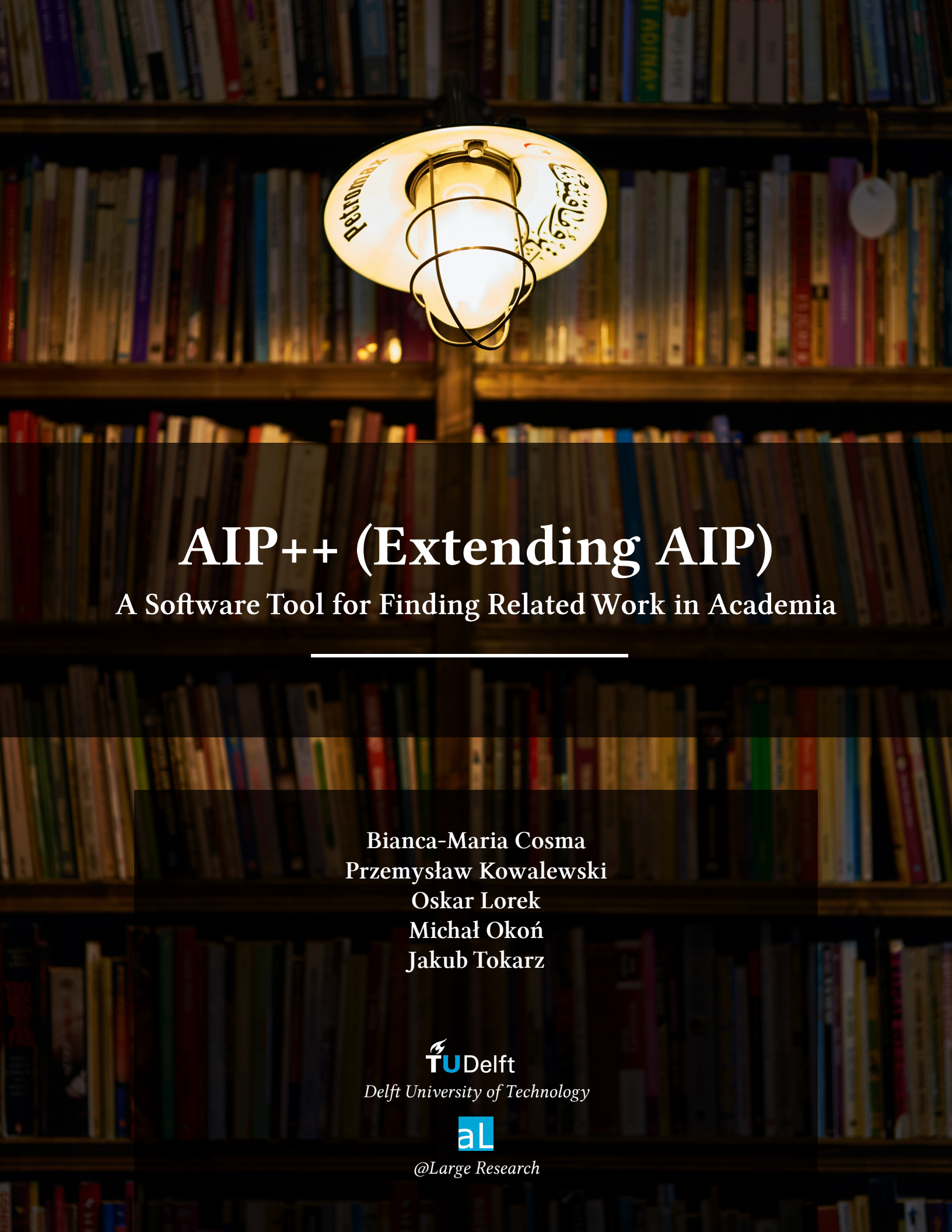

AIP++ (Extending AIP)

A Software Tool for Finding Related Work in Academia

Bianca-Maria Cosma
Przemysław Kowalewski
Oskar Lorek
Michał Okoń
Jakub Tokarz

TU Delft
Delft University of Technology

aL
@Large Research

Faculty of Electrical Engineering, Mathematics and Computer Science,
Delft University of Technology

CSE2000 Software Project

Final Project Report

AIP++ (Extending AIP)

A Software Tool for Finding Related Work in Academia

Date: June 17, 2021

Technical Writing Group 5C

Bianca-Maria Cosma
Przemysław Kowalewski
Oskar Lorek
Michał Okoń
Jakub Tokarz

Project Coach | assoc. prof. dr. ir. Bart H. M. Gerritsen

Responsible Teaching Assistant | Dan Andreescu

Clients | prof. dr. ir. Alexandru Iosup and ir. Laurens Versluis
@ Large Research - https://atlarge-research.com/

Preface

This report was written by a group of five students at Delft University of Technology, during the second year of our bachelor in Computer Science and Engineering. As part of our Software Project in quarter 4, we developed AIP++, an open source academic search engine for articles in the field of computer systems. AIP++ was implemented on top of AIP, a project built by members of the @Large Research team and led by ir. Laurens Versluis.

Throughout this text, we assume that the reader is familiar with the Python programming language and has basic knowledge of web development, database technologies, and algorithm design. A glossary is included at the end of the report, before the bibliography, to clarify the most important and frequently used technical terms, alongside a list of acronyms.

Readers who want to know more about the architecture and design of our application can find relevant information in chapter 4. This part of the report includes our insight about the implementation of an academic search engine that can be run both locally and remotely. In chapter 6, we give the reader a detailed description of the algorithms that form the foundations of important functionality in AIP++, such as detection of rising stars in academia.

We would like to sincerely thank our clients from the @Large Research team, prof. dr. ir. Alexandru Iosup and ir. Laurens Versluis, for their patience and insightful advice throughout this project. We are also grateful for the rigorous guidance of our coach, assoc. prof. dr. ir. Bart H. M. Gerritsen, and our teaching assistant, Dan Andreescu. Lastly, we mention that writing this report was an invaluable learning experience, due to the constructive feedback of our Technical Writing lecturer, drs. Mariëtte Bliekendaal.

Delft, 17 June 2021
Bianca-Maria Cosma, Przemysław Kowalewski, Oskar Lorek, Michał Okoń, Jakub Tokarz

Summary

Scientific research strongly depends on the quality and accessibility of resources published on the internet. Today, there are hundreds of millions of scientific publications arranged into a multitude of data sets accessed through search engines. The search engines can be categorized by a number of factors, such as fields of science they are focused on or the range of search options. Moreover, many of them do not provide their services for free and may require payment for users to gain access to their resources. The rapid development in the fields of data science and data mining has had a lasting effect on almost every existing technology, the publication browsers being no exception. With the help of data manipulation techniques, the range of functionalities offered by search engines can be greatly expanded to include new, research-oriented features.

The aim of this report is to look into an implementation of a search engine operating on articles from the field of computer systems. This search engine, aside from offering the most common browsing functionalities, includes the following innovative features:

- **Rising Stars.** The ability to detect promising, young scientists is crucial to all the senior researchers hoping to supervise developing talents and strengthen the world of science. The *Rising Stars* problem can be split into two categories, namely the global ones, with scientists originating from all the entire of the computer systems field, and local ones, falling into specific sub-fields. In the case of the global rising stars, the performance of three algorithms is compared. The first algorithm employs z-scores, the second one makes use of a PageRank algorithm, while the last one utilizes author-based clustering techniques. Each algorithm resulted in a slightly different outcome and favored different groups of scientists.

- **Hot Keywords.** In the network of words building scientific publications, we can identify the ones that best characterize groups of articles in regard to the scientific fields they belong to (e.g. machine learning, internet of things, cloud). The goal of the *Hot Keywords* feature is to take a step further and detect the ones among these words, that have become the most popular in recent years and can be utilized to predict currently shaping trends in science. To that end, keywords derived from clustered publications are filtered out using three different criteria: number of citations, number of publications, and a growth ratio. Similar to the case of *Rising Stars*, those criteria can be modified depending on the definition of emergence and attractiveness of a keyword.

- **Author Exploration** There is no doubt that the networks of scientific citations cannot be described as regular with scientists preferring to create publications in their known groups of colleagues. One of our aims is to visualize connections of researchers in Bibliometric Networks in a form of a user-friendly graph.

The aforementioned features are a result of long-term discussions with experienced researchers from the @Large Research group, who have shown an avid interest in seeing their possible implementation. Moreover, most of these functionalities are not available in any of the existing free search engines, so their application can bring innovation to the market of search engines.

To conclude, we have managed to build an application containing the mentioned features and substantial efforts have been put into extensive research and the validation of the obtained results. Moreover, we have laid the foundation for future works aiming at expanding the functionalities of search engines with the use of data mining techniques.

Contents

1 Introduction

The ability to find related work is a pillar of today's research communities. Academic search engines, formally defined as "search products that localize scientific information on the web", are tools that fulfill this purpose [1, p. 3]. Many products of this type are available nowadays. However, a lot of them are either paywalled or offer a narrow range of advanced search options. It is important to lay the foundations of an academic search engine that is open-source, free to use, and nonrestrictive of its users' querying permissions, while also providing novel features that could potentially be useful for researchers. To this end, our team developed AIP++, an extension of Article Information Parser (AIP), a project created by the @Large Research team and currently available on GitHub [2].

With the help of the @Large Research team, we have identified features that modern academic search engines lack and which could prove invaluable to researchers. Firstly, one of the main points raised by our clients was helping experienced scientists find their younger and less accomplished, yet promising colleagues – the rising stars of academia. Young researchers eagerly join the ever-growing scientific society, which releases thousands of new publications each day [3]. This results in an enormous surge of new papers, many of which go completely unnoticed despite their substantive values. It is hard to imagine how much potential input is regularly overlooked by scientists searching for sources, only because the authors of valuable study materials are not yet well-known in the world of science. Helping researchers with a small academic age be noticed by their more experienced colleagues would, undoubtedly, prove to be a great deed to the whole scientific community [4].

Secondly, in a technology-driven world, scientific trends tend to change rapidly. Staying updated with regard to what topics are currently gaining in popularity is crucial to properly navigate through the intricate world of science. In other words, finding new research topics and conducting research is easier, when you know the fields where the biggest advancements take place, as well as the directions in which these fields are heading. Therefore, our client also requested the implementation of a so-called "emerging keywords" feature, which aims to highlight all trending topics in a field of research.

Lastly, it is important to acknowledge that the social skills of a researcher represent a very meaningful aspect of his/her career. Some authors may rarely cite or collaborate with people outside of their closed circle of colleagues, whereas others will actively seek connections and opportunities to evade their comfort zones. A visualization tool for the authors in a bibliometric network can help distinguish between these two types of researchers, and provide encouragement for all authors to expand their social circles.

This report answers the question of how an academic search engine with novel features, such as rising stars and emerging keywords detection, custom querying, or author network visualizations, can be implemented. One of the first steps in our method is a brief literature review regarding state-of-the-art technologies and algorithms that are relevant for the project, after which we conduct an experimental analysis, in order to validate the functionality of the application. We focus our attention on finding and comparing algorithms addressing the identification of rising stars, both global and residing in specific fields of research. The datasets used to supply AIP++ are DBLP, AMiner, Semantic Scholar, and MAG (Microsoft Academic Graph). As requested by the client, only publications from certain venues are added to the database of the application, which limits it to the field of computer systems.

The report begins with a thorough analysis of the problem in chapter 2. This includes the general outline of the project and a discussion on the involved stakeholders. Subsequently, in chapter 3, the functional and non-functional requirements of AIP++ are documented using the MoSCoW prioritisation strategy. The full structure of the project, consisting of the front-end, back-end, and the parser, is characterized in chapter 4, which also contains a section regarding the architecture of the system. The agile development process followed throughout the AIP++ project is outlined in chapter 5, where the reader can additionally find one section regarding testing practices employed by the team. Algorithms used in the identification of rising stars, emerging keywords, and author relationships in the bibliometric network can be found in chapter 6. Chapter 7 contains two different benchmarks that were performed during the project, and chapter 8 gives a detailed account of value-sensitive design in AIP++, by delving deeper into the problem of fairness and bias in software. Limitations and possible further improvements of our project are discussed in chapter 9. Finally, the main takeaways are delineated in chapter 10, which concludes the report.

2 Problem Analysis

This chapter lays the foundations of project AIP++, by explaining the context in which the application was developed and describing its most important features. Section 2.1 provides some background information on AIP, the basis on which AIP++ is developed. Consequently, the importance of AIP++ is outlined in section 2.2, which begins with a short description of similar products and highlights the novel features of the application. Section 2.3 contains a detailed explanation regarding the functionality of the tool, followed by a list of project objectives. Finally, an analysis of project stakeholders can be found in section 2.4.

2.1 From AIP to AIP++

Article Information Parser (AIP) is a project which facilitates parsing of article metadata, developed by the @Large Research team and currently available on GitHub [2]. The parser is designed specifically for articles in the field of computer systems. Three different data sources are parsed and unified, namely DBLP, Semantic Scholar, and AMiner, and the data is afterwards stored in a PostgreSQL database. The use of multiple datasets contributes to the completeness of the database, and in some cases helps with missing or inconsistent data. Publications are filtered by the Venue-Mapper Python library [1], developed by the authors of AIP, and those with recognized venues are then added to the database. In 2019, the number of articles in the AIP database was 117,058 [5].

AIP++, the application described throughout this report, is built on top of AIP. In essence, AIP++ has the functionality of an academic search engine, limited to the field of computer systems and the selection of venues determined by the developers of AIP [2]. Therefore, executing queries on the article database is a core feature of the application, and this can be done either through the graphical user interface or by writing a custom SQL query. Some other important features, like detection of rising stars and trending topics in academia, rely on Natural Language Processing (NLP) and data mining techniques. More details regarding the functionality of AIP++ are given in the remainder of this chapter, and a full overview of the application's features can be derived from its list of requirements (section 3). A clear outline of the different components of AIP and AIP++ can be found in chapter 4, section 4.1, which includes an architecture diagram (figure 4.1) and an explanation of how AIP++ was integrated on top of AIP.

2.2 The Relevance of AIP++

The goal of this section is to highlight the importance of an application like AIP++, in the context of today's research communities. To this end, subsection 2.2.1 includes a brief overview of the most widely used modern academic search engines, focusing on their shortcomings. Consequently, subsection 2.2.2 mentions the unique and novel features of AIP++, and explains how the application differs from other products of its kind.

[1] https://pypi.org/project/Venue-Mapper/

[2] This selection of venues is essential for our clients and enforced by the Venue-Mapper library, which was linked in the previous footnote.

2.2.1 Shortcomings of Similar Software Products

After taking the time to study the current market, we determined that there is a wide range of products that researchers typically use to look for scientific publications. A lot of them are free, such as the well-known *Google Scholar* [3], *Semantic Scholar* [4], *Microsoft Academic* [5], or *CiteSeerX*[6]. All of these academic search engines have large article databases that are maintained and updated regularly. Unfortunately, users of these tools are quite restricted in their search options. This often makes it impossible to execute personalized queries on the databases, and slows down the process of finding related work.

Other tools, like Elevier's *Scopus*[7], *Web of Science*[8] or *IEEE Xplore*[9], are less restrictive in terms of querying. As an example, figure 2.1 includes a screenshot of the basic search functionality in *Scopus*. The query can be customized with more search fields, and these are connected by a binary operator of the user's choice. Search fields include, but are not limited to, the abstract, the title, authors, keywords, references, language, and the source of the publication. Nonetheless, a major drawback of *Scopus* and the other tools mentioned at the beginning of this paragraph is the fact that they are paywalled. Therefore, only researchers from registered institutions have access to the their full functionality.

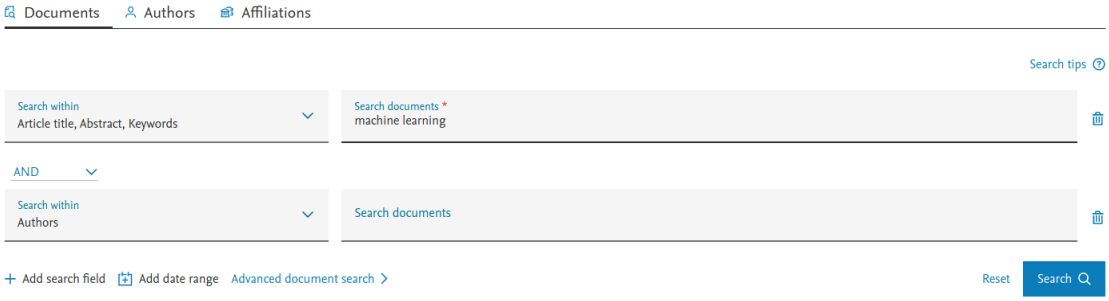

Figure 2.1: A screenshot of the basic search functionality in *Scopus*, taken at https://www.scopus.com, on June 6, 2021. Only two search fields are included in this picture, but more can be added and customized.

None of the previously mentioned tools is open-source, and the great majority of them do not exploit the full potential of their article databases, which could be mined to produce significant results that are informative for the research community. Perhaps an exception is *Microsoft Academic*. Among other useful features, this tool showcases top authors in a research field, related topics for each article, a graph-based visualization tool for research topics, relationships based on citations, and author-journal relationships. However, *Microsoft Academic* still lacks functionality for rising stars detection, a feature that is considered very important by our clients.

[3]https://scholar.google.com/
[4]https://www.semanticscholar.org/
[5]https://academic.microsoft.com/home
[6]https://citeseerx.ist.psu.edu
[7]https://www.scopus.com/
[8]https://login.webofknowledge.com
[9]https://ieeexplore.ieee.org

2.2.2 Unique Functionality of AIP++

As briefly mentioned in the previous subsection, rising stars detection is a novel feature among academic search engines today, and one of the main highlights of AIP++. Finding junior researchers with a lot of potential can be very valuable for senior researchers and industry leaders, who might be interested in offering them jobs or positions in academia [6]. Therefore, our clients believe that laying the foundations for such a feature will be a great help for the computer systems research community. As a result, a lot of the team's research and implementation efforts were dedicated to this functionality, which is further documented in section 6.1.

Another unique feature of AIP++ is custom querying, implemented in a way that is different from other academic search engines. Aside from having basic querying functionality, as shown in figure B.3 of appendix B, the application also allows users to write their own read-only SQL queries and execute them on the article database. In order to make this possible, the schema of the database is disclosed to the user. As is the case with basic queries, these raw SQL queries can be marked as favorite and reused later. Query results can be exported to a CSV (Comma-Separated Values) file, whose title includes the version of the database on which the query was executed. This is an essential feature of the application, as it ensures that all results are reproducible and the user is informed about the context in which they were calculated.

2.3 Project Outline

This section builds on the previous sections of this chapter, and includes a clear overview of the application and the project as a whole. Subsection 2.3.1 contains a description of the application's functionality, as determined by our interviews with the client, and subsection 2.3.2 lists the objectives of the project, which served as guidance for the team.

2.3.1 A Feature Overview of AIP++

As previously mentioned in this chapter, AIP++ is an extension of AIP (see section 2.1). On the one hand, this means that new functionality has to be added to the base application. Rising stars detection is an important part of this functionality, but there are many other features that were requested by the clients. Some of them were briefly touched upon in sections 2.1 and 2.2, but the reader can find a complete list in chapter 3, where the functional requirements with the highest priorities are included in subsections 3.1.2 and 3.1.3. Detailed descriptions of the most interesting features of the application are provided in chapter 6, and visualizations can be found in appendix B.

On the other hand, extending AIP also entails making certain improvements to the application. These could not be fully captured in the list of requirements, although many of them appear to be non-functional, since the clients delegated a lot of major design decisions to the team. A first point of concern is the usability of AIP. The Graphical User Interface (GUI) has to be rebuilt from scratch, as it is currently outdated[10]. Moreover, running the application is currently difficult for the average user. All of the data sources have to be downloaded, which requires a lot of memory, a PostgreSQL server has to be running in the background, and the file paths included in the code have to be manually changed for the parser to be able to find the necessary files. Another point of concern is the performance of the database. The database of AIP does not contain any foreign

[10]Previous developers of AIP initially designed the GUI for a MySQL database, but the application migrated to PostgreSQL. The GUI has not been updated and no longer works.

key constraints. This significantly slows down the parser. The team came up with solutions to the problems discussed in this paragraph, and the reader can find more information in chapter 4.

2.3.2 Project Goals

As requested by our clients, the team aims to build an application tailored to academics, with an intuitive user interface and good performance. To this end, the following goals are established for the project:

- **Build a functioning application that meets at least all *Must Have* requirements, and the majority of *Should Have* requirements.** We acknowledge that problems may arise during the course of the project, and so we might not be able to meet all of the requirements outlined in chapter 3. However, we aim to at least cover those features which are essential to our client.
- **Deliver a product that is maintainable, documented, and thoroughly tested.** We believe that the right thing to do is to focus on quality, rather than quantity. As this is an open-source project, it is most important that other developers can build upon our code and improve the application further.
- **Write a final report which documents our research and the project's functionality.** All of the features of AIP++ will be described in this report, and major design decisions of the team will be fully and thoroughly documented.

2.4 A Discussion on 7 Key Stakeholders of the Product

A product stakeholder can be defined as a person influenced by our product [7]. This influence can be either positive or negative. In other words, a stakeholder either benefits or loses something as a consequence of the project. Identification of the stakeholders is a non-trivial task requiring us to look at the whole structure of our project from a wider perspective. A thorough analysis of all the factors impacting and being impacted by our work not only allows us to develop a functional application, but also ensures it is correct from an ethical standpoint.

We have managed to identify the following stakeholders:

1. **Authors of articles in the databases.** The databases that are already incorporated into AIP, as well as the ones that are still to be added, contain names of researchers who have devoted their valuable time to their work. The way our application works could potentially discriminate against some of them. We should give special attention to ensuring that all of their work is uniformly accessible to the users reaching the publication database through our interface.
2. **Owners of the data sets used.** Semantic Scholar, DBLP and AMiner are the entities currently involved in providing AIP with data sets. These data sets contain research information used to supplement the main database. We are responsible for handling these archives appropriately and, therefore, we should strongly avoid modifying the information contained in them. As we add more data sources to the project, we will have to apply a similar approach to new data owners.

3. **Application users.** Undoubtedly, most of the potential users of our application are people searching for scientific articles in order to conduct research. The validity of the research can be highly influenced by the kind of results they find. Moreover, some of the features may contribute to rising popularity of particular scientific fields and scientists.

4. **Our clients - @Large Research.** @Large Research is a research group specializing in Massivizing Computer Systems. Our clients expect us to develop an application on top of the components they have already created. They demand the application to be scalable and maintainable, such that it can be expanded in the future. We communicate with our client on a regular basis to control whether our results meet their expectations.

5. **Our teacher mentor and teaching assistant.** These are the people responsible for assessing the process of the application development, as well as grading the final product. Their feedback can possibly influence our workflow and introduce changes to the product.

6. **Delft University of Technology.** Since the project is a part of the Computer Science and Engineering curriculum at Delft University of Technology, our work is bound to represent the name of the university itself.

