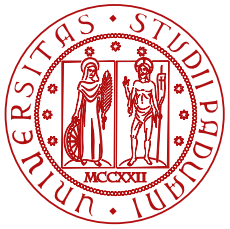

Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering

Analyzing the Impact of RDF Graph Structure on Dataset Search: A Case Study with ACORDAR

Master Candidate

Riccardo Forzan

Student ID 2057453

Supervisor

Prof. Gianmaria Silvello

University of Padova

Co-supervisor

Ing. Laura Menotti

Academic Year
2022/2023

It is not because things are difficult that we do not dare,
it is because we do not dare that they are difficult.
Seneca

Abstract

Resource Description Framework (RDF) plays a central role in the era of the Semantic Web, enabling a structured representation of datasets and their relationships. The complex nature of RDF graph structures significantly influences the retrieval of datasets, offering a blend of both challenges and possibilities. Delving deeply into the ACORDAR case study, the work unveils how graph structures influence dataset retrieval and the organization of data. Furthermore, it introduces serialization methods within RDF, emphasizing the importance of Uniform Resource Identifier (URI) and the capabilities of the SPARQL Protocol and RDF Query Language (SPARQL). Presenting the ACORDAR reproducibility, the research underscores the significance of metadata in dataset search. Exploring potential avenues for future research in dataset search, the investigation integrates graph structures and harnesses emerging technologies from the Semantic Web era.

Sommario

Nel mondo del Semantic Web, RDF si pone come elemento cardine per la modellazione precisa dei dati e dei loro legami. L'obiettivo centrale di questo lavoro è esplorare le dinamiche dei grafi RDF, mettendo in luce le principali problematiche e potenzialità nell'ambito della ricerca di dataset.
Il caso studio di ACORDAR viene esaminato per illustrare l'effetto delle strutture a grafo sull'organizzazione dei dati. Vengono analizzate le tecniche di serializzazione in RDF, sottolineando la centralità di elementi quali gli URI e le capacità avanzate offerte da SPARQL. Si affronta il tema della riproducibilità di ACORDAR, mettendo in risalto l'importanza dei metadati nella fase di ricerca dei dataset. In conclusione, si delineano prospettive future per ottimizzare la ricerca di dataset, arricchendo l'analisi con informazioni tratte dalle strutture a grafo e avvalendosi delle tecnologie emergenti.

Contents

List of Figures **11**

List of Tables **13**

List of Acronyms **17**

1 Introduction **19**

2 Background **21**

2.1 Dataset Search 21

2.2 Graph Structures for Organizing Data 22

2.3 Resource Description Framework 23

2.3.1 Serialization 24

2.3.2 URI 25

2.4 SPARQL 26

2.5 Stream processing 27

2.5.1 Management of Extensive RDF Graphs 28

2.6 Betweenness Centrality 29

2.6.1 Approximation Method 30

3 ACORDAR reproducibility **33**

3.1 Acquiring ACORDAR collection 34

3.2 Extracting Data from the Downloaded Files 36

3.3 Indexing 40

3.3.1 Metadata-Based Fields 41

3.3.2 RDF-Based Fields 41

3.4 Searching 43

4 Enhancing Dataset Search Performance using Graph Structure **47**
4.1 Re-Ranking based on Numerical Features 48
4.1.1 Numerical Feature Extraction 49
4.1.2 Numerical Feature Utilization 50
4.2 Leveraging Important Nodes in Graph Structure 51

5 Experimental Evaluation **53**
5.1 Reproduced Experiments 53
5.2 Impact of Missing Datasets 79
5.3 Re-Ranking based on Numerical Features 81
5.4 Leveraging Important Nodes 84

6 Conclusions and Future Work **89**
6.1 General Considerations On ACORDAR 89
6.2 Reproduced Results 90
6.3 Conclusions and Future Work 91
6.3.1 Key Discoveries 92
6.3.2 Semantic Embeddings 92
6.3.3 Future Directions 93

References **95**

Acknowledgments **99**

List of Figures

2.1 Betweenness Centrality 30

3.1 Downloaded Files 35
3.2 Example of `metadata.json` 38
3.3 Dataset Refinement 39

5.1 Missing Datasets Impact 79
5.2 Feature Importance 83

List of Tables

3.1 Boost Weights for Metadata 45
3.2 Boost Weights for Data 46
3.3 Boost Weights for Full Data 46

5.1 Performance Metrics of ACORDAR across all queries 53
5.2 Performance comparison using combined metadata and data across all queries in the raw collection 54
5.3 Performance comparison using metadata only across all queries in the raw collection 55
5.4 Performance comparison using data only across all queries in the raw collection 56
5.5 Performance comparison using combined metadata and data across all queries in the refined collection without files that are larger than 200MB 57
5.6 Performance comparison using metadata only across all queries in the refined collection without files that are larger than 200MB . 58
5.7 Performance comparison using data only across all queries in the refined collection without files that are larger than 200MB 59
5.8 Performance comparison using combined metadata and data across all queries in the refined collection 60
5.9 Performance comparison using metadata only across all queries in the refined collection 61
5.10 Performance comparison using data only across all queries in the refined collection 62
5.11 Performance comparison using combined metadata and data across all queries in the refined collection with Kaggle stoplist 63

5.12 Performance comparison using metadata only across all queries in the refined collection with Kaggle stoplist 64
5.13 Performance comparison using data only across all queries in the refined collection with Kaggle stoplist 65
5.14 Performance comparison using combined metadata and data across all queries in the refined collection with NLTK stoplist 66
5.15 Performance comparison using metadata only across all queries in the refined collection with NLTK stoplist 67
5.16 Performance comparison using data only across all queries in the refined collection with NLTK stoplist 68
5.17 Performance comparison using combined metadata and data across all queries in the refined collection with boosting 69
5.18 Performance comparison using metadata only across all queries in the refined collection with boosting 70
5.19 Performance comparison using data only across all queries in the refined collection with boosting 71
5.20 Performance comparison using combined metadata and data across all queries in the refined collection with Kaggle stoplist and boosting 72
5.21 Performance comparison using metadata only across all queries in the refined collection with Kaggle stoplist and boosting 73
5.22 Performance comparison using data only across all queries in the refined collection with Kaggle stoplist and boosting 74
5.23 Performance comparison using combined metadata and data across all queries in the refined collection with NLTK stoplist and boosting 75
5.24 Performance comparison using metadata only across all queries in the refined collection with NLTK stoplist and boosting 76
5.25 Performance comparison using data only across all queries in the refined collection with NLTK stoplist and boosting 77
5.26 Divergence of configurations from the baseline 78
5.27 Divergence of configurations from the baseline without considering empty files 80
5.28 Performance comparison using combined metadata and data across all queries in the refined collection with re ranking based on numerical features 81

5.29 Performance comparison using metadata only across all queries in the refined collection with re-ranking based on numerical features 82
5.30 Performance comparison using data only across all queries in the refined collection with re-ranking based on numerical features . . 82
5.31 Re-Ranking Improvements Evaluation 83
5.32 Enrichment of Metadata and Data with Node URIs 85
5.33 Selective Use of Top Node URIs and Metadata 86
5.34 Usage of Nodes versus Extracted Data 87

List of Acronyms

API Application Programming Interface

FSDM Fielded Sequential Dependence Model

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

IR Information Retrieval

JSON JavaScript Object Notation

LLM Large Language Model

LMD Language Model with Dirichlet Smoothing

LOD Linked Open Data

MAP Mean Average Precision

NDCG Normalized Discounted Cumulative Gain

RDF Resource Description Framework

SPARQL SPARQL Protocol and RDF Query Language

TREC Text Retrieval Conference

TF-IDF Term Frequency-Inverse Document Frequency

URI Uniform Resource Identifier

URL Uniform Resource Locator

WWW World Wide Web

XML Extensible Markup Language

1

Introduction

In the period marked by the advent of the Semantic Web, the plenitude of datasets, especially those organized using RDF, has presented us with a multitude of both challenges and prospects. Efficiently retrieving, exploring, and utilizing these datasets is of great significance. This document investigates these challenges with an emphasis on the ACORDAR case study. The primary focus of this thesis is to investigate the impact of RDF graph structures on the outcomes of dataset searches.

The opening chapters lay a foundational understanding of the central concepts and technologies pertinent to the topic. They encompass discussions on dataset retrieval, the influence of graph structures in data organization, the manner in which RDF structures data, the importance of URI, and the potentialities of SPARQL. Subsequent chapters delve into the ACORDAR system, strategies to refine dataset retrieval using graph structures, and evaluations of these strategies.

We delve into the influence of RDF graph structures on dataset search outcomes and investigate whether an inclusive search methodology, encompassing both data and metadata as opposed to solely relying on metadata, can enhance the precision and quality of the search results. Our research underscores the profound influence of metadata on system efficiency. Regardless of integrating provided metadata with exhaustive data from RDF graphs or merely with the URIs of select central nodes, the performance metrics remain notably consistent. This suggests that it is the metadata, rather than the granularity of data extraction, that primarily determines the results.

The document concludes by offering reflections on future research trajectories in this domain. Our investigation highlights the value of graph summarization techniques and semantic embeddings, suggesting avenues for elevating dataset metadata quality and search relevance. Furthermore, the potential integration of large language models with graph summarization tools emerges as a notable direction for enhanced data processing and analysis. Readers will acquire a comprehensive understanding of the challenges and solutions related to RDF dataset retrieval. The findings and methodologies detailed could influence the trajectory of dataset retrieval and exploration within the evolving landscape of the Semantic Web.

The code pertinent to the experiments undertaken in this thesis can be found in the associated GitHub repository [1].

Chapter Overview The work is structured into six distinct chapters.

Chapter 1, sets the stage by outlining the research scope, giving readers a clear understanding of the topics to be discussed.

Chapter 2 delves into the "Background", introducing the Dataset Search problem, the significance of graph structures in data organization, and more technical aspects such as RDF, URI, and SPARQL queries. It also presents techniques for managing extensive RDF graphs and the concept of Betweenness Centrality.

The focus shifts in Chapter 3 to "ACORDAR Reproducibility", offering a comprehensive analysis of the challenges encountered during the replication of ACORDAR.

Chapter 4, titled "Enhancing Dataset Search Performance using Graph Structure", discusses strategies devised to improve dataset retrieval, with an emphasis on the integration of numerical features for re-ranking and indexing methods exploiting nodal centrality.

Chapter 5, titled "Experimental Evaluation", presents and discusses the outcomes of replication attempts and the efficacy of proposed enhancement strategies. It also analyzes the implications on the performance of missing datasets.

Finally, Chapter 6 offers insights on "Conclusions and Future Work", summarizing the main discoveries and suggesting potential avenues for future research in the dataset search domain.

[1] https://github.com/riccardoforzan/RDFDS

2

Background

This chapter provides an overview of the foundational concepts and technologies pertinent to the analysis of the impact of RDF graph structures on dataset search, with a specific focus on the ACORDAR [11] case study. We delve into the complexities of dataset search, the significance of graph structures, the pivotal role of the RDF, and the relevance of stream processing in RDF graph analysis.

2.1 Dataset Search

The proliferation of datasets in the Semantic Web era, especially those structured using the RDF, presents challenges and opportunities in search, discovery, and utilization. RDF datasets, rich in information, can be harnessed for various applications, but their volume and diversity necessitate efficient search mechanisms. Keyword search is a primary method for querying RDF datasets. Casanova et al. [2] underscore the complexity of keyword search over RDF datasets and introduce the concept of fortuitous search, which diversifies answers and enhances discovery. The entity relatedness problem is another challenge, aiming to understand connections between entities. Ellefi et al. [5] emphasizes the importance of RDF dataset profiling, which offers insights into dataset quality, coverage, and dynamics. Such profiling is crucial for tasks like entity linking and semantic search. Mehdi Zrhal et al. [22] highlights the significance of efficient dataset search in the spatial domain. They propose a geographic Knowledge Graph to enhance the search process for spatial datasets.

Metadata Metadata provides structured, descriptive information about datasets. In the context of RDF datasets, metadata offers insights into the content, structure, provenance, and other attributes of the data, facilitating efficient search, discovery, and utilization. The rise of the Linked Open Data (LOD) cloud and various governmental efforts have led to an exponential increase in the number of open datasets available on the web. However, the metadata associated with these datasets is often minimal, heterogeneous, and distributed across various sources. This makes the task of finding a suitable dataset for specific needs challenging. Frosterus et al. [7] address this challenge by introducing DataFinland [1], a semantic portal that employs a distributed content creation model and tools for annotating and publishing metadata about both RDF and non-RDF datasets on the web. The system leverages a modified version of the VoID vocabulary [2], specifically designed for describing linked RDF datasets, and integrates an online metadata editor for semantic annotations. Such efforts underscore the importance of comprehensive metadata in enhancing dataset discovery.

2.2 Graph Structures for Organizing Data

In the context of dataset search and Information Retrieval (IR), the choice of data representation plays a crucial role in determining the efficiency and effectiveness of the search process. Traditional tabular databases and document-centric approaches have been widely used for organizing and managing data. However, with the expansion of linked data and the Semantic Web, a novel approach based on graph structures can be exploited. This section delves into the fundamental aspects of graph structures and their relevance in the context of organizing data for dataset search. Graph structures, as the name suggests, represent data as a collection of nodes interconnected by edges. In the context of RDF, these nodes typically represent entities, while the edges denote relationships or predicates between these entities. This graph-based representation allows for a more flexible and semantically rich way of encoding knowledge [4].

[1] https://www.ldf.fi/
[2] https://www.w3.org/TR/void/

The adoption of graph structures for organizing data offers several advantages:

- **Semantic Richness**: RDF graphs provide a mechanism for expressing rich semantic relationships between entities. This semantic richness enables more precise and context-aware searches [14].
- **Flexibility**: Graphs can accommodate evolving data schemas with ease. As new relationships or properties emerge, they can be seamlessly incorporated into the existing graph structure [12].
- **Query Efficiency**: Graph-based querying languages, such as SPARQL, are tailored for navigating RDF graphs efficiently. This facilitates expressive and powerful queries, bridging the gap between the semantic web and big data [12].

2.3 Resource Description Framework

The Semantic Web [9] is an extension of the World Wide Web (WWW) that aims at making web content not only accessible to humans but also understandable by machines. It was envisioned by Tim Berners-Lee, one of the creators of the World Wide Web, as a way to enrich web data with meaning, context, and semantics. In essence, the Semantic Web strives to create a web of data that can be processed and interpreted by computers, enabling more intelligent and automated interactions. Key elements of the Semantic Web include:

- **RDF**: RDF serves as the foundational data model for the Semantic Web. It provides a structured way to describe resources and their relationships on the web.
- **Ontologies**: Ontologies are formal knowledge representations that define the concepts and relationships within a specific domain. They play a crucial role in adding semantic richness to data.
- **Linked Data**: The Semantic Web encourages the publication of data in a format that is linked to other related data sources. This creates a web of interconnected data, making it easier to discover and navigate information.
- **SPARQL**: SPARQL is a query language designed for querying RDF data. It enables complex searches and retrievals from Semantic Web datasets.

Resource Description Framework (RDF) is a data modeling framework that allows to represent information about resources in a structured and standardized manner. RDF achieves this by using a graph-based approach, where data is organized into triples:

- **Subject**: The resource you're describing
- **Predicate**: The property or attribute of the resource
- **Object**: The value or target of that property

For example, consider the triple: "John (subject) knows (predicate) Mary (object)." This triple represents a basic statement about a relationship between two resources.

