RDF files);
- Total number of not parsed files: 462 (2% of the RDF files).

Instead, from a datasets point of view:

- Total number of full datasets: 26363 (83% of the datasets);
- Total number of partial datasets: 100 (1% of the datasets);
- Total number of empty datasets: 5126 (16% of the datasets).

Note that the total number of parsed RDF files is intended over the whole number of RDF files of the downloaded collection after the recovering phase, thus 28476. The results obtained on the data search configuration are shown in table 6.3 and the results obtained on the combined one are shown in table 6.4.
From the results we can see that the reproducibility is not reached in the data run, especially using the FSDM ranking, indeed FSDM is the only retrieval model that is performing quite poorly with respect to the other models and this is quite strange and suspicious. In general, the results have a significant performance gap compared to the reference paper for all the retrieval models. The paired t-tests conducted with a p-value of 0.05 on the nDCG@5 measure show statistical differences for all the retrieval models.
The same trend can be seen also in the combined run results where FSDM is getting even farther. In the combined run the performances are in general much better and a little bit closer to the ACORDAR baseline but, after a quick and simple analysis of the boosting weights, the reason is explained: the search (thanks to the boosting weights) is more focused on the metadata fields, so this apparently good performance is obtained thanks to the metadata. Indeed in the metadata search configuration, the reproducibility is reached and the developed systems are performing well and are very close to the ACORDAR results. The paired t-tests conducted on the nDCG@5 scores with a p-value of 0.05 show statistical differences for all the retrieval models except for BM25.

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.1500†	0.1616	0.0762	0.0948
	Difference	-0.0410	-0.0347	-0.0236	-0.0251
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1805†	0.1809	0.1112	0.1248
	Difference	-0.0358	-0.0387	-0.0273	-0.0302
FSDM	Original	0.2497	0.2606	0.1478	0.1758
	Reproduced	0.1641†	0.1545	0.0983	0.1060
	Difference	-0.0856	-0.1061	-0.0495	-0.0698
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.2106†	0.2153	0.1198	0.1404
	Difference	-0.0292	-0.037	-0.0217	-0.0268

Table 6.3: Basic Parsing, Data search configuration results. In the table are reported the original and reproduced results. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4809†	0.5099	0.2720	0.3715
	Difference	-0.0279	-0.0353	-0.0151	-0.0261
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.5468	0.5758	0.3161	0.4265
	Difference	-0.0070	-0.0119	-0.0037	-0.0093
FSDM	Original	0.5932	0.6151	0.3592	0.4602
	Reproduced	0.4852†	0.4502	0.2865	0.3253
	Difference	-0.1080	-0.1649	-0.0727	-0.1349
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.5006†	0.5393	0.2925	0.3950
	Difference	-0.0459	-0.0412	-0.0341	-0.0374

Table 6.4: Basic Parsing, Combined search configuration results. In the table are reported the original and reproduced results. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

6.2.2 Labels Parsing

After the Labels Parsing strategy the following statistics about the parsed RDF files are retrieved::

- Total number of parsed RDF files: 28002 (97%);
- Total number of not parsed RDF files: 474 (3%).

From a datasets point of view:

- Number of full datasets: 26361 (83%);
- Number of partial datasets: 99 (1%);
- Number of empty datasets: 5129 (16%).

This parsing strategy was developed to test how the `rdfs:label`, the `schema:name` and in general all the human labels assigned to the RDF resources could be used to improve the search in the data search configuration. Furthermore, since the indexing code in the ACORDAR 2.0 repository cites an unclear ”label” concept, the purpose of this parsing strategy was testing if this kind of label was intended by the ACORDAR systems developers.
The results obtained on the data and combined search configurations are shown in the tables 6.5 and 6.6. Unfortunately, in spite of predictions, this parsing strategy is working worse in the data search configuration than the previous strategy despite the indexed information are more human readable. The reasons why this attempt did not produce the desired results may be:

- the parsing is not streamed but is conducted by using SPARQL queries;
- the parsing order of the RDF files is discarding too much information.

Also in this parsing strategy, we can observe the same trend as before: FSDM is the model that is working worst and the combined search configuration results are closer to the ACORDAR ones thanks to the boosting weights. In particular, in the data search configuration, the results gap is bigger than the previous parsing strategy and the paired t-tests conducted with a p-value of 0.05 on the nDCG@5 scores show a statistical difference in all the retrieval models.
In the combined search configuration the results are better thanks to the metadata and the t-tests show statistical differences for all the retrieval models except for BM25 as before.
The labels extracted are very human-readable so they can potentially improve the search if used in the correct way. This attempt was not entirely useless because it allows us to verify that a lot of different labels are assigned to the RDF Resources in the datasets collection.