7. **We - the developers.** Finally, we, as the developers, are influenced by the outcome of our work as the entities directly involved in the development process and responsible for the delivery of a functioning product.

3 Software Requirements of AIP++

This chapter includes all of the software requirements of AIP++. In section 3.1, the functional requirements of the application are listed and prioritized using the MoSCoW method. Section 3.2 follows up with a complete overview of the system's non-functional requirements.

3.1 Prioritisation of Functional Requirements

The prioritisation strategy for the functional requirements of AIP++ is discussed in this section. First, a brief explanation of the MoSCoW method is given in subsection 3.1.1, after which the functional requirements are distributed into *Must Have*'s, *Should Have*'s, *Could Have*'s, and *Won't Have*'s, in subsections 3.1.2-3.1.5.

3.1.1 The MoSCoW Method

Functional requirements describe the features of the application, as well as its behavior and reactions to user inputs [8]. We have decided to prioritize functional requirements using the MoSCoW method, which is common for teams that use an agile approach to software development. Under time pressure and unforeseeable obstacles, it is our intention to deliver a product that can still be useful to our clients, whose approval was required for the requirement prioritization scheme of AIP++. Functional requirements can be distinguished into four categories [9]:

- *Must Have* - requirements that have to be implemented for the final product to be accepted by the clients;
- *Should Have* - requirements that are almost as important as the previous category, but can be overlooked if the team faces an extreme challenge;
- *Could Have* - requirements that describe features the clients would like to have, but do not prioritize as much as the first two categories;
- *Won't Have* - requirements that can be added to the application in the future, but will not be implemented by our team during this run of the project.

3.1.2 *Must Have* Requirements

The user is able to...

1. browse the database using the GUI.
2. filter based on columns which have a numerical data type, by specifying a range.
3. search for papers in the database, by specifying the value of a field (for example: Digital Object Identifier (DOI), author).
4. see all of the available information about a paper in the database, that is: the id, the title, the authors, the publication venue, the volume of the publishing journal (only for some papers), the DOI, the abstract, and the number of citations.

5. see all of the available information about an author in the database, that is: the id, the name, and the ORCID, the latter only if available.
6. sort the database table lexicographically.
7. execute custom SQL queries on the databases, by typing them out and running them.
8. only query and view the data, not modify it.
9. export the results of a query. The exported file will contain the version of the database on which the query was executed (that is, information regarding when the database and data sources were last updated).
10. update the database when a new version is available. This means that the database automatically and periodically is checked for new versions.
11. see which versions of the data sources are being used, as well as when the database was last modified.

3.1.3 *Should Have* Requirements

The user is able to...

1. click on the DOI of a paper, to be redirected to the web-page where the publisher posted that paper.
2. access data from the MAG database, through our application.
3. save a query, marking it as favorite.
4. see the list of his/her favorite queries.
5. execute a favorite query by clicking on it.
6. remove a query from his/her list of favorites.
7. export his/her list of favorite queries.
8. import a list of favorite queries.
9. see the SQL query that was executed after every search.
10. see who are the rising stars in the research field, based on the articles contained in the database.
11. see who are the rising stars for articles containing a given keyword. [1]

[1] This was initially part of the *Could Have* requirements for the application. In week 7 of the project, the team and the clients gave this requirement a higher priority.

3.1.4 *Could Have* Requirements

The user is able to...

1. see the authors with whom an author has collaborated.
2. see a graph visualization of an author's network, meaning that (s)he can see if there is a strong connection between certain groups of authors.
3. see what the trending topics are (we call these emerging keywords). These keywords are found through analysis of publications' popularity over the years and extraction of common, non-stop keywords from clustered papers.
4. click on an emerging keyword and see which articles it is linked to (this relates to the previous requirement, which provides the list of keywords).
5. see a list of articles which are cited by a given article.
6. see which articles cite a given article.
7. give a name to each favorite query.
8. search for a favorite query in his/her list, by providing its name.
9. click on an author to reveal more information about him/her.

3.1.5 *Won't Have* Requirements

The user is able to...

1. access articles from fields other than computer systems.
2. see the history of his/her queries.
3. receive recommendations regarding papers that might interest him/her, based on his/her queries.

3.2 Non-Functional Requirements

Non-functional requirements describe how the system as a whole is defined by certain constraints imposed by the client. They can be related to the performance of the application in terms of speed or memory, the programming language that has to be used, the operating system on which the application should run, or security standards. Unlike functional requirements, which can be dropped if they have a low priority, the non-functional requirements of an application typically have to be fulfilled in their entirety.

There are three categories of non-functional requirements: product requirements, organizational requirements and external requirements [8]. Firstly, product requirements focus the behaviour of the system, defining how fast, secure, or reliable it should be. Secondly, organizational requirements are specific to the organization requesting the product, and can sometimes determine things like programming languages, software licensing, or the team's development methodology. Lastly, external requirements are out of the scope of this report, and do not apply to AIP++. The non-functional requirements of AIP++ are listed below and labelled according to the category to which they belong

1. The system will be open-source. The source code of the application will be published after the project is completed. (organizational)
2. The final version of the source code will be available on the @Large Research GitHub page. (organizational)
3. The product will be developed as a web application, and users will be able to run the application locally, or by connecting to a remote server. (organizational)
4. The system will be protected against malicious users. The database will be available as read-only, and no queries should modify it. (product)
5. The latency of the application will be in the order of seconds, that is, it should never exceed one minute. (product)
6. As compared to the current version of AIP, the performance of the database will show improvement in terms of latency, due to the use of indexing and other suitable techniques. (product)

4 The Design of AIP++

This chapter gives an overview of AIP++ as a software application, starting from its architecture and subsequently revealing the details of its implementation. Section 4.1 describes the chosen architecture of the system, as well as its components. Next, section 4.2 outlines the technologies, libraries and frameworks used in the development of the application on the back-end, front-end, and database side.

4.1 The High-Level Architecture of the System

Looking deeper into this project, it becomes clear that the application can be easily divided into three distinguishable layers. These layers perfectly align with the Model-View-Controller (MVC) architecture. The first layer, the view, aligns with the front-end of AIP++. The controller corresponds to the back-end, which is the second layer. Lastly, the database represents the third layer – the model. MVC is an example of a Layered Architectural Pattern, which represents one of the most widely used approaches nowadays [10, p. 80]. Following this pattern should satisfy our project's need for simultaneous development, ease of modification and frictionless testability.

Regarding simultaneous development, which is one of the advantages mentioned previously, this division directly leads to a smooth distribution of work among the group members, considering that every layer acts as a separate entity. As a result, each component can be developed by specific team members, independently of one another, and integration of these three layers is effortless.

Figure 4.1 shows how the MVC architectural pattern was applied to AIP++. This high-level overview highlights seven different components, three of which align with the layers identified at the beginning of this section: the GUI (view), the REST API application (controller), and the database (model). The parser is a legacy component, used to process data sources and write to the database, and the local storage is used for managing favorite queries. Descriptions of all of these components are given below:

1. **The GUI.** The GUI of AIP++ was built from scratch and developed as a web application in HTML, JavaScript and CSS. Popular frameworks as React, Material.UI and D3.js were used.

2. **The Server (REST API Application).** The server of AIP++ was developed using the Django REST Framework. Models communicate with the relational database and replicate the entities and relationships in it. Views implement the logic of the application, and represent API endpoints.

3. **The Parser.** The parser is a legacy component, which originally belonged to AIP. As the name suggests, this component parses the data sources that make up the application's database.

4. **Local Storage.** A user of AIP++ can mark certain queries as favorite. To avoid the need for managing user accounts, the front-end of the application stores these queries in the browser's local storage.

5. **The Database.** The database of the application can be accessed both locally, as well as remotely. Its underlying technology is PostgreSQL.

6. **The Venue Mapper.** As mentioned in section 2.1, the Venue-Mapper is an external Python library created by the developers of AIP. It is used to mitigate naming inconsistencies of publication venues, and to select the most important venues in the field of computer systems.

7. **The Datasets.** AIP included support for parsing of the DBLP, AMiner, and Semantic Scholar datasets. In AIP++, Microsoft Academic Graph (MAG) was added as a fourth dataset. DBLP is used as a main dataset in the application, complemented with additional information from the other datasets.

To implement the MVC architecture, a typical use case of our application would look as follows:

- The controller receives an HTTP request sent by the user.
- The controller contacts the model to retrieve the requested data.
- The controller performs appropriate operations on the data.
- The results of the data manipulations are sent to the view layer.
- The view layer renders a display for the user to see.

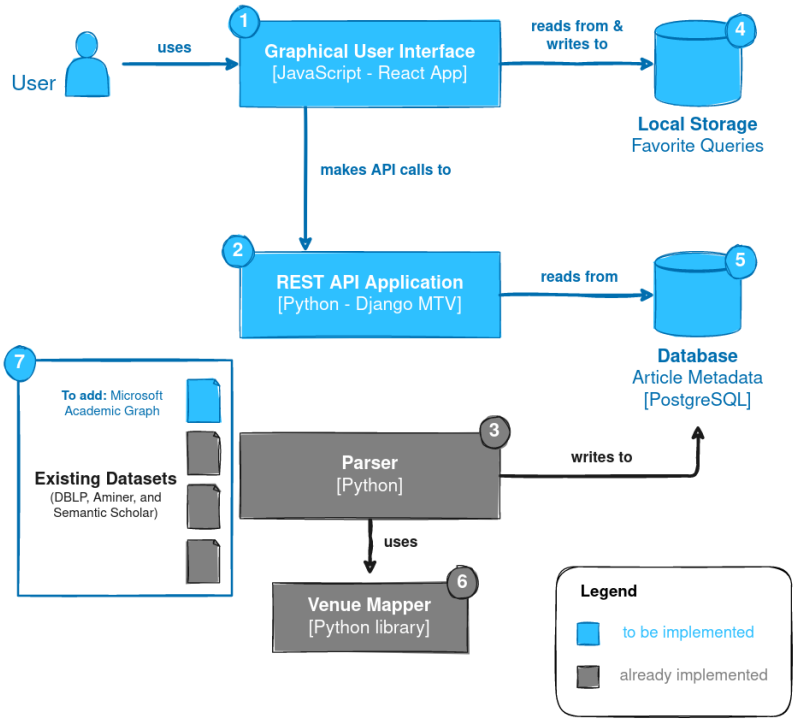

Figure 4.1: A high-level overview of the system architecture of AIP++.

4.2 The Implementation Details of AIP++

The goal of this section is to familiarize the reader with the implementation details of AIP++. To this end, subsection 4.2.1 describes how Django is used as the main framework for the back-end of

the application, whereas subsection 4.2.2 explains how the client was built with React. Details about the database and the parser are included in subsection 4.2.3. Finally, subsection 4.2.4 outlines the team's experience using Docker throughout this project.

4.2.1 The Back-End of the Application

A Comparison Between 4 Back-End Frameworks

There are various open-source frameworks available, offering a range of features which aim to simplify the development process for the server side of the application [11]. In this section, we are going to mention and summarize the most popular ones.

- **Ruby on Rails** - Ruby-based open-source framework designed for web applications. Rails support multiple pre-made modules, reducing the time needed for the construction of the most generic code. Moreover, the framework provides special tools designed to facilitate testing. We have rejected this framework, as we have no knowledge about Ruby and learning its basics could waste too much of the resources at hand.
- **Spring Boot** - framework well-known to TU Delft students. Spring Boot helps set up Java web applications and provides programmers with more advanced functionalities, which we will most likely not need. Besides, the already existing database code is written in Python. We abstain from using many programming languages in order to reduce the complexity of our project and to avoid the need to integrate components written in two different languages.
- **Flask** - simple and flexible Python-based framework that does not require components from external sources to function. Unfortunately, this framework does not support a database abstraction layer, making communication with the databases difficult to implement.
- **Django** - Python-based framework combining all the favorable aspects of the frameworks mentioned before. Its gentle learning curve and support for various web appliances should greatly accelerate the development process and allow us to create an application with every core feature mentioned by our client. The key characteristics of this framework are explored further in the next section.

Django's Strengths as a Back-End Framework

To facilitate the development process, we looked at the most popular back-end frameworks. Undoubtedly, all of them offer range of functionalities that we could find helpful. After careful research, we have opted for a solution using REST API along with the Django framework - one of the most popular frameworks used by websites such as Instagram and Pinterest. There are plenty of reasons for this choice.

A familiar programming language. Django is a framework for Python. The existing code, which we are expected to expand with new functionalities, is written in Python. That makes the coding process more seamless, as we do not have to overcome issues connected with combining multiple programming languages in one project. Moreover, Python is an easy-to-use language, which we have already used throughout the course of our bachelor's degree, therefore, we we will not incur any time overhead unavoidable when learning a new language.

Scalability, maintainability, and reusability. Django meets some of the most important requirements of our project - it is both scalable and maintainable. In the case of Django, back-end is divided into so-called applications. Each application is treated as a separate entity, which can be easily modified, replaced or removed without any interference with other parts of the code. Furthermore, new components can be freely added as the project grows bigger. That guarantees scalability. Similarly, code reusability is highly encouraged by the framework, as the applications can be copied and reused in other programs.

A framework that can be easily integrated with the rest of the project. Lastly, Django is one of the most versatile frameworks available, providing tools to interact with databases easily and work with any user-side frameworks, making back-end and front-end completely independent in terms of the framework choice.

4.2.2 The Front-End of the Application

The major choice regarding the front-end of our application was between a standalone desktop application and a web-based application in the browser. The second approach was deemed more advantageous, as it seemed to offer better user experience and more flexibility. A web-based application could be run both remotely or locally, if someone wishes to host it. Having both of these options available is a non-functional requirement of AIP++ (see subsection 3.2).

The second choice we had to make is choosing a component-based UI library. It is a very important part of the process, as making the right choice will accelerate the UI development process greatly by decreasing the amount of effort put into establishing the basic logic of our application. There are three major competitors on this market - **React**, **Angular** and **Vue.js**. We chose React for the reasons below.

- It has been developed by Facebook and it is the most popular framework among the ones considered [12]. That means it is the most widely tested and has the greatest support from the community. It also possesses a large developer tool set.
- It is free and open-source.
- Our team members are most familiar with it.
- It is very flexible, which means that, if we decide to add a new functionality during the development process, it will be easily incorporated into the existing application.

4.2.3 The Database

Integration With Django. Other than being used as the backbone of the application, Django also includes default ORM. We will use it to directly communicate with the database from the controller layer. This greatly improves the maintainability of the project, allowing for complex business logic while still having effective and efficient queries.

PostgreSQL. PostgreSQL is used for the current version of the system. It makes no sense changing the RDBMS to a different one, since, primarily, PostgreSQL is among the most widely used ones [13]. Thus, its SQL dialect should be familiar for the majority of stakeholders that already know SQL. Moreover, the use of a relational database, in comparison to a no-SQL one like MongoDB, is desired, because the system is highly unlikely to become distributed, thus we do not have to trade the *consistency* guaranteed by an RDBMS for the *partition tolerance* offered by no-SQL systems.

Neo4J. Neo4J is the most popular graph database [14]. It is very popular in the industry and its libraries can be used to visualize the results of the queries. During the interview with the client, we discovered that currently the database is often queried recursively (think of queries specific to networks, such as finding cliques). Graph databases perform significantly better for this type of queries, in comparison with relational databases [15]. Thus, we want to investigate possible performance gains, while trying Neo4J as an alternative. Conversion of data to Neo4J will be done through a Neo4J Python plugin.

Psycopg2. Psycopg2 is the most popular PostgreSQL adapter for Python. It is already being used in the base parser we have to extend. A major advantage of this technology is the thread-safety it guarantees. This aspect will be crucial when introducing parallelism to the application.

4.2.4 Integration With Docker

Docker was meant to be one of the fundamental parts of our project. The process of setting up the database, starting the back-end, compiling, and hosting the client might be a straightforward task for experienced users, but it is a very complex process for users that are not familiar with the architecture or technical caveats of the application we have created in the past 10 weeks. This can be vastly simplified with a containerization tool like Docker. Unfortunately, no one in the team had much experience with the tool, and a lot of research and experimentation efforts went into this.

At the start of the development process, we created a simple Docker image containing the relational database and the back-end. This image would just connect to the database, import the backup file into it, and in the end start the development server running the API. The configuration was very useful for both team members working on the back-end, as well as those who wanted to test out the application.

The next step was adding, compiling, and hosting the static files of the client, and then combining them with the API server. This was achieved using Nginx, the reverse proxy server that enables hosting of two different applications on one port. A configuration was written to make Nginx forward requests whose path-name started with */api* to the port on which the API was hosted. For all the other paths, the static files of the web application were served. The new docker image was a great testing tool for all the team members that were not experienced with the deployment of web applications, but still wanted to use the graphical interface locally. Manual configuration was no longer required, as it was enough for anyone to just have Docker installed on his/her machine, in order to run the application with just one command. Docker also turned out to be very useful in configuring continuous development later on in the development process.

The last step in our Docker configuration was the most challenging for us. We wanted to add the Neo4J database, which we intended to use for a author exploration feature. The goal was to have this set up similarly to how our Postgres database was initially configured with Docker. Although a few team members tried implementing this for a whole week, there was no success. We encountered several problems, ranging from failing to run the fresh image, to lacking the permissions to import the data. It is hard to tell if the docker image was underdeveloped and poorly documented, or if all of this was just caused by our lack of experience, more details about this are described in section 9.1.2 In the end, we have concluded that we cannot spend any more time on this issue and, as a team, we have decided to move forward and have the Neo4J database hosted separately.

5 The Development Process

The aim of this chapter is to give insight into the development process of project AIP++, by giving an account of the core values in the team's development methodology, as well as the guidelines that were followed in order to ensure that the software product was of high quality. Section 5.1 begins with a discussion of the main elements of the Scrum framework within the project, following up with a detailed description of the team's agile practices. Subsequently, section 5.2 shows how these practices were employed in order to guarantee that the functionality in the final version of AIP++ will be thoroughly tested.

5.1 The Agile Software Development Methodology of AIP++

The goal of this section is to provide several details about the team's development methodology are given. To this end, subsection 5.1.1 emphasizes the most important aspects of the Scrum framework in AIP++, and subsection 5.1.2 explores the core values of the team's agile development methodology.

5.1.1 Highlights of the Scrum Process

Scrum is a well-known and reliable methodology for agile project management, allowing teams to focus on dynamic, incremental development [8, p. 58]. Firstly, the dynamic nature of this approach lies in its core philosophy. The client is continuously involved in the development process, and requirements may change over time, which is something that an agile team is expected to handle. Secondly, versions of the software are released incrementally, typically at the end of each sprint[1]. The team tracks the completion status of a product backlog, which contains the full set of requirements for the application under development. Based on this backlog, planning is done at the beginning of each sprint, and tasks are distributed among team members.

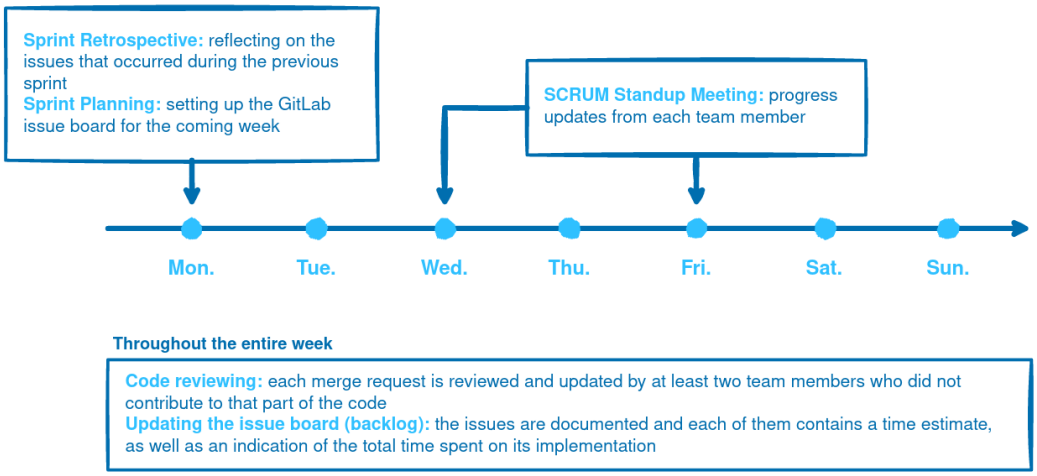

Figure 5.1: The main highlights of a Scrum sprint in the AIP++ project.

[1]In Scrum, a sprint represents a time period during which the team works on another increment of the system. A Scrum project usually spans multiple sprints.

Several elements of the Scrum methodology were used to develop the AIP++ project, which spanned a total of ten weeks. Figure 5.1 shows an overview of these elements, by giving a general outline of a typical sprint within the project. Each sprint lasted one week, which was a suitable choice given the scale of the project. At the beginning of each sprint, on Mondays, the team reflected on the previous sprint, and all important observations were recorded in the **Sprint Retrospective**. This gave an indication of what went well, what could be improved, and what adjustments would be made for the next sprint. After the retrospective, the team focused on **Sprint Planning**. GitLab issue boards were used to convey the requirements of the application, and new issues were created and assigned each week, during planning. A GitLab milestone was the equivalent of a sprint. Each milestone had its own set of issues, but these could also be delayed and moved to the next milestone. Two times a the week, on Wednesdays and Fridays, **Standup Meetings** were held for progress updates.

Additionally, to ensure smooth development, a set of guidelines was followed throughout each sprint. All branches in the GitLab repository were created from issues, and no changes were pushed directly to the development branch. Integration of new features was done through merge requests, which had to be reviewed and approved by at least two team members which did not contribute to the code that was being merged. All issues could be officially closed when their corresponding branches were merged into the development branch. A team member assigned to an issue had the responsibility to provide proper documentation of his/her work, and to keep track of the time (s)he spent on that issue.

5.1.2 Reprioritisation of Features and Dynamic Adjustments Within the Project

Aside from mentioning standard Scrum practices followed during the AIP++ project, it is important to emphasize why the team's development process was agile in nature, and how it differed from a classic plan-driven approach. Agile teams are able to distinguish themselves from others due to their adaptability and dynamic planning. Two main aspects of the AIP++ project highlighted these traits: the continuous restructuring of the project schedule, and the reprioritisation of an important requirement in a later stage of the development process.

The Flexibility of the Project Schedule

A Gantt chart was drafted up at the beginning of the project, to give an indication of time estimates for the most important features of the application, as well as other activities involved in the development process, such as stakeholder interviews, continuous testing, and meetings with the client. This schedule is illustrated in figure A.1 of appendix A. It might seem like such a planning strategy somehow contradicts the principles of agile development mentioned previously in this chapter, and is more aligned with plan-driven project management. However, that is a false impression. It is important to note that faded colors in the figure signify that, for each feature, there is always room for variation within the schedule, so no time estimates are fixed. This by itself is a distinguishing characteristic of the team's agile development practices.