2.3.1 Serialization

Serializing a graph for RDF involves representing structured data in a way that can be easily shared and processed by computers. Possible formats to serialize a graph for RDF encompass:

- **RDF/XML**: RDF/XML is the most widely used serialization format for RDF data. It uses Extensible Markup Language (XML) to represent RDF triples, where subjects, predicates, and objects are enclosed in XML elements. While RDF/XML is human-readable, it can be verbose and complex for machines to process.
- **Turtle**: Turtle is a more human-friendly and compact serialization format for RDF. It uses simple syntax to represent RDF triples, making it easier to read and write than RDF/XML. Turtle is often preferred for writing RDF data by hand or for smaller datasets.
- **N-Triples**: N-Triples is a minimalistic serialization format for RDF that represents triples as plain text lines. Each line consists of a subject, predicate, object, and a period. It's primarily used for machine-to-machine data exchange and is not as human-readable as Turtle.
- **N-Quads**: N-Quads is an extension of N-Triples that adds support for named graphs. Named graphs allow you to associate RDF triples with a specific context or dataset. N-Quads includes an additional component to represent the graph name.
- **JSON-LD**: JSON-LD is a popular RDF serialization format that uses JavaScript Object Notation (JSON) syntax to represent RDF data. It's both human-readable and machine-friendly, making it suitable for web applications and Application Programming Interface (API). JSON-LD also provides a way to express context and semantics, making it more flexible than other formats.

2.3.2 URI

URI are a fundamental concept in the world of web technologies and play a crucial role in the RDF. A URI is a string of characters that uniquely identifies a resource on the internet. These resources can be anything from web pages, documents, images, to abstract concepts, and more. URIs provide a standardized way to reference and access these resources. URIs must be unique globally, this means that no two resources should have the same URI. This uniqueness ensures that resources can be precisely located and distinguished from one another. URIs follow a standardized format, making them machine-readable and interpretable by web browsers, search engines, and other software.

URIs in RDF In RDF, every resource is identified by a URI. This means that when you describe something in an RDF triple, such as "John (subject) knows (predicate) Mary (object)," both "John" and "Mary" would typically be represented as URIs. These URIs uniquely identify John and Mary, making it clear which specific individuals or things are being referred to. URIs serve as the glue that connects resources within RDF graphs. When you have multiple triples that share the same URI as their subject or object, it indicates relationships and connections between those resources. For example, if you have several triples with "John" as the subject, it implies that all these triples are describing the same individual, John. URIs provide semantic clarity by ensuring that resources are precisely defined and linked. This is particularly important in the Semantic Web context, where the goal is to enable machines to understand and reason about data. With well-defined URIs, it's easier to establish the meaning of resources and their relationships.

Human readable URIs URIs serve as a foundational element in web technologies, and their primary purpose is to enable machine readability. It's generally considered good practice to craft URIs that are not only machine-readable but also human-readable. This practice enhances the usability and accessibility of web resources by allowing humans to intuitively infer context from the URI itself. When a URI is designed to be human-readable, it often incorporates meaningful words or phrases that bring information about the resource it identifies. For instance, a URI like "https://www.example.com/blog/how-to-create-semantic-uris" is not only understandable to machines but also provides

valuable context to a human observer. By taking a quick look at this URI, one can deduce that it likely points to a blog post discussing the creation of semantic URIs. This dual-purpose approach to URI design promotes a harmonious relationship between humans and machines. While machines rely on the uniform structure of URIs for precise resource identification, humans benefit from the added clarity and user-friendliness of URIs that convey meaning. It simplifies navigation, assists in search engine optimization, and improves the overall experience when interacting with web content.

In summary, while the primary role of URIs is to facilitate machine-readable resource identification, it's advisable to make them human-readable as well. This dual readability ensures that both machines and humans can effectively engage with web resources, promoting accessibility, understanding, and efficient data utilization on the internet.

2.4 SPARQL

SPARQL is a powerful query language specifically designed for querying and manipulating RDF data. As the Semantic Web has grown, so too has the volume of data created, published, and managed using Semantic Web standards, especially via the RDF. This growth has made efficient processing of vast RDF datasets a challenge. SPARQL has become an integral tool for researchers and developers working with RDF datasets, such as the one used in ACORDAR, due to its ability to express complex queries and extract specific information from RDF graphs. SPARQL's versatility extends beyond simple queries. It supports filtering, aggregation, and even updating RDF data, offering a comprehensive suite of operations for efficiently handling RDF data.

Furthermore, as more data is made available in RDF format, there's a growing need for data analytics tools that go beyond traditional semantic search. Ferré's work on "Analytical Queries on Vanilla RDF Graphs with a Guided Query Builder Approach" [6] addresses this need. This approach directly answers analytical queries on unmodified RDF graphs by leveraging the computation features of SPARQL 1.1. It introduces a query builder that hides the complexities of SPARQL behind a natural language verbalization, providing intermediate results and suggestions at each step, making it more user-friendly and intuitive.

These advancements underscore the evolving nature of SPARQL and its adaptability to the challenges posed by the ever-growing Semantic Web.

2.5 Stream processing

Stream processing is a crucial approach in the realm of RDF (Resource Description Framework) graph analysis, particularly when considering its potential applications within dataset search. Stream processing involves the continuous analysis of data as it flows into a system, making it especially valuable for handling large RDF graphs efficiently. This methodology offers significant performance advantages compared to traditional methods that involve loading the entire dataset into a triple store or materializing the graph in memory. One key benefit of stream processing for RDF graphs lies in its ability to handle data incrementally. Instead of waiting to load an entire RDF dataset into memory or into a triple store, stream processing allows you to process incoming data as it arrives. This not only saves time but also conserves system resources, making it more memory-efficient. Since RDF datasets can be massive, loading them entirely can lead to memory bottlenecks and slow query performance. Stream processing mitigates these issues by only holding relevant portions of the graph in memory, reducing the strain on computational resources.

Furthermore, stream processing is well-suited for real-time updates and dynamic data sources, which are common scenarios in RDF data. In a real world context, where dataset search may involve continuously changing RDF datasets, stream processing ensures that your system stays up to date with the latest information. This dynamic nature aligns with the nature of RDF graphs, making stream processing a natural choice for maintaining the freshness and accuracy of your dataset search results. Numerous libraries are available for processing RDF datasets as streams, and Apache Jena, which is employed in ACORDAR, is a prime example.

In the processing of RDF datasets, libraries often adhere to stringent error-handling protocols when confronted with RDF serializations, prevalent in real-world datasets. These protocols usually entail terminating the stream upon detecting an error. Nonetheless, even in the presence of syntax errors, stream-based processing of RDF graphs can yield valuable insights. Directly loading these problematic files into a triple store would typically render the file unanalyzable due to errors. An intrinsic challenge when handling RDF files as sequences of triples involves the necessity for methods such as regular expressions or pattern matching. This is because using SPARQL queries isn't feasible when processing files as streams, given their demand to materialize the entire graph

in memory.

There is a notable tradeoff in RDF dataset processing between processing speed and query complexity. While streaming analysis of RDF datasets offers unparalleled speed, it often limits the depth of analysis to more superficial information. In contrast, the process of ingesting files into a triple store for the purpose of employing SPARQL queries offers a comprehensive investigative approach, enabling an in-depth exploration of intricate patterns within the data. This in-depth method, however, requires more time both to load and to query the graph, indicating a balance between speed of execution and depth of analytical examination.

Reservoir Sampling Reservoir sampling is a randomized algorithmic method aimed at selecting a representative sample from large datasets, especially when the total number of elements in the dataset is unknown in advance [20]. This makes reservoir sampling especially useful for data streams, which are inherently dynamic, constantly changing, and typically uncertain in their total volume. When examining RDF graphs, encompassing entities, literals, properties, and classes, reservoir sampling can be an effective tool for obtaining a significant subset from these.

2.5.1 Management of Extensive RDF Graphs

Managing expansive RDF graphs is challenging due to the significant data scale they encompass. Storing these datasets requires more resources, as it's not only about their large size but also the necessity for efficient indexing and query capabilities. Furthermore, the computational cost rises when analyzing them, especially when dealing with complex tasks like SPARQL queries.

Motivated by the complexities and growth of RDF datasets, researchers have been inspired to develop more flexible and effective methods for evaluating SPARQL queries. One notable contribution in this field is the "Scalable Semantic-Based Distributed Approach for SPARQL Query Evaluation" by Sejdiu et al. [17]. This approach utilizes a semantic-based partitioning strategy within the SANSA framework [10], with a focus on horizontal scalability and efficient query operations, particularly in distributed environments. In prior research, Sejdiu et al. [16] highlighted these challenges, suggesting an innovative resolution. Leveraging the distributed in-memory capabilities of Apache Spark, a platform

recognized for its rapid in-memory data operations, they devised a technique specifically tailored for computing metrics on vast RDF datasets. Their proposal effectively addresses the challenges posed by the large-scale nature of RDF datasets, ensuring scalability and efficient processing.

Beyond mere processing capability, in-depth analysis of such extensive RDF datasets can place a significant computational load, including concerns related to memory allocation. The task of collecting, preserving, and managing triples from an RDF graph can consume substantial storage and computational resources. This highlights the need for advanced algorithms and data structures to ensure accurate and streamlined analysis.

In conclusion, as the scope of the semantic web expands and RDF datasets grow, the demand for innovative methods and instruments to efficiently handle these datasets intensifies. These contributions serve as a example of the potential of distributed in-memory processing in tackling the challenges associated with analyzing extensive RDF data.

2.6 Betweenness Centrality

Betweenness centrality stands as a significant metric in the field of network analysis, offering a means to quantify the importance of a node within the intricate web of a graph (see Figure 2.1). This metric precisely measures how often a particular node takes on the role of a connector or bridge on the shortest paths connecting other nodes. In the specialized domain of Dataset Search, especially when dealing with the complex structure of RDF graphs, betweenness centrality becomes an important metric. It identifies nodes that serve as essential connectors or intermediaries, linking distinct segments of the dataset. Such insights, derived from the calculation of betweenness centrality, hold great value for dataset search mechanisms. They enable these systems to prioritize or emphasize nodes that are inherently central to the dataset's structure. Additionally, due to their high connectivity, these nodes often serve as primary gateways for users searching for specific data. This should enhance the overall efficiency and accuracy of the IR process.

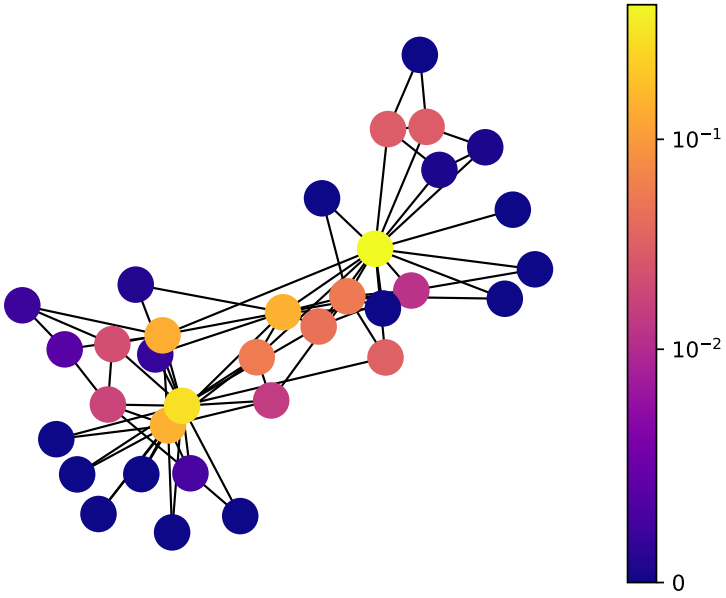

Figure 2.1: Betweenness Centrality

Formally, the betweenness centrality of a node v is defined as the sum of the fraction of all-pairs shortest paths that pass through v. Mathematically, it is given by the equation:

$$C_B(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the total number of shortest paths from node s to node t, and $\sigma_{st}(v)$ is the number of those paths that pass through node v. The sum is taken over all pairs of nodes s, t in the graph, excluding the node v itself. This metric indicates how effectively a node serves as a channel for facilitating the flow of information within the network.

2.6.1 Approximation Method

Computing exact betweenness centrality for nodes in large networks is computationally demanding. To mitigate these challenges, various approximate methods have been developed by the community. Among these, the `Approximate Betweenness Centrality Algorithm` [3] stands out as, effectively providing es-

timates of betweenness centrality while significantly reducing computational demands.

The core concept of this approximation method involves the random selection of a subset of nodes from the entire graph. The size of this subset is typically determined by either a user-defined parameter or a predefined sampling rate. Once this subset is obtained, the algorithm focuses on identifying the shortest paths connecting these selected nodes. Subsequently, it counts how frequently each node appears on these identified paths. These raw counts are then adjusted and normalized to produce an estimate of the node's betweenness centrality.

It is important to highlight that the accuracy of the results generated by the approximate betweenness centrality method is closely related to the size of the sampled subset and the scaling coefficients used. Opting for smaller subsets along with aggressive scaling approaches may accelerate computation but could compromise accuracy. Conversely, larger subsets and more conservative scaling methods, while ensuring higher precision, may impose greater computational demands.

In situations where dealing with massive networks makes exact betweenness centrality calculations virtually impossible due to computational constraints, these approximate strategies prove to be invaluable tools. They effectively address a delicate balance between accuracy and computational efficiency, empowering network analysts to gain profound insights into the pivotal roles of nodes within complex networks.

The `Approximate Betweenness Centrality Algorithm`, implemented in NetworkX, is based on the work of Brandes [1], but it leverages sampling to make computation faster.

3

ACORDAR reproducibility

In this chapter, we focus on replicating the foundational architecture of ACORDAR, an IR system tailored for RDF datasets. Our reproducibility efforts are organized into four sequential phases, each designed to reconstruct the ACORDAR system's original functionality.

- **Downloading the Collection**: Obtaining an accurate representation of the ACORDAR dataset collection serves as the initial and foundational phase of our reproduction work.
- **Extracting the Data**: This phase involves navigating the RDF structures to extract data, mirroring the methodology in the original system.
- **Creating the Index**: An index aligned with the design principles of the original ACORDAR system is created to streamline search operations.
- **Searching and Performance Comparison**: The final phase encompasses evaluating the replicated system's search performance to assess its fidelity to the original system.

3.1 Acquiring ACORDAR collection

The first step in the reproducibility work involved acquiring the datasets necessary for replicating the results obtained in the ACORDAR dataset search system. We created a Python script to streamline this task, utilizing the popular Requests [1] library. This choice was made not only because of the library's widespread use but also due to its user-friendly API, robust network connection handling, and its ability to gracefully handle temporary failures. To guarantee a seamless data retrieval process, we intentionally avoided implementing parallel downloads. The main objective of the script was to fetch resources from a list of URLs specified in a JSON file generously provided by ACORDAR. The server's status codes function as signals reflecting the state of the request. To elaborate, HTTP status codes within the range of 200 to 399 indicate either successful responses or redirections. A code belonging to the 2xx category indicates that the request has been received, understood, and confirmed as successful. In contrast, 3xx codes indicate that additional actions are required by the user agent to fulfill the request, often involving URL redirection.