6.2.3 Stream Parsing

After the Stream Parsing strategy, the following statistics about the parsed RDF files are retrieved:

- Total number of parsed files by Jena: 27855;
- Increment of parsed files with LightRDF: 54;
- Total number of parsed files (Jena + LightRDF): 27909 (97.8% of the RDF files);
- Total number of not parsed files: 628 (2.2% of the RDF files).

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.1453†	0.1525	0.0742	0.0900
	Difference	-0.0457	-0.0438	-0.0256	-0.0299
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1623†	0.1676	0.1005	0.1148
	Difference	-0.0540	-0.0520	-0.0380	-0.0402
FSDM	Original	0.2497	0.2606	0.1478	0.1758
	Reproduced	0.1349†	0.1337	0.0804	0.0890
	Difference	-0.1148	-0.1269	-0.0674	-0.0868
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.1843†	0.1932	0.1076	0.1259
	Difference	-0.0555	-0.0591	-0.0339	-0.0413

Table 6.5: Labels Parsing, Data search configuration results. In the table are reported the original and reproduced results. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4778†	0.5065	0.2722	0.3697
	Difference	-0.0310	-0.0387	-0.0149	-0.0279
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.5428	0.5693	0.3146	0.4210
	Difference	-0.0110	-0.0184	-0.0052	-0.0148
FSDM	Original	0.5932	0.6151	0.3592	0.4602
	Reproduced	0.4291†	0.3916	0.2547	0.2789
	Difference	-0.1641	-0.2235	-0.1045	-0.1813
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.4960†	0.5206	0.2958	0.3805
	Difference	-0.0505	-0.0599	-0.0308	-0.0519

Table 6.6: Labels Parsing, Combined search configuration results. In the table are reported the original and reproduced results. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

And, from a datasets point of view:

- Total number of full datasets: 26224 (83% of the datasets);
- Total number of partial datasets: 145 (0.5% of the datasets);
- Total number of empty datasets: 5220 (16.5% of the datasets).

The results obtained by using the Stream Parsing with triple deduplication are shown in the tables 6.7 and 6.8, while the results obtained by using the Stream Parsing with triple deduplication and

the labels are shown in the tables 6.9 and 6.10. From the tables 6.7 and 6.8 it is possible to note how the Stream Parsing with triple deduplication is performing better than the other parsing strategies by achieving the best results on the data and combined search configurations. The paired t-tests (conducted on the nDCG@5 scores with a p-value of 0.05) still highlight statistical differences from the original results in the data search configuration for all the retrieval models: the reproducibility is still not reached but with this parsing strategy we are closer to the baseline results. This parsing strategy is probably more similar to the one used by the ACORDAR baseline: the triples are analyzed and deduplicated in a stream way and the blank nodes are discarded. Also in this case the results trend is the same: FSDM is the retrieval model that is performing worst and the combined results are better, closer to the baseline results, and statistically different from the original except for the BM25 retrieval model.
Also in this parsing strategy, in spite of predictions, the label variant is not performing very well and, after this last test, we can conclude that the label concept present in the ACORDAR 2.0 repository code is still not clear. In particular, in the data and combined search configuration, the gap is bigger with respect to the variant without labels. Apart from performing the worst, the results trend is the same as in the variant without labels.
Stream Parsing with triple deduplication is thus the best parsing strategy developed but there is still a small difference between the original and the reproduced results: the possible causes of this delta are analyzed in the next section.

		NDCG@5	**NDCG@10**	**MAP@5**	**MAP@10**
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.1637†	0.1700	0.0807	0.0989
	Difference	-0.0273	-0.0263	-0.0191	-0.0210
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1810†	0.1847	0.1116	0.1261
	Difference	-0.0353	-0.0349	-0.0269	-0.0289
FSDM	Original	0.2497	0.2606	0.1478	0.1758
	Reproduced	0.1748†	0.1667	0.1108	0.1192
	Difference	-0.0749	-0.0939	-0.0370	-0.0566
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.2161†	0.2197	0.1231	0.1439
	Difference	-0.0237	-0.0326	-0.0184	-0.0233

Table 6.7: Stream Parsing with triple deduplication, Data search configuration results. In the table are reported the original and reproduced results. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4799†	0.5097	0.2714	0.3709
	Difference	-0.0289	-0.0355	-0.0157	-0.0267
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.5476	0.5770	0.3158	0.4274
	Difference	-0.0062	-0.0107	-0.0040	-0.0084
FSDM	Original	0.5932	0.6151	0.3592	0.4602
	Reproduced	0.4957†	0.4621	0.2946	0.3365
	Difference	-0.0975	-0.1530	-0.0646	-0.1237
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.5072†	0.5463	0.2973	0.4022
	Difference	-0.0393	-0.0342	-0.0293	-0.0302