As a follow up, figure A.2, depicting the final project schedule modified in week 9, is proof of these practices. Visible changes were made to the schedule, and the initial plan was disregarded on multiple occasions, in close consultation with the clients. Still, almost all features were completed successfully, one exception being the implementation of the graph database. This particular functionality was too time consuming, and negatively impacted the completion of other features. The implementation

problems that were encountered are described in more detail in subsection 4.2.4, as well as in chapter 9, section 9.1.2.

Requirement Reprioritisation

It is important for agile teams to be equipped to take on new requirements later on in the software development process, or to make changes to the already existing requirements, if the client makes a reasonable request. In AIP++, a great deal of emphasis was placed on detection of rising stars in a bibliometric network. Two variants of this algorithm were suggested by the clients: global and local detection of rising stars. The former would be run on the entire database, and output a list of rising stars for the field of computer systems. The latter would be keyword-based: for instance, if a user gave the keyword "serverless" as input, then (s)he would be able to see the rising stars whose publications contain that keyword in their abstracts.

Initially, global rising stars detection was marked as a *Should Have* requirement, whereas local rising stars detection was only labelled a *Could Have* requirement. In week 7 of the project, the clients made the decision to assign a higher priority to local rising stars detection, and make it a *Should Have* requirement (see subsection 3.1.3). Using the clustering technique implemented for the functionality of emerging keywords identification, and adapting the global rising stars algorithm, it was possible to fulfill this requirement in a relatively short time, adding an interesting, novel feature to the application.

5.2 Software Quality and Testing

An important characteristic of high-quality software is that it is thoroughly tested. In an agile development setting, this is strongly linked to the way in which the team members establish their own definition of "done". Such a guideline needs to be taken into account each time new functionality is added to the code base. In the implementation of AIP++, a new feature was marked by an issue on GitLab, as well as a corresponding branch. As established by the team, the feature would be considered "done" after the branch was merged into the main development branch.

Merge requests had three essential components that contributed to code quality: an approval rule, a proof of validation, and a passing pipeline. These are summarized in figure 5.2, which includes an interactive diagram that captures the essences of subsequent paragraphs. Firstly, as briefly mentioned in section 5.1, the approval rule for merge requests in AIP++ was a way to ensure that code reviews were properly conducted, and that all team members had at least basic knowledge of the features in the application, even if they were not directly involved in their implementation. Therefore, each merge request was carefully reviewed and approved by at least two team members who did not work on that portion of the code.

Secondly, proving that the code is validated means that tests need to be provided in the contents of the merge request, to check the correctness of the implemented feature. For some parts of the application, such as the parser and the back-end, the logic was validated mainly with unit tests. However, a few methods relied heavily on external libraries, and were not tested. Minimal automated tests are written for complex data mining features, whose validation is not straightforward, like detection rising stars and emerging keywords. Such methods are mostly validated manually, as explained in chapter 6. Some basic tests are also included in the front-end of the application, to check that rendering of the GUI works properly.

Lastly, the merge request needs to pass the GitLab CI/CD pipeline. Since the beginning of the project, this tool was used to support Continuous Integration and Continuous Development of AIP++. The first means that tests and checkstyle validation scripts are automatically executed each time new commits are pushed to the repository. This is done separately for the three main components in the application: the front-end, the back-end, and the parser. The second offers support for automatic deployment of the software version in the main development branch, after a new feature is added through a merge request.

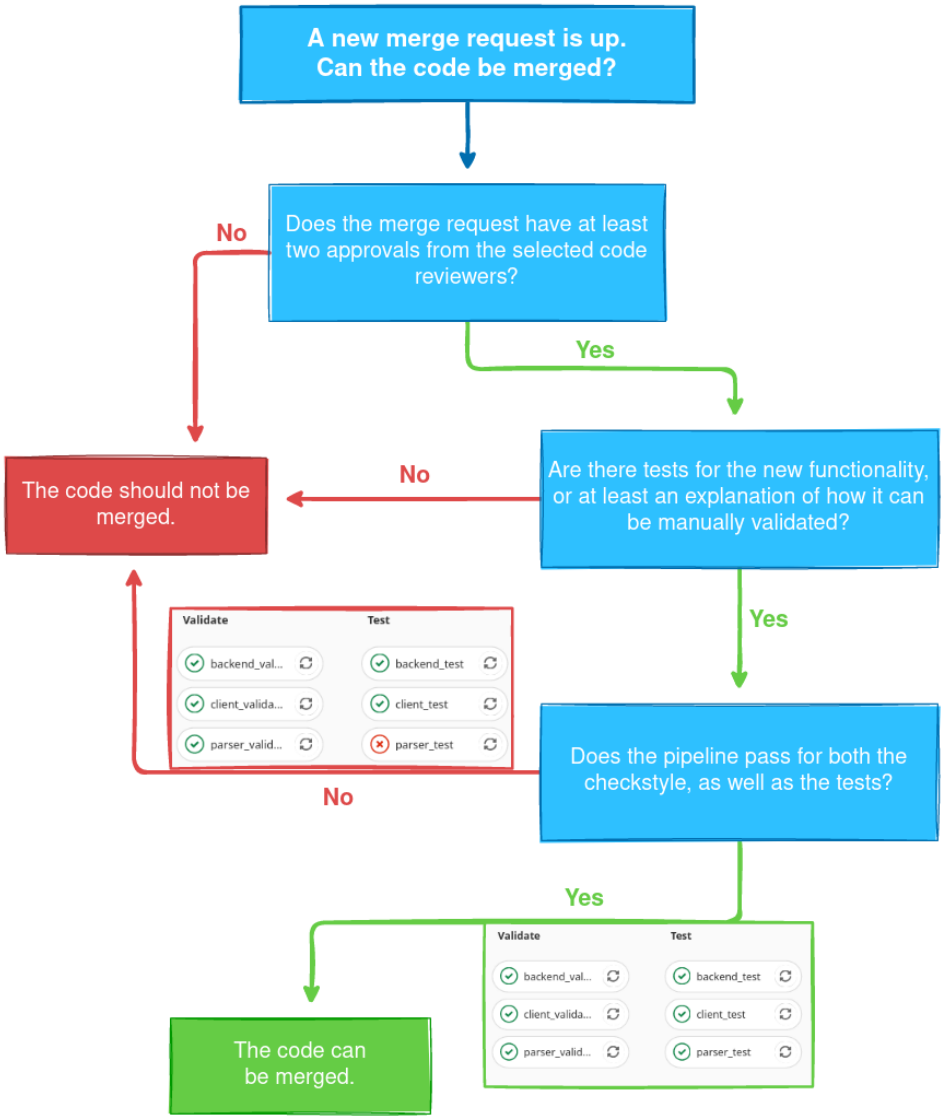

Figure 5.2: An interactive decision tree that can be used by the developers of AIP++ to determine if a branch should be merged into the main development branch of the application. This takes into account the tests performed on the code, the number of approvals, and the pipeline status of a merge request.

6 The Main Features of AIP++

The aim of this chapter is to analyze the algorithms employed in the features provided by our application. Section 6.1 describes the algorithm used in AIP++ to identify rising stars in academia. As this was our main research focus, the section also presents the research that was already conducted in that area and compares it to the approach and research methods adopted by the team. Section 6.2 illustrates methods used in the identification of the trending topics (the hot keywords). Lastly, section 6.3 concludes our findings in terms of the detection of strongly connected groups of authors.

6.1 Identifying Rising Stars in Academia

The purpose of this section is to highlight one of the main features of AIP++, which is identification of rising stars in academia. Subsection 6.1.1 puts our algorithms in the context of other works in on the topic. Research is done to show other possible approaches to the topic. Subsection 6.1.2 presents four different versions of our proposed rising stars detection algorithms: citation-based, PageRank approach, clustering approach, and rising stars obtained by keywords. Subsection 6.1.3 aims at validating these approaches to rising stars detection.

6.1.1 Research on Related Work

The majority of publications on the topic of rising stars detection in academia agree on the criteria which constitute a rising star, although a few differences apply. Firstly, an essential condition for a researcher to be considered a rising star is that his/her work is quickly gaining popularity within the community. This can be quantified by the number of citations, the quality of the venues where the author publishes, or the renown of his/her co-authors. The h-index [1] is also listed by some publications as a reliable characteristic [16], although others believe that it is an inconsistent metric [17]. Secondly, it is important for a rising star to be a researcher who is making a recent breakthrough in his/her career, and is not yet well-established and known within the field. Some approaches to rising stars detection regard academic age as a determining factor, and do not include authors whose academic age is higher than a specified number of years [2] [18] [19] [20]. This can be one way of ensuring that only junior researchers are labelled as rising stars. Different methods track a researcher's work over time, in order to detect a breakthrough, but do not incorporate academic age in their detection process [16] [17].

Figure 6.1 illustrates the different ways in which a researcher's career can evolve over time, and shows how rising stars differ from other types of authors. As explained previously, some detection algorithms use academic age, and some do not. Regardless of the chosen method, it is crucial that rising stars are identified during the first major breakthrough in their career, before they become well-established authors.

[1] https://en.wikipedia.org/wiki/H-index

[2] Typically, the maximum academic age is set to 5 years, but this can vary.

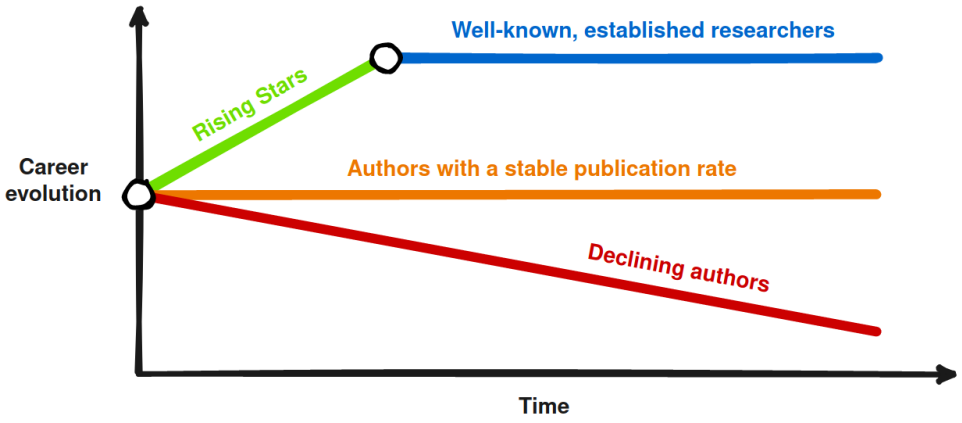

Figure 6.1: A simplified view of the different ways in which a researcher's career can evolve over time. Clearly, it is essential that rising stars are identified before they become senior researchers, who are already well-established within the field. [6, p. 634]

Methods that have been devised for rising stars detection can be divided into four categories: ranking, prediction, clustering, and analysis methods [6]. The remainder of this subsection will briefly describe some algorithms which belong to the first three categories. Table 6.1 includes a summary of these algorithms. Firstly, as the name suggests, ranking methods work by first assigning a score to each researcher in the bibliometric network, and using this to rank authors and select the rising stars among them. Secondly, prediction methods generally rely on an underlying supervised machine learning model, which uses a labelled training set and carefully selected features. Lastly, in contrast to prediction methods, clustering methods use algorithms which belong to the area of unsupervised learning techniques.

Ranking Methods for Rising Stars Detection

StarRank (2013), CocaRank (2016), and ScholarRank (2016) are all ranking algorithms that propose variations of PubRank, with the aim of achieving better detection accuracy. These ranking algorithms built upon each other and tackled previous limitations. In StarRank, two additions are made to the PubRank algorithm: using the order of authors in a publication to determine the extent of their contribution, and considering the evolution of a venue's rank over time [3] [19]. CocaRank introduces a new factor, the collaboration caliber, calculated in a way that reflects to what extent a researcher has the initiative to collaborate with others at the beginning of his/her career [20]. Lastly, among other new variations, ScholarRank runs the PageRank algorithm on a citation network, in order to determine the quality of a publication [18].

Another recent ranking algorithm, published in 2021, is the Hot Topics Rising Star Rank, abbreviated as the HTRS-Rank [21]. The authors argue that all previous rising stars detection methods overlooked a very important factor – the textual content of a publication, which can be used to determine its research topic. The reasoning behind this observation is the following: if an author has publications that belong to emerging topics in his/her area of research, then it is more likely that (s)he is a rising star. HTRS-Rank first clusters publications based on the topics that can be extracted from

[3] PubRank used a static ranking of venues, whereas StarRank tracks how the reputation of a venue changes over time.

their titles, and then identifies those which belong to hot topics. Authors are then ranked depending on their contribution to these topics, which leads to identification of rising stars.

Table 6.1: A brief description of all rising stars detection methods discussed in subsection 6.1.1, including ranking, prediction, and clustering methods.

A Summary of Existing Methods for Rising Stars Detection	
Method Name	Method Highlights
Ranking Methods	
PubRank [16]	One of the first methods for rising stars detection. The algorithm uses a modified version of PageRank to rank authors based on their co-author relationships and the quality of the venues where they publish.
StarRank [19]	An extension of the PubRank algorithm. StarRank also considers the order of authors in a publication, and ranks venues dynamically over a period of time.
CocaRank [20]	Introduces a new indicator to measure the social skills of a researcher, called the collaboration caliber.
ScholarRank [18]	Uses more available information from the bibliometric network, including citation relationships.
HTRS-Rank [21]	A recent ranking algorithm which mines the textual content in publications and finds rising stars depending on their contributions to hot topics.
Prediction Methods	
Binary Classification [22]	Uses supervised machine learning models to perform binary classification, and labels some authors as rising stars.
Clustering Methods	
Clustering of Authors [17]	After identifying key performance indicators for each author, the method uses clustering algorithms to create author profiles based on the changes in the indicators over time.

Prediction and Clustering Methods for Rising Stars Detection

Unlike previously described algorithms, prediction methods do not provide any ranking of authors, and instead aim to find rising stars with the help of supervised machine learning models. One example of such a method is that presented in [22]. In essence, the authors solve a binary classifica-

tion task, which determines if a scholar is a rising star or not. Discriminative and generative models were compared, and the feature space comprised of 11 dimensions. Features were selected based on a researcher's contributions, co-author relationships, citations, and publication venues. DBLP and AMiner datasets from 1996-2000 were used for training, validation, and testing. Results showed that venue-based features had the most significant impact.

Clustering methods, such as the one proposed in [17], are developed using unsupervised techniques. The methodology given as an example relies on the discovery of key performance indicators for each author. Similar to other methods, these indicators capture how well a scholar performs in his/her career, as well as the extent to which (s)he collaborates with other researchers. The next important step in the algorithm is tracking the changes in these indicators across a number of years. This information is then used in a clustering framework, where each cluster represents a different author profile. Lastly, identification of rising stars is based on the characteristics of these profiles, as these scholars are expected to show great improvements in their key performance indicators over time.

6.1.2 Detection of Rising Stars in AIP++

AIP++ has a dedicated tab for displaying rising stars in the field of computer systems, as shown in figure B.13. First of all, in this view, the user can specify a year as input. Out of all authors in the AIP++ database, only those who started publishing during that year, or subsequent years, will be considered by the rising stars detection algorithm. As mentioned in the previous section, some existing detection methods take into account the academic age of an author, and some do not. It is a good choice to let the user have control over this parameter. Moreover, the scores used to rank researchers and determine rising star are shown to the user.

Second of all, it is important to note that rising stars can be determined both globally, on the entire AIP++ database, or locally. The latter option works as follows. The user can enter a keyword, which may be something like "serverless", or "cloud". The rising stars algorithm will be run only on the set of authors who published papers containing the specified keyword in their abstract. The remainder of this section will provide more details regarding the implementation of these algorithms.

Implementation Details for the Rising Stars Algorithms

Four main variants of the rising star detection algorithm are available in the source code of AIP++:

- **Finding rising stars using sum of authors' citations** The user makes a choice regarding the publication year that will be passed as a parameter to the rising stars detection algorithm, which works as follows. First, each author with a suitable academic age is assigned to a specific band, based on the year of his/her first publication. Next, authors are compared to other authors of the same academic age. In particular, for each author, the number of citations extracted from the AIP++ database [4] is calculated and z-scored against authors with the same academic age. Lastly, the authors are sorted based on their z-score.
- **Finding rising stars using the PageRank algorithm.** For the PageRank version of the rising stars algorithm, page-ranks of publications are considered instead of citation numbers.

[4]Note that this database is limited to specific venues in the field of computer systems.

PageRank is first run on the publication graph, in which publications are nodes and citation indicate directed edges. A page-rank is then assigned to each author, equal to the sum of page-ranks for all of his/her publications. Author z-scores are calculated in the end.

- **Finding rising stars using clustering.** In this version of the algorithm, authors are not just divided into bands based on their academic age, but also depending on their community. Communities are extracted by running a clustering algorithm on the publications graph, as described in subsection 6.2.1. The z-scores of the authors are then calculated against other authors of the same band, similarly to the basic version of the algorithm.
- **Finding rising stars by keywords.** For this algorithm variant, previously mentioned algorithms are modified such that the calculation of a z-score is multiplied by a weight that relates authors to a specific keyword, depending on the user input. This weight is found by performing citation-based clustering of papers and extracting keywords for each cluster using the TF-IDF metric.

Incorporating order of authorship in these algorithms was considered, because many publications in the field of computer systems order authors based on the extent of their contribution. An analysis was performed, in which only the contribution of the first author was taken into account, and others were disregarded. As a result, some exceptions were identified. One of them is the Hardy-Littlewood rule, which states that all author contributions are always considered equal [23]. To avoid discriminating against researchers that publish in communities which adhere to similar rules, the order of authorship was no longer incorporated in the algorithms.

6.1.3 Validation of Rising Stars Detection in Our Application

Validation of the rising stars detection algorithms is a difficult task to carry out given the time constraints in the AIP++ project. This is due to the novelty of the feature, but also because it is difficult to properly outline the defining characteristics of rising stars, as mentioned at the start of this section. Efforts were made to create automated tests for this feature, and perform manual validation on a real dataset, as explained below.

Mock Dataset Validation

Automated tests were written to check the correctness of the global rising stars detection algorithms in AIP++ on a basic level. These relied on the use of a mock dataset of researchers and publications, conveying citations and co-authorships. Although this mock dataset was quite simplistic, and did not capture the complex nature of a real bibliometric network, it still helped reveal the presence of bugs in the rising stars algorithms. For instance, the algorithm did not perform a check on the number of authors in a band, and tried computing the standard deviation for sets consisting of just one data point, which resulted in an error message. Unchecked divisions by 0 were also identified during automated testing.

The mock dataset contained 13 authors and 56 publications, and it was constructed as follows. Four types of researchers were distinguished, as outlined in figure 6.1: senior researchers, stable authors, rising stars, and declining authors. The years in which papers were published ranged from 2005 to 2020. Defining characteristics of a rising star were the following: an academic age of less than 5 years, a rapidly growing citation count, and collaborations with renowned authors (or, as

previously called, senior researchers). All variants of the global rising star detection algorithm were able to distinguish these successful young scholars among those that met the academic age criteria.

Real Dataset Validation

Variants of the rising stars algorithms were compared using manual validation on a dataset representative of a real bibliometric network. The top performing researchers were examined for each variant. The procedure is as follows. The three versions of the rising star algorithms (basic, clustering version, page-rank version) are run with the keyword "serverless" and first year of publication equal to 2015. In each case, the first ten authors, in order of their calculated z-score, are analysed according to three metrics.

Firstly, it is checked whether the authors are indeed rising stars. It may happen that the authors have a publication history that dates back beyond the desired maximum year. This is mainly caused by the fact that the AIP++ parser omits some of the works of the author [5], which may date back past the year 2015. Thus, as one of our metrics, we examine the average academic age of the top 10 authors.

Secondly, we look at the average number of citations of the top authors for each algorithm. To quantify this number, we search for the authors' names in *Google Scholar* and record the total number of citations for each of them. We decided to use this academic search engine for validation purposes, instead of the AIP++ database, in order to have an external oracle that prevents potential biases introduced by the selection of publications in the database of the application.

Lastly, we check to what extent the top rising stars are related to the given keyword – "serverless". To do so, we examine the abstracts of all papers produced by the authors found by the algorithms and count the ratio of papers that include the desired word, "serverless", in their abstracts.

Table 6.2: Results of real dataset validation with comparison of all keyword-based rising stars algorithms. Citation data and academic age were extracted using *Google Scholar*, and validation was performed on the AIP++ PostgreSQL database, version 6066, last updated on 09.06.2021. The keyword used in this validation is "serverless".

A Comparison of Proposed Methods for Keyword-Based Rising Stars Detection			
Method Name	Average Academic Age	Average Number of Citations	Ratio of Works Related to Keyword
Basic version	5.4	388	0.67
Page rank approach	4.1	259	0.74
Clustering approach	3.9	280	0.71

Results and analysis As it can be seen in table 6.2, the average number of citations achieved by the basic algorithm is higher than for the page-rank and clustering approaches. This can be explained by the fact that the basic algorithm solely aims at finding young researchers who have a high number of citations in their papers. It does not take into account whether important papers cite their work, as is the case for the page-rank approach, nor does it consider in which type of community the paper is published, which is the case for the clustering approach.

For the ratio of works related to keywords, it can be seen that it is the highest for the page-rank approach, and the lowest for the basic version of the algorithm. This means that for this particular run

[5] As explained at the beginning of the report, the AIP++ database only contains a selection of articles from certain venues in the field of computer systems.

of the algorithm, the page-rank approach produces authors that are more related to the searched keyword - "serverless". It is worth mentioning the difference between algorithms is small – a maximum difference of 7% – and statistically insignificant. To possibly show statistical significance between the versions of the algorithms, the validation must be repeated many times for different keywords as input.

Finally, it can be observed that the average academic age for the basic version of the algorithm is the highest. This can be explained by the fact that 2 authors were falsely identified as authors that had their first publication after the year 2015. This is a result of the fact that the parser only considers articles that are published within certain computer systems venues, omitting the ones that belong to different fields or venues. Those 2 authors had published papers before in different fields, thus they cannot be considered young researchers. To combat this issue, it is recommended that users, after manual inspections, should be able to change the first publication year of authors in their local databases, if they feel that the AIP++ tool wrongly identified it, and re-run the algorithms.

6.2 Detection of Trending Topics

Detection of emerging keywords in science is an important matter that has already been tackled by several scientists. Used methods vary from machine learning approaches [24] to the use of two citation models simultaneously [25]. Due to the time constraints, we were not allowed to implement each of the algorithms, and, having thoroughly analyzed all the available options, we opted for the one that is flexible and suitable for our data set, both in terms of size and the covered fields of science. The bibliometric approach proposed by Qi Wing [26] falls exactly into that category. However, in our implementation, some elements of the algorithm are modified to better suit our case and adhere to the recent findings in the field of clustering algorithms. The implemented clustering algorithm is described in section 6.2.1 and the remaining part of the hot keywords algorithm is covered in section 6.2.2.

6.2.1 Clustering method for Trending Topics Detection

In order to group publications covering similar topics and outline communities among them, we have chosen to utilize a clustering algorithm for a non-directed graph. With the use of the database, this graph is constructed with vertices denoting publications and edges as citations between those. So far, a Louvain algorithm was considered to be one of the most efficient methods of finding strong connections in networks. It was widely used to find communities in convoluted networks such as social [27] and customer networks [28]. However, in recent years, significant progress has been made leading to the introduction of many improvements to the Louvain algorithm.