This stage came with its own set of challenges. In particular, out of the 34,283 URLs embedded within the datasets, a total of 5,302 URLs failed to yield a valid HTTP status code, primarily falling within the 400 or 500 error code categories (refer to Figure 3.1). Such codes typically indicate issues like resource unavailability, server timeouts, or other server-side obstacles. Given these conditions, a single retrieval attempt was conducted for each link. URLs that did not return a status code within the 200-399 range were recorded in the `metadata.json` file, under the `failedURLs` list.

Despite encountering server-related errors, a total of 28,981 files were successfully downloaded. Among these downloaded files, there were 20 archives that have the potential to generate one or more files when extracted.

[1] `https://pypi.org/project/requests/`

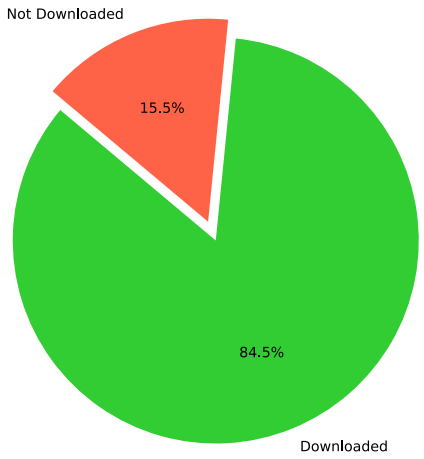

Figure 3.1: Downloaded Files

In addition to the dataset acquisition process, it's important to highlight the metadata that was provided (by ACORDAR's creators) within the same JSON file in which download URLs were reported. The following fields were included as metadata:

- **ID**: For unique identification and tracking of datasets.
- **Title**: A descriptor indicating the dataset's subject matter.
- **Description**: A concise summary of the dataset's content.
- **Author**: Information for assessing dataset provenance.
- **Tags**: Keywords for easier dataset categorization and retrieval.

Notably, the JSON file occasionally contained additional field such as 'Download Date'. This field could be useful for tracking when the datasets were last updated or accessed, or it may help in ensuring that you are working with the correct version of the dataset.

A limitation present in ACORDAR's JSON metadata is the lack of indicators for graphs that are serialized in multiple formats but encapsulate identical data. Addressing this shortcoming could optimize the system's efficiency, allowing for the selective download of datasets in preferred formats. This would enhance both temporal and computational resource management in subsequent phases.

To optimize the data acquisition process, a structured folder system has been implemented for each dataset. Within these dedicated folders, one can find not only the downloaded data files but also a JSON metadata file (`metadata.json`) that provides comprehensive information about the dataset.

The accompanying JSON file extends beyond the standard metadata fields provided by ACORDAR's initial JSON format, which encompass details such as title, description, author, and tags. It also incorporates additional information, such as the source URLs from which the data was retrieved, the filenames of the downloaded data, and a list of URLs that returned errors during the download.

While some challenges were encountered, the large volume of successfully downloaded files lays a robust foundation for the ensuing steps in the replication effort.

3.2 Extracting Data from the Downloaded Files

The fundamental element for replicating the ACORDAR baseline is the extraction of four primary data elements from the RDF datasets, a precursor for the next phases of the dataset search system. These elements include:

- **Entities**: Distinct objects or concepts within the RDF data.
- **Classes**: Categories to which entities belong.
- **Literals**: Non-resource data such as text or numbers linked to entities or properties.
- **Properties**: Relationships between entities defined through attributes or links.

To achieve this, the ACORDAR paper utilized `Apache Jena` [2], a powerful Java framework designed for RDF data manipulation. Apache Jena's extensive toolkit simplifies RDF data parsing, querying, and manipulation. Drawing from insights gained from the Search Engine course, the extraction pipeline was constructed utilizing Python, due to its robust ecosystem tailored for data processing pipeline development. This choice presents the advantage of enhanced code reusability in subsequent phases of the research. The goal of the extraction pipeline is to systematically segment the data into four separate UTF-8 encoded files, each representing one of the primary elements: Entities, Classes, Literals,

[2] `https://jena.apache.org/`

and Properties. During this phase, URIs were indexed for Entities, Classes, and Properties, while Literals were extracted in their raw form.

In the context of datasets with a manageable size, the in-memory method utilizing RDFLib combined with SPARQL queries was employed to extract the data fields. For more extensive datasets, a custom streaming mechanism was devised to guarantee efficient data extraction without compromising data integrity. We purposefully avoided in-depth extraction and indexing of human-readable fields like associated labels. This was due to two reasons: the ACORDAR paper's not mentioning such a process, and the lack of evidence supporting this method in the codebase. Indexing nodes using their labels leads to duplication since labels, which are literals, get indexed both individually and again as node labels. This method serves two critical functions: It addresses potential encoding discrepancies and ensures consistent textual data handling across RDF datasets. The UTF-8 encoding selection is a proactive measure to mitigate potential encoding-related issues.

Optimizing Data Extraction Pipeline for Large Datasets During the initial development phase, the data extraction process was projected to take about 72 hours. However, the size of some datasets, some surpassing 10GB, posed significant challenges. To address this, optimizations were implemented in the pipeline. RDFLib [3] was chosen for files under 200MB. For larger files, LightRDF[4] was used. Its streaming approach reduces computational workload and decreases processing time. The 200MB threshold was determined through practical tests, particularly assessing main memory usage. This boundary ensures efficient pipeline operation without straining computer resources. Leveraging the Python Multiprocessing library further sped up the extraction. Datasets were categorized into discrete tasks, with each viewed as an independent job. A dedicated function managed data extraction and storage for each dataset. A pool of worker processes, equal to the CPU core count, was established. This parallel processing approach significantly enhanced the pipeline's overall performance. By employing this strategy, the data extraction time was reduced from the estimated 72 hours to just 16 hours, underscoring the effectiveness of parallelization in optimizing the data extraction pipeline.

[3] https://github.com/RDFLib/rdflib
[4] https://github.com/ozekik/lightrdf

```
"id": "1",
"title": "[ 2017 a 2020 ] Cursos da Pós-Graduação ...",
...
"extracted": [
  {
    "file": "curso-sf-dump.ttl",
    "size": 6209230,
    "classesFile": "20230608-181949-curso-sf-dump-ttl-classes.txt",
    "literalsFile": "20230608-181949-curso-sf-dump-ttl-literals.txt",
    "entitiesFile": "20230608-181949-curso-sf-dump-ttl-entities.txt",
    "propertiesFile": "20230608-181949-curso-sf-dump-ttl-properties.txt",
    "connections": 70553,
    "connectedVertices": 15544,
    "averageLiteralsPerVertex": 5.998,
    "extractedWith": "RDFLib"
  },
  ...
],
"unusedFiles": []
```


Figure 3.2: Example of `metadata.json`

Collecting Extracted Data: A Shift in Strategy for Sustainable Data Management Expanding upon the improvements in data extraction, the next critical section involved the storage and organization of the extracted data. The initial approach was to store dataset entities, literals, properties, and classes in temporary memory locations, with the ultimate aim of consolidating them into JSON files. However, this method had two significant drawbacks. First, it proved to be highly memory-intensive, which conflicts with earlier efforts to optimize computational resources during the extraction phase. Secondly, as the dataset sizes grew, some resulting JSON files expanded to over 5GB, making them impractical for subsequent processing.

Recognizing these limitations, it became mandatory to reassess the data storage strategy. The updated strategy moved away from using large, all-in-one JSON files and instead adopted a more segmented solution. For each processed graph, four distinct text files were generated, each serving as a dedicated repository for entities, literals, properties, and classes, respectively. As a further step to ensure appropriate tracking and management of these text files, the names of the four aforementioned files are then saved inside the `metadata.json` file, as reported

in the Figure 3.2.
This not only resolved the problem of bulky file sizes but also aligned with the previous improvements made in the data extraction process. As a result, it established a more efficient and end-to-end data management strategy.

Data Extraction Challenges and Refinement Process Challenges were encountered during the initial stages of the data extraction pipeline, specifically regarding the utility of the downloaded files. To systematically address these issues, log files were employed to closely monitor the pipeline's operations. This analysis revealed that a subset of the downloaded files was unsuitable for subsequent data processing stages. The utilization of the `magic` [5] library, a Linux-based tool with Python bindings, facilitated an in-depth examination of these files, enabling a comprehensive assessment of their usability within the dataset. As outcome of the refinement process, two versions of the collection emerged:

- *Raw Collection*, that contains all files as downloaded
- *Refined Collection*, that derives from *Raw Collection*

A visual representation of this dataset refinement process can be seen in Figure 3.3.

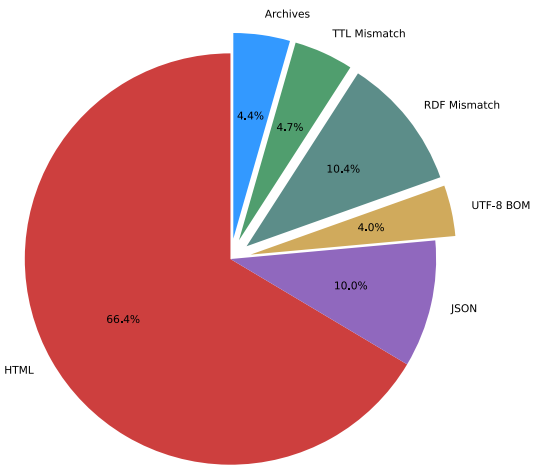

Figure 3.3: Dataset Refinement

[5] `https://github.com/ahupp/python-magic`

Dataset Refinement in Detail A Jupyter notebook was employed to encapsulate all essential functions for the transformation from *Raw Collection* to *Refined Collection*, aiming to enhance the reproducibility of the research. These functions encompass:

- **Elimination of Hypertext Markup Language (HTML) Files (299 elements)**: Utilizing the `magic` library, a custom function was developed to identify and remove 299 HTML-labeled files.
- **Pruning of Misconfigured JSON Files (45 elements)**: Some web servers responded with misleading Hypertext Transfer Protocol (HTTP) 200 codes while delivering JSON files with errors. Such files were systematically identified and excluded.
- **Correction of UTF-8 BOM Encoding (18 elements)**: Files with the unsupported UTF-8 BOM encoding were converted to standard UTF-8 to be compatible with RDFLib.
- **Classification of Files Without Extensions (47 elements)**: Using heuristic techniques and the `magic` library, 47 files lacking extensions were confidently categorized and formatted.
- **Conversion to Turtle (TTL) Format (21 elements)**: A subset of files was transformed into the TTL format using code heuristics and the `magic` library.
- **Decompression of Archives (20 files)**: A range of compressed archives, including TAR, ZIP, TAR-GZ, and BZ2 formats, were encountered. Specialized methods enabled their successful unpacking and incorporation into the dataset.

Thus, the Jupyter notebook plays an integral role in the dataset refinement process, contributing significantly to the research's reproducibility.

3.3 INDEXING

Indexing serves as the backbone of ACORDAR's dataset search mechanism, supporting the entire IR process. Employing the Lucene framework, ACORDAR ensures an efficient foundation for both indexing and querying RDF datasets. This section delineates the indexing strategy employed by ACORDAR, specifically focusing on the eight indexed fields crucial for comprehensive and accurate search results. These fields are categorized into two groups: metadata-based and RDF-based fields, each serving unique roles in dataset identification and retrieval.

3.3.1 Metadata-Based Fields

Metadata-based fields encapsulate auxiliary but crucial information about the dataset, facilitating the recognition of pertinent datasets. These fields include:

- **Title**: This field captures the dataset's title, facilitating dataset identification based solely on its name.
- **Author**: The author field records the names of dataset authors, enabling users to search for datasets created by specific individuals or groups.
- **Description**: Containing textual overviews of datasets, this field serves as a guide for users to comprehend the content of datasets.
- **Tags**: Tags provide a structured way to categorize datasets, making it easier for users to discover datasets related to specific topics or themes.

3.3.2 RDF-Based Fields

In contrast, fields rooted in RDF are explicitly crafted to capture and index the structural components present within RDF datasets. They comprise:

- **Entities**: This field includes identifiers for entities present in RDF dataseta. It includes information about the entities present in the RDF data, facilitating semantic search and entity-specific queries.
- **Literals**: The literals field indexes RDF literals, which are data values such as numbers, dates, and plain text.
- **Classes**: Capturing RDF classes, this field is instrumental for classifying datasets and facilitating topic-based searches.
- **Properties**: This field indexes RDF properties, documenting the relationships between entities and literals within the datasets.

The Indexing Pipeline To replicate the ACORDAR baseline, an in-depth analysis of the indexing pipeline is crucial. The pipeline operates in a sequence of steps beginning with the traversal of dataset folders, uniquely identified by IDs, within the ACORDAR collection. This facilitates data access and ensures a consistent indexing process. The pipeline initiates its operation by parsing the `metadata.json` files, which act as repositories for salient dataset-related attributes such as Title, Author, Description, and Tags. These metadata attributes

offer crucial insights into the datasets' nature, laying the foundations for the subsequent phases of indexing. After extracting the metadata attributes, the pipeline fetches the names of the four text files containing extracted data for each processed file from the `metadata.json` file. These documents relate to the key data elements (entities, properties, literals, and classes) that have been previously extracted for each file processed in the dataset. The pipeline then reads these files, employing reservoir sampling with a maximum size constraint of 10,000,000 elements to mitigate memory limitations. The sampled elements populate the corresponding fields in a Lucene document, which serves as the backbone of the IR system. In the replication efforts, Lucene's `StandardAnalyzer` was employed. This selection aligns with the analyzer used in the ACORDAR-2[6] codebase. It is presumed that a similar analyzer was chosen for ACORDAR, even though explicit documentation in the original paper is lacking and the source code is unavailable.

Index Configuration Within the ACORDAR-2 repository, we've discovered the configuration employed for field storage [7]. Considering the possibility that ACORDAR might have employed a similar configuration, we have opted to implement the same setup as detailed in ACORDAR-2 for our project. The following explanations detail the chosen configuration:

- **Stored:** Ensures retention of the field's original value in the index, crucial for accessing the field value post-search.
- **Tokenized:** Flags the field for tokenization before indexing, vital for text search and matching.
- **Store Term Vectors:** Enables term vector storage for the field, accelerating text analyses.
- **Store Term Vector Positions:** Allows storing term positions with term vectors, aiding in phrase querying and term proximity operations.
- **Index Options:** Set to `DOCS_AND_FREQS_AND_POSITIONS` to index document IDs, term frequencies, and positions, facilitating efficient querying and ranking. This option supports exact phrase and proximity searches, which are crucial for ranking algorithms like BM25F and ClassicSimilarity. Alternatives like `DOCS_AND_FREQS` and `DOCS` provide less indexing detail, suitable for different scenarios or algorithms. Specifically, `DOCS_AND_FREQS`

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

[7] `https://github.com/nju-websoft/ACORDAR-2/blob/main/Code/sparse/indexing/DatasetIndexer.java#L295`

omits term positions, aligning with algorithms like LMD, while `DOCS` solely indexes document IDs, making it lightweight but less suited for algorithms that leverage term frequencies and positions for scoring.

Undisclosed System Details The ACORDAR research paper lacks comprehensive information regarding essential operational aspects, such as tuple deduplication and the handling of large files. This poses a significant obstacle in achieving an accurate replication of their work.

Upon closely analyzing ACORDAR-2 codebase, it becomes apparent that they have employed a relational database structure. This structure encompasses columns for subjects, predicates, and objects, which are likely utilized for tuple deduplication. However, the paper itself lacks the necessary level of detail regarding this crucial element, impeding a precise replication process. Furthermore, the ACORDAR-2 codebase examination reveals that ACORDAR may have been adopting a strategy of limiting its analysis to the initial 1,000,000 triples within each RDF file. Nonetheless, this approach remains unaddressed within the original paper's discourse, thereby withholding details that replicators need for precise data ingestion and processing. Providing a more exhaustive clarification of these techniques would greatly elevate the potential for reproducing ACORDAR's research with precision.