Table 6.8: Stream Parsing with triple deduplication, Combined search configuration results. In the table are reported the original and reproduced results. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.1473†	0.1586	0.0750	0.0932
	Difference	-0.0437	-0.0377	-0.0248	-0.0267
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1702†	0.1731	0.1039	0.1180
	Difference	-0.0461	-0.0465	-0.0346	-0.0370
FSDM	Original	0.2497	0.2606	0.1478	0.1758
	Reproduced	0.1641†	0.1545	0.0983	0.1060
	Difference	-0.0856	-0.1061	-0.0495	-0.0698
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.2024†	0.2096	0.1138	0.1349
	Difference	-0.0374	-0.0427	-0.0277	-0.0323

Table 6.9: Stream Parsing with triple deduplication and labels, Data search configuration results. In the table are reported the original and reproduced results. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

6.3 Impact of the empty datasets

Obviously, the empty datasets impact the results in the data search configuration (and as a result also in the combined search configuration) for every parsing strategy. An empty dataset is a dataset for which not a single RDF file was parsed due to download or parsing errors. For these datasets no RDF files are available and thus there is no content information associated with them: this makes it impossible to return this kind of dataset in any output rank, for any query,

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4798†	0.5102	0.2715	0.3718
	Difference	-0.0290	-0.0350	-0.0156	-0.0258
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.5449	0.5718	0.3142	0.4229
	Difference	-0.0089	-0.0159	-0.0056	-0.0129
FSDM	Original	0.5932	0.6151	0.3592	0.4602
	Reproduced	0.4443†	0.4053	0.2587	0.2901
	Difference	-0.1489	-0.2098	-0.1005	-0.1701
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.4977†	0.5232	0.2923	0.3817
	Difference	-0.0488	-0.0573	-0.0343	-0.0507

Table 6.10: Stream Parsing with triple deduplication and labels, Combined search configuration results. In the table are reported the original and reproduced results. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

in the data search configuration except for rare cases. Indeed, thanks to the stream parsing, it is possible to extract from the problem RDF files some information until the first syntax error, thus in some cases also the empty datasets have some content data associated.
The main focus of the following analysis is to understand how big is the impact of these datasets to understand if they are an important factor that prevents the reproducibility of the ACORDAR baselines. The whole analysis can be seen in the `data_based_systems.ipynb` notebook in the ARA repository. In this analysis we are going to consider the best parsing strategy obtained, the Stream parsing with triple deduplication, thus its set of empty datasets and its runs.
From a first simple analysis of the qrels, 232 queries (out of 493) have at least one empty dataset marked as relevant: this is a considerable number. In particular, in figure 6.1 it is possible to see a histogram about how many queries have a given number of empty datasets marked as relevant in the qrels: from the plot, it is evident that the empty relevant datasets are a big problem. Furthermore, after an analysis of the ACORDAR baseline runs, these empty relevant datasets are returned in the output ranks of a considerable number of queries and, more importantly, the number of empty relevant datasets in the output ranks of these queries varies from 1 to 7. The number of queries that have a given number of empty relevant datasets in the output ranks is reported for every ACORDAR baseline run in the plots of figure 6.2. Since the empty datasets cannot be returned by the reproduced systems except for rare cases, it is easy to state that this problem could affect the overall reproduced system performances.
In order to evaluate the impact of the empty datasets the following analysis steps were conducted:

1. A new qrels file without the empty datasets was produced: from the original qrels all the rows that involve an empty dataset were removed;