One of the most remarkable examples of such improvements is the algorithm we have chosen- the Leiden algorithm [29]. Its implementation not only solves the issue related to badly connected and disconnected graphs, which the Louvain Algorithm erroneously created, but also significantly improves its computation time, which is key in the case of large networks. In addition, it is highly flexible allowing to modify the clustering criteria and define the maximum size of communities. In our case, in order to properly capture the relations between, we have decided to limit the maximum number of members in a single community to 100 and set the resolution parameter to 1.

6.2.2 Algorithm for Detection of Emerging Keywords

Before delving deeper into the specifics of the algorithm, we should define what a hot keyword is. There are plenty of definitions used in the literature, therefore, to maintain a clear picture, we explicitly characterize it similarly to Rotolo et al. Namely, "radically novel and relatively fast-growing technology characterized by a certain degree of coherence persisting over time and with the potential to exert a considerable impact on the socio-economic domain(s) which is observed in terms of the composition of actors, institutions and patterns of interactions among those, along with the associated knowledge production processes. Its most prominent impact, however, lies in the future and so in the emergence phase is still somewhat uncertain and ambiguous." [30, p. 13]. From now on, the words 'hot' and 'emerging' will be used interchangeably, when referring to trending topics in science.

In the following paragraphs, we will take a closer look at each of the steps of the algorithm, which looks as follows:

1. All publications in the database are clustered using the clustering method described in section 6.2.1.

2. Each cluster is assigned with 25 more suitable features based on the contents of the abstracts and the titles of articles comprising them.

3. Keywords are aggregated. Each keyword is assigned to a list of publications characterized by it.

4. Keywords are filtered out based on 3 criteria- number of publications containing them, number of citations received in a specific period, and the relative growth of the publication count over the years.

With clusters of strongly related publications built, keywords are assigned to every cluster by employing a TF-IDF numerical statistic method. Juan Ramos [31] argues that TF-IDF algorithm is highly efficient in matching keywords closely related to a given document. However, he also mentions that this method does not recognize synonyms, therefore some important aspects of the publications can be overlooked. This is the reason why publication clustering has been implemented. By using direct-citation-based clustering with the size of communities limited to 100 publications, we are able to capture publications covering very similar topics, while not losing the sense of detail. Using clusters of larger size could potentially lead to losing part of important, yet less popular keywords, whereas using smaller clusters could result in insignificant features being treated as important keywords. Abstracts and titles of the grouped publications are aggregated and combined so that they can be used as a TF-IDF corpus. Once the weights of keywords of each cluster are calculated, 25 keywords with the highest grades are designated to each consecutive cluster. Limiting the number of keywords to 25 allows us to prune the list of keywords by eliminating the ones of less importance. It is important to notice, that a list of most common stop words is used to reduce the noise resulting from the existence of words such as 'so', 'she', 'it', 'this' etc. Appendix D.1 contains a partial list of keywords assigned to each cluster. One keyword can be assigned to multiple clusters, therefore, for every keyword, a list of all the publications characterized by it is collected.

The next step is to filter out all the keywords not eligible to be declared emerging. To that end, a slightly modified version of the process described by Qi Wing [26] is conducted. The proposed algorithm is flexible, as it allows to calculate the emerging topics in the past years. Therefore, every

stage detailed below can be performed from the perspective of any year available in the database. This aspect also proves useful the validation part, since one can see whether the emerging topics from the past kept gaining in popularity. Moreover, it provides the user with a unique insight into the history of Computer Systems, which can be found invaluable by many researchers. For instance, Erwin van Eyk et al.[32] investigated the evolution process of serverless computing. With the use of our tool, they could find the same results more easily by tracking the emerging topics over the years.

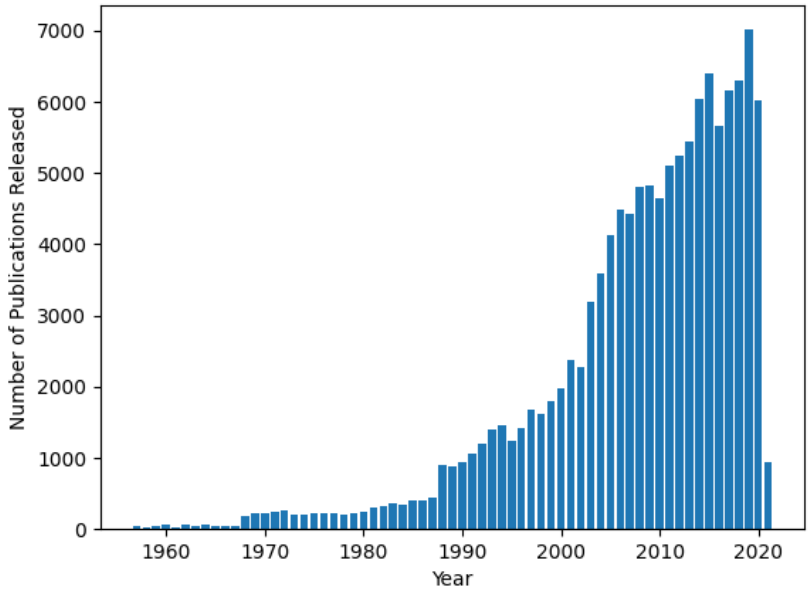

Figure 6.2: Graph depicting number of publications in the database released each year. The spread of the publications release years is not consistent over time.

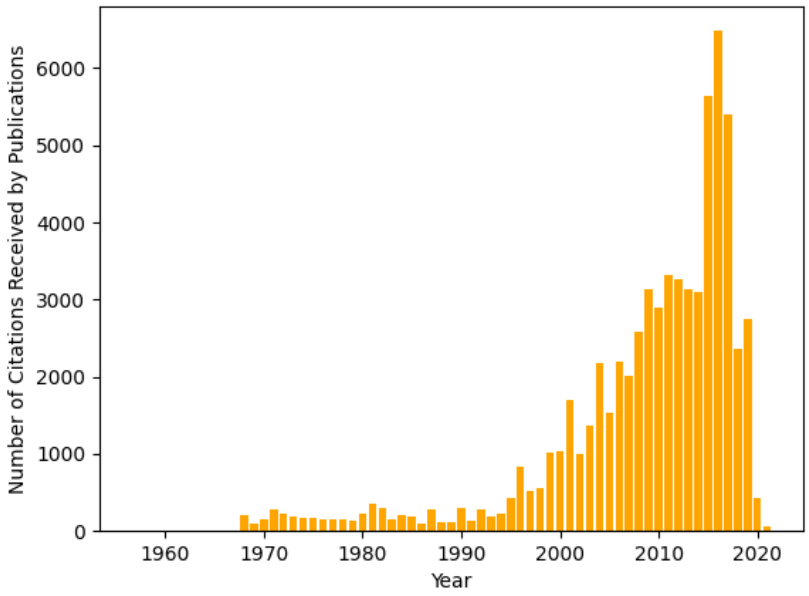

Figure 6.3: Graph depicting number of citations received by publications released in specific years. It is evident, that publications released in recent years are the most popular among researchers.

Firstly, for the given year and every separate keyword, the average p_1 of the publications released over the past three years is calculated. By doing so, it is ensured, that the spikes and irregularities visible in the figure 6.2 will not negatively contribute to the outcome of the calculations. Next, if p_1 of a keyword is bigger than the p_{max} parameter, the keyword is eliminated, as it does not show clear signs of novelty, having many publications assigned to it.

Secondly, the number of citations c received by all the publications released in the past dt years is summed up. Logically, the citations received after the chosen year t are not considered, as this would give us an unfair advantage in predicting past emerging keywords and introduce bias to validation. All keywords with the c value below c_{min} are disqualified because they do not seem to be attractive enough for us to consider. As seen in the figure 6.3, one needs to be very careful when choosing proper values for the p_{min} parameter, since the citations tend to fluctuate sharply. Our approach to the determination of used parameters will be discussed in detail in one of the coming paragraphs.

Lastly, the keywords, whose publication count growth is below the r_{min} parameter are excluded. The growth is calculated by dividing the number of keyword-associated publications released in t by the ones released in $t - dt$ (the publication count is averaged, as mentioned before). Thereby, publications showing a lack of development are not considered hot. It is easy to notice, that the r_min factor can be adversely biased by the number of publications released every year. For example, if the number of papers released in one year is smaller than the number of articles released in the previous year, it would be more difficult for a keyword to show a sufficient growth ratio, as the number of publications published one year is highly correlated with the number of publications associated with certain keywords. To neutralize this negative effect, an adjustable growth ratio can be used. The algorithm is run multiple times until a sufficient number of keywords is identified. At each consecutive run, the r_{min} parameter is decremented resulting in the emerging keywords constraints

becoming less strict.

We have decided not to incorporate the last step outlined by Qi Wing [26], as checking coherence does not fit into our clustering algorithm. Since one keyword may be assigned to multiple clusters, there is no need to check if it is fully coherent. That is also the result of using a very specific type of dataset, which only covers the field of Computer Systems. This narrows down the topical range of found keywords and is, by nature, more integrally coherent. We have arrived at this conclusion after implementing the coherence filter and realizing, that none of the keywords were affected by it. Therefore, utilizing it would unnecessarily increase the complexity of the process.

As mentioned before, finding the optimal values for parameters is not a trivial task, as these are strongly dependant on the year we are considering. To determine the optimal values of p_{max} and c_{min} two new parameters are introduced- p_{ratio} and c_{ratio}. Let p_t and c_t denote the total number of publications released in the year t and the citations received by those publications, respectively. The values of p_{max} and c_{min} are then calculated as follows:

$$p_{max} = p_{ratio} * p_t$$

$$c_{min} = c_{ratio} * c_t$$

This method ensures, that the constraints on publications remain consistent no matter what year is being taken into consideration.

Table 6.3: The most accurate parameters and emerging keywords found in the year 2020.

Parameters and emerging keywords in 2020				
dt	p_{ratio}	c_{ratio}	r_{min}	Emerging Keywords
5	0.01	0.1	2 (adjusted)	'layer', 'imagenet', 'trained', 'cnns', 'layers', 'dnn', 'view', 'mask', 'panoptic', 'coco', 'pixel', 'optical', 'action', 'frame', 'body'

In appendix D.2, we perform an analysis aiming to find the most accurate set of parameters. To that end, the developed algorithm is performed on the year 2000 with multiple sets of parameters. To support the validity of selected parameters, popularity of each resulting keyword is depicted on graphs. The most precise parameters are presented in table 6.3 along with the emerging keywords of the year 2020.

It is important to highlight, that each aforementioned parameter (i.e. t, dt, p_{ratio}, c_{ratio}) can be manually defined by the user through the advanced options incorporated into the GUI of AIP++. If none are chosen, the default parameters are selected. This allows the user to either define their own unique definition of a hot keyword or simply see the emerging topics computed with the predefined parameters.

6.3 Visualizing Author Connections in a Bibliometric Network

The author exploration functionality in AIP++ relies heavily on the visualization of a graph structure, which reveals a researcher's network, based on his/her citation and co-author relationships. A clear illustration of this feature is shown in screenshot B.9, included in appendix B. Author exploration works as follows. First, the user specifies the name of an author. Next, the graph is rendered and displayed, and can be navigated with the mouse. The user can also click on a researcher's name to find out more information about him/her. There are three different views of this graph: one that shows citation relationships between researchers, one that shows co-author relationships between researchers, and one that combines the two. Navigating between these views gives the user a clearer picture, and limits the sizes of the graphs that are displayed.

Another way to limit the size of this graph is by specifying a certain percentage of authors to be rendered. It is important to note that an edge between two authors is, in fact, a weighted edge. The weight represents either the amount of times the two researchers cited each other, or the number of publications they wrote together, depending on which view is rendered. Therefore, if the user only chooses to see 20% of the authors in a researcher's network, these authors will be ranked and selected based on the weights assigned to graph edges.

Lastly, the author exploration feature also gives the user the option to aggregate networks of different authors. For example, if the user is exploring the network of author A, and this author collaborated with author B, then the user can choose to add the network of author B to the rendered graph. This allows for visualization of indirect relationships between the authors in the bibliometric network.

7 Performance Comparisons

This chapter presents two important benchmarks that were performed during the AIP++ project. One of them is described in section 7.1, highlighting the optimizations that were made to the database to allow for faster keyword search within publication abstracts. The other, outlined in section 7.2, relates to a different topic: a performance comparison between Neo4J and PostgreSQL, for exploration of author connections in a bibliometric network.

7.1 Searching for Keywords in Publication Abstracts

Before delving into the contents of this section, it is important to clarify that the benchmarks which will be discussed in subsequent paragraphs were performed on AIP++, and do not reflect the performance of AIP. Several changes were made to the database of the original application, regarding the schema, addition and removal of indices, and foreign-key constraints. All comparisons were carried out after these changes were applied.

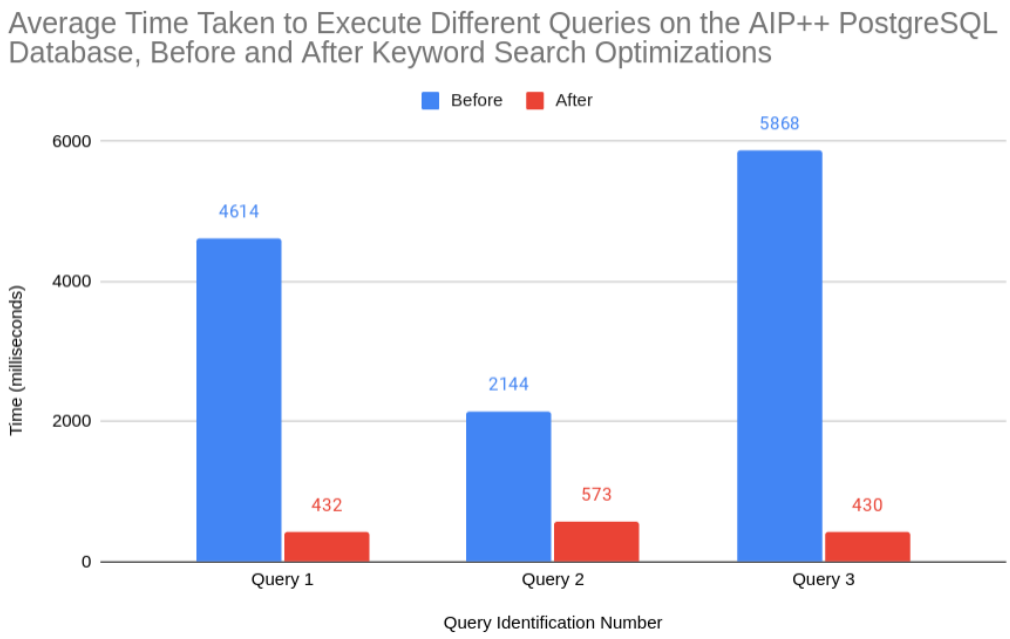

Figure 7.1: A comparison of average execution times for different queries executed on the AIP++ PostgreSQL database, measured in milliseconds, before and after implementing optimizations for keyword search within publication abstracts. Each average is computed across five runs of a query, and all queries went through a "warm-up" round consisting of three runs. The full queries are given in table C.1 of appendix C. This comparison was performed on version 6066 of the AIP++ PostgreSQL database, last updated on 09.06.2021.

In the current implementation of AIP++, searching for publications that contain certain keywords in their abstracts can be done in two different ways. A first approach is to search each abstract, and check if it contains the specified keyword. A typical abstract of a scientific publication ranges from 150 to 200 words [33]. At the time of writing this report, the count of publications in the AIP++

database is around 130'000, and the great majority of them have abstracts. These numbers give an indication that such a query could potentially be time-consuming and inefficient. A different approach involves the addition of two new tables to the database: *words*, which contains all of the words that appear in the abstracts, as well as their count, and *paper_word_pairs*, which maps each publication to the words in its abstract and their frequency. The computation of these two new tables is performed during parsing, the results are stored in the database, and proper indices are added.

A benchmark of these two approaches was conducted by performing three different queries on the AIP++ database, version 6066, last updated on 09.06.2021. These queries are all outlined in table C.1 of appendix C, and they were selected in such a way that they reflect common use cases for keyword searches. All queries were run five times, and the runtime was averaged across these executions. In addition to these five runs, each query also had a "warm-up" round consisting of three runs, which was necessary for setting up the database cache. It is also relevant to note that the SQL *EXPLAIN ANALYZE* tool was used to visualize intermediate steps in query executions, and, most importantly, to provide accurate measurements of the runtime of each query, in miliseconds. The comparison results are illustrated in figure 7.1. The performance improvement given by the optimized approach is significant in all cases. Many features of AIP++, such as finding emerging keywords or keyword-based rising stars detection, rely on queries of this sort. Therefore, this performance boost is very beneficial for the application.

7.2 A Benchmark of Two Types of Databases for Author Exploration

Another performance comparison that was conducted during the AIP++ project was a benchmark of two databases: a graph database, Neo4J, and a relational database, PostgreSQL. The queries used in this comparison actually form the basis of the author exploration feature in the application, where it is necessary to retrieve data concerning author relationships in the bibliometric network. More specifically, there are two different queries that fulfill this purpose: one which returns citation-based relationships between authors, and one which lists all pairs of authors who ever worked on a publication together. Each of these had to be written in both SQL, as well as Cypher, resulting in a total of four queries, outlined in table C.2 of appendix C. For each database, results of the two queries are returned together, and the execution times are summed up.

The outcomes of the benchmark are depicted in figure 7.2. Citations and co-authorships were computed for three different researchers in the AIP++ database, selected with regard to the size of their networks. On version 6066 of the AIP++ PostgreSQL database [1], updated on 09.06.2021, Rajkumar Buyya was part of a large author network, Alexandru Iosup was part of a medium-sized author network, and Derek Abbott was part of a smaller author network. For all queries used in this comparison, the Neo4J database outperformed PostgreSQL, almost halving the execution times. A possible explanation for the great performance of the Neo4J database is the underlying graph structure of the author networks that were analysed. Connections that occur in structures like these are not as naturally depicted in a relational database, like PostgreSQL. Additionally, it is noteworthy to mention that no significant differences in performance are noticeable for the three authors, which suggests that the size of a researcher's network does not have a considerable influence on query runtimes in this kind of benchmark.

[1] It is important to mention that the relational database was converted to Neo4J when this benchmark was performed. Therefore, the data contained in these two databases was consistent, which ensures that the results of this comparison are reliable.

Although the graph database performed better than the relational one, adding support for Neo4J in the author exploration feature was not possible during the scope of this project, mainly due to time constraints. Considerable efforts were made towards the implementation of this functionality, but they had to be abandoned, as some major challenges were encountered. This is further documented in subsection 4.2.4, but also at the end of the report, in chapter 9.

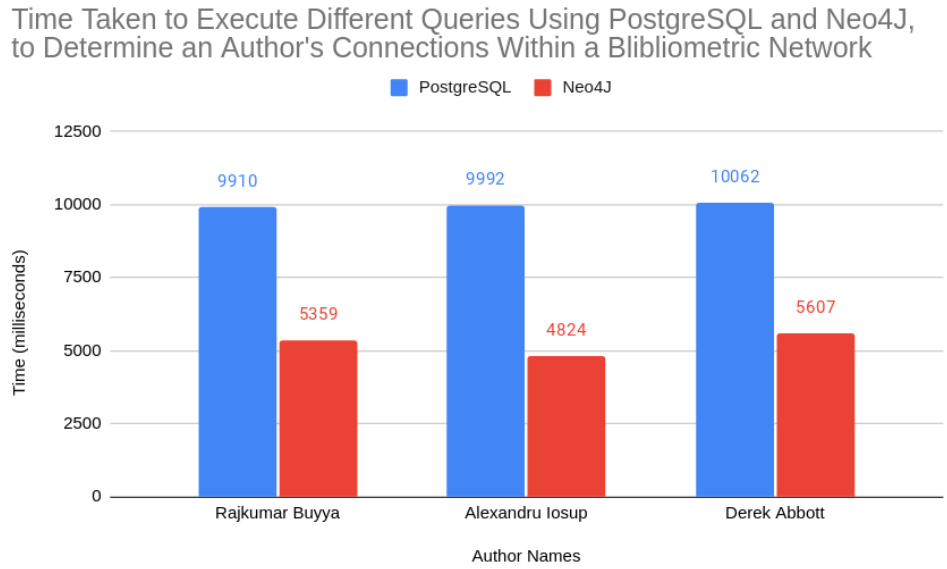

Figure 7.2: A comparison of execution times for different queries, measured in milliseconds, executed on a graph database, Neo4J, and a relational database, PostgreSQL. Two queries were run on each database: one for retrival of citation-based relationships for each author, and one that listed all co-authorships. The execution times of these two queries were summed up, and reported for each of the three authors. The full queries can be found in table C.2, appendix C. This comparison was performed on version 6066 of the AIP++ PostgreSQL database, last updated on 09.06.2021. The Neo4J database contained the same data as the PostgreSQL database, as conversion was perfomed for the purpose of conducting this benchmark.

8 Value Sensitive Design in AIP++

The goal of this chapter is to describe how fairness, as a fundamental value in software engineering, is incorporated in the design process of one of the key features in our application: rising stars detection. Section 8.1 formalizes the concept of bias in computer systems, and presents some examples of situations in which biased software tools exhibited unfair behavior towards specific groups of people. Consequently, section 8.2 includes a discussion on how bias can occur in the design of algorithms for detection of rising stars in academia, and explains to what extent the developers of AIP++ are responsible for mitigating its potentially harmful impact.

8.1 An Overview of Bias in Software

An application is considered biased "if it both unfairly and systematically discriminates against one group in favor of another" [34, p. 141]. First of all, the root cause of unfair discrimination is often a malicious decision making process, which intends on treating people based on their ethnicity, sexuality or religious belief etc. For instance, one can think of a tool that determines if someone should be granted a loan. If this tool disqualifies applicants who have a history of not fulfilling their payments on time, it is considered fair. On the other hand, if for example people of a certain ethnicity are intentionally excluded, the software is labelled unfair. Second of all, an important part of this definition is marked by the word "systematically". A software system that is not consistently unfair, meaning it does not make flawed decision recurrently on a subset of subjects with a single or more specific characteristics. Or simply makes mistakes by chance, should not be considered biased.

In accordance with the definition given in the previous paragraph, several software tools used in today's society turned out to be biased. One recent example is that of an algorithm used by hospitals in the USA to recommend specialized programs to people with severe health problems [35]. On average, this algorithm assigned lower risk scores to black people, even though the gravity of their illness was equal to that of white people. As a result, many people of color were not given the opportunity for better healthcare, whereas white people were privileged in this regard. Another infamous case is that of Amazon's recruiting tool, which filtered applicants based on their resumes [36]. In this scenario, it was apparent that the recruiting process heavily discriminated against women. The main causes of this were the underlying machine learning model of the software system and the lack of balance in the training data, where the amount of true positives for women was much smaller than in men.