3.4 Searching

The architecture of the retrieval system culminates in a search process designed to evaluate its performance. Within this context, the search mechanism of the ACORDAR system leverages the Lucene framework and incorporates four similarity metrics. These metrics are as follows:

- **Term Frequency-Inverse Document Frequency (TF-IDF)**: Known as `ClassicSimilarity` in Lucene.
- **BM25F**: Extends BM25 for field-specific weighting, implemented as `BM25Similarity` in Lucene.
- **Language Model with Dirichlet Smoothing (LMD)**: Utilizes probabilistic language models with Dirichlet smoothing for unseen terms.
- **Fielded Sequential Dependence Model (FSDM)**: Goes beyond term matching, emphasizing sequential term relationships.

Only TF-IDF, BM25F, and LMD are natively supported by Lucene. FSDM [21], although not a default Lucene similarity, is remarkable for outperforming other metrics in the ACORDAR system's experimental evaluations. FSDM can be particularly useful in scenarios where the structure of documents is important, and the relationships between terms can provide additional insights into their relevance to a query. In ACORDAR-2, this non-standard similarity appears to have been implemented as a re-ranking model on top of BM25F[8]. It's possible that they might have employed a similar implementation from ACORDAR-2 for the original ACORDAR system, but this information remains undisclosed, mainly due to the unavailability of the source code for ACORDAR.

To comprehensively evaluate performance, three query sets are used:

1. Sourced from the Text Retrieval Conference (TREC).
2. Curated by contributors of the ACORDAR system.
3. *ALL QUERIES*: A combination of the first two sets.

The evaluation process is broken down into three specific search scopes, each covering different Lucene document fields:

- **Metadata-only search**
- **Data-only search**
- **Comprehensive search** involving both metadata and data across all eight Lucene document fields

Incorporation of Stoplists In the context of IR, the incorporation of stoplists plays a crucial role in removing terms that tend to diminish the importance of more informative words. Consequently, this has an impact on the efficiency of retrieval algorithms in discerning relevant documents. The variation in performance metrics (with respect to the baseline), especially noticeable when using the BM25 and LMD similarity algorithms, suggested an investigation of the potential benefits of utilizing stoplists to reduce this gap, so an external stoplist sourced from Kaggle[9] was introduced into the experimental configuration. By

[8] `https://github.com/nju-websoft/ACORDAR-2/blob/main/Code/sparse/models/SparseRetrievalModels.java#L241`

[9] `https://www.kaggle.com/datasets/rowhitswami/stopwords`

employing this stoplist during the tokenization process, frequently occurring yet low-information words were removed, with the aim of enhancing the system's retrieval precision and relevance.

During the examination of the ACORDAR-2 repository, a reference to the Natural Language Toolkit (NLTK) stoplist was identified. This discovery highlighted the possible use of stoplists within the ACORDAR system. Consequently, this finding prompted the integration of the NLTK stoplist into subsequent configurations for comparative analysis.

Analysis of Boosting Techniques During the in-depth analysis of the repository (ACORDAR-2), we identified the usage of boosting techniques as a component for enhancing IR performance. The boosting process was realized through the assignment of specific weights to different query fields, and these weights are summarized in the table presented below. The implementation of boosting strategies remarkably contributed to the system's performance, achieving results that closely approximated the baseline metrics of the original ACORDAR system.

Table 3.1 displays the boost weights assigned specifically for metadata using three methods: BM25, TF-IDF, and LMD. It can be observed that the weights for the fields vary depending on the method used. For instance, both TF-IDF and LMD assign a full weight (1.0) for the 'Title', suggesting its significance in these methods.

Table 3.1: Boost Weights for Metadata

Method	Title	Description	Author	Tags
BM25	0.5	0.3	0.2	0.2
TF-IDF	1.0	0.6	0.4	0.5
LMD	1.0	0.8	0.9	0.7

In contrast, Table 3.2 demonstrates the boost weights when only considering data. A noticeable observation is the high weight assigned to the 'Literal' field, especially when using TF-IDF and LMD methods where it attains the maximum value of 1.0. This indicates a possible inclination towards 'Literal' data during the boosting process when metadata is excluded, that is the field having the highest chance of containing human readable values.

Table 3.2: Boost Weights for Data

Method	Entity	Literal	Class	Property
BM25	0.1	0.7	0.2	0.2
TF-IDF	0.3	1.0	0.6	0.3
LMD	0.3	1.0	0.1	0.6

However, when combining both metadata and data for boosting, as presented in Table 3.3, an interesting trend emerges. The weights for metadata fields, are significantly higher than most of the data-specific fields. Particularly, the 'Title' field maintains a consistent high weight of 1.0 across all three methods. This skewed weighting towards metadata in the combined boosting scenario implies that, when additional data (extracted from graphs) are incorporated into the system, there is a need to deliberately steer the system towards prioritizing matches on metadata. This is likely because blindly adding more data could deteriorate overall performance, necessitating a control mechanism in the form of higher boost weights for metadata. In conclusion, the boosting weights for the overall configuration are rewarding more matches on metadata, implicitly admitting that adding data makes the overall performance worse if the system isn't directed to deemphasize them.

Table 3.3: Boost Weights for Full Data

Method	Title	Description	Author	Tags	Entity	Literal	Class	Property
BM25	1.0	0.9	0.9	0.6	0.2	0.3	0.1	0.1
TFIDF	1.0	0.7	0.9	0.9	0.8	0.5	0.1	0.4
LMD	1.0	0.9	0.1	1.0	0.2	0.3	0.2	0.1

4

Enhancing Dataset Search Performance using Graph Structure

The effort to replicate the functionality of ACORDAR served as a foundational platform for exploring the impact of RDF graph structures on the improvement of dataset search performance. This exploration was driven by the hypothesis that a more in-depth analysis of RDF graph structures could result in enhancements in system performance.

The core of this improvement effort can be encapsulated by two fundamental concepts:

- **Feature Engineering for Re-Ranking**: Initially, our focus was on extracting numerical features from RDF graphs. These features were designed to enhance the ranking mechanism of the ACORDAR system by providing additional metrics for re-ranking. This analytical step aimed to generate a more relevant set of search results.
- **Leveraging Important Nodes in RDF Graph Structure**: The second aspect of the research delved into the exploration of interconnections within RDF graph structures. These complex networks feature nodes with distinct significance, often serving as pivotal connectors or hubs. Our aim was to discern these influential nodes and integrate them strategically into the search process.

4.1 Re-Ranking based on Numerical Features

In the context of IR, ranking documents is often conceptualized as a regression task. The primary objective of this task is to predict a document's relevance score with high accuracy, utilizing various features intrinsic to the document itself. Once these predictive scores are generated, the documents are subsequently arranged in descending order based on these scores, thereby highlighting the document with the highest score as the most pertinent to the search query [15].

To improve the ranking process, the data extraction pipeline goes beyond retrieving fundamental elements like classes, entities, literals, and properties from RDF datasets. It also captures a variety of metrics that provide insights into the structural details of the datasets. These metrics play an important role in the subsequent re-ranking of documents. Specifically, the following metrics were observed:

- **Class Count**: This metric reveals the quantity of classes within the RDF graph.
- **Literal Count**: It assesses the number of literal values, such as text strings or numerical data, in the RDF dataset.
- **Entity Count**: Representing the fundamental elements in RDF data, this metric provides insights into the dataset's size and complexity.
- **Property Count**: This quantifies the relationships or connections between entities and literals, assisting in evaluating the dataset's degree of connectivity and interdependency.
- **Total Connection Count**: This includes all forms of connections in the graph, including those among entities, classes, and literals, offering a comprehensive perspective on the graph's connectivity.
- **Connected Vertex Count**: This metric reflects the level of interconnections within the graph. A higher value indicates a more intricately linked and complex dataset.
- **Average Literals per Vertex**: This metric measures the distribution of literal values across vertices

4.1.1 Numerical Feature Extraction

To obtain the previously outlined metrics, SPARQL queries were utilized on RDF graphs loaded via RDFLib. SPARQL queries present a robust and adaptable mechanism for exploring relationships and entities found within RDF data. These queries ease precise retrieval of specific features, including the number of connection among vertices, number of connected vertices (unique nodes that are either subjects or non-literal objects in any triple), and the average literals per vertex.

```
SELECT (count(*) as ?count)
WHERE {
    ?s ?p ?o.
    FILTER(!IsLiteral(?s))
    FILTER(!IsLiteral(?o))
}
```

Code 4.1: Example of SPARQL query to extract the number of connection among vertices

The SPARQL query in Listing 4.1 serves as an example of how numerical features of the graph, particularly the number of connections among vertices, are extracted. This query targets all triples (?s ?p ?o) within the graph. To ensure that the subject (?s) and object (?o) represent vertices rather than literals, filters `FILTER(!IsLiteral(?s))` and `FILTER(!IsLiteral(?o))` are applied. The query then counts the number of such triples which gives the total number of connections among vertices. A different approach became necessary when dealing with datasets exceeding the 200MB limit. Instead of using SPARQL queries, a custom method was integrated into the Python script designed for processing these larger files. This method uses a streaming approach to process the data incrementally. For instance, to compute the same metric demonstrated above, a counter is incremented upon detecting a triple connecting two vertices. It was crucial to extract the same structural metrics for these larger files files, mirroring the process used for smaller ones with SPARQL queries. As previously mentioned, custom logic was devised to continuously compute and update metrics, namely class count, literals, entities, properties, total graph connections, connected vertices, and average literals per vertex. This approach guarantees scalable analyses despite dataset sizes, ensuring consistent metric derivation regardless of file dimensions.

4.1.2 Numerical Feature Utilization

To discern the correlation between the aforementioned features and the relevance of datasets, the *Random Forest Regressor* machine learning model was employed. At its core, this model is an intelligent algorithm designed to discern patterns and relationships within data, utilizing its insights to predict values based on prior information.

ACORDAR's relevance judgments serve as indicators of a dataset's utility in relation to specific queries, assigning scores of 1 for partial relevance and 2 for complete relevance. However, to ensure data reliability, judgments related to unavailable or non usable datasets were excluded from consideration.

Given this context, it becomes essential to introduce a scoring mechanism that combines both Lucene's scores and the model's predictions. This is encapsulated by the formula for the Final Score of a document d_i:

$$\text{Final Score}(d_i) = \frac{\text{Normalized Document Score}(d_i) + \text{RandomForest Score}(d_i)}{2}$$

Where:

- *Final Score* for a document d_i represents the average of its *Normalized Document Score* and *RandomForest Score*
- Both the *Normalized Document Score* and the *RandomForest Score*, as well as the resultant *Final Score*, lie in the range [0,1].

The *Normalized Document Score* for a given document d_i is articulated as:

$$\text{Normalized Document Score}(d_i) = \frac{\text{Score}(d_i)}{\max(\text{Score}(d_j) \text{ for all documents } d_j)} \quad (4.1)$$

Given Equation 4.1, the normalization procedure adjusts the score attributed to a document by Lucene relative to the highest score observed across the ten documents denoted as d_j.

4.2 Leveraging Important Nodes in Graph Structure

The process of identifying and extract important nodes from the RDF graph structure involved the usage of RDFLib and NetworkX to manipulate and analyze the graph.

To effectively leverage the graph structure and identify crucial nodes within it, I employed the RDFLib library, specifically utilizing the `rdflib to networkx graph` function. This conversion enabled a streamlined process for subsequent analysis.

Having the dataset in the form of a NetworkX graph, the betweenness centrality algorithm, as provided by NetworkX was employed. This algorithm calculates betweenness centrality, which effectively signifies a node's function as a bridge connecting different parts of the network. A higher value of betweenness centrality indicates that a node plays an important role in enhancing connectivity and promoting the flow of information.

Upon computing betweenness centrality for nodes, these were then ranked in descending order of their centrality values. This ranking enabled the extraction of URIs for the most significant nodes.

To optimize computational resources and ensure feasible processing durations, a cap was set on the number of extracted nodes: a consistent 20 nodes per RDF file. Recognizing the computational intensity of precise betweenness centrality computation, an approximate method was employed. Using 100 nodes as starting points, this approach not only accelerated the computation but also retained a respectable accuracy level.

In the course of implementing the aforementioned optimizations, certain limitations in processing time were observed, necessitating the introduction of specific constraints. To maintain reasonable computing time, two thresholds were established:

- RDF files necessitating a loading time exceeding 100 seconds via RDFLib were precluded from subsequent analysis.
- Files that necessitated more than 200 seconds for processing through the NetworkX betweenness centrality algorithm were omitted.

These thresholds were established upon analysis of running times, ensuring that only feasible computations were included in the analysis.

Even with the application of these restrictions and the use of multiprocessing techniques to streamline processing, the total running time for processing all files exceeded the 10-hour mark. This emphasizes the computational complexity of the task and highlights the need for careful resource management and optimization throughout the research process.

It is noteworthy that, as with prior analyses, relevance judgments for unavailable or unusable datasets were disregarded. Further examination of the results, along with discussion of the findings, will be provided in the upcoming chapter.

5 Experimental Evaluation

5.1 Reproduced Experiments

This section aims to assess the performance of the replicated ACORDAR system by utilizing key metrics: NDCG@5, NDCG@10, MAP@5, and MAP@10, as emphasized in the original study. We aim to comprehend ACORDAR's behavior by conducting different experiments, and then we will compare our findings with the results established in the initial research as a reference point.

Baseline Performance Metrics To establish a point of comparison, Table 5.1 summarizes the baseline performance metrics evaluated on *ALL QUERIES* as reported in the original ACORDAR study.

Table 5.1: Performance Metrics of ACORDAR across all queries

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
BM-25F	0.5538	0.5877	0.3198	0.4358
BM-25F[m]	0.5045	0.5250	0.2859	0.3838
BM-25F[d]	0.2163	0.2196	0.1385	0.1550
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

In Table 5.2, which presents the overall results, the following observations can be made:

1. **TF-IDF**: The discrepancy between the original and reproduced results is the smallest among all methods across all metrics. The most notable difference is -0.0410 for NDCG@10.
2. **BM25F**: This method demonstrates the greatest divergence in NDCG@5 and NDCG@10, with differences amounting to -0.1781 and -0.1737, respectively.
3. **LMD**: LMD exhibits the most pronounced deviation across all metrics, particularly for NDCG@5 and NDCG@10, with differences of -0.2750 and -0.2823, respectively.

Note: In all the tables of this section, the `Difference` rows are calculated as the experimental reproduced value minus the reference value. Positive values signify that results are better than the baseline, while negative values signify the opposite.

Table 5.2: Performance comparison using combined metadata and data across all queries in the raw collection

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4756	0.5042	0.2692	0.3669
	Difference	-0.0332	-0.0410	-0.0179	-0.0307
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.3757	0.4140	0.2087	0.2793
	Difference	-0.1781	-0.1737	-0.1111	-0.1565
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.2715	0.2982	0.1549	0.1926
	Difference	-0.2750	-0.2823	-0.1717	-0.2398

Table 5.3 showcases the results using metadata only. The observations are as follows:

1. **TF-IDF**: The discrepancies remain relatively minor. The most significant difference is noted in NDCG@10, with a value of -0.0279.
2. **BM25F**: Notably, the deviations are minimal, and in some cases, the reproduced results marginally outperform the original ones, as seen with NDCG@5 and MAP@5.
3. **LMD**: The reproduced results exhibit marginal improvements over the original results, with the most significant improvement of +0.0175 observed in NDCG@5.