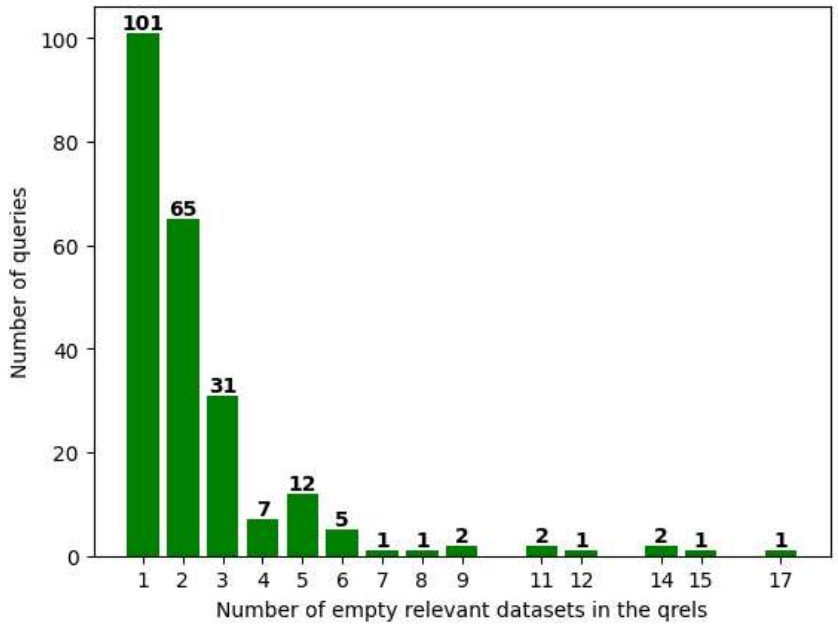


Figure 6.1: Histograms about the number of queries that have a given number of empty datasets in the qrels.

2. The ACORDAR and reproduced runs (obtained by using the Stream Parsing with triple deduplication for the data and combined search configurations) were analyzed again with the `trec_eval` tool by using this new qrels file.

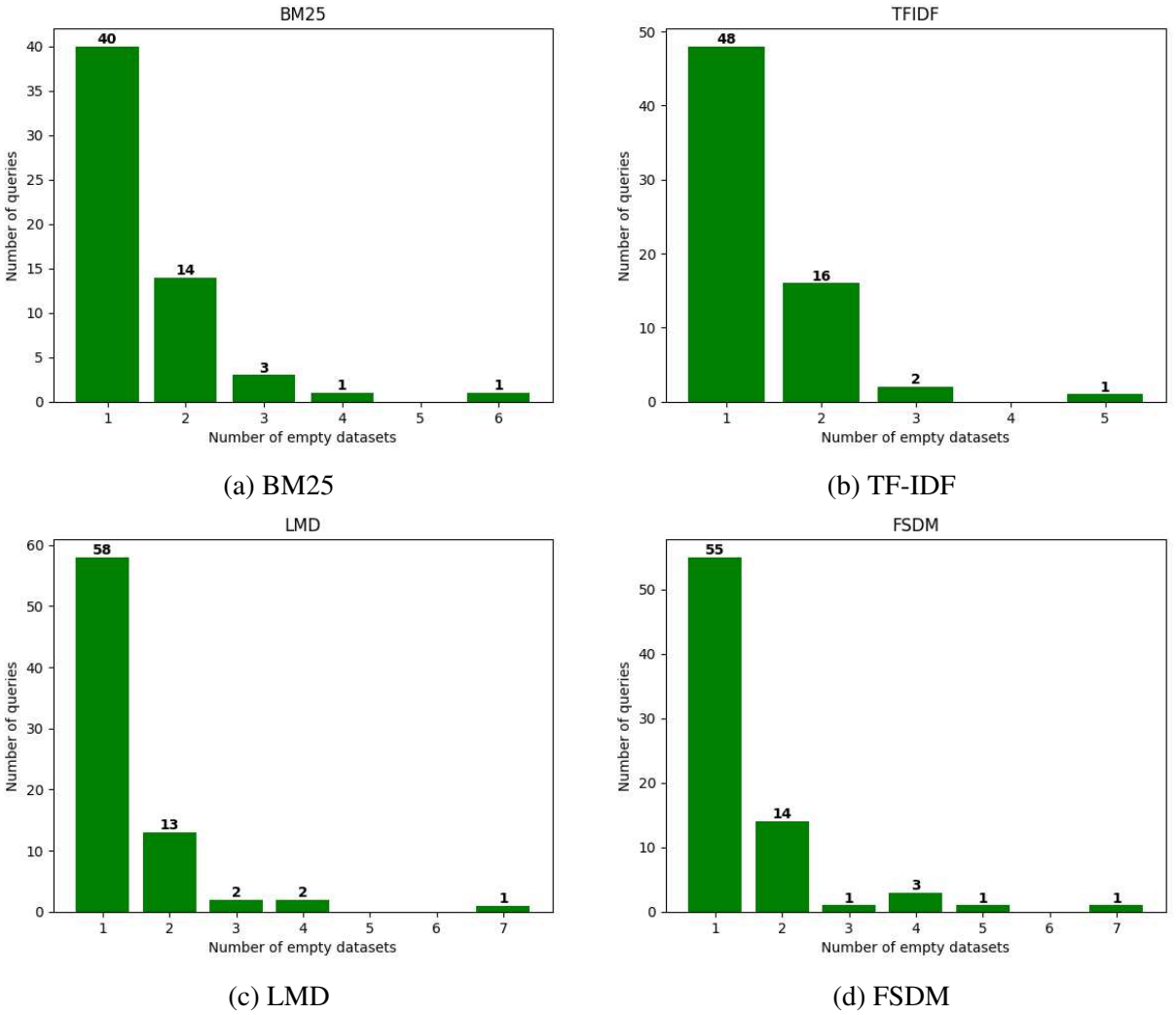
The results obtained in the metadata, data and combined search configurations are reported in the tables 6.11, 6.12 and 6.13. For every table, the ACORDAR baselines and the reproduced values indicated are obtained with the new qrels file.