To understand how such discrimination occurs in software and how it can be avoided, it is important to distinguish between three categories of bias in the context of computer systems: preexisting, technical, and emergent bias [37]. Firstly, as the name suggests, preexisting bias is established prior to the development of the software. This type of bias may originate in society or institutions, but can also be enforced by a major stakeholder in the project. Secondly, technical bias is usually rooted in the software design process, which may include the use of flawed algorithms or randomness in the source code of the application. Lastly, emergent bias relates to how the application is used after deployment, and in most cases is difficult to predict.

8.2 A Discussion on Fairness in Rising Stars Detection

In this section, we discuss the possible causes of unfairness in our rising stars algorithm (8.2.1), and give a set of general guidelines that we will follow in order to mitigate them (8.2.2).

8.2.1 Identifying Possible Causes of Bias

As it was mentioned throughout this document, one key feature of our application is the identification of rising stars in the field of computer systems. To this end, users will have the option to see a list of researchers that are considered rising stars, as determined by an algorithm. These are often new academics who are quickly gaining popularity in the field, and whose contribution can be described as innovative, despite the fact that their career is not fully established yet. Senior researchers and leaders in the industry might be interested in rising stars as potential candidates for PhD positions or jobs.

Technical bias is a main concern when implementing a data mining algorithm for a feature such as the one described in the previous paragraph. Many approaches studied in the past propose solutions that rely on variations of the PageRank algorithm, although some novel methods also consist of supervised or unsupervised machine learning algorithms. Regardless of the approach we use, it might be possible that our algorithm for example ends up discriminating against researchers from smaller, more isolated communities. That are not as popular as other communities and consequently would receive less citations and a lower score by our algorithm.

This does not mean that we completely disregard the possibility of preexisting or emerging bias in rising stars detection. Firstly, the data sources we use are not our own, and it is likely that preexisting bias could be present in them. An example of such bias would be the exclusion of articles from journals that are not as well known to the western world, or written in other languages, which would to an extent discriminate against researchers from certain ethnic groups. Secondly, it is still a challenge to predict how this feature is going to be used after the application is released. Emerging bias could arise if some authors are not considered worthy of a job position simply because our algorithm did not categorize them as rising stars.

8.2.2 The Team's Measures Against Bias

Technical bias is something that we can control, to a certain extent, as developers of AIP++. The team's efforts will be focused on ensuring that our rising stars algorithm does not use information which could directly or indirectly refer to a researcher's ethnicity, race, or gender. Testing will also be performed to validate that the researchers identified by our algorithm come from diverse backgrounds, and no group of people is preferred over another.

With regard to preexisting and emerging bias, it is important for the application to be transparent with its users. The source code will be publicly available and open to contributors, and the data sources will be disclosed, alongside their versions. Also, each time rising stars results are computed, z-scores will be made available. To mitigate the problem of emerging bias, should it occur after deployment, a disclaimer will be included alongside this feature, clarifying its intended use.

9 Recommendations for Further Development of AIP++

This chapter examines some of the improvements that could be made during future development of the application, and highlights its current limitations. In section 9.1, suggestions are brought forward regarding the parts of AIP++ that could not be polished or implemented to their full potential, due to time constraints within the project. Subsequently, section 9.2 includes a discussion about the limitations of the project, which relate to its design, scope, and purpose.

9.1 Suggested Improvements

The aim of this section is to elaborate on further improvements of the application, from extending the current functionality, to comprehensive research and validation. To this end, subsection 9.1.1 presents a more reliable approach for testing the rising stars algorithm in AIP++, and subsection 9.1.2 includes suggestions regarding future uses of the Neo4J database in the application. Finally, in subsection 9.1.3, some other areas of improvement are discussed, related to the usability of the tool and new research directions.

9.1.1 Thorough Validation of the Rising Stars Detection Algorithm

Identifying rising stars in a bibliometric network is a complex task, partially due to its novelty, but also because its validation is not straightforward. There are no datasets that clearly label these scholars, which could be potentially used to assess the accuracy of a detection method. Moreover, as mentioned in subsection 6.1.1, there are some subtle disagreements in the literature regarding the defining characteristics of a rising star, and this inevitably adds ambiguity to the validation process. The AIP++ project spanned ten weeks, during which there was little time to perform extensive validation of the rising stars detection algorithm, although some efforts were made in this regard (see subsection 6.1.3).

To address the need for further assessment of the rising stars detection algorithm in AIP++, a first step would be setting up a testing pipeline which tracks the evolution of authors over time. The algorithm can be run on a DBLP dataset from five years ago, complemented by AMiner, Semantic Scholar, and MAG data. The identified rising stars should be junior researchers, still in the first years of their career at the time of the prediction. Upon selecting the top ten scholars determined by the algorithm, developers can manually check if they have become well-established researchers in the present, which is the final step in the evolution of a rising star's career, as illustrated in figure 6.1.

9.1.2 Exploring the Potential of the Graph Database

Initial plans for the project included the maintenance of two databases: a graph database and a relational database. It was established that they would both contain the same data, and that experiments would be performed to assess their capabilities and performance on different queries and algorithms. Each database would then be used for those scenarios in which it outperformed the other. For instance, if the graph database allowed for faster exploration of author relationships in the bibliometric network, then it would replace the relational database in the implementation of this feature. The goal was to exploit the reliability of a relational database, and, at the same time, to

benefit from the flexibility and speed of a graph database on a network structure, such as the one created within research communities.

Two main difficulties were encountered while setting up this functionality, and they could not be overcome due to lack of time. First of all, a crucial problem was database synchronization. Essentially, this means that the data contained in these two databases would have to be consistent after each update. Although this is a major challenge by itself, the task also had an additional layer of complexity: conversion of data from PostgreSQL to Neo4J. Second of all, a significant impediment was the scarce documentation available for graph databases, with regard to the use case in AIP++. As explained in chapter 4, users of the application can run it both locally, as well as remotely. The team made great efforts to allow for Neo4J local deployment, using Docker, but they were all unsuccessful. Several peculiar errors occurred, ranging from availability problems to failures in the restoration of the database from a backup file. Unfortunately, no answers were available in the Neo4J documentation regarding these topics.

Even though this feature had to be abandoned during this run of the project, it is certainly an area for improvement that can play an important role in future developments of the application. The graph database was set up remotely, and some performance comparisons were made, as highlighted in chapter 7 of this report. These clearly show the potential of the Neo4J database, which outperformed PostgreSQL for the selected queries on the AIP++ dataset. With enough time, consistency and deployment issues can be overcome by future developers, and the graph database may likely become a valuable asset of AIP++.

9.1.3 Other Functionality Improvements and Research Directions

Adding support for automatic updates of the AIP++ database is another feature of high priority that should be considered during future runs of the project, as the data sources which supply the application are updated regularly. Ideally, such functionality would be available for both users with remote access, as well as those who run the application locally. The current implementation only allows users to check for updates, and does not perform them automatically, which is a downside in terms of usability. The reason for this is the difficulty of managing the large datasets required for parsing, which in total amount to approximately one terabyte of memory. Automatic updates will either require a lot of free storage, which can be expensive, or a strategy for gradual data fetching and processing.

In terms of research, the AIP++ project offers a lot of future possibilities. One of them is conducting an analysis of semantic-based clustering for publications, to enable detection of local rising stars or emerging keywords. It can be interesting to see what results will be produced this way, as the current versions of the algorithms cluster articles based on citation relationships, which might be favouring certain groups of authors. Another research direction can be related exclusively to detection of emerging topics. The implemented version of the algorithm only has support for identification of single keywords, and misses important groups of words that often go together, such as “machine learning” or “computer vision”. Moreover, output sanitation can be employed in order to differentiate between plural and singular version of one word.

9.2 Limitations of the Application

Due to its design and dependencies, AIP++ is limited to publications in the field of computer systems, written in English, and hence built for a specific target user, as envisioned by the clients.

As documented throughout this report, the main data source of the application is DBLP, augmented with additional data from AMiner, Semantic Scholar, and Microsoft Academic Graph. The use of DBLP as a central dataset already narrows down the range of articles to the field of computer science. In addition, the VenueMapper library aids the selection of publications from certain venues, a dependency which further limits the available data to the computer systems research community, a sub-community of computer science.

There are two main consequences of this limitation. The first is rooted in the design of the application's features. A great example is the rising stars detection algorithm, which is difficult to generalize to fields of research other than computer systems. Other examples of such features are detection of emerging topics, or custom SQL querying. It is inevitable that these features are tailored to researchers in the field of computer systems, and optimized using a dataset specific to the community, to align with the requirements of the clients. Hence, the application cannot be easily extended to other fields.

A second consequence is that AIP++ is only accessible to the English-speaking world, and mainly benefits researchers who are part of this circle. The same issue was raised in chapter 8, where it was argued that this can be a possible cause of bias against authors who publish in different languages. Unfortunately, time constraints in the AIP++ project did not allow for the integration of a different dataset, containing scientific publications written in languages other than English. Moreover, the team was not professionally equipped to design algorithms for NLP and data mining features for such data sources.

10 Conclusion

The aim of this project was to examine how an academic search engine may be implemented to include novel features such as finding the rising stars in academia, detection of emerging keywords in research communities, graph visualizations of authors in a bibliometric network, and custom SQL querying. To this end, AIP++ was developed as an extension of AIP, a project initiated by the members of the @Large Research team, some of which were our clients.

The approach for the implementation of this application relied on extensive research and validation, integrated with an agile development process. Several state-of-the-art algorithms and technologies were analyzed and compared, especially in the case of those features with a higher degree of complexity, like emerging keywords and rising stars detection. The system as a whole was studied and optimized, with the use of frameworks and architectural patterns.

The Scrum framework was a suitable choice for agile project management in the development of AIP++, alongside the MoSCoW prioritization strategy. This enabled for seamless creation of a detailed product backlog, which will serve as a fundamental guideline for future developers and the members of the @Large Research team. Following such a development methodology, the team was also able to reprioritize an important requirement, keyword-based detection of rising stars, and to successfully complete several other requirements, which brought the application one step closer to being fully ready for deployment.

Several modern research areas related to data mining and Natural Language Processing were examined in the course of this project, to lay the foundations for the functionality of AIP++. Detection of rising stars in the field of computer systems was performed using variations of the PageRank algorithm and clustering techniques for graph-like structures. These were adapted to offer additional support for local detection of rising stars, based on keywords which correspond to sub-fields in computer systems. Moreover, it was determined that applying clustering methods on a bibliometric network should not be limited to just one feature of AIP++. By utilizing clusters created by the Leiden algorithm and the TF-IDF statistical method, we have managed to capture some of the most promising emerging keywords in the field of computer systems, which will most likely shape the future of this ever-growing research community.

The design and architecture of AIP++ were clearly thought out, to guarantee the testability and maintainability of the application. The well-known and reliable Django REST Framework was used to build the server, enabling easy communication with the database layer, thanks to Django's Object-Relational Mapping (ORM). The front-end of the application was developed in React, a well-documented, powerful JavaScript framework. The back-end, front-end, and database were aligned with the Model-View-Controller (MVC) architecture.

Some research directions still remain unexplored in AIP++, and the product backlog has yet to be fully implemented. As detailed in section 9.1, recommendations are made for future development teams to further explore and validate the features of the application, focusing on rising stars and emerging keywords detection, and to continue working on those features that had to be abandoned during this run of the project. With further extensions enabled by the @Large Research team, it is quite likely that AIP++ will reach its full potential as a reliable, open-source academic search engine.

Glossary

academic age Typically, the number of years that have elapsed since the year of a researcher's first publication. 1, 21, 24–27

AMiner Also known as ArtnetMiner, AMiner is a search engine for scientific publications. The dataset provided by AMiner is free to download, and used in addition to MAG and Semantic Scholar, to completent the DBLP dataset in AIP++. 1, 3, 6, 13, 38, 40, 42, 43

bibliometric network A network describing a research community [38]. Nodes in a bibliometric network are typically authors, publications, or keywords. Edges can convey citations or co-author relationships. 1, 2, 19, 22, 25, 26, 32–34, 38, 41

DBLP A shorter name for the "DBLP Computer Science Bibliography", maintained by the Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH (LZI), Wadern. As its full title suggests, DBLP is a bibliography for publications in the field of computer science, and also represents the main dataset used to supply the AIP++ database. 1, 3, 6, 13, 24, 38, 40, 42, 43

emerging keyword In essence, a new keyword within a research community, which is quickly gaining popularity, but still characterized by a certain degree of ambiguity. An emerging keyword has five defining characteristics: (1) radical novelty, (2) relatively fast growth, (3) coherence, (4) prominent impact, and (5) uncertainty and ambiguity [30]. Also referred to as 'hot keyword'. 1, 2, 10, 19, 27, 34, 41, 55

MAG An abbreviation for the Microsoft Academic Graph. As its name suggests, this graph is maintained by Microsoft, and used by the Microsoft Academic search engine. It is free to download and represents one of the datasets used to complement DBLP in AIP++, alongside Semantic Scholar and AMiner. 1, 9, 13, 38, 42, 43

Neo4J A graph database management system owned by Neo4J, Inc. The query language of Neo4J is Cypher. 16, 33–35, 38, 39, 58

PageRank An algorithm used by Google Search to determine the importance of a web page on the internet, based on the pages that have a link to it. PageRank can be adapted and computed on any graph structure, not just the World Wide Wide. 21–25, 41

PostgreSQL A widely used, reliable, and open source relational database management system, also known as Postgres. As a query language, Postgres uses its own SQL dialect. 3, 5, 12, 26, 33–35, 39, 57, 58

rising star A junior researcher who has just entered a field, and whose publications are gaining citations and popularity in the community. A rising star usually publishes in a top journal early in his/her career, due to the innovative nature of his/her research. i, 1–5, 9, 19, 21–26, 34, 36–41, 56

Semantic Scholar An academic search engine developed and maintained by the Allen Institute for Artificial Intelligence. Like MAG and AMiner, the Semantic Scholar dataset, which can be downloaded for free, is used to complement DBLP as one of the data sources for the AIP++ database. 1, 3, 4, 6, 13, 38, 40, 42

TF-IDF A numerical statistic that aims to show the importance of a word in a document being part of a collection of documents. 25, 28, 41

z-score In statistics, the z-score is used to assess how far a data point is located from the mean of a distribution. 24–26

Acronyms

AIP Article Information Parser. i, 1, 3, 5, 6, 12, 13, 33, 41

API Application Programming Interface. 12, 16

DOI Digital Object Identifier. 8, 9

GUI Graphical User Interface. 5, 8, 12, 19, 51

HTRS Hot Topics Rising Star. 22

MVC Model-View-Controller. 12, 13, 41

NLP Natural Language Processing. 3, 40, 41

ORM Object-Relational Mapping. 41

REST REpresentational State Transfer. 12

Bibliography

[1] L. J. Ortega, *Academic Search Engines: a Quantitative Outlook*, 1st ed. Amsterdam: Elsevier, 2014.

[2] "Github - atlarge-research/aip: An instrument to combine, unify, and correct (scientific) article meta-data," (Accessed on: Apr. 22, 2021). [Online]. Available: https://github.com/atlarge-research/AIP

[3] E. Landhuis, "Scientific literature: Information overload," *Nature*, vol. 535, no. 7612, pp. 457–458, 2016.

[4] G. Panagopoulos, G. Tsatsaronis, and I. Varlamis, "Detecting rising stars in dynamic collaborative networks," *Journal of Informetrics*, vol. 11, no. 1, pp. 198–222, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1751157716300645

[5] L. Versluis and A. Iosup, "A survey and annotated bibliography of workflow scheduling in computing infrastructures: Community, keyword, and article reviews – extended technical report," 2020, (Accessed on: Apr. 22, 2021). [Online]. Available: https://export.arxiv.org/abs/2004.10077v1

[6] A. Daud, M. Song, M. K. Hayat, T. Amjad, R. Abbasi, H. Dawood, and A. Ghani, "Finding rising stars in bibliometric networks: a survey," *Scientometrics*, vol. 124, pp. 633–661, Apr. 2020.

[7] I. Alexander and S. Robertson, "Understanding project sociology by modeling stakeholders," *Software, IEEE*, vol. 21, pp. 23 – 27, Feb. 2004.

[8] I. Sommerville, *Software Engineering*, 9th ed. USA: Addison-Wesley Publishing Company, 2010.

[9] A. Cline, *Agile Development in the Real World.* Berkeley, CA: Apress, 2015.

[10] M. Keeling, *Design It!: From Programmer to Software Architect.* Pragmatic Bookshelf, 2017.

[11] J. Clark, "Top 10 backend frameworks," (Accessed on: Apr. 22, 2021). [Online]. Available: https://blog.back4app.com/backend-frameworks

[12] W3Techs, "Market share trends for javascript libraries," (Accessed on: Apr. 22, 2021). [Online]. Available: https://w3techs.com/technologies/history_overview/javascript_library

[13] S. Juba and A. Volkov, *Learning PostgreSQL 11: a beginner's guide to building high-performance PostgreSQL database solutions.* Packt Publishing Ltd, 2019.

[14] D. Fernandes and J. Bernardino, "Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb." in *DATA*, 2018, pp. 373–380.

[15] S. Batra and C. Tyagi, "Comparative analysis of relational and graph databases," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 2, no. 2, pp. 509–512, 2012.

[16] X.-L. Li, C. S. Foo, K. L. Tew, and S.-K. Ng, "Searching for rising stars in bibliography networks," in *Database Systems for Advanced Applications*, X. Zhou, H. Yokota, K. Deng, and Q. Liu, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 288–292.

[17] G. Panagopoulos, G. Tsatsaronis, and I. Varlamis, "Detecting rising stars in dynamic collaborative networks," *Journal of Informetrics*, vol. 11, no. 1, pp. 198–222, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1751157716300645

[18] J. Zhang, Z. Ning, X. Bai, W. Wang, S. Yu, and F. Xia, "Who are the rising stars in academia?" in *2016 IEEE/ACM Joint Conference on Digital Libraries (JCDL)*, 2016, pp. 211–212.

[19] A. Daud, R. Abbasi, and F. Muhammad, "Finding rising stars in social networks," in *Database Systems for Advanced Applications*, W. Meng, L. Feng, S. Bressan, W. Winiwarter, and W. Song, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 13–24.

[20] J. Zhang, F. Xia, W. Wang, X. Bai, S. Yu, T. M. Bekele, and Z. Peng, "Cocarank: A collaboration caliber-based method for finding academic rising stars," in *Proceedings of the 25th International Conference Companion on World Wide Web*, ser. WWW '16 Companion. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2016, p. 395–400. [Online]. Available: https://doi.org/10.1145/2872518.2890524

[21] A. Daud, F. Abbas, T. Amjad, A. A. Alshdadi, and J. S. Alowibdi, "Finding rising stars through hot topics detection," *Future Generation Computer Systems*, vol. 115, pp. 798–813, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X20329903

[22] A. Daud, M. Ahmad, M. S. I. Malik, and D. Che, "Using machine learning techniques for rising star prediction in co-author network," *Scientometrics*, vol. 102, no. 2, pp. 1687–1711, Feb. 2015.

[23] J. T. da Silva and J. Dobránszki, "Should the hardy–littlewood axioms of collaboration be used for collaborative authorship," *Asian Australas. J. Plant Sci. Biotechnol*, vol. 7, pp. 72–75, 2013.

[24] C. Lee, O. Kwon, M. Kim, and D. Kwon, "Early identification of emerging technologies: A machine learning approach using multiple patent indicators," *Technological Forecasting and Social Change*, vol. 127, pp. 291–303, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0040162517304778

[25] H. Small, K. W. Boyack, and R. Klavans, "Identifying emerging topics in science and technology," *Research Policy*, vol. 43, no. 8, pp. 1450–1467, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0048733314000298

[26] Q. Wang, "A bibliometric model for identifying emerging research topics," *Journal of the Association for Information Science and Technology*, vol. 69, 07 2017.

[27] J. Pujol, V. Erramilli, and P. Rodriguez, "Divide and conquer: Partitioning online social networks," 05 2009.

[28] T. Raeder and N. Chawla, "Market basket analysis with networks," *Social Netw. Analys. Mining*, vol. 1, pp. 97–113, 04 2011.

[29] V. Traag, L. Waltman, and N. J. van Eck, "From louvain to leiden: guaranteeing well-connected communities," *Scientific Reports*, vol. 9, p. 5233, 03 2019.

[30] D. Rotolo, D. Hicks, and B. Martin, "What is an emerging technology?" *Research Policy*, vol. 44, pp. 1827–1843, 12 2015.

[31] J. E. Ramos, "Using tf-idf to determine word relevance in document queries," 2003.

[32] E. van Eyk, L. Toader, S. Talluri, L. Versluis, A. Uță, and A. Iosup, "Serverless is more: From paas to present cloud computing," *IEEE Internet Computing*, vol. 22, no. 5, pp. 8–17, 2018.

[33] P. Koopman, "How to write an abstract," (Accessed on: June 16, 2021). [Online]. Available: https://users.ece.cmu.edu/~koopman/essays/abstract.html

[34] B. Friedman and H. Nissenbaum, "Discerning bias in computer systems," in *INTERACT '93 and CHI '93 Conference Companion on Human Factors in Computing Systems*, ser. CHI '93. New York, NY, USA: Association for Computing Machinery, 1993, p. 141–142. [Online]. Available: https://doi.org/10.1145/259964.260152

[35] H. Ledford, "Millions of black people affected by racial bias in health-care algorithms," Oct. 2019, (Accessed on: May 29, 2021). [Online]. Available: https://www.nature.com/articles/d41586-019-03228-6

[36] J. Dastin, "Amazon scraps secret ai recruiting tool that showed bias against women," Oct. 2018, (Accessed on: May 29, 2021). [Online]. Available: https://www.reuters.com/article/us-amazon-com-jobs-automation-insight-idUSKCN1MK08G

[37] B. Friedman and H. Nissenbaum, "Bias in computer systems," *ACM Trans. Inf. Syst.*, vol. 14, no. 3, p. 330–347, Jul. 1996. [Online]. Available: https://doi.org/10.1145/230538.230561

[38] N. J. van Eck and L. Waltman, *Visualizing Bibliometric Networks*. Cham: Springer International Publishing, 2014, pp. 285–320.