Table 5.3: Performance comparison using metadata only across all queries in the raw collection

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.4743	0.5019	0.2676	0.3685
	Reproduced	0.4599	0.474	0.2583	0.3436
	Difference	-0.0144	-0.0279	-0.0093	-0.0249
BM25F	Original	0.5045	0.5250	0.2859	0.3838
	Reproduced	0.5059	0.5222	0.2865	0.3820
	Difference	+0.0014	-0.0028	+0.0006	-0.0018
LMD	Original	0.4363	0.4573	0.2543	0.3325
	Reproduced	0.4538	0.4699	0.2679	0.3446
	Difference	+0.0175	+0.0126	+0.0136	+0.0121

In Table 5.4, which presents the results based on extracted data only, the following observations are made:

1. **TF-IDF**: The deviations are more pronounced compared to those in the metadata results, with the largest difference being -0.0606 for NDCG@5.
2. **BM25F**: The reproduced results trail notably behind the original ones, particularly for NDCG@5, which presents a difference of -0.0857.
3. **LMD**: Similarly to BM25F, a significant gap exists between the original and reproduced results, with NDCG@5 showing the largest difference of -0.0901.

Table 5.4: Performance comparison using data only across all queries in the raw collection

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.1304	0.1384	0.0677	0.0802
	Difference	-0.0606	-0.0579	-0.0321	-0.0397
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1306	0.1431	0.0747	0.0906
	Difference	-0.0857	-0.0765	-0.0638	-0.0644
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.1497	0.1655	0.0801	0.0998
	Difference	-0.0901	-0.0868	-0.0614	-0.0674

The earlier experimental results showed significant differences, leading us to suspect that there might be issues with incomplete or missing data. These variations not only affect the accuracy and reliability of our findings but also make it difficult for others to replicate the experiments. As detailed in Chapter 3, we improved our collection by addressing compression and file extension issues. With this refined dataset in hand, we proceed to scrutinize the impact of these improvements on the discrepancies observed in the results.

In the table that presents the overall results on the refined collection without big files (files that are larger than 200MB) (Table 5.5), we observe:

1. **TF-IDF**: The discrepancies are modest. NDCG@10 stands out with a difference of -0.0410, indicating a slight underperformance in the reproduced results.
2. **BM25F**: The differences are more pronounced here. The largest gap is evident in NDCG@5 at -0.1834, suggesting a significant decline in the quality of reproduced results for the top 5 documents.
3. **LMD**: The results reveal substantial variations, with NDCG@10 showcasing a difference of -0.2808, indicating decreased performance in document rankings.

Table 5.5: Performance comparison using combined metadata and data across all queries in the refined collection without files that are larger than 200MB

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4751	0.5042	0.2691	0.3667
	Difference	-0.0337	-0.0410	-0.0180	-0.0309
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.3704	0.4099	0.2051	0.2752
	Difference	-0.1834	-0.1778	-0.1147	-0.1606
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.2775	0.2997	0.159	0.1952
	Difference	-0.2690	-0.2808	-0.1676	-0.2372

In the table that presents results on metadata only on the refined collection without big files (Table 5.6), we observe:

1. **TF-IDF**: The differences between the original and reproduced results are minor. The most prominent variance appears in NDCG@10, which has a difference of -0.0279, hinting at a modest underperformance in the reproduced top 10 document rankings.

2. **BM25F**: Interestingly, the differences between original and reproduced are very minimal, with the Reproduced results occasionally performing slightly better than the Original. For instance, the reproduced result for NDCG@5 is marginally higher than the Original by +0.0014, although this difference is so small that it might not be of practical significance.

3. **LMD**: Unlike the other models, LMD's reproduced results tend to outperform the original ones. The variance in NDCG@5 stands out with a difference of +0.0175, indicating that the reproduced results provide slightly better precision across the top 10 documents.

It should be noted that the negative values in the difference rows for BM25F and LMD imply that the Reproduced results are better than the Original in those instances.

Table 5.6: Performance comparison using metadata only across all queries in the refined collection without files that are larger than 200MB

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.4743	0.5019	0.2676	0.3685
	Reproduced	0.4599	0.474	0.2583	0.3436
	Difference	-0.0144	-0.0279	-0.0093	-0.0249
BM25F	Original	0.5045	0.5250	0.2859	0.3838
	Reproduced	0.5059	0.5222	0.2865	0.3820
	Difference	+0.0014	-0.0028	+0.0006	-0.0018
LMD	Original	0.4363	0.4573	0.2543	0.3325
	Reproduced	0.4538	0.4699	0.2679	0.3446
	Difference	+0.0175	+0.0126	+0.0136	+0.0121

Considering the data results for all queries on the refined collection without accounting for files larger than 200MB (Table 5.7), we can summarize:

1. **TF-IDF**: The Reproduced results exhibit an evident underperformance relative to the original scores. The largest discrepancy is seen in NDCG@5 with a difference of -0.0630, pointing to a substantial decline in the effectiveness of the top 5 retrieved documents in the reproduced version.

2. **BM25F**: The deviations between original and reproduced results are significant. The disparity in NDCG@5 stands out with a difference of 0.0900, emphasizing a decline in the average precision across queries in the Reproduced results.

3. **LMD**: For LMD, the differences between the two sets of results are noticeable across all metrics. The NDCG@5 value showcases a difference of -0.0917, indicating a decline in the ranking effectiveness of retrieved documents.

Table 5.7: Performance comparison using data only across all queries in the refined collection without files that are larger than 200MB

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.1280	0.1362	0.0667	0.0789
	Difference	-0.0630	-0.0601	-0.0331	-0.0410
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1263	0.1383	0.0724	0.0880
	Difference	-0.0900	-0.0813	-0.0661	-0.0670
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.1481	0.1624	0.0799	0.0992
	Difference	-0.0917	-0.0899	-0.0616	-0.0680

In our ongoing analysis, we now turn our attention to data that includes files exceeding 200MB. To effectively process such large files, they were parsed using a streaming methodology.

Assessing the outcomes for all queries in the Raw Collection, as displayed in Table 5.8, the observations are as follows:

1. **TF-IDF**: The reproduced scores exhibit a decline relative to the original across all metrics, with NDCG@10 showcasing the most pronounced drop of -0.0460, signaling reduced ranking efficiency for the top 10 documents.
2. **BM25F**: There's a noticeable discrepancy between original and reproduced results. Specifically, the NDCG@10 difference of -0.0914 highlights a decline in precision of the results in the reproduced set.
3. **LMD**: Deviations between original and reproduced scores are most significant here. The variance of -0.3286 in NDCG@5 underscores a marked decrease in the efficacy of the top 5 results in the reproduced set.

Table 5.8: Performance comparison using combined metadata and data across all queries in the refined collection

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4747	0.4992	0.2681	0.3635
	Difference	-0.0341	-0.0460	-0.0190	-0.0341
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.4677	0.4963	0.2644	0.3523
	Difference	-0.0861	-0.0914	-0.0554	-0.0835
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.2179	0.2409	0.1292	0.1576
	Difference	-0.3286	-0.3396	-0.1974	-0.2748

Examining the metadata-only results for the refined collection excluding larger files, as depicted in Table 5.9, we note:

- **TF-IDF**: The original consistently (but slightly) outperforms the reproduced version across all measured metrics.
- **BM25F**: Mixed results are observed; NDCG@5 is nearly identical, while the original has a slight edge in NDCG@10 and MAP@10.
- **LMD**: Contrarily, the reproduced results excel beyond the original in every metric.

Table 5.9: Performance comparison using metadata only across all queries in the refined collection

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.4743	0.5019	0.2676	0.3685
	Reproduced	0.4599	0.4742	0.2583	0.3440
	Difference	-0.0144	-0.0277	-0.0093	-0.0245
BM25F	Original	0.5045	0.5250	0.2859	0.3838
	Reproduced	0.5053	0.5220	0.2862	0.3824
	Difference	+0.0008	-0.0030	+0.0003	-0.0014
LMD	Original	0.4363	0.4573	0.2543	0.3325
	Reproduced	0.4544	0.4699	0.2680	0.3443
	Difference	+0.0181	+0.0126	+0.0137	+0.0118

Regarding the results focusing only on data within the refined collection without larger files, as shown in Table 5.10, our findings are:

- **TF-IDF**: reproduced scores are notably lower with differences spanning from -0.0553 in MAP@5 to a significant -0.1091 in NDCG@5.
- **BM25F**: The original consistently ranks higher than the reproduced, with the peak difference at -0.1035 in NDCG@5.
- **LMD**: This model exhibits the most prominent disparities of all, particularly in metrics like NDCG@5 and NDCG@10, showing differences of -0.1400 and -0.1388, respectively.

Table 5.10: Performance comparison using data only across all queries in the refined collection

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.0819	0.0937	0.0445	0.0556
	Difference	-0.1091	-0.1026	-0.0553	-0.0643
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1128	0.1211	0.0669	0.0777
	Difference	-0.1035	-0.0985	-0.0716	-0.0773
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.0998	0.1135	0.0591	0.0702
	Difference	-0.1400	-0.1388	-0.0824	-0.0970

To mitigate the discrepancy observed between the original and reproduced results, we introduced the previously mentioned Kaggle stoplist. The following analysis outlines the impact of this adjustment.

Analyzing the cumulative outcomes for all search queries on the Refined Collection using the Kaggle stoplist (Table 5.11), the observations are:

- **TF-IDF**: Utilizing the Kaggle stoplist reduced the disparities in results. The variations now span from -0.0085 in MAP@5 to -0.0272 in NDCG@10.
- **BM25F**: An evident enhancement in the replicated results is seen, but a considerable difference still persists, particularly in NDCG@10 and MAP@10 with deviations of -0.0678 and -0.0625 respectively.
- **LMD**: Despite the introduction of the stoplist, the divergence in LMD remains significant. The NDCG@10 and MAP@10 metrics deviate by 0.3449 and 0.2796 respectively, implying that the stoplist's contribution to reducing the gap in this model was minimal.

Table 5.11: Performance comparison using combined metadata and data across all queries in the refined collection with Kaggle stoplist

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4899	0.5180	0.2786	0.3792
	Difference	-0.0189	-0.0272	-0.0085	-0.0184
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.4932	0.5199	0.2803	0.3733
	Difference	-0.0606	-0.0678	-0.0395	-0.0625
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.2162	0.2356	0.1264	0.1528
	Difference	-0.3303	-0.3449	-0.2002	-0.2796

From the outcomes achieved by utilizing the Kaggle stoplist on the refined collection focusing only on metadata (Table 5.12), we infer the following:

- **TF-IDF**: The stoplist has brought the reproduced results closer to the original values, leading to minimal discrepancies. The differences are marginal, lying between -0.0033 in MAP@5 to- 0.0082 in MAP@10.
- **BM25F**: Interestingly, the reproduced results with BM25F have not just approached the original, but in some metrics have slightly surpassed them. This is reflected in a negative difference value for NDCG@5 and MAP@5. The largest difference is a mere -0.0001 in NDCG@10, indicating a very close match.
- **LMD**: For the LMD model, the reproduced results after employing the stoplist are better than the original across all metrics. The difference ranges from -0.0134 in NDCG@10 to -0.0189 in NDCG@5.

Table 5.12: Performance comparison using metadata only across all queries in the refined collection with Kaggle stoplist

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.4743	0.5019	0.2676	0.3685
	Reproduced	0.4708	0.4947	0.2643	0.3603
	Difference	-0.0035	-0.0072	-0.0033	-0.0082
BM25F	Original	0.5045	0.5250	0.2859	0.3838
	Reproduced	0.5076	0.5249	0.2872	0.3847
	Difference	+0.0031	-0.0001	+0.0013	+0.0009
LMD	Original	0.4363	0.4573	0.2543	0.3325
	Reproduced	0.4552	0.4707	0.2689	0.3469
	Difference	+0.0189	+0.0134	+0.0146	+0.0144

From the results obtained by employing the Kaggle stoplist on the refined collection considering only extracted data (Table 5.13), we can derive the following observations:

- **TF-IDF**: Incorporating the Kaggle stoplist seems to have slightly improved the reproduced results compared to the prior refined version. The discrepancies remain with differences from -0.0535 in MAP@5 to -0.1051 in NDCG@5.
- **BM25F**: The discrepancies have reduced as well. While the reproduced results are still lower than the original, the difference is notably less across all metrics, ranging from -0.0658 in MAP@5 to -0.0965 in NDCG@5.
- **LMD**: The application of the Kaggle stoplist has led to improvements in the reproduced results when compared to the original LMD metrics. However, significant discrepancies persist with the largest difference being -0.1351 in NDCG@10.

Table 5.13: Performance comparison using data only across all queries in the refined collection with Kaggle stoplist

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.0859	0.0947	0.0463	0.0566
	Difference	-0.1051	-0.1016	-0.0535	-0.0633
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1198	0.1287	0.0727	0.0836
	Difference	-0.0965	-0.0909	-0.0658	-0.0714
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.1065	0.1172	0.0648	0.0752
	Difference	-0.1333	-0.1351	-0.0767	-0.0920

An in-depth examination of the ACORDAR-2 codebase revealed the utilization of the NLTK stoplist during its development. With this new information, it is plausible to speculate that the original ACORDAR may have also incorporated the NLTK stoplist into its data processing procedures. Based on this supposition, we integrated the NLTK stoplist into our analysis to investigate potential changes in result disparities.

Referring to Table 5.14, which presents the overall results of all queries on the raw collection post-NLTK stoplist integration:

- **TF-IDF**: The inclusion of the NLTK stoplist has led to a slight reduction in the disparity between the original and reproduced results across all metrics. The variation ranges from -0.0108 in MAP@5 to -0.0282 in NDCG@10, suggesting a minor enhancement in similarity.
- **BM25F**: Post-NLTK stoplist integration, the reproduced results remained consistently lower than the original results. However, the discrepancies were comparable to the previous version, ranging from -0.0394 in MAP@5 to -0.0664 in NDCG@10.
- **LMD**: Despite the addition of the NLTK stoplist, the reproduced LMD results exhibited notable discrepancies in comparison to the original values. A significant difference of -0.3439 was observed in NDCG@10.

Table 5.14: Performance comparison using combined metadata and data across all queries in the refined collection with NLTK stoplist

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4873	0.5170	0.2763	0.3799
	Difference	-0.0215	-0.0282	-0.0108	-0.0177
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.4940	0.5213	0.2804	0.3741
	Difference	-0.0598	-0.0664	-0.0394	-0.0617
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.2167	0.2366	0.1268	0.1540
	Difference	-0.3298	-0.3439	-0.1998	-0.2784

Upon analyzing the results for the metadata after the incorporation of the NLTK stoplist, as shown in Table 5.15:

- **TF-IDF**: The disparities between the original and reproduced results diminished, with the highest discrepancy being -0.0090 in NDCG@10.
- **BM25F**: Interestingly, the reproduced results exceeded the original across all metrics. The most notable improvement was at +0.0064 in MAP@10.
- **LMD**: Following the trend set by BM25F, the LMD model also demonstrated enhancements, with differences extending to +0.0151 in MAP@5.