As can be seen from the tables 6.11 6.12 6.13 after removing the empty datasets from the qrels the ACORDAR and reproduced systems have more or less the same performances. In particular:

- On the metadata search configuration the results are practically the same on the BM25, TF-IDF and LMD similarities, and this is confirmed also by the t-tests conducted on the nDCG@5 scores (p-value: 0.05) that indicate a statistical equivalence between these runs. FSDM is actually performing better in the reproduced run with respect to the ACORDAR run and the obtained results are statistically different from the original.

- On the data and combined search configurations the situation is the same as in the metadata configuration except for the FSDM model where the reproduced results are slightly worse. The results obtained with that retrieval model are the only ones considered statistically different from the original results.

After this test it is clear that the empty datasets are actually impacting in a quite strong way the performances of the reproduced systems, indeed, after removing them from the qrels, the



(a) BM25

(b) TF-IDF

(c) LMD

(d) FSDM

Figure 6.2: Histograms about the number of queries that have a given number of empty datasets in the baseline runs

reproducibility is reached.

One last point to note is still the FSDM ranking, in all the results shown in the previous section and for every parsing strategy, the results obtained with it were always a little bit far from the ACORDAR results and also here it is the only model with a significant difference from the baseline performances. This makes us question whether the implementation provided is actually the one used in the ACORDAR tests that reach the excellent nDCG@5 of almost 0.6 in the run based on the combined search configuration.

6.4 Impact of partial datasets

Quantifying the impact of the partial datasets is quite difficult for two reasons:

- their scores in the final rank are already influenced by the presence of empty datasets;

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.2580	0.3201	0.1746	0.2260
	Reproduced	0.2549	0.3091	0.1711	0.2160
	Difference	-0.0031	-0.0110	-0.0035	-0.0100
BM25F	Original	0.2736	0.3308	0.1841	0.2328
	Reproduced	0.2744	0.3313	0.1855	0.2337
	Difference	0.0008	0.0005	0.0014	0.0009
FSDM	Original	0.2478	0.2906	0.1632	0.1987
	Reproduced	0.2776†	0.3358	0.1873	0.2369
	Difference	0.0298	0.0452	0.0241	0.0382
LMD	Original	0.2303	0.2836	0.1591	0.2011
	Reproduced	0.2365	0.2867	0.1641	0.2038
	Difference	0.0062	0.0031	0.0050	0.0027

Table 6.11: Metadata search configuration results obtained without empty datasets. In the table are reported the original and reproduced results obtained by considering the new qrels file. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.0858	0.1056	0.0516	0.0633
	Reproduced	0.0867	0.1065	0.0528	0.0646
	Difference	0.0009	0.0009	0.0012	0.0013
BM25F	Original	0.1088	0.1201	0.0766	0.0837
	Reproduced	0.1060	0.1194	0.0754	0.0837
	Difference	-0.0028	-0.0007	-0.0012	0.0000
FSDM	Original	0.1360	0.1570	0.0910	0.1060
	Reproduced	0.0991†	0.1034	0.0721	0.0757
	Difference	-0.0369	-0.0536	-0.0189	-0.0303
LMD	Original	0.1239	0.1468	0.0815	0.0952
	Reproduced	0.1213	0.1435	0.0796	0.0933
	Difference	-0.0026	-0.0033	-0.0019	-0.0019

Table 6.12: Data search configuration results obtained without empty datasets and by using Stream Parsing with Triple Deduplication. In the table are reported the original and reproduced results obtained by considering the new qrels file. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

- this type of datasets, despite not all the RDF files were parsed (or available), can still appear in the final ranks (in contrast with the empty datasets) but their positions can be affected by how many RDF files were parsed for them.