A Project Schedule

A.1 Initial Project Schedule

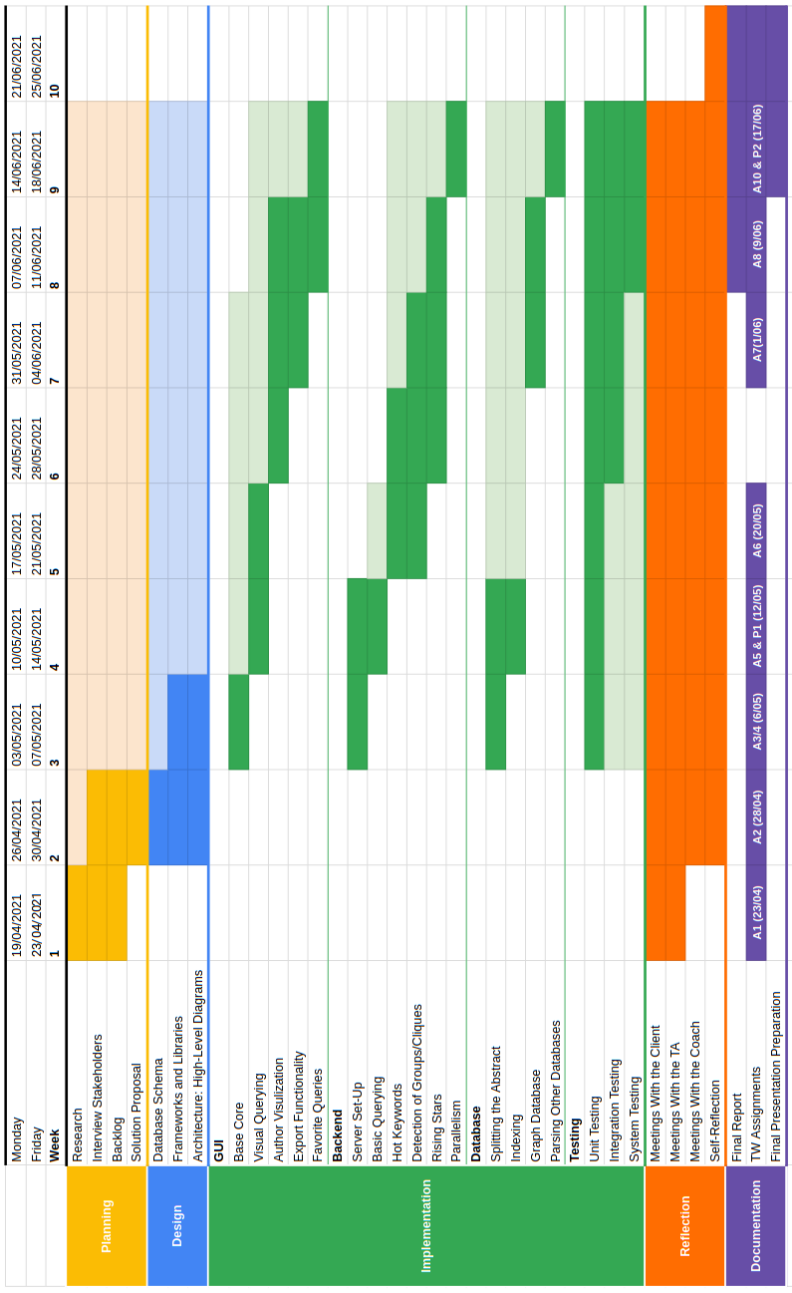

Figure A.1: A Gantt chart for the schedule of AIP++, as planned at the start of the project. Faded colors mark the fact that the implementation of a feature may extend longer than initially estimated. Weeks correspond to sprints.

A.2 Final Project Schedule

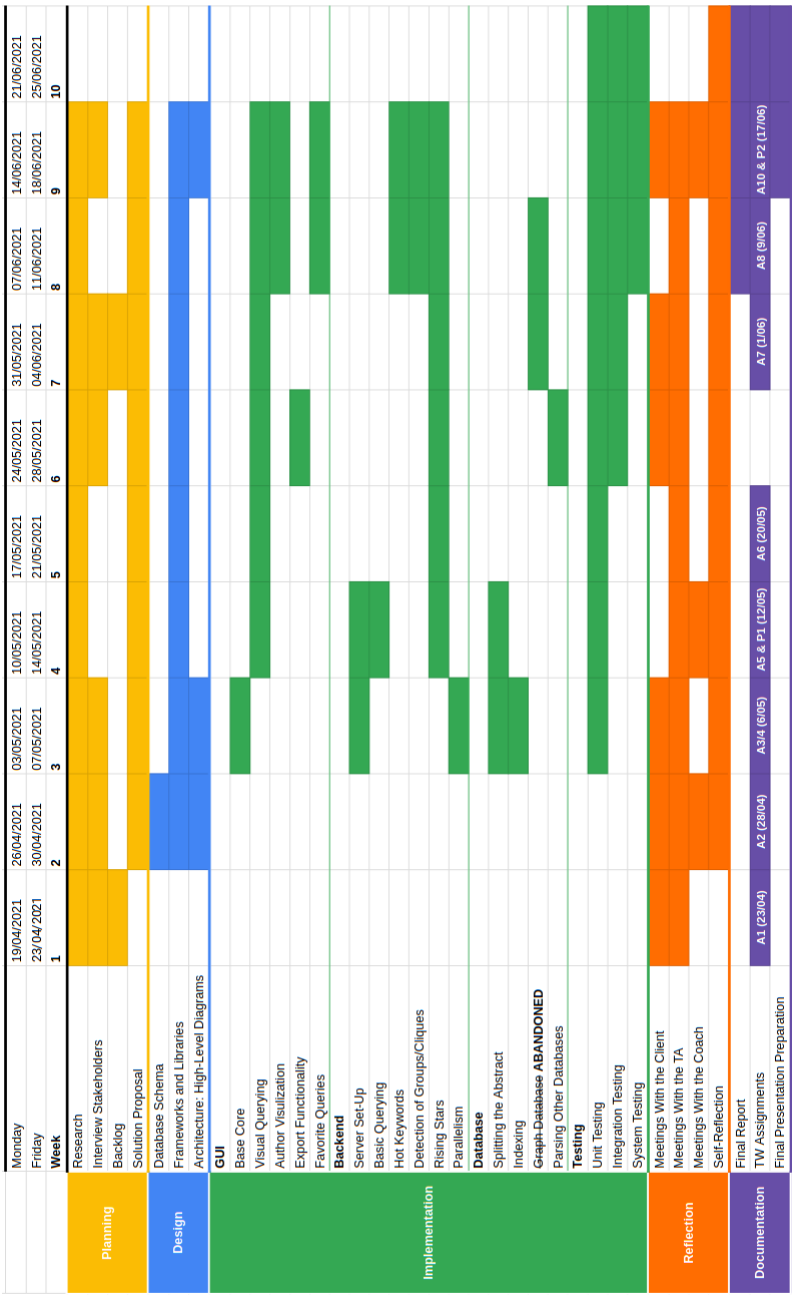

Figure A.2: A Gantt chart for the final schedule of AIP++, as modified in week 9 of the project. Weeks correspond to sprints.

B The Graphical User Interface

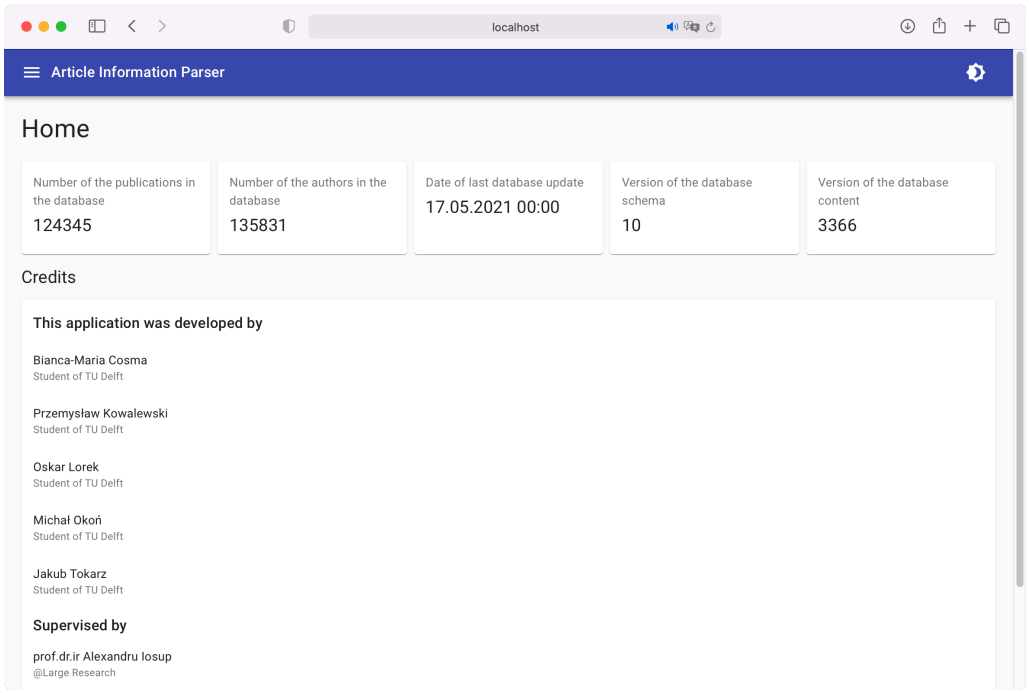

Figure B.1: A screenshot of the home page of AIP++, taken on 17.06.2021.

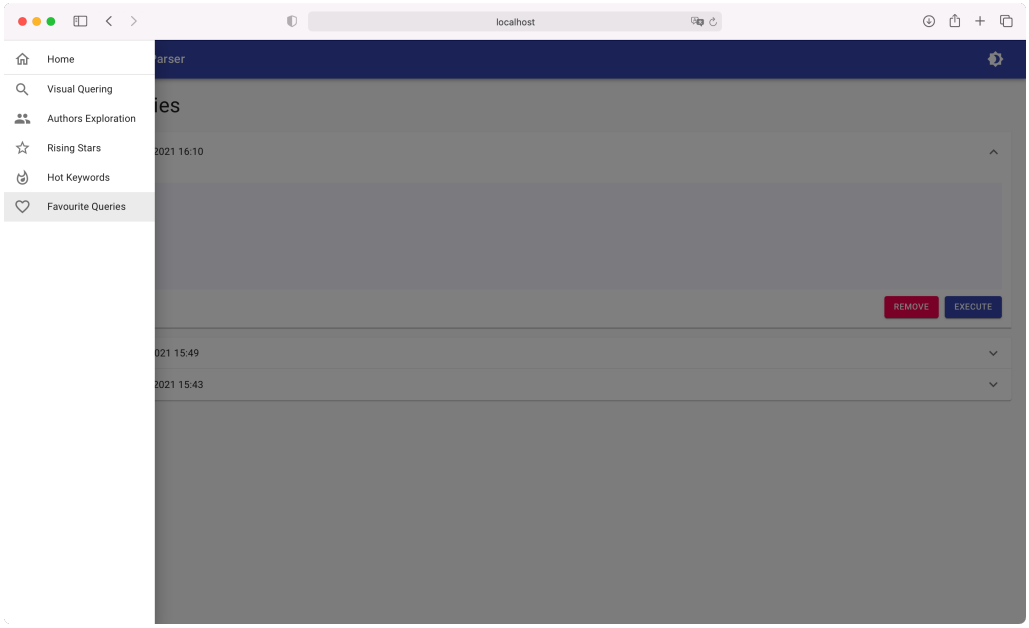

Figure B.2: A screenshot of the expanded application menu of AIP++, taken on 17.06.2021.

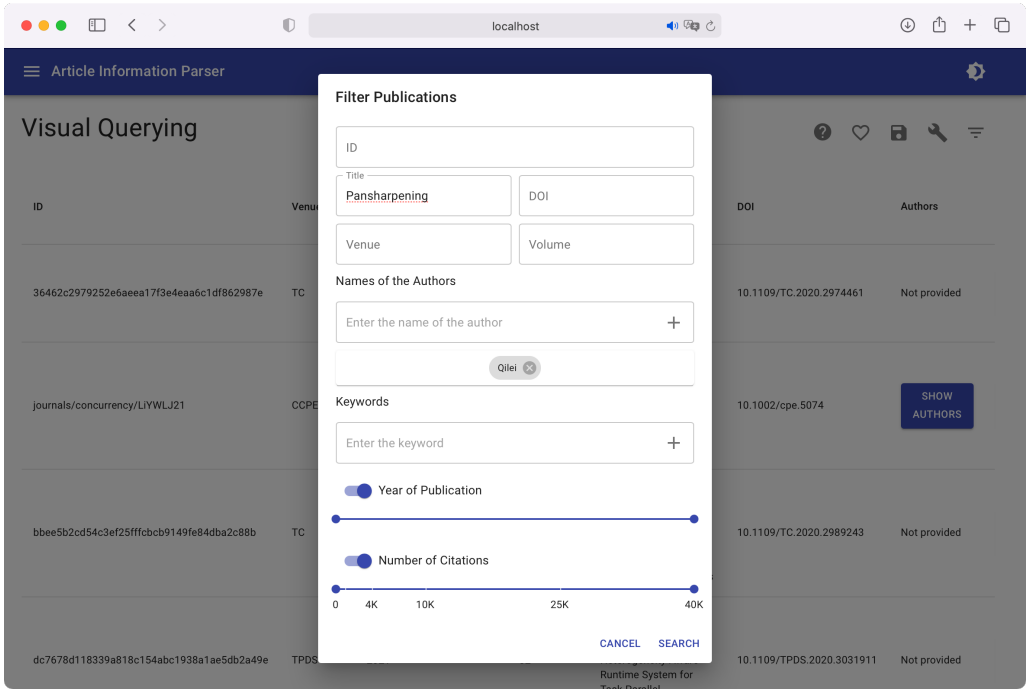

Figure B.3: A screenshot showing the GUI filters that can be used to query the AIP++ database, taken on 17.06.2021.

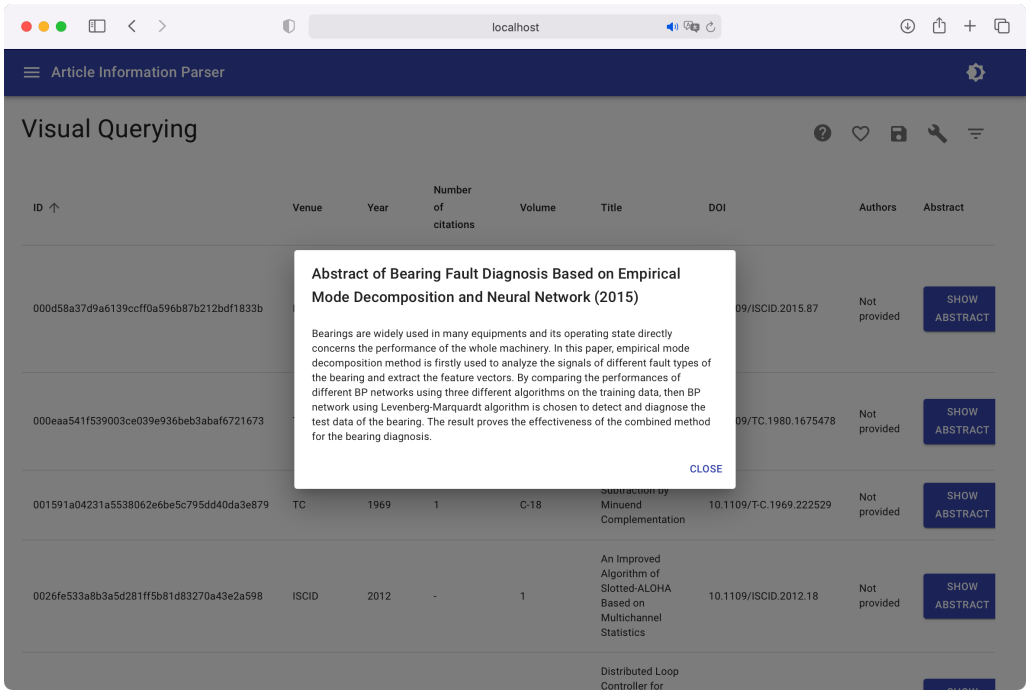

Figure B.4: A screenshot of the abstract of a publication in the AIP++ database, taken on 17.06.2021.

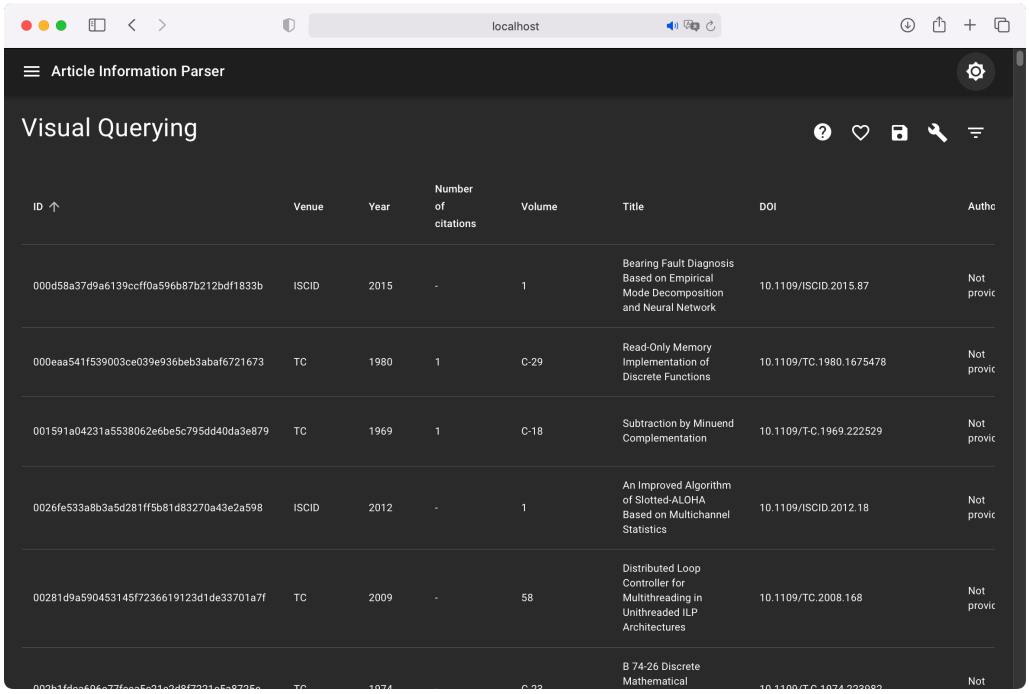

Figure B.5: A screenshot of the first page in the list of publications in the AIP++ database, taken on 17.06.2021. The screenshot was taken when the user interface was displayed in dark mode.

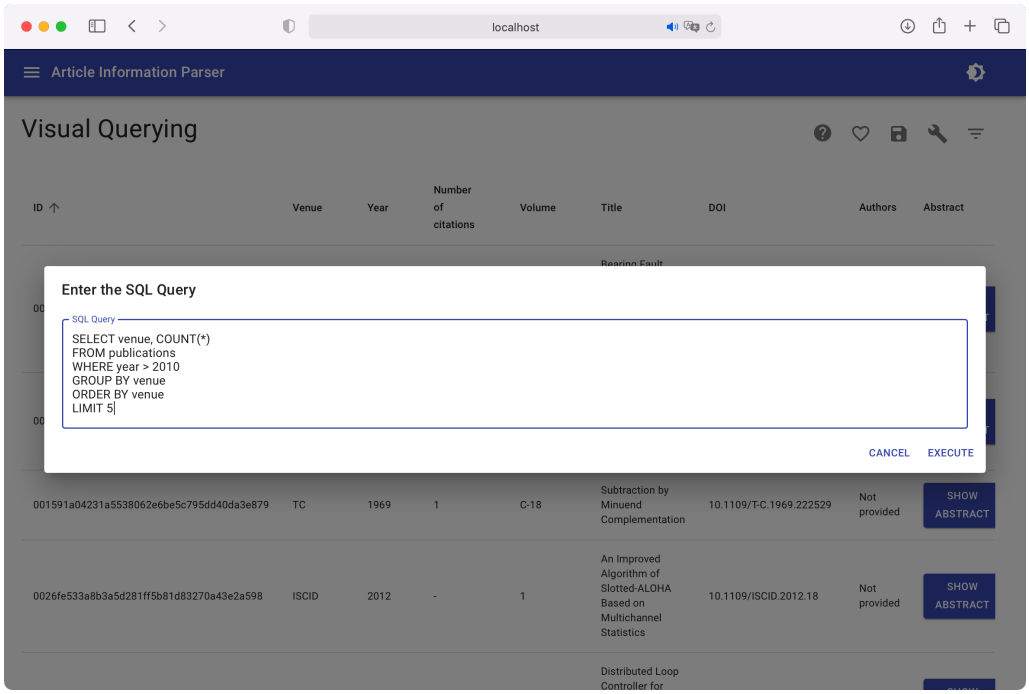

Figure B.6: A screenshot of the custom SQL query functionality of AIP++, taken on 17.06.2021.

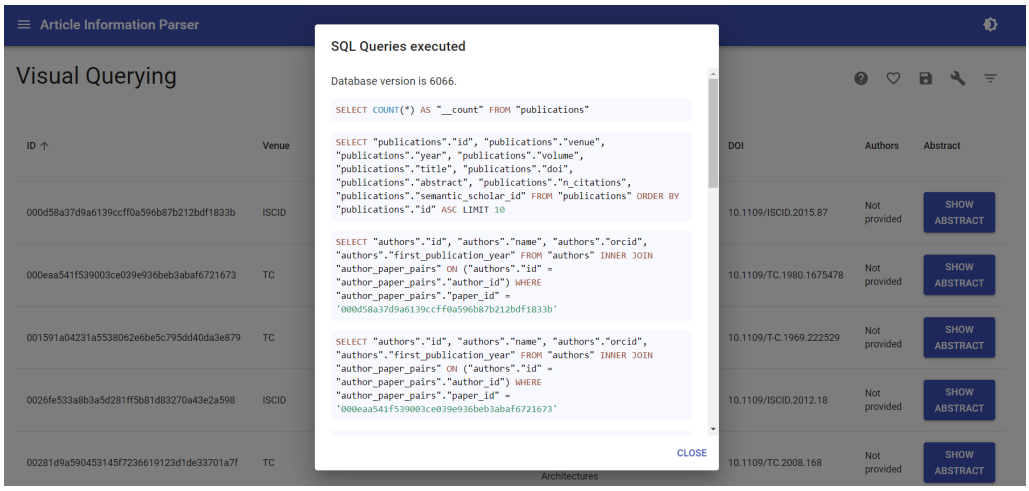

Figure B.7: A screenshot of the SQL queries executed internally for visual querying, taken on 17.06.2021.