Table 5.15: Performance comparison using metadata only across all queries in the refined collection with NLTK stoplist

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.4743	0.5019	0.2676	0.3685
	Reproduced	0.4717	0.4929	0.2649	0.3615
	Difference	-0.0026	-0.0090	-0.0027	-0.0070
BM25F	Original	0.5045	0.5250	0.2859	0.3838
	Reproduced	0.5108	0.5305	0.2913	0.3902
	Difference	+0.0063	+0.0055	+0.0054	+0.0064
LMD	Original	0.4363	0.4573	0.2543	0.3325
	Reproduced	0.4552	0.4715	0.2694	0.3475
	Difference	+0.0189	+0.0142	+0.0151	+0.0150

When examining the results derived from the refined collection, focusing only on extracted data and factoring in the NLTK stoplist (as presented in Table 5.16):

- **TF-IDF**: There was an increase in the disparity between the original and reproduced results across all metrics. The largest discrepancy observed was -0.1059 for NDCG@5.
- **BM25F**: The discrepancies expanded across all metrics, with the maximum divergence being -0.0963 for NDCG@5.
- **LMD**: In alignment with the previous findings, the results displayed greater variance relative to the original. The difference peaked at -0.1362 for NDCG@10.

Table 5.16: Performance comparison using data only across all queries in the refined collection with NLTK stoplist

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.0851	0.0955	0.0451	0.0561
	Difference	-0.1059	-0.1008	-0.0547	-0.0638
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1200	0.1290	0.0725	0.0837
	Difference	-0.0963	-0.0906	-0.0660	-0.0713
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.1043	0.1161	0.0630	0.0736
	Difference	-0.1355	-0.1362	-0.0785	-0.0936

A detailed analysis of the ACORDAR-2 repository unveiled the usage of boosting. Presuming its application in the original ACORDAR, it was incorporated into the pipeline. The addition of boosting deeply affected all evaluation metrics, narrowing the gap between the reproduced results and the baseline.

After examining the outcomes (as displayed in Table 5.17) for all queries within the refined collection, and taking into account the boosting weights detailed in Table 3.3, we can derive the following observations:

- **TF-IDF**: The incorporation of boosting resulted in notable discrepancies between the original and reproduced results. The largest divergence is observable in NDCG@10 with a difference of -0.0554.
- **BM25F**: With the application of boosting, the BM25F algorithm exhibited a more consistent alignment with the original, especially when compared with the other methods. Specifically, the smallest discrepancy was seen in MAP@5, measuring at -0.0101.
- **LMD**: The reproduced LMD algorithm results, while improved with boosting, still displayed a considerable difference from the original. The most pronounced deviation is observed in NDCG@10 with a difference of -0.0705.

Table 5.17: Performance comparison using combined metadata and data across all queries in the refined collection with boosting

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4647	0.4898	0.2618	0.3532
	Difference	-0.0441	-0.0554	-0.0253	-0.0444
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.5396	0.5604	0.3097	0.4142
	Difference	-0.0142	-0.0273	-0.0101	-0.0216
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.4880	0.5100	0.2876	0.3717
	Difference	-0.0585	-0.0705	-0.0390	-0.0607

The results on metadata (see Table 5.18) highlight on the effect of boosting (using the weights reported in Table 3.1) on the alignment between original and reproduced results:

- **TF-IDF**: The inclusion of boosting slightly increased the differences for all metrics when compared to the original. The largest discrepancy is evident in NDCG@10 with a difference of -0.0313.
- **BM25F**: Boosting enhanced the alignment between original and reproduced results. Particularly, the smallest difference was observed in MAP@5, which stands at -0.0017.
- **LMD**: Interestingly, for this model, the reproduced results outperformed the original in all metrics. The highest deviation from the original is found in NDCG@5 with a difference of +0.0202.

Table 5.18: Performance comparison using metadata only across all queries in the refined collection with boosting

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.4743	0.5019	0.2676	0.3685
	Reproduced	0.4525	0.4706	0.2555	0.3393
	Difference	-0.0218	-0.0313	-0.0121	-0.0292
BM25F	Original	0.5045	0.5250	0.2859	0.3838
	Reproduced	0.4980	0.5146	0.2842	0.3751
	Difference	-0.0065	-0.0104	-0.0017	-0.0087
LMD	Original	0.4363	0.4573	0.2543	0.3325
	Reproduced	0.4565	0.4706	0.2697	0.3443
	Difference	+0.0202	+0.0133	+0.0154	+0.0118

The results on data (see Table 5.19) offer insights into the variations observed due to the application of boosting (using the weights reported in Table 3.2):

- **TF-IDF**: Boosting brought about notable differences between the original and reproduced results, especially for NDCG@5 and NDCG@10 with discrepancies of -0.0980 and 0-.0970 respectively. The improvements, however, might not be substantial enough to consider the boosted results analogous to the original.
- **BM25F**: Among all the models, BM25F observed the least variation from the original when boosting was implemented. Nevertheless, the reproduced results are still considerably distant from the original, with the NDCG@10 metric having a difference of -0.0739.
- **LMD**: The LMD model experienced the most pronounced deviation in the NDCG@10 metric with a difference of -0.1101. The application of boosting, while reducing the gap, still yields reproduced results noticeably different from the original.

Table 5.19: Performance comparison using data only across all queries in the refined collection with boosting

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.0930	0.0993	0.0492	0.0598
	Difference	-0.0980	-0.0970	-0.0506	-0.0601
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1435	0.1457	0.0869	0.0982
	Difference	-0.0728	-0.0739	-0.0516	-0.0568
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.1383	0.1422	0.0831	0.0946
	Difference	-0.1015	-0.1101	-0.0584	-0.0726

Upon inspecting the overall results for all queries on the refined collection, with the application of a boosting strategy (using weigths reported in Table 3.3) and Kaggle stoplist (Table 5.20), the following insights can be deduced:

- **TF-IDF**: The reproduced results fall slightly short of the original figures. The difference in performance is particularly noticeable in the NDCG@5 and NDCG@10 metrics, with discrepancies of -0.0305 and -0.0405, respectively.
- **BM25F**: BM25F's reproduced outcomes are commendably close to the original results. Specifically, the disparity in the NDCG@10 metric is just -0.0244, indicating that the reproduced strategy effectively captures most of the original BM25F's performance.
- **LMD**: The LMD model presents the most pronounced gaps between the original and reproduced results, particularly in the NDCG@10 metric, with a difference of -0.0675. This larger discrepancy suggests challenges in faithfully reproducing the original LMD's efficiency.

Table 5.20: Performance comparison using combined metadata and data across all queries in the refined collection with Kaggle stoplist and boosting

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4783	0.5047	0.2687	0.3646
	Difference	-0.0305	-0.0405	-0.0184	-0.0330
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.5391	0.5633	0.3087	0.4164
	Difference	-0.0147	-0.0244	-0.0111	-0.0194
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.4852	0.5130	0.2882	0.3768
	Difference	-0.0613	-0.0675	-0.0384	-0.0556

Upon examination of the metadata results for all queries on the refined collection, with the application of a boosting strategy (using weigths reported in Table 3.1) and Kaggle stoplist (refer to Table 5.21), we can draw the following conclusions:

- **TF-IDF**: The reproduced results show a modest drop from the original figures. The metrics like NDCG@5 and NDCG@10 depict a difference of -0.0131 and -0.0203, respectively.
- **BM25F**: Remarkably, the reproduced results of the BM25F model are quite close to the original values. In some cases, such as NDCG@5, the reproduced results even surpass the original, albeit by a slim margin. The tiny differences, like -0.0008 in NDCG@10, illustrate the precision achieved in the replication process for BM25F in this setup.
- **LMD**: For the LMD model, the reproduced results approach the original values, signifying a competent replication. With a difference of only +0.0121 in the NDCG@10 metric, it is evident that the integration of Kaggle stoplist and boosting, provides a suitable environment for LMD's efficient performance.

Table 5.21: Performance comparison using metadata only across all queries in the refined collection with Kaggle stoplist and boosting

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.4743	0.5019	0.2676	0.3685
	Reproduced	0.4612	0.4816	0.2616	0.3494
	Difference	-0.0131	-0.0203	-0.0060	-0.0191
BM25F	Original	0.5045	0.5250	0.2859	0.3838
	Reproduced	0.5076	0.5242	0.2895	0.3844
	Difference	+0.0031	-0.0008	+0.0036	+0.0006
LMD	Original	0.4363	0.4573	0.2543	0.3325
	Reproduced	0.4516	0.4694	0.2675	0.3451
	Difference	+0.0153	+0.0121	+0.0132	+0.0126

Upon examination of the data results for all queries on the refined collection, with the application of a boosting strategy (using weights reported in Table 3.2) and Kaggle stoplist (refer to Table 5.22), we can deduce the following insights:

- **TF-IDF**: The difference between the original and reproduced results remains substantial, albeit slightly reduced from prior assessments. Specifically, discrepancies in metrics such as NDCG@5 and NDCG@10 amount to -0.0920 and -0.0906 respectively, pointing to the challenges inherent in replicating the original TF-IDF approach within this dataset configuration.
- **BM25F**: The BM25F model's reproduced results show a closer alignment with the original, even though differences exist. For instance, the variation in the NDCG@10 metric is -0.0754. The data suggests that, while this model benefits from the Kaggle stoplist and boosting, there's still room for improvement to attain full alignment with the original setup.
- **LMD**: In the case of the LMD model, the disparity between original and reproduced outcomes, particularly with a -0.1120 difference in NDCG@10, indicates notable challenges in the replication process. However, the results hint at the benefits of integrating Kaggle stoplist and boosting, especially when compared to previous configurations.

Table 5.22: Performance comparison using data only across all queries in the refined collection with Kaggle stoplist and boosting

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.0990	0.1057	0.0532	0.0646
	Difference	-0.0920	-0.0906	-0.0466	-0.0553
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1421	0.1442	0.0856	0.0964
	Difference	-0.0742	-0.0754	-0.0529	-0.0586
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.1347	0.1403	0.0825	0.0939
	Difference	-0.1051	-0.1120	-0.0590	-0.0733

By integrating the NLTK stoplist and the boosting technique inferred from the ACORDAR-2 baseline, we've achieved a synthesis that yielded results more congruous with the baseline than any other configuration we've explored.

Upon inspecting the overall results (Table 5.23) for all queries on the refined collection while using boosting (using weights reported in Table 3.3), we can derive the following observations:

- **TF-IDF**: The gap between the original and the reproduced results has narrowed considerably. Differences in metrics like NDCG@5 and NDCG@10 have been reduced to -0.0275 and -0.0385, respectively. This clearly demonstrates the effectiveness of the combined approach, especially given the greater disparities seen in prior setups.
- **BM25F**: BM25F, under the combined influence of the NLTK stoplist and boosting, has shown the closest proximity to the original. The difference in the NDCG@10 metric, for instance, is a mere -0.0209. This combination almost replicates the efficacy of the original configuration for the BM25F model.
- **LMD**: For the LMD model, while the discrepancies between the original and reproduced results have reduced, they remain relatively significant, especially with a -0.0624 difference in the NDCG@10 metric. Nonetheless, the combined approach still presents an enhancement over prior configurations.

Table 5.23: Performance comparison using combined metadata and data across all queries in the refined collection with NLTK stoplist and boosting

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.5088	0.5452	0.2871	0.3976
	Reproduced	0.4813	0.5067	0.2719	0.3686
	Difference	-0.0275	-0.0385	-0.0152	-0.0290
BM25F	Original	0.5538	0.5877	0.3198	0.4358
	Reproduced	0.5450	0.5668	0.3130	0.4205
	Difference	-0.0088	-0.0209	-0.0068	-0.0153
LMD	Original	0.5465	0.5805	0.3266	0.4324
	Reproduced	0.4906	0.5181	0.2905	0.3801
	Difference	-0.0559	-0.0624	-0.0361	-0.0523

Upon examination of the metadata results for all queries on the refined collection, with the application of a boosting strategy (using weights reported in Table 3.1) and NLTK stoplist (refer to Table 5.24), we can draw the following conclusions:

- **TF-IDF**: The original results slightly outpace the reproduced versions across all the evaluated metrics. For metrics such as NDCG@5 and NDCG@10, the differences are -0.0085 and -0.0168, respectively. These discrepancies, though small, indicate areas where the reproduction may have varied from the original.
- **BM25F**: The reproduced results for BM25F are nearly on par with the original, especially in the NDCG@10 metric where the difference is as slight as +0.0008. This suggests that the reproduced approach manages to capture most of the efficacy of the original BM25F implementation.
- **LMD**: The reproduced results for the LMD algorithm surpass the original across all metrics. With differences such as +0.0168 in NDCG@5, the reproduced version seems to offer improvements or variations that might have led to these enhanced outcomes.

Table 5.24: Performance comparison using metadata only across all queries in the refined collection with NLTK stoplist and boosting

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.4743	0.5019	0.2676	0.3685
	Reproduced	0.4658	0.4851	0.2632	0.3515
	Difference	-0.0085	-0.0168	-0.0044	-0.0170
BM25F	Original	0.5045	0.5250	0.2859	0.3838
	Reproduced	0.5094	0.5258	0.2898	0.3856
	Difference	+0.0049	+0.0008	+0.0039	+0.0018
LMD	Original	0.4363	0.4573	0.2543	0.3325
	Reproduced	0.4531	0.4698	0.2681	0.3452
	Difference	+0.0168	+0.0125	+0.0138	+0.0127

Upon inspecting the data results for all queries on the refined collection, with the application of a boosting strategy (using weights reported in Table 3.2) and NLTK stoplist (refer to Table 5.25), we can draw the following conclusions:

- **TF-IDF**: The reproduced results significantly lag behind the original ones. For metrics like NDCG@5 and NDCG@10, the discrepancies are considerable, with differences of -0.0935 and -0.0915, respectively. These discrepancies indicate that the replicated setup for data extraction deviates from ACORDAR's original configuration.
- **BM25F**: While the BM25F algorithm's reproduced results are also lower than the original, the gaps are slightly smaller than those observed with TF-IDF. Specifically, the NDCG@10 metric shows a difference of -0.0757. This indicates that while the reproduced version still underperforms, it is closer to the original configuration's performance compared to TF-IDF.
- **LMD**: The LMD algorithm shows the largest discrepancies between the original and reproduced results. With a difference of -0.1110 in the NDCG@10 metric, it's evident that the reproduced LMD implementation has significant room for improvement. This discrepancy underscores the challenge in reproducing the original setup's efficacy.

Table 5.25: Performance comparison using data only across all queries in the refined collection with NLTK stoplist and boosting

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Original	0.1910	0.1963	0.0998	0.1199
	Reproduced	0.0975	0.1048	0.0523	0.0633
	Difference	-0.0935	-0.0915	-0.0475	-0.0566
BM25F	Original	0.2163	0.2196	0.1385	0.1550
	Reproduced	0.1408	0.1439	0.0842	0.0954
	Difference	-0.0755	-0.0757	-0.0543	-0.0596
LMD	Original	0.2398	0.2523	0.1415	0.1672
	Reproduced	0.1354	0.1413	0.0821	0.0937
	Difference	-0.1044	-0.1110	-0.0594	-0.0735

Table 5.26: Divergence of configurations from the baseline

Configuration	**%**
Raw Collection	7.00
Refined Collection without files over 200 MB	7.09
Refined Collection	7.65
Refined Collection with NLTK stoplist	6.90
Refined Collection with Kaggle stoplist	6.94
Refined Collection with boosting	3.99
Refined Collection with NTLK stoplist and boosting	3.47
Refined Collection with Kaggle stoplist and boosting	3.64

Through this analysis, it is observed that the replication of metadata results demonstrates a high degree of alignment with the original study. Nonetheless, greater variability manifests in the outcome concerning the extracted data. This inconsistency can be due to differences in processing the RDF files. Regrettably, the reference paper only briefly touches upon this topic.