In order to simplify this analysis only the datasets with less than 50 % of the original RDF files parsed have been considered: 41 datasets of this type were identified after the execution of the best parsing strategy. In the qrels only 26 queries have at least one of these partial relevant

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.2702	0.3381	0.1806	0.2356
	Reproduced	0.2688	0.3302	0.1821	0.2316
	Difference	-0.0014	-0.0079	0.0015	-0.0040
BM25F	Original	0.2793	0.3505	0.1869	0.2452
	Reproduced	0.2956	0.3670	0.2012	0.2594
	Difference	0.0163	0.0165	0.0143	0.0142
FSDM	Original	0.2947	0.3521	0.2003	0.2491
	Reproduced	0.2652†	0.2856	0.1859	0.2073
	Difference	-0.0295	-0.0665	-0.0144	-0.0418
LMD	Original	0.2748	0.3408	0.1873	0.2389
	Reproduced	0.2747	0.3411	0.1859	0.2395
	Difference	-0.0001	0.0003	-0.0014	0.0006

Table 6.13: Combined search configuration results obtained without empty datasets and by using Stream Parsing with Triple Deduplication. In the table are reported the original and reproduced results obtained by considering the new qrels file. Symbol † represents results that are statistically different from the original, according to a paired t-test conducted on the nDCG@5 scores with a p-value of 0.05.

datasets marked as relevant. Already from this simple data, it is easy to guess that the impact of this problem is very little for the final performances.
The partial datasets problem impacts the order of the final ranks: a partial dataset can still appear in the final ranks but probably with a lower position due to the absence of RDF files. This can obviously impacts the nDCG@5 and nDCG@10 metrics. By comparing the reproduced and the ACORDAR baselines runs on the data search configuration, the average number of positions difference for a partial dataset between the two runs are:

- for BM25: 2 positions,
- for TF-IDF: 2.23 positions,
- for LMD: 1.69 positions,
- for FSDM: 1.61 positions.

This means that a partially relevant dataset in the ACORDAR runs is usually returned in higher positions with respect to the reproduced runs. This is obviously a problem for the reproducibility but it is actually impacting the systems in a very little way with respect to the empty datasets problem.

Chapter 7

Reproducibility Solutions

In this Chapter, we provide some simple solutions to solve the reproducibility problems reported in Chapter 5.

7.1 Download

In order to solve the reproducibility problems highlighted in the download phase of the ACORDAR collection, the whole collection could be provided entirely as a compressed archive that contains all the datasets RDF files. The internal structure of the archive can be the same used in this reproducibility study: one directory per dataset which contains all the datasets RDF files and a json file with all the dataset metadata. With this solution only one URL must be guaranteed active, which ensures the download of the archive: this will completely solve the problem. Furthermore, by providing the collection in this way, the download time could be lowered a lot and the collection datasets versions could be fixed.

7.2 Parsing

In the parsing phase, two main problems have been encountered: the RDF files syntax errors and the unknown parsing strategy used by ACORDAR.
For the first problem, the solution is the same proposed in the previous section: if the whole collection is provided in a big compressed archive all the RDF files used by ACORDAR in its tests are available and, if they did not actually encounter parsing errors, the problem is solved. Since they are not reporting syntax errors in the paper we are assuming that they did not encounter any.
For the second problem, the solution is to provide the whole parsing code used in the tests: this is the only part of the code that is missing and it should explain in a very exhaustive way how the RDF files are parsed by explaining which library was used, which information is extracted

for entities, literals and classes and how the relational database is structured and populated. The code can be released in the ACORDAR 2.0 repository and it should be commented better than the other code parts to avoid any doubts on how it works.
In order to solve at the same time all the problems about the collection download and parsing, the collection files and the code could be also provided inside a Docker container that can define a lightweight and standalone execution environment that includes everything needed to run (and thus reproduce) the ACORDAR baselines.

All this reproducibility study was projected to be fully reproducible, indeed the code for the collection download and parsing is provided in the ADE repository and the code needed for the collection indexing and searching is provided in the ARDFS repository. The downloaded and curated version of the collection that was used could be provided as a compressed archive.

Chapter 8

Discussion and Conclusion

This work tests ACORDAR: A Test Collection for Ad Hoc Content-Based (RDF) Dataset Retrieval by reproducing the baseline methods reported in its reference paper [11]. The main goal of this work was to reproduce the baseline systems in order to:

- Test the collection and evaluate its overall quality for the Ad Hoc Content-Based RDF Dataset retrieval task during the systems development.
- Reach the paper declared results and performances.

The collection has been tested from the download to the development of the reproduced systems and the following problems and limitations arise:

- The collection is not ”easy to use”: the download phase must be performed by using a custom script that can organize the collection datasets and files in a curated and automated way. Furthermore, the downloaded files must be curated by assigning a possible RDF extension to all the files that have an empty format and all the archives must be extracted and treated in the correct way.
- The collection datasets files are provided as a list of download links and, as reported in the previous chapters, this means that a lot of URLs can possibly not work from one day to another since they can change or they could be not maintained over time. Furthermore, this ensures a big download time for the collection and the continuous variability of the collection.
- The collection contains semantically duplicated datasets probably due to an error during the datasets crawling.
- The relevance judgments are not so good: the number of queries is good but the number of evaluated datasets for every query is very small, furthermore for every query there is an imbalance in the number of datasets considered relevant and those that are irrelevant.