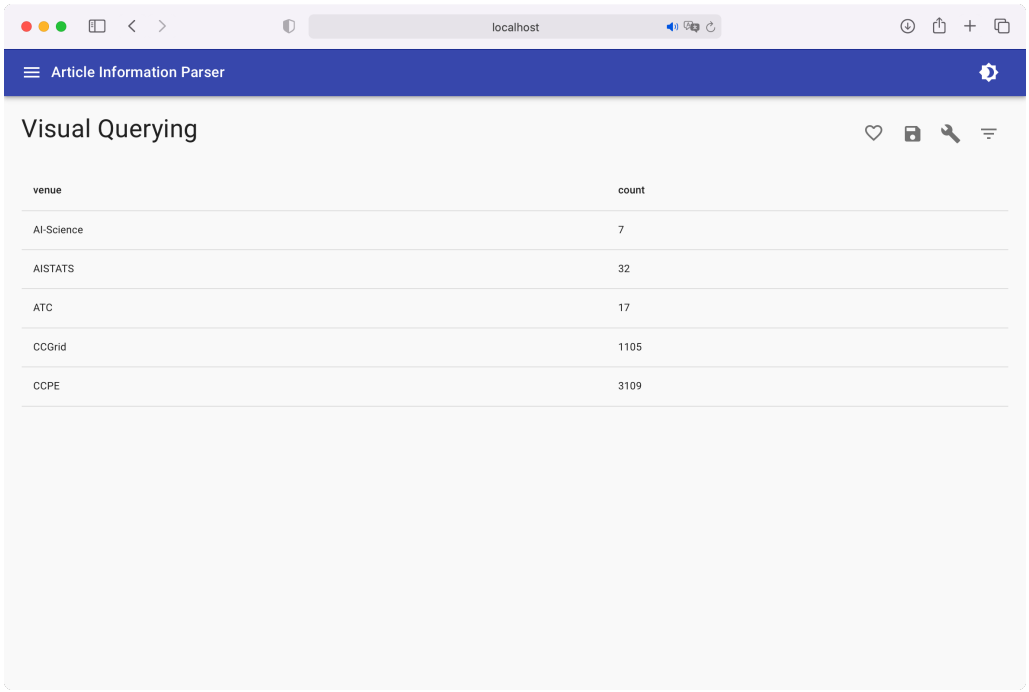

Figure B.8: A screenshot of the results of a custom SQL query on the AIP++ database, taken on 17.06.2021.

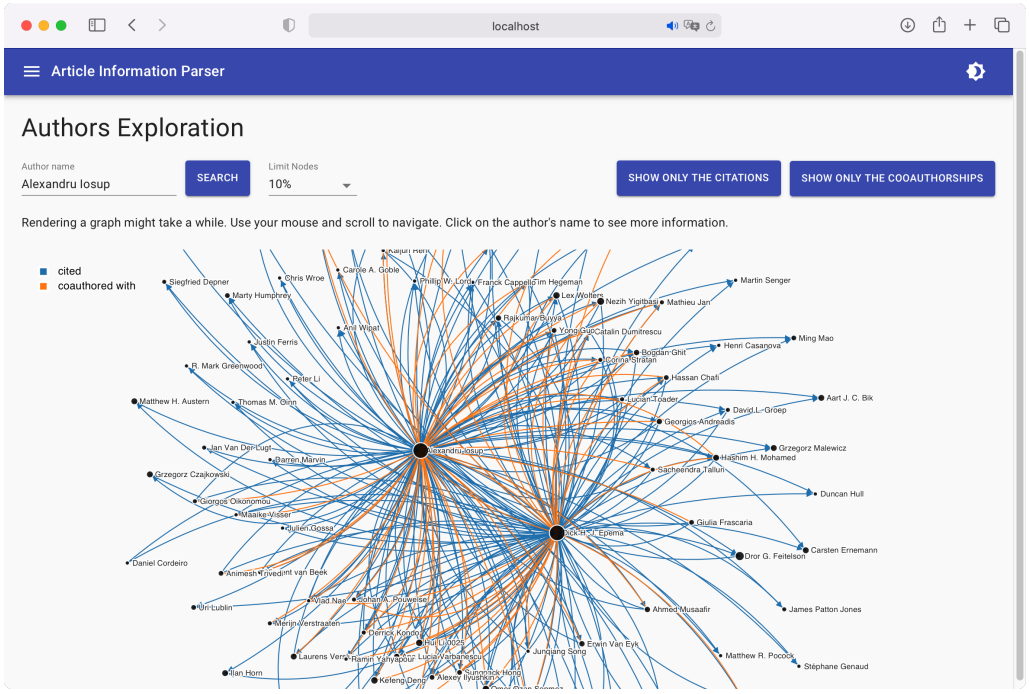

Figure B.9: A screenshot of the author exploration functionality in AIP++, taken on 17.06.2021.

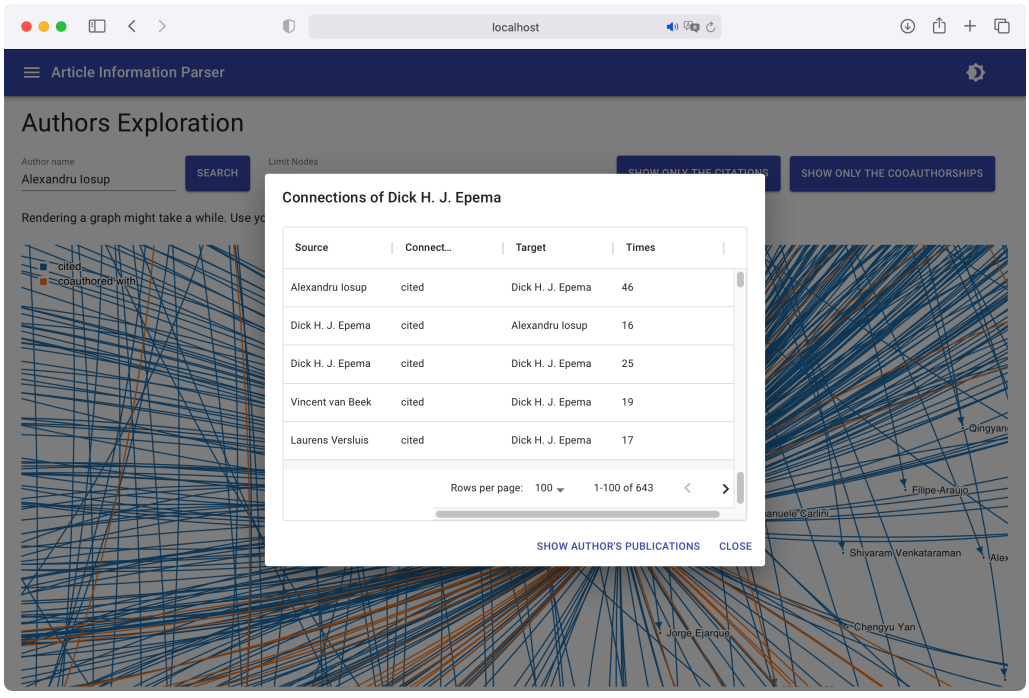

Figure B.10: A screenshot showing more details about an author, in the author exploration tab in AIP++, taken on 17.06.2021.

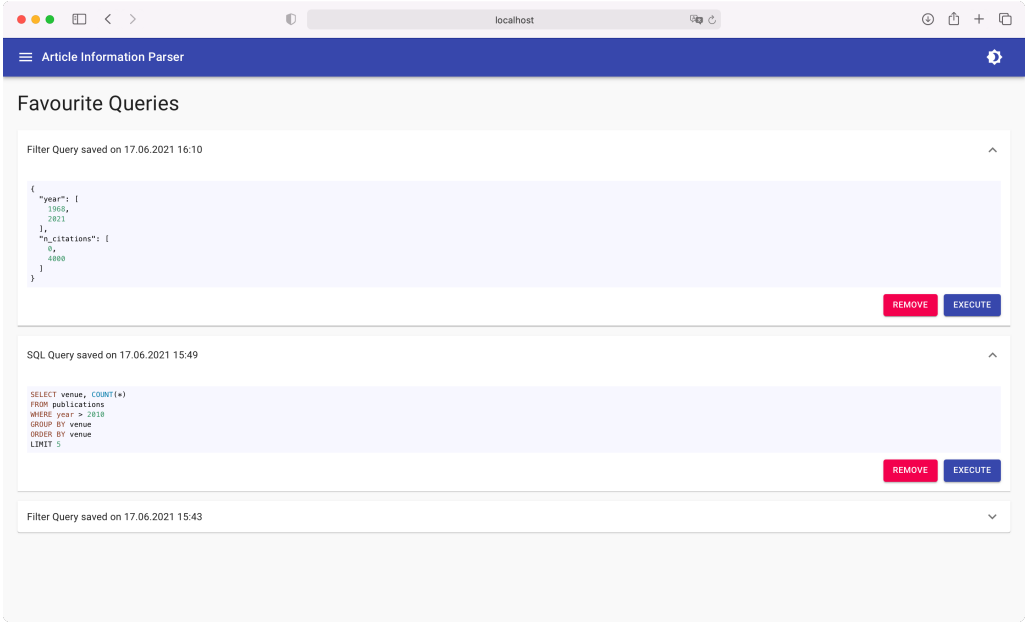

Figure B.11: A screenshot of the favorite queries functionality in AIP++, taken on 17.06.2021.

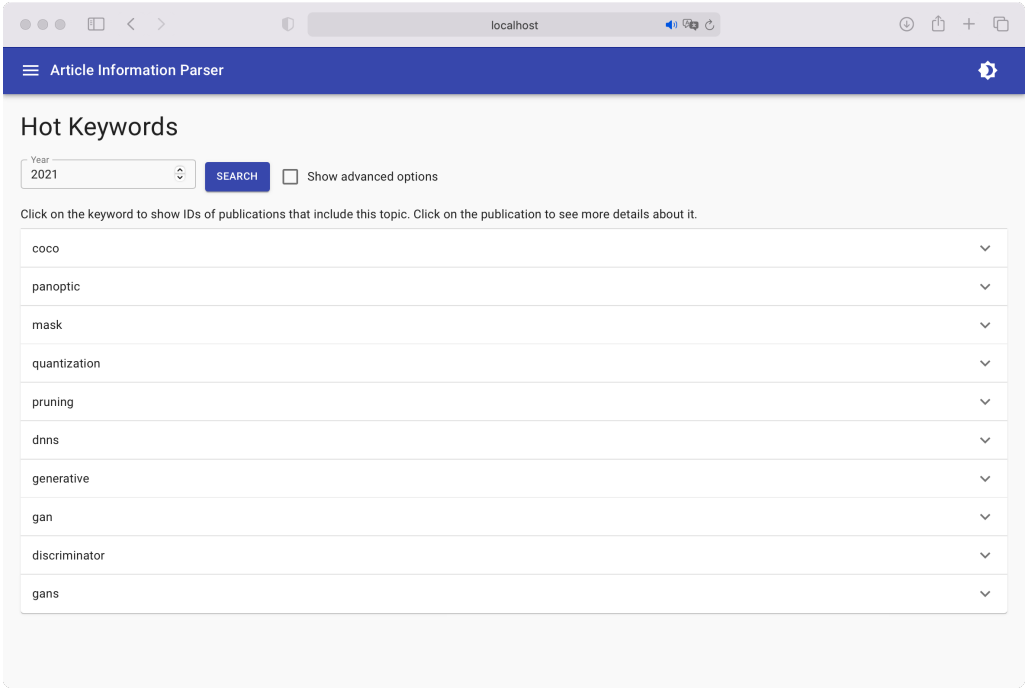

Figure B.12: A screenshot of the hot/emerging keywords functionality in AIP++, taken on 17.06.2021.

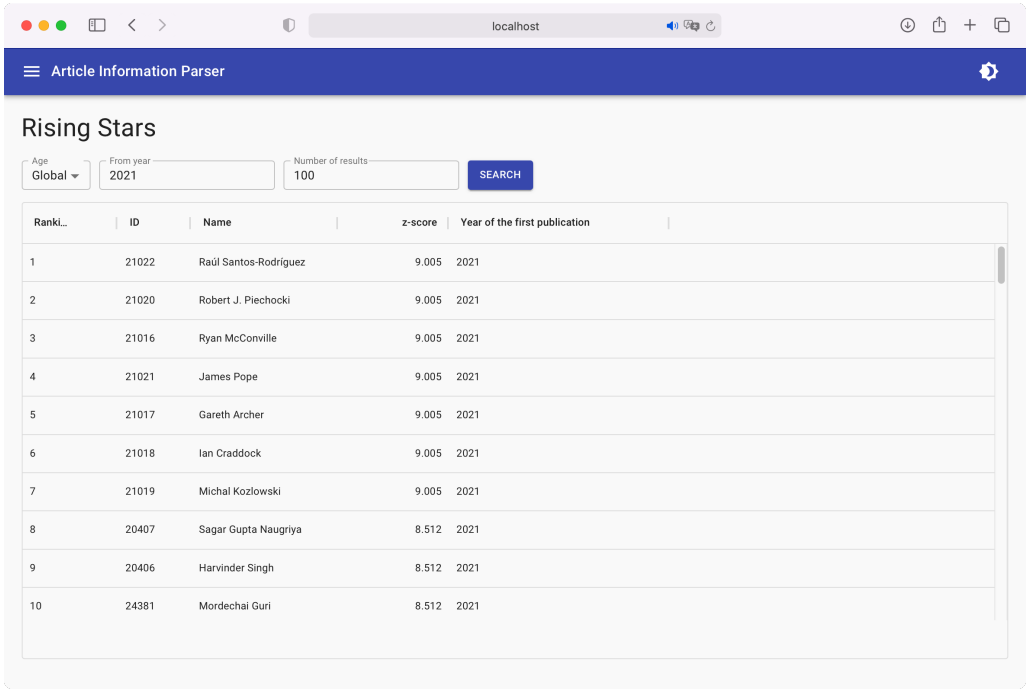

Figure B.13: A screenshot of the rising stars functionality in AIP++, taken on 17.06.2021.

C Queries Used for Performance Comparisons

C.1 Queries That Highlight Keyword Search Optimizations

Table C.1: The exact queries that were executed on the AIP++ PostgreSQL database before and after keyword search optimizations were implemented. These optimizations are explained in section 7.1. It is important to note the relation between this table and figure 7.1. The queries all have a number, specified on the horizontal axis of the figure, and in the first column of this table. There are two queries for each number: a query that was run before the optimizations (colored blue in the figure), and a query that was run after the optimizations (colored red in the figure).

Queries Executed on the AIP++ PostgreSQL Database, Before and After Keyword Search Optimizations		
No.	Before Optimizations	After Optimizations
1	EXPLAIN ANALYZE SELECT COUNT(*) FROM publications WHERE LOWER(abstract) LIKE '%distributed%';	EXPLAIN ANALYZE SELECT COUNT(*) FROM publications JOIN paper_word_pairs pwp ON publications.id = pwp.paper_id JOIN words w ON pwp.word_id = w.id WHERE word = 'distributed';
2	EXPLAIN ANALYZE SELECT publications.id FROM publications WHERE LOWER(abstract) LIKE '%distributed%' AND (LOWER(abstract) LIKE '%system%' OR LOWER(abstract) LIKE '%systems%');	EXPLAIN ANALYZE SELECT publications.id FROM publications JOIN paper_word_pairs pwp1 ON publications.id = pwp1.paper_id JOIN paper_word_pairs pwp2 ON publications.id = pwp2.paper_id WHERE pwp1.word_id = 'distributed' AND (pwp2.word_id = 'system' OR pwp2.word_id = 'systems');
3	EXPLAIN ANALYZE SELECT id AS paper_id, word AS word_id, COUNT(*) AS cnt FROM (SELECT id, regexp_split_to_table(REGEXP_REPLACE(LOWER(abstract), '[^a-z|\\-|\\/]', '', 'g'), '[|\\-|\\/]') AS word FROM publications) AS a WHERE word = 'data' GROUP BY paper_id, word_id ORDER BY cnt DESC;	EXPLAIN ANALYZE SELECT * FROM paper_word_pairs WHERE word_id = 'data' ORDER BY cnt DESC;

C.2 Queries for Benchmarking of the Graph and Relational Databases

Table C.2: The exact queries that were run during the benchmarking of the Neo4J and PostgreSQL databases, described in section 7.2. It is important to note the relation between this table and figure 7.2, where Neo4J queries are colored red, and PostgreSQL queries are colored blue. Two queries were executed on each of the two databases: one which retrieves citation-based connections between authors, and one that lists pairs of co-authors. Their results were summed up and reported for three different authors. Note that any author name can be filled in instead of *<author name>* in the queries below.

Queries Executed for Benchmarking of the PostgreSQL and Neo4J Databases

PostgreSQL (SQL Queries)

```
SELECT a1.name, a2.name, COUNT(*)
FROM author_paper_pairs app1
JOIN authors a1 ON app1.author_id = a1.id
JOIN publications p1 ON app1.paper_id = p1.id
JOIN cites c ON p1.id = c.paper_id
JOIN publications p2 ON p2.id = c.cited_paper_id
JOIN author_paper_pairs app2
ON app2.paper_id = p2.id
JOIN authors a2 ON app2.author_id = a2.id
WHERE a1.name = '<author name>'
OR a2.name = '<author name>'
GROUP BY a1.name, a2.name;
```

Neo4J (Cypher Queries)

```
MATCH (a2:Author) - [:PUBLISHES] -> (p2:Publication)
<- [:CITES] - (p1:Publication)
<- [:PUBLISHES] - (a1:Author)
WHERE a1.name = '<author name>'
OR a2.name = '<author name>'
RETURN a1.name, a2.name, count(*) as weight
```

Queries for Co-Author Relationships Between Authors

PostgreSQL (SQL Queries)

```
SELECT a1.name, a2.name, COUNT(*)
FROM author_paper_pairs app1
JOIN authors a1 ON app1.author_id = a1.id
JOIN publications p ON p.id = app1.paper_id
JOIN author_paper_pairs app2
ON p.id = app2.paper_id
JOIN authors a2 ON app2.author_id = a2.id
WHERE app1.author_id != app2.author_id
AND a1.name = '<author name>'
OR a2.name = '<author name>'
GROUP BY a1.name, a2.name
ORDER BY COUNT(*) DESC;
```

Neo4J (Cypher Queries)

```
MATCH (a2:Author) - [:PUBLISHES] ->(:Publication)
<- [:PUBLISHES] - (a1:Author)
WHERE a1.id <>a2.id
AND a1.name = '<author name>'
OR a2.name = '<author name>'
RETURN a1.name, a2.name, count(*) AS weight
```

D Hot Keywords Detection Process and Results

D.1 Keywords Assigned to Citation-Based Clusters

Table D.1: Examples of clusters resulting from using a TF-IDF method on clusters. Documents were built from the strings consisting of every abstract and title of publications in specific clusters. Do notice, that some of them do not have a high bibliometric value and are filtered out later on in the keyword filtering process.

Clusters with Keywords Assigned to them and the Size of each Cluster			
No.	Size	Keywords	Exemplary Publications
1	100	'job' 'scheduling' 'jobs' 'grid' 'parallel' 'resource' 'workload', 'backfilling', 'scheduler', 'time', 'allocation', 'systems', 'performance', 'resources', 'different', 'requests', 'policy', 'utilization', 'multicluster', 'waiting', 'average', 'gang', 'schedulers', 'policies', 'processor'	'Workload Analysis of a Cluster in a Grid Environment', 'Modeling the Latency on Production Grids with Respect to the Execution Context', 'Resource availability-aware advance reservation for parallel jobs with deadlines'
2	96	'attacks', 'ddos', 'security', 'bgp' 'attack', 'network', 'internet', 'traffic', 'traceback' 'ip', 'routing', 'service', 'packet', 'marking', 'dns', 'cloud', 'denial,' 'addresses', 'prefix', 'data', 'networks', 'dos', 'path', 'attackers' '	'Robbing the Bank with a Theorem Prover", "Instant attack stopper in InfiniBand architecture', 'Detection, classification, and analysis of inter-domain traffic with spoofed source IP addresses'
3	69	'face' '3d' 'facial' 'shape', 'image', 'model', 'surface', 'method', 'images', 'pose', 'faces', 'methods', 'depth', 'expression', 'art', 'landmark', 'state', 'reconstruction', 'alignment', 'geometry', 'head', 'data', 'albedo', 'dataset', 'expressions'	'Self-Adaptive Matrix Completion for Heart Rate Estimation from Face Videos under Realistic Conditions', 'Support Vector Machines in face recognition with occlusions', Face alignment using cascade Gaussian process regression trees

D.2 Analysis of Emerging Keywords from the Year 2000

Table D.2: The table presents emerging keywords resulting from running the hot keywords algorithm using different parameters. Meaning of each of the parameter is meticulously explained in section 7.2. As seen below, dt is one of the parameters that affects the found keywords most strongly, as it completely alters the outcome. Smaller values of dt lead to the detection of more recent trends, whereas greater ones find more sustainable tendencies.

Emerging Keywords Extracted Using Different Parameters				
dt	p_{ratio}	c_{ratio}	r_{min}	Emerging Keywords
5	0.01	0.05	5	'attacks', 'secure', 'sensors', 'sensing', 'proxy', 'grids', 'skeleton', 'skeletons', 'environment', 'p2p', 'content', 'heterogeneous', 'mapreduce', 'overlay', 'routers', 'human', 'aqm', 'openmp', '802', '11', 'checkpoint', 'hoc', 'ad', 'mobility', 'proxies', 'lrd', 'reality', 'hand', 'augmented', 'ieee', 'edca'
5	0.005	0.05	5	'attacks', 'proxy', 'grids', 'skeleton', 'skeletons', 'p2p', 'heterogeneous', 'mapreduce', 'aqm', '802', '11', 'checkpoint', 'proxies', 'lrd', 'ieee', 'edca'
5	0.01	0.1	5	'sensors', 'sensing', 'skeleton', 'skeletons', 'environment', 'p2p', 'content', 'heterogeneous', 'overlay', 'aqm', 'openmp', '802', '11', 'proxies', 'lrd', 'reality', 'hand', 'augmented', 'ieee', 'edca'
5	0.01	0.05	7	'attacks', 'sensors', 'sensing', 'proxy', 'grids', 'skeleton', 'skeletons', 'environment', 'content', 'heterogeneous', 'mapreduce', 'overlay', 'aqm', 'openmp', '802', 'hoc', 'mobility', 'proxies', 'reality', 'hand', 'augmented', 'ieee', 'edca'
3	0.01	0.02	5	'rdma', 'smr', 'grids', 'rdf', 'overlay', 'social', 'forwarding', 'routers', 'string', 'aqm', 'handoff', 'hoc', 'ad', 'vr', 'trie', 'prefixes', 'rtt', 'subscription', 'toeplitz', 'multicluster', 'multihoming', 'isp'

Graphs presenting popularity of words resulting from performing the algorithm with parameters $dt = 5$, $p_{ratio} = 0.01$, $c_{ratio} = 0.1$ and $r_{min} = 5$. After careful examination of all the extracted emerging keywords, we have decided that this setting results in the most accurate prediction., as seen below. Most of them show rapid growth in the beginning of 21st century. Note, that 3 words containing numbers were not considered.