Table 5.26 provides a comparative look at the divergence percentages from the baseline across different configurations.

Interestingly, when files are recovered and additional information is added to the data fields of Lucene's documents, there is a noticeable decline in performance compared to the baseline. This behavior is evident in the run "Refined Collection without files over 200 MB", which exhibits a slightly higher divergence compared to the run "Raw Collection". The data demonstrates that configurations employing boosting techniques lead to a significant reduction in disparity when compared to the results of the ACORDAR baseline. The "Refined Collection with NTLK stoplist and boosting" achieves the least divergence, at 3.47%. Based on the evidence, we can draw the conclusion that the use of boosting techniques has notably diminished this disparity, effectively narrowing the gap between the experimental findings and the results presented in the reference paper.

5.2 Impact of Missing Datasets

In investigating the role of absent datasets on the performance of the replicated dataset search system, an analysis was conducted to determine how the absence of particular datasets affects the accuracy of search outcomes. This investigation aimed to identify the factors contributing to deviations from the system's reference performances.

One of the crucial aspects of this analysis was to assess whether there exists a direct correlation between the number of missing datasets and the subsequent lack of precision for a given query.

In contrast to the initial assumptions that a greater number of absent datasets would necessarily lead to a decline in precision, the empirical evidence did not definitively support this hypothesis.

These findings highlight the intricate nature of how missing datasets influence search precision within the ACORDAR. The analysis underscores that factors beyond mere dataset availability play a significant role in shaping search precision, thus illustrating the complexity of dataset retrieval.

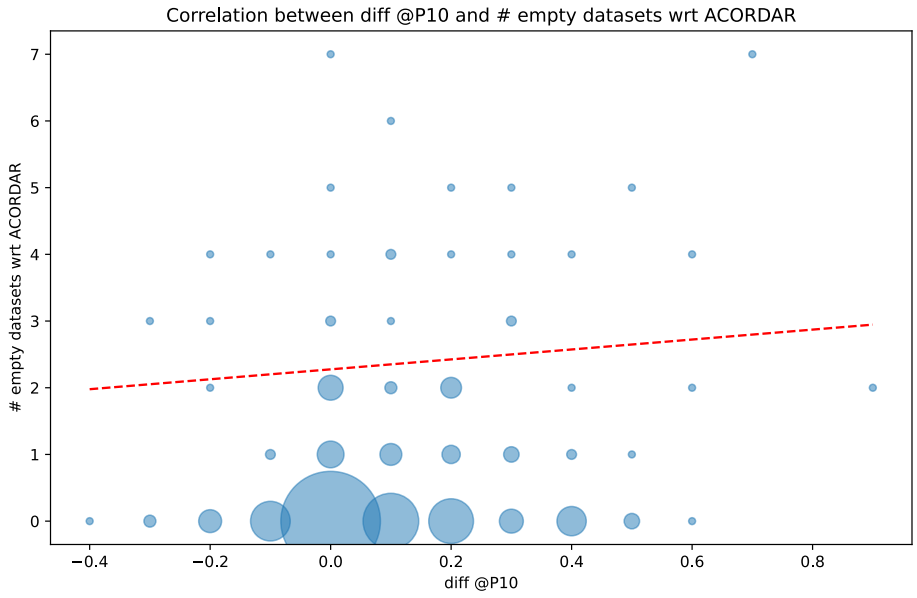

Figure 5.1: Missing Datasets Impact

Analyzing the visual representation of the data in Figure 1, it becomes evident that the relation between missing datasets and the system's performance

is not linear as one might initially presume. If missing datasets had a considerably significant impact on performance, one would anticipate the graphical representation to showcase a steeper incline when interpolating the data points. This observation provides additional support for the idea that the quantity of missing datasets does not exhibit a linear correlation with the degradation in the system's precision.

To support the argument that missing datasets have a negligible impact on overall performance, we conducted an evaluation in which empty datasets (i.e., those for which no file was downloaded) were excluded from the relevance judgements.

Table 5.27: Divergence of configurations from the baseline without considering empty files

Configuration	**%**
Refined Collection	7.75
Refined Collection with NLTK stoplist	6.93
Refined Collection with Kaggle stoplist	6.99
Refined Collection with boosting	4.12
Refined Collection with NTLK stoplist and boosting	3.51
Refined Collection with Kaggle stoplist and boosting	3.70

Moreover, when examining the results presented in Table 5.27, it becomes evident that even when empty datasets are excluded from the relevance judgements, the divergence in performance from the complete ground truth remains minimal. The results indicate that the performance delta, when compared to the complete ground truth, is less than 0.1%. This minor variance reaffirms that the presence or absence of specific datasets does not influence the precision of the ACORDAR's search mechanism.

By combining the information obtained from both the visual representation and the data presented in tables, we can infer that there is a complex connection between the availability of data and the accuracy of search results.

5.3 Re-Ranking based on Numerical Features

Within the context of enhancing the IR system in ACORDAR, it is essential to continually evaluate and improve search results. A crucial aspect of this process involved re-evaluating the relevance of retrieved datasets. Previous steps revealed that some datasets, although ranked by the system, were either inaccessible or impractical for the intended application. Consequently, there was needed to revisit the reference results that lead to the establishment of a fresh baseline.

Table 5.28 provides a performance comparison when using combined metadata and data across all queries in the refined collection. It is evident from the table that the incorporation of re-ranking based on numerical features offers a consistent improvement in all the ranking metrics.

Table 5.28: Performance comparison using combined metadata and data across all queries in the refined collection with re ranking based on numerical features

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Reference	0.4752	0.4986	0.2865	0.3759
	Re-Ranking	0.5575	0.5416	0.3541	0.4280
	Difference	+0.0823	+0.0430	+0.0676	+0.0521
BM25F	Reference	0.5282	0.5558	0.3212	0.4266
	Re-Ranking	0.6133	0.6020	0.3897	0.4788
	Difference	+0.0851	+0.0462	+0.0685	+0.0522
LMD	Reference	0.4785	0.5084	0.2989	0.3846
	Re-Ranking	0.5495	0.5445	0.3428	0.4194
	Difference	+0.0710	+0.0361	+0.0439	+0.0348

In Table 5.29, the focus shifts to performance comparisons when only metadata is utilized. There is a noticeable enhancement in metrics when re-ranking is implemented, underscoring the effectiveness of incorporating graph structural features into the ranking procedure.

Table 5.29: Performance comparison using metadata only across all queries in the refined collection with re-ranking based on numerical features

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Reference	0.4607	0.4768	0.2793	0.3609
	Re-Ranking	0.5464	0.5204	0.3481	0.4122
	Difference	+0.0857	+0.0436	+0.0688	+0.0513
BM25F	Reference	0.4946	0.5103	0.2992	0.3889
	Re-Ranking	0.5775	0.5539	0.3665	0.4399
	Difference	+0.0829	+0.0436	+0.0673	+0.0510
LMD	Reference	0.4259	0.4446	0.2653	0.3366
	Re-Ranking	0.5051	0.4851	0.3172	0.3768
	Difference	+0.0792	+0.0405	+0.0519	+0.0402

Lastly, Table 5.30 presents the results using only the data for the comparisons. It's interesting to note that even when only data is utilized, re-ranking leads to performance improvements, although with a more modest effect compared to combined data and metadata scenarios.

Table 5.30: Performance comparison using data only across all queries in the refined collection with re-ranking based on numerical features

		NDCG@5	NDCG@10	MAP@5	MAP@10
TF-IDF	Reference	0.1018	0.1108	0.0573	0.0696
	Re-Ranking	0.1418	0.1325	0.0826	0.0898
	Difference	+0.0400	+0.0217	+0.0253	+0.0202
BM25F	Reference	0.1453	0.1508	0.0902	0.1026
	Re-Ranking	0.1712	0.1643	0.1052	0.1145
	Difference	+0.0259	+0.0135	+0.0150	+0.0119
LMD	Reference	0.1397	0.1481	0.0877	0.1006
	Re-Ranking	0.1660	0.1598	0.1019	0.1110
	Difference	+0.0263	+0.0117	+0.0142	+0.0104

Table 5.31: Re-Ranking Improvements Evaluation

Configuration	**Improvement %**
Refined Collection	3.88
Refined Collection with NLTK stoplist	3.85
Refined Collection with Kaggle stoplist	3.84
Refined Collection with boosting	4.30
Refined Collection with NTLK stoplist and boosting	4.51
Refined Collection with Kaggle stoplist and boosting	4.51

In a comprehensive analysis, enhancements have been observed across all metrics, manifesting an approximate increase of 4% when compared to the newly established reference configuration results. Particularly noteworthy are the improvements in the NDCG@5 values for BM25 and LMD retrieval models. The improving trend holds true even for other configurations, as reported in Table 5.31.

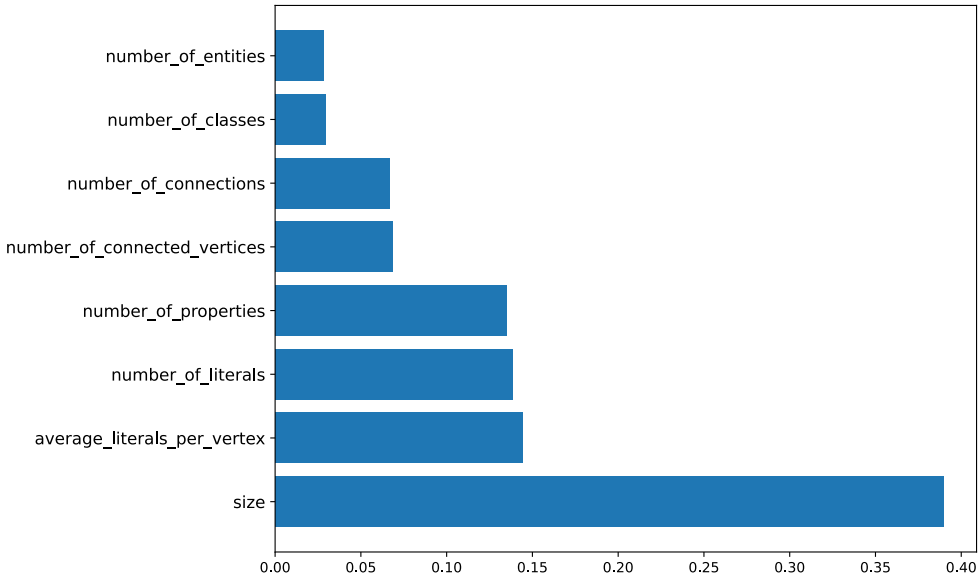

Figure 5.2: Feature Importance

An interesting pattern can be observed in Figure 5.2. It appears that the size is strongly related to the relevance of a dataset. The observed relationship between dataset size and relevance may be connected to the manner in which relevance judgments were produced. Including datasets with substantial collections may introduce a potential bias in the outcomes, as these extensive datasets might encompass be relevant (or partially relevant) to a broad spectrum of queries.

5.4 Leveraging Important Nodes

As mentioned in Chapter 2, the task of extracting salient information from RDF datasets offers significant prospects for enhancing the capabilities of dataset search systems. To this end, the NetworkX library was employed to identify and retrieve the URI of the most influential nodes within each RDF file for subsequent analysis.

To evaluate the advantages of focusing on these central nodes, two experimental configurations were formulated to measure potential improvements.

Facing challenges similar to prior scenarios, the issue of data usability was addressed by refining the test collection to exclude datasets lacking usable files. Due to computational limitations, compromises were necessary: files that required more than 100 seconds for ingestion into RDFLib or exceeded 200 seconds for betweenness centrality computation using the approximate algorithm were excluded. This time constraint ensured that the computational workload remained manageable. Consequently, there was a need to establish a new baseline for future performance comparisons. Building on the foundational work and adjustments made to the data usability, two primary experiments were conducted to delve deeper into the performance optimization of the system.

Enrichment of Metadata and Data with Node URIs The first experiment aimed at evaluating the performance of the IR system while leveraging both the comprehensive data from the RDF datasets and the top 20 URIs extracted from each usable file. The objective was to investigate whether integrating key nodes could yield an improvement in system performance.

Selective Use of Top Node URIs and Metadata The second experiment explored the efficacy of replacing the all the extracted data, comprising Entities, Classes, Literals, and Properties, with only the URIs of the top 20 nodes extracted from each usable file. The goal was to determine if the combination of metadata and chosen URIs could match or surpass the set performance benchmark.

In the following tables, the results presented are obtained from the system's operation without the application of any stoplist and without the implementation of boosting. The evaluation is conducted using *ALL QUERIES*. As previously delineated, the dataset collection was refined, excluding datasets that contained no usable files.

Note: In the tables below:

- **[m]** stands for metadata only.
- **[d]** represents data only.
- **[N]** signifies the usage of node URIs.
- **[m+d]** denotes a combination of metadata and data.
- **[m+N]** denotes a combination of metadata and node URIs.
- **[m+d+N]** denotes a combination of metadata, data and node URIs.

In Table 5.32, the initial group of rows refers to the baseline results, while the succeeding group reports the outcomes achieved upon the integration of nodes as a new document field.

Table 5.32: Enrichment of Metadata and Data with Node URIs

Method	**NDCG@5**	**NDCG@10**	**MAP@5**	**MAP@10**
TF-IDF [m+d]	0.4474	0.4648	0.2865	0.3608
BM25 [m+d]	0.4387	0.4749	0.2805	0.3633
LMD [m+d]	0.2005	0.2272	0.1260	0.1530
TF-IDF [m+d+N]	0.4475	0.4708	0.2868	0.3668
BM25 [m+d+N]	0.4095	0.4446	0.2563	0.3300
LMD [m+d+N]	0.2119	0.2363	0.1284	0.1572

From the performance measures detailed in Table 5.32, several significant observations can be highlighted:

1. **Impact of Node URIs:** The integration of node URIs shows varied effects on the retrieval methods. While it marginally enhances the performance for the TF-IDF and LMD methods, the scores for BM25F experience a slight decline across all metrics.
2. **Magnitude of Change:** The degree of alteration in scores with the addition of node URIs differs between methods. This variance suggests that the influence of node URIs on performance is likely method-dependent.

In Table 5.33, three rows are presented for each similarity measure employed:

1. The initial row corresponds to the system's performance using metadata exclusively.
2. The subsequent row illustrates the results associated with ACORDAR default system configuration.
3. The final row reports the system's results when augmenting the four metadata fields with node fields.

Table 5.33: Selective Use of Top Node URIs and Metadata

Method	**NDCG@5**	**NDCG@10**	**MAP@5**	**MAP@10**
TF-IDF [m]	0.4247	0.4363	0.2681	0.3371
TF-IDF [m+d]	0.4474	0.4648	0.2865	0.3608
TF-IDF [m+N]	0.4296	0.4447	0.2722	0.3451
BM25 [m]	0.4616	0.4808	0.2961	0.3761
BM25 [m+d]	0.4387	0.4749	0.2805	0.3633
BM25 [m+N]	0.4390	0.4685	0.2730	0.3576
LMD [m]	0.4102	0.4269	0.2702	0.3322
LMD [m+d]	0.2005	0.2272	0.1260	0.1530
LMD [m+N]	0.3913	0.4107	0.2517	0.3113

From Table 5.33 we can observe:

- *TF-IDF:* Its best performance is with the combined metadata and data (**[m+d]**), with a marginal improvement when node URIs augment the metadata.
- *BM25:* It excels most when relying solely on metadata (**[m]**).
- *LMD:* The method's best performance is with metadata only, but adding node URIs (**[m+N]**) produces results that are similar, unlike the significant dip seen in the **[m+d]** setup.