From a content point of view, the datasets RDF files sometimes contain well build ontologies, sometimes some big and general-purpose ontologies and sometimes some tabular datasets dumps: there is a good variety of datasets that reflect the overall quality of datasets that is now present on the Web. This variety of datasets must be managed in the correct way.
ACORDAR is the very first test collection for the task of Ad Hoc RDF datasets retrieval so a lot of improvement can be done in the future, thanks also to the increased research effort in this area.
The ACORDAR baseline systems were reproduced to the best of our knowledge, by tuning all the parameters according to the repositories and paper directives. The baselines based on the metadata search configuration were reproduced instead the systems based on the data and combined search configurations were not reproduced perfectly except when excluding the empty datasets. In order to push the development of new systems on the ACORDAR collection and fix a common starting point for future works, the whole code produced during this work has been made available on Github in 3 different repositories.
The last take-home message of this work is that currently, the datasets content is not impacting a lot the systems performances with respect to those based only on the metadata: the data runs results are very low and as a result, the data impact in the combined run is limited. The ACORDAR baselines are very simple and limited systems that threat the RDF files basically as text files without considering their intrinsic semantics but, also from here, it is possible to note that actually the RDF data weight is very limited and it is not overturning the performances.

8.1 Future Work

The future work directions comprise the development of better semantic representations of the datasets content with respect to the simple lists of entities, classes, literals and properties. In particular, having an idea of the data schema of the graph and a brief summary of the graph content can help the search. The main ideas for enhancing the search are thus:

- Extract data schemas from the datasets RDF files in order to have a birds eye view of the datasets content in terms of classes, properties and relations [23],
- Extract graph snippets [22] that can summarize the main content of the graph.

The extracted information, that would be presented as RDF triples, could be then translated into text by using RDF2Text[1] or Triple2Text [7] models in order to search over them with Text Search techniques (Bag Of Words based or Neural based).

[1]https://github.com/badrex/rdf2text

List of Figures

List of Tables

Bibliography

[1] Dean Allemang, Jim Hendler, and Fabien Gandon. *Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL*. 3rd ed. Vol. 33. New York, NY, USA: Association for Computing Machinery, 2020. ISBN: 9781450376174.

[2] Marco Angelini et al. "VIRTUE: A Visual Tool for Information Retrieval Performance Evaluation and Failure Analysis". In: *J. Vis. Lang. Comput.* 25.4 (Aug. 2014), pp. 394–413. ISSN: 1045-926X. DOI: 10.1016/j.jvlc.2013.12.003. URL: https://doi.org/10.1016/j.jvlc.2013.12.003.

[3] Cambridge Semantics. *Learn RDF*. [Online; accessed 6-September-2023]. 2023. URL: https://cambridgesemantics.com/blog/semantic-university/learn-rdf/.

[4] Zhiyu Chen et al. *Leveraging Schema Labels to Enhance Dataset Search*. 2020. arXiv: 2001.10112 [cs.IR].

[5] Otis Gospodnetic Erik Hatcher. *Lucene in Action*. Manning Pubns Co, 2004. ISBN: 1932394281.

[6] Nicola Ferro. "Search Engines Course, Lection 4: Basic Concepts in Information Retrieval". University Lecture. 2022.

[7] Hanning Gao et al. *Triples-to-Text Generation with Reinforcement Learning Based Graph-augmented Neural Networks*. 2022. arXiv: 2111.10545 [cs.CL].

[8] *Google Datasets Website*. https://developers.google.com/search/docs/appearance/structured-data/dataset?hl=it. Accessed: 06-09-2023.

[9] Laura M. Koesten et al. "The Trials and Tribulations of Working with Structured Data: -A Study on Information Seeking Behaviour". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI '17. Denver, Colorado, USA: Association for Computing Machinery, 2017, pp. 1277–1289. ISBN: 9781450346559. DOI: 10.1145/3025453.3025838. URL: https://doi.org/10.1145/3025453.3025838.

[10] Laura M. Koesten et al. “The Trials and Tribulations of Working with Structured Data: -A Study on Information Seeking Behaviour”. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. CHI ’17. Denver, Colorado, USA: Association for Computing Machinery, 2017, pp. 1277–1289. ISBN: 9781450346559. DOI: 10.1145/3025453.3025838. URL: https://doi.org/10.1145/3025453.3025838.