Within Table 5.34, a noteworthy phenomenon becomes apparent: there is an average performance variation of roughly 2% when comparing the results obtained from extracted data to those derived from a substantially limited set of URIs. This disparity might be attributed to the inherent quality or relevance of data present in nodes with high betweenness centrality. Nodes of this nature play a crucial role in facilitating the flow of information across the graph, potentially resulting in more precise outcomes. Conversely, the entire RDF graph might include noise or less relevant data, compromising the integrity of the search results. While prioritizing nodes with high betweenness centrality can yield results that are central and pertinent to numerous paths in the graph, it's worth noting that it bypass some peripheral yet potentially significant information.

Table 5.34: Usage of Nodes versus Extracted Data

Method	**NDCG@5**	**NDCG@10**	**MAP@5**	**MAP@10**
TF-IDF [d]	0.0780	0.0878	0.0483	0.0572
TF-IDF [N]	0.0849	0.0826	0.0417	0.0459
BM25 [d]	0.1056	0.1158	0.0653	0.0759
BM25 [N]	0.0825	0.0840	0.0401	0.0460
LMD [d]	0.0942	0.1091	0.0601	0.0710
LMD [N]	0.0857	0.0868	0.0426	0.0487

The efficiency of the system is significantly influenced by metadata. While integrating all the extracted data offers marginal improvements, including only the URIs of the most central nodes yields nearly identical enhancements. This implies that concentrating on these core nodes might be an approach to achieve optimal results. Nonetheless, the minimal difference in performance between using all the extracted data and only the URIs of the central nodes highlights the critical role of metadata in the system's efficiency.

6 Conclusions and Future Work

6.1 General Considerations On ACORDAR

During the course of this research, multiple challenges and opportunities were observed in the ACORDAR framework. The dataset download phase was marked by several limitations, including download speed caps from some servers and IP address bans due to high request frequency. A significant overhead was also noted when downloading multiple small files, as it necessitated opening numerous HTTP connections. To improve the reproducibility of ACORDAR's results, a consolidated approach involving a single compressed file containing the dataset could be a significant step forward. Furthermore, the time-dependent nature of some RDF files, which are likely serializations of relational databases, raises concerns about the stability of provided relevance judgements.

6.2 Reproduced Results

Duplicated Datasets The quality of the collection raises some concerns, given that numerous datasets (over 200) appear to be duplicates or subsets of others, all accessing the same URLs. This lack of proper deduplication could be attributed to the crawling of open data portals. This issue reflects also on other problems, such as inconsistent relevance judgements for queries across duplicate datasets. A refined curation process involving syntax validation and file deduplication is advised to enhance the quality of the collection. Offering a more detailed JSON file, which includes metadata about the collection and the URLs for downloading the assets, and possibly indicating various representations of the same graph, would be beneficial for researchers who utilize the collection.

Deduplication The ACORDAR paper refers to tuple deduplication without discolosing the specific methodologies employed. Based on the code analysis of ACORDAR-2, a relational database table was likely used for this task. However, this approach adds significant computational overhead to the collection processing. Implementing probabilistic approaches like a Bloom Filter could potentially offer adequate tuple deduplication while alleviating computational costs.

Influence of Boosting and Stoplist The introduction of boosting brings ACORDAR's performance closer to the baseline, although this was not discussed in the original paper. Moreover, an interesting observation arises from this analyzing the boosting weights: an inclination towards prioritizing matches in metadata as opposed to data extracted from fields, substantially diminishes the contribution of the latter. Furthermore, the implementation of the NLTK stoplist successfully reduced deltas with respect to baseline results. These enhancements, however, were overlooked in the published paper, and have been inferred by looking the ACORDAR-2 repository.

Partial Usage of Files In the ACORDAR paper, there is an ambiguity surrounding the methodology employed for handling files with syntax errors. The paper does not provide explicit details on how these invalid files were managed during their research process.

By scrutinizing the ACORDAR-2 repository, a pattern of handling these files can be inferred. It appears that these files are used in a streaming fashion. When the parser, specifically the Jena framework, encounters an error within the file (be it due to a syntax irregularity or any other anomaly) it immediately terminates the parsing process. Thus, only the data preceding the initial error gets retained and utilized in the study. This implies that any file containing an error at its inception would be largely overlooked, and a file with an error towards its end would almost be entirely accepted, with possibly only the latter segment omitted. Such a methodology might lead to data being either underrepresented, depending on the positioning of the error within the file.

Furthermore, another observation from the ACORDAR-2 repository suggests that the streaming process for any file has a predetermined limit. The parser only processes the initial 1,000,000 triples found within a file [1], irrespective of its total length or content beyond this point. This thresholding raises concerns about the representativeness and comprehensiveness of the dataset. Arbitrarily truncating files after a specific number of triples might not provide an accurate representation of the data within. This, indeed, could mean that certain insights or patterns present in the omitted segments are disregarded, thereby potentially deviating the results. It is necessary to highlight that even this practice was omitted from the paper.

6.3 Conclusions and Future Work

In this section, we present outcomes of our research and outline possible directions for future investigations. The discussion encloses factors like the computational challenges posed by additional data, the effectiveness of graph summarization techniques in enhancing metadata, and innovative approaches to extracting features from RDF graphs. We also delve into the potential use of large language models and semantic embeddings.

[1] https://github.com/nju-websoft/ACORDAR-2/blob/main/Code/sparse/until/GlobalVariances.java#L35

6.3.1 Key Discoveries

Computational Complexity and Performance Impact

Our research has revealed that integrating extra data into the IR system results in significant computational overhead. This is evident in the expansion of index dimensions, leading to a decrease in search speed, and a huge extension of computational time during data extraction stages. Interestingly, we observed only marginal improvements in system efficiency when compared to an approach exclusively focused on metadata.

Graph Summarization and Metadata Enhancement

Graph summarization methods [8, 19], offer promising avenues for enriching metadata. These methods not only enhance the quality of generated summaries but also incorporate effective algorithms, providing valuable tools for exploring and understanding extensive knowledge graphs.

6.3.2 Semantic Embeddings

Semantic embeddings constitute a specialized field that aims to capture the inherent semantic relationships between entities. These vector representations leverage machine learning algorithms to analyze both the graph's structure and attribute characteristics. Such embeddings provide a robust computational mechanism capable of revealing semantic connections between dataset entities, thereby enhancing the precision and relevance of search results. Two notable methodologies within the realm of embeddings are pyRDF2Vec and Literal2Feature, which will be discussed in detail below.

pyRDF2Vec

pyRDF2Vec [18] is a Python library that utilizes Word2Vec models to create embeddings for RDF graphs. This library offers a variety of graph traversal strategies, making it a versatile tool for different applications.

Literal2Feature

Literal2Feature [13] represents another innovative approach, focusing on automated feature extraction from RDF graphs. Using machine learning algo-

rithms, it identifies key literals and then uses them to generate a descriptive feature set.

6.3.3 Future Directions

Augmenting Node Descriptions

Enhancing the richness of node descriptions can be achieved through various methods. One promising approach is data augmentation, which can be executed by extending nodes using resources such as API requests, web scraping techniques, and leveraging open data collections. However, while these augmentations can offer more comprehensive node descriptions, it is crucial to be aware of the computational and time costs that such methods may entail.

Prospective Research Avenues and Harnessing Large Language Models

A captivating avenue is the integration of graph summarization techniques, as illustrated by systems like SumMER [19], with Large Language Model (LLM). Such a combination has the potential to craft a hypothetical description of a dataset's underlying content, using the summary extracted via graph summarization tools.

Optimization Strategies

For handling large RDF datasets, future development could consider the utilization of the Rust programming language for its performance benefits. Specifically, the Rio library [2] allows for resilient stream processing of RDF files, ensuring that processing can continue even in the presence of syntactic errors. The usage of such instrument can optimize the time required for data manipulation.

[2] `https://github.com/oxigraph/rio`

References

[1] U. Brandes. "A faster algorithm for betweenness centrality". In: *Journal of Mathematical Sociology* 25.2 (2001), pp. 163–177. DOI: 10.1080/0022250X.2001.9990249. URL: https://kops.uni-konstanz.de/bitstream/123456789/5739/1/algorithm.pdf.

[2] M. Casanova. "Keyword Search over RDF Datasets - (Extended Abstract)". In: *Conference Proceedings*. 2019. DOI: 10.1007/978-3-030-33223-5_2. URL: https://dx.doi.org/10.1007/978-3-030-33223-5_2.

[3] M. H. Chehreghani. "An Efficient Algorithm for Approximate Betweenness Centrality Computation". In: *The Computer Journal* 57.11 (2013), pp. 1691–1705. DOI: 10.1093/comjnl/bxu003. URL: https://lirias.kuleuven.be/bitstream/123456789/439323/1/cikm13.pdf.

[4] Tạ Duy Công Chiến. "AN APPROACH TO EXTENDING QUERY SENTENCE FOR SEMANTIC ORIENTED SEARCH ON KNOWLEDGE GRAPH". In: *Journal of Science and Technology* 50 (08 2021). DOI: 10.46242/JST-IUH.V50I08.980. URL: https://dx.doi.org/10.46242/JST-IUH.V50I08.980.

[5] M. B. Ellefi et al. "RDF dataset profiling - a survey of features, methods, vocabularies and applications". In: *Semantic Web Journal* (2018). DOI: 10.3233/SW-180294. URL: https://dx.doi.org/10.3233/SW-180294.

[6] Sébastien Ferré. "Analytical Queries on Vanilla RDF Graphs with a Guided Query Builder Approach". In: *Flexible Query Answering Systems*. Ed. by Troels Andreasen et al. Cham: Springer International Publishing, 2021, pp. 41–53. ISBN: 978-3-030-86967-0. DOI: 10.1007/978-3-030-86967-0_4. URL: https://link.springer.com/chapter/10.1007/978-3-030-86967-0_4.

[7] Matias Frosterus, E. Hyvönen, and Joonas Laitio. "DataFinland - A Semantic Portal for Open and Linked Datasets". In: *Conference Proceedings*. 2011. DOI: 10.1007/978-3-642-21064-8_17. URL: https://dx.doi.org/10.1007/978-3-642-21064-8_17.

[8] François Goasdoué, Pawel Guzewicz, and I. Manolescu. "RDF graph summarization for first-sight structure discovery". In: *International Journal on Very Large Data Bases* (2020). DOI: 10.1007/s00778-020-00611-y. URL: https://hal.inria.fr/hal-02530206v2/file/revised_HAL_inlined.pdf.

[9] G. Kuck. "Tim Berners-Lee's Semantic Web". In: *South African Journal of Information Management* 6 (Dec. 2004). DOI: 10.4102/sajim.v6i1.297.

[10] Jens Lehmann et al. "Distributed Semantic Analytics using the SANSA Stack". In: Oct. 2017. ISBN: 978-3-319-68203-7. DOI: 10.1007/978-3-319-68204-4_15.

[11] Tengteng Lin et al. "ACORDAR: A Test Collection for Ad Hoc Content-Based (RDF) Dataset Retrieval". In: July 2022, pp. 2981–2991. DOI: 10.1145/3477495.3531729.

[12] Hakim El Massari, Sajida Mhammedi, and Noreddine Gherabi. "Bridging the gap between the semantic web and big data: answering SPARQL queries over NoSQL databases". In: *International Journal of Electrical and Computer Engineering* (2022). DOI: 10.11591/ijece.v12i6.pp6829-6835. URL: https://dx.doi.org/10.11591/ijece.v12i6.pp6829-6835.

[13] Farshad Bakhshandegan Moghaddam et al. "Literal2Feature: An Automatic Scalable RDF Graph Feature Extractor". In: *Semantic Web* (2021). DOI: 10.3233/ssw210036. URL: https://dx.doi.org/10.3233/ssw210036.

[14] S. Natarajan et al. "Schema-Based Mapping Approach for Data Transformation to Enrich Semantic Web". In: *Wireless Communications and Mobile Computing* (2021). DOI: 10.1155/2021/8567894. URL: https://dx.doi.org/10.1155/2021/8567894.

[15] Ö. Şahin, I. Çiçekli, and Gonenc Ercan. "Learning term weights by overfitting pairwise ranking loss". In: (2022). DOI: 10.55730/1300-0632.3913. URL: https://dx.doi.org/10.55730/1300-0632.3913.

[16] Gezim Sejdiu et al. "DistLODStats: Distributed Computation of RDF Dataset Statistics". In: *Conference Proceedings*. 2018. DOI: 10.1007/978-3-030-00668-6_13. URL: https://dx.doi.org/10.1007/978-3-030-00668-6_13.

[17] Gezim Sejdiu et al. "Towards a Scalable Semantic-Based Distributed Approach for SPARQL Query Evaluation". In: *Semantic Systems. The Power of AI and Knowledge Graphs*. Ed. by Maribel Acosta et al. Cham: Springer International Publishing, 2019, pp. 295–309. ISBN: 978-3-030-33220-4. DOI: 10.1007/978-3-030-33220-4_21. URL: https://link.springer.com/chapter/10.1007/978-3-030-33220-4_21.

[18] Bram Steenwinckel et al. "pyRDF2Vec: A Python Implementation and Extension of RDF2Vec". In: *European Semantic Web Conference*. Springer Nature Switzerland, 2023, pp. 471–483. DOI: 10.1007/978-3-031-33455-9_28. URL: https://arxiv.org/abs/2205.02283.

[19] Georgia Eirini Trouli et al. "SumMER: Structural Summarization for RDF/S KGs". In: *Algorithms* 16.1 (2023). ISSN: 1999-4893. DOI: 10.3390/a16010018. URL: https://www.mdpi.com/1999-4893/16/1/18.

[20] J. Vitter. "Random sampling with a reservoir". In: *Theory of Computing Systems* 31.5 (1985), pp. 497–514. URL: https://dl.acm.org/doi/pdf/10.1145/3147.3165.

[21] Nikita Zhiltsov, Alexander Kotov, and Fedor Nikolaev. "Fielded Sequential Dependence Model for Ad-Hoc Entity Retrieval in the Web of Data". In: Aug. 2015. DOI: 10.1145/2766462.2767756. URL: https://www.researchgate.net/publication/280042927_Fielded_Sequential_Dependence_Model_for_Ad-Hoc_Entity_Retrieval_in_the_Web_of_Data.

[22] M. Zrhal et al. "Spatial Dataset Search: Building a dedicated Knowledge Graph". In: *AGILE: GIScience Series* 2 (2021), p. 43. DOI: 10.5194/agile-giss-2-43-2021. URL: https://agile-giss.copernicus.org/articles/2/43/2021/.

Acknowledgments

I express my profound gratitude to my family for their support throughout my academic journey, especially to my mother for her endless encouragement. I extend my heartfelt gratitude to my brother, Pietro, and Sara for standing by me during our most challenging moments. Equally, my deepest thanks go to my closest friends who have supported me during such times.

Special thanks to Professor Gianmaria Silvello and Laura Menotti for their invaluable guidance, mentorship, and consistent support throughout this process.

To everyone who believed and stood by me, thank you.