[11] Tengteng Lin et al. “ACORDAR: A Test Collection for Ad Hoc Content-Based (RDF) Dataset Retrieval”. In: *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’22. Madrid, Spain: Association for Computing Machinery, 2022, pp. 2981–2991. ISBN: 9781450387323. DOI: 10.1145/3477495.3531729. URL: https://doi.org/10.1145/3477495.3531729.

[12] Emmanuel Pietriga et al. “Browsing Linked Data Catalogs with LODAtlas”. In: *The Semantic Web – ISWC 2018*. Ed. by Denny Vrandečić et al. Cham: Springer International Publishing, 2018, pp. 137–153. ISBN: 978-3-030-00668-6.

[13] RDF W3C. *Learn RDF*. [Online; accessed 6-September-2023]. 2023. URL: https://www.w3.org/RDF/.

[14] RDFS W3C. *Learn RDF*. [Online; accessed 6-September-2023]. 2023. URL: https://www.w3.org/TR/rdf12-schema/.

[15] RDFS W3C. *Learn RDF*. [Online; accessed 6-September-2023]. 2023. URL: https://www.w3.org/TR/2010/NOTE-curie-20101216/.

[16] RDFS W3C. *Learn RDF*. [Online; accessed 6-September-2023]. 2023. URL: https://www.w3.org/TR/rdf11-concepts/.

[17] Gerard. Salton. *Automatic Information Organization and Retrieval.* McGraw Hill Text, 1968. ISBN: 0070544859.

[18] Gianmaria Silvello. “Database 2 Course, Lection 3: Introduction to RDF”. University Lecture. 2023.

[19] *Text REtrieval Conference (TREC) Data - English Test Questions (Topics) File List*. [Online; accessed 9-September-2023]. 2023. URL: https://trec.nist.gov/data/topics_eng/index.html.

[20] *Understanding the BM25 Ranking Algorithm*. https://medium.com/@evertongomede/understanding-the-bm25-ranking-algorithm-19f6d45c6ce. Accessed: 06-09-2023.

[21] Trevor Strohman W. Bruce Croft Donald Metzler. *Search Engines. Information Retrieval in Practice*. Pearson Education Inc., 2015. URL: https://ciir.cs.umass.edu/irbook/.

[22] Xiaxia Wang et al. “BANDAR: Benchmarking Snippet Generation Algorithms for (RDF) Dataset Search”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.2 (2023), pp. 1227–1241. DOI: 10.1109/TKDE.2021.3095309.

[23] Marc Weise, Steffen Lohmann, and Florian Haag. “LD-VOWL: Extracting and Visualizing Schema Information for Linked Data Endpoints”. In: *VOILA@ISWC*. 2016. URL: https://api.semanticscholar.org/CorpusID:2757616.

[24] Wikipedia contributors. *Resource Description Framework — Wikipedia, The Free Encyclopedia*. [Online; accessed 6-September-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=Resource_Description_Framework&oldid=1171438417.

[25] Wikipedia contributors. *SPARQL — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-September-2023]. 2023. URL: https://en.wikipedia.org/w/index.php?title=SPARQL&oldid=1171113278.

[26] Chengxiang Zhai and John Lafferty. “A Study of Smoothing Methods for Language Models Applied to Ad Hoc Information Retrieval”. In: *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’01. New Orleans, Louisiana, USA: Association for Computing Machinery, 2001, pp. 334–342. ISBN: 1581133316. DOI: 10.1145/383952.384019. URL: https://doi.org/10.1145/383952.384019.

[27] Nikita Zhiltsov, Alexander Kotov, and Fedor Nikolaev. “Fielded Sequential Dependence Model for Ad-Hoc Entity Retrieval in the Web of Data”. In: *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’15. Santiago, Chile: Association for Computing Machinery, 2015, pp. 253–262. ISBN: 9781450336215. DOI: 10.1145/2766462.2767756. URL: https://doi.org/10.1145/2766462.2767756.

Acknowledgements

First of all, I would like to thank Professor Gianmaria Silvello for his professionalism, helpfulness, and valuable advice that guided me in the realization of this research project and the writing of the thesis.
My sincere thanks also go to Eng. Laura Menotti for always carefully supervising this work.
I would like to thank all my friends from the university with whom I have shared large parts of the challenges and milestones I have achieved over the past two years, but most of all I would like to thank my close friends who have always helped me to move forward on this journey.
I sincerely thank my parents who have always supported me; without their sacrifices, none of this would have been possible. Thank you for always believing in my abilities and always picking me up in the darkest moments.
Finally, I would like to dedicate this achievement to myself because despite all the difficulties and sacrifices I never gave up. I hope this is just the beginning of something great.