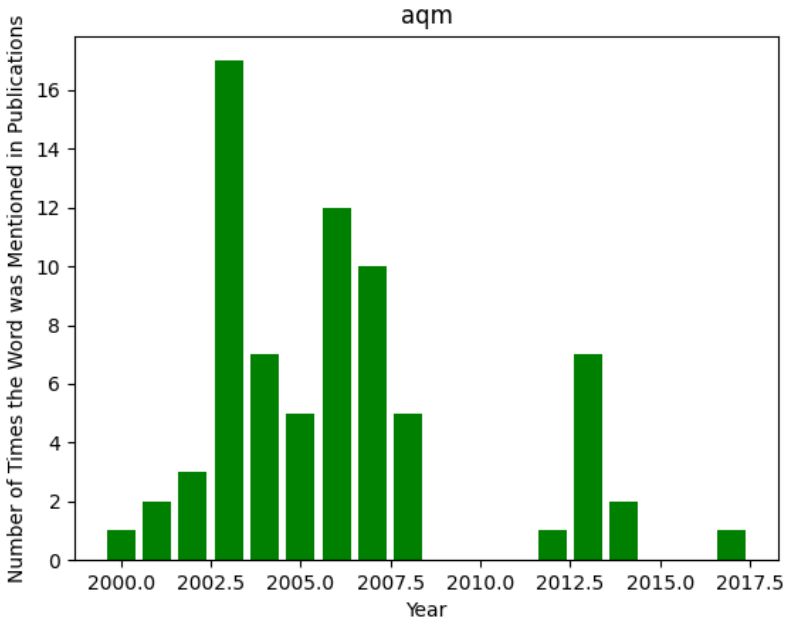

Figure D.1: Number of occurrences of the word 'aqm' over the past years. 'Aqm' stands for 'Active queue management'.

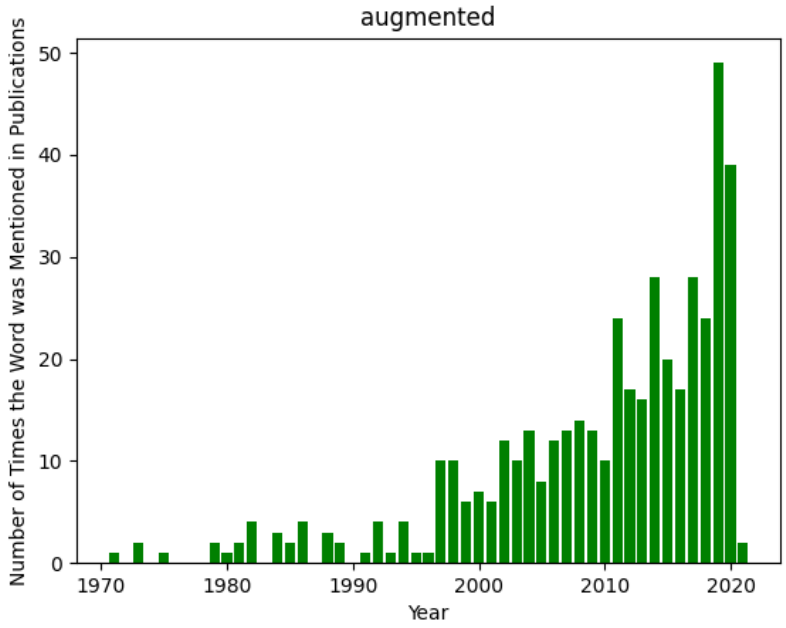

Figure D.2: Number of occurrences of the word 'augmented' over the past years.

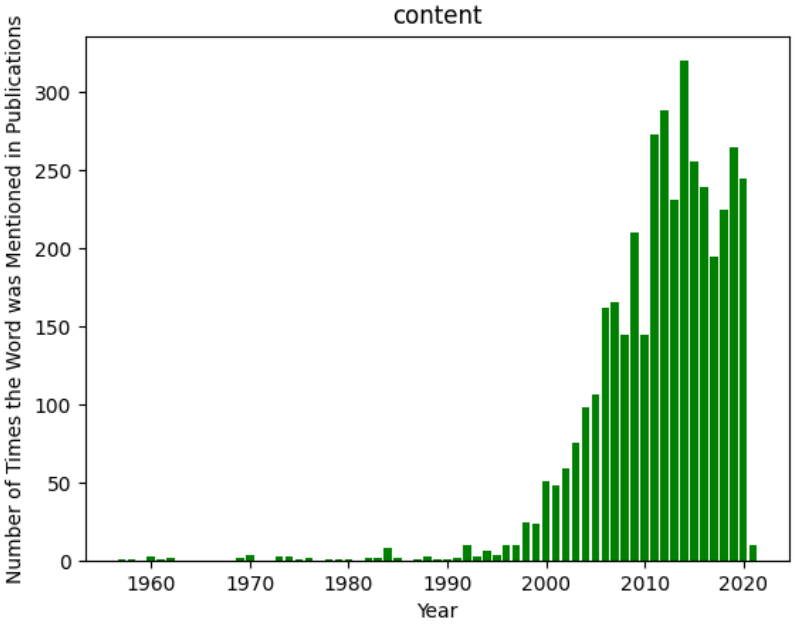

Figure D.3: Number of occurrences of the word 'content' over the past years.

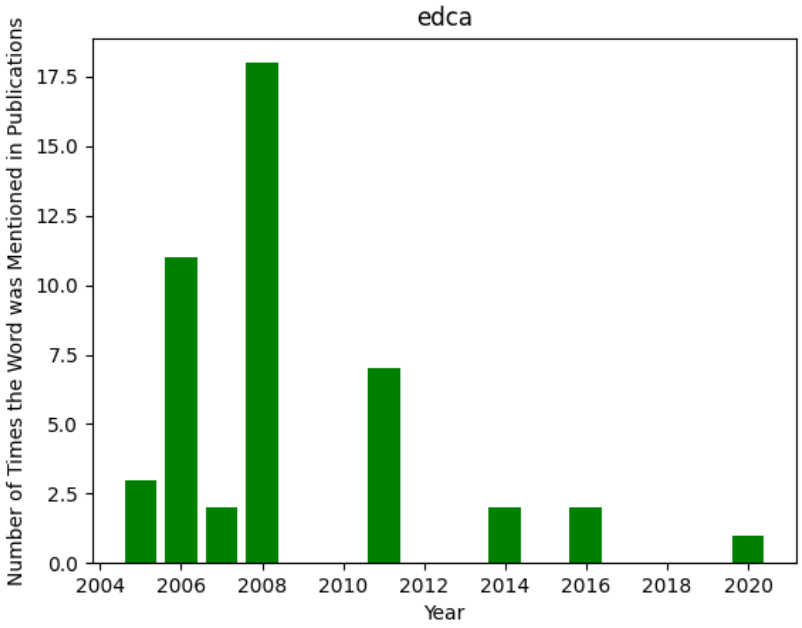

Figure D.4: Number of occurrences of the word 'edca' over the past years. 'Edca' stands for 'Enhanced distributed channel access'.

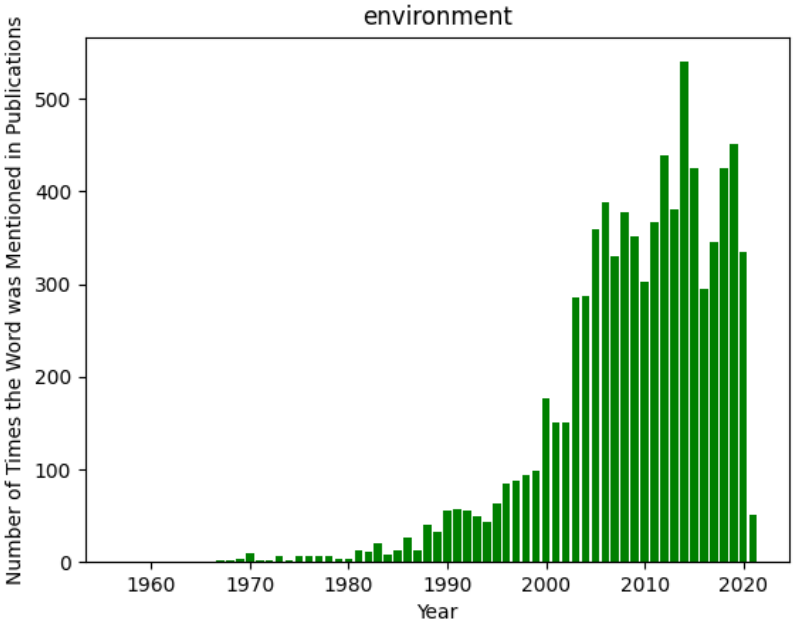

Figure D.5: Number of occurrences of the word 'environment' over the past years.

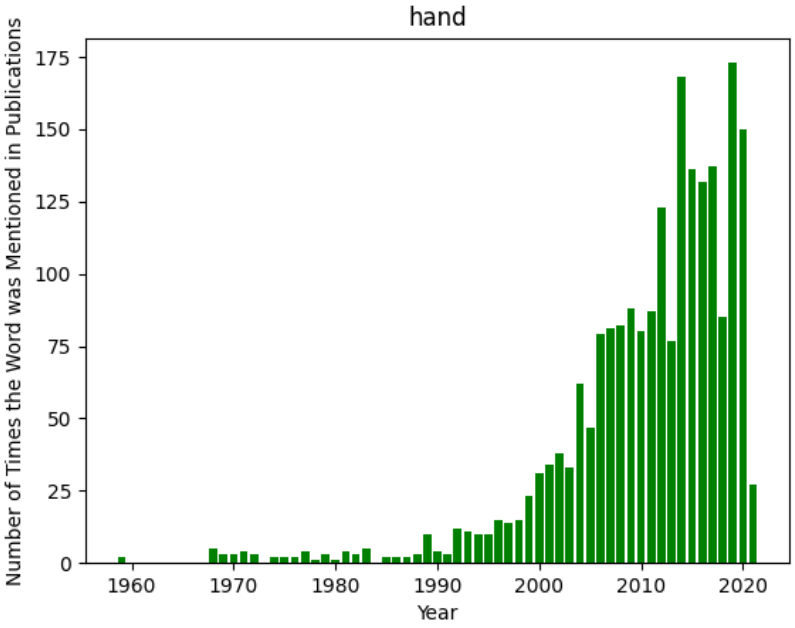

Figure D.6: Number of occurrences of the word 'hand' over the past years.

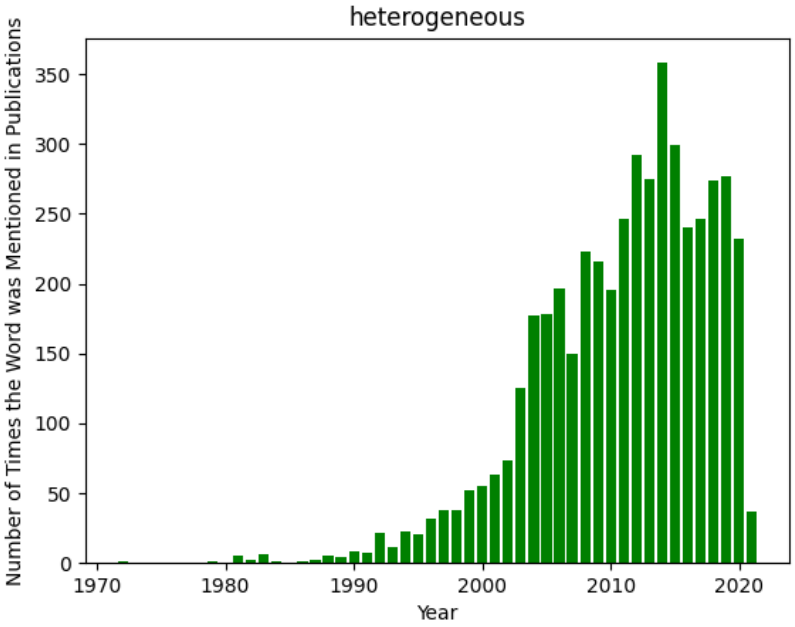

Figure D.7: Number of occurrences of the word 'heterogeneous' over the past years.

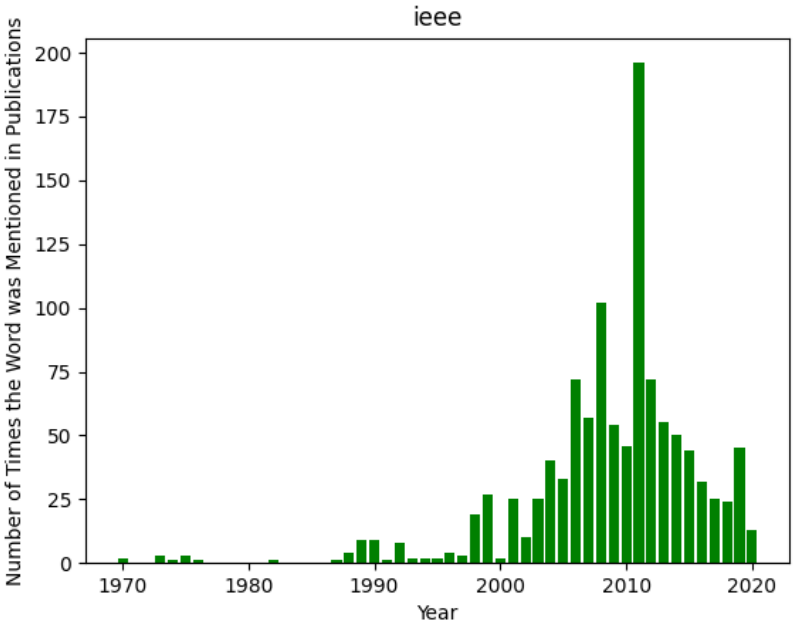

Figure D.8: Number of occurrences of the word 'ieee' over the past years. 'Ieee' stands for 'Institute of Electrical and Electronics Engineers'

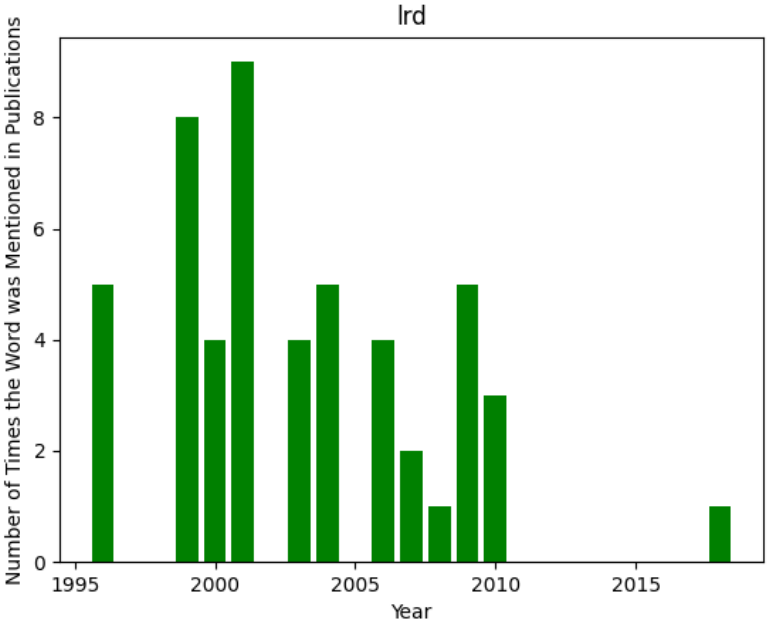

Figure D.9: Number of occurrences of the word 'lrd' over the past years. 'Lrd' may stand for 'Long-range dependence'.

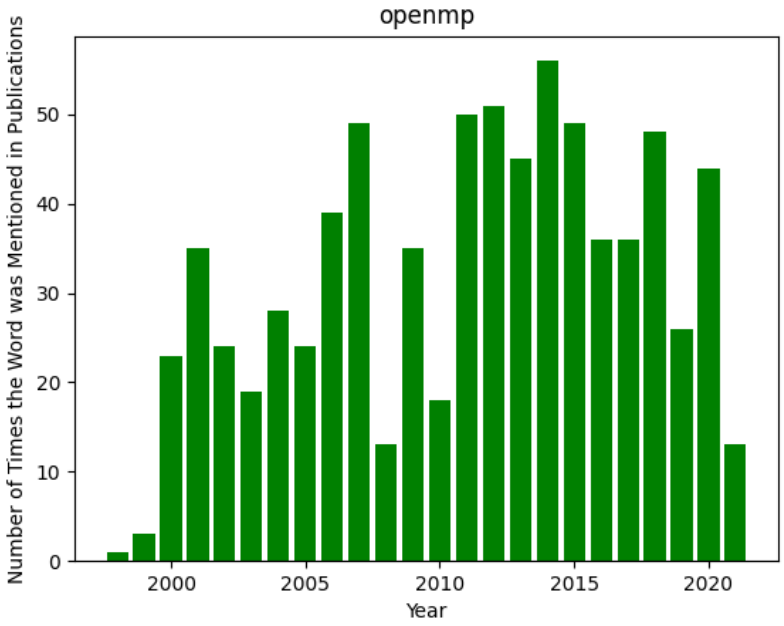

Figure D.10: Number of occurrences of the word 'openmp' over the past years. 'Openmp' may stand for 'Open Multi-Processing'.

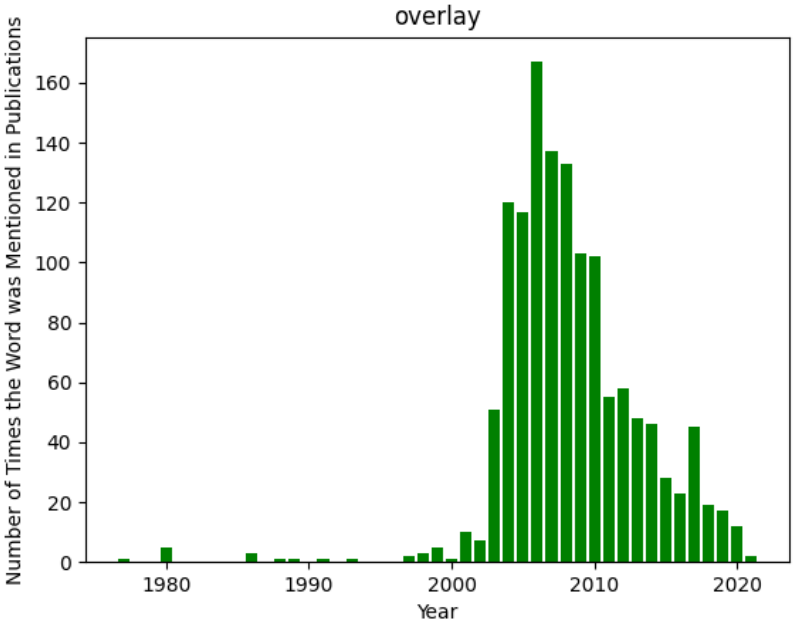

Figure D.11: Number of occurrences of the word 'overlay' over the past years.

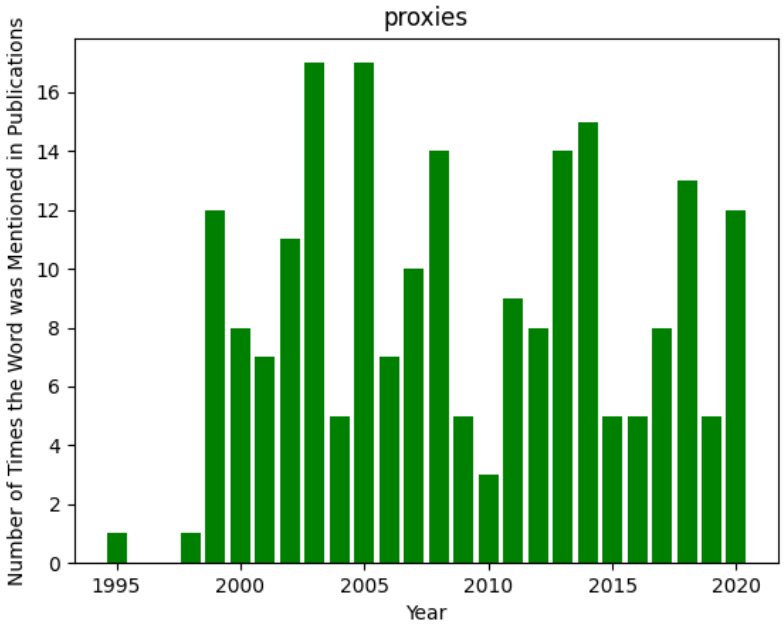

Figure D.12: Number of occurrences of the word 'proxies' over the past years.

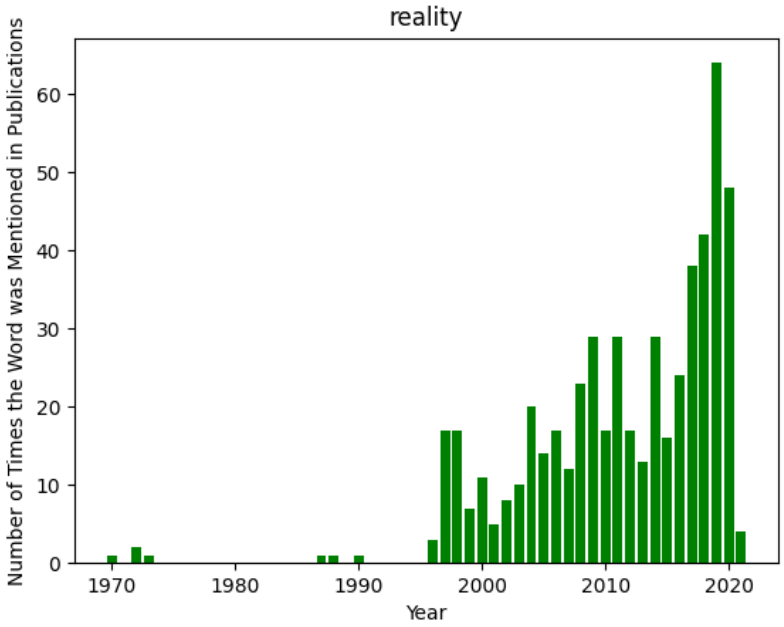

Figure D.13: Number of occurrences of the word 'reality' over the past years.

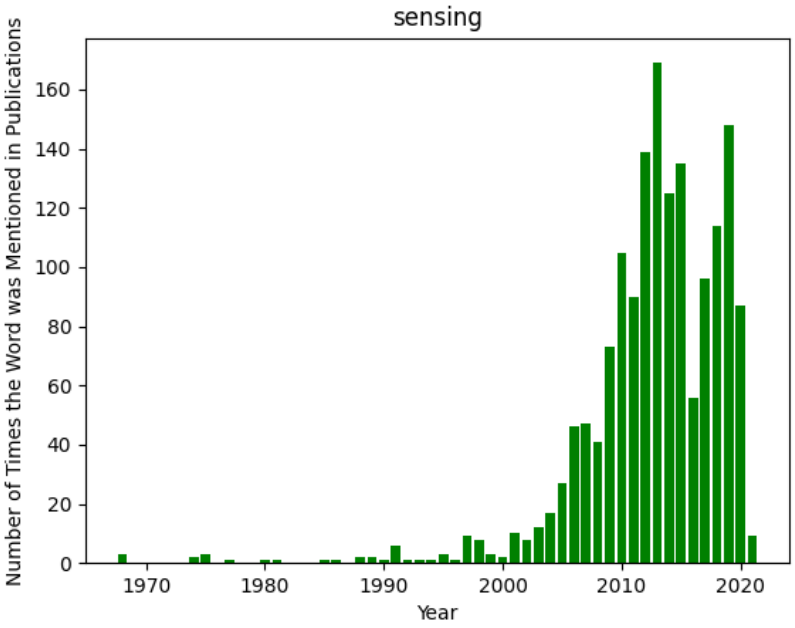

Figure D.14: Number of occurrences of the word 'sensing' over the past years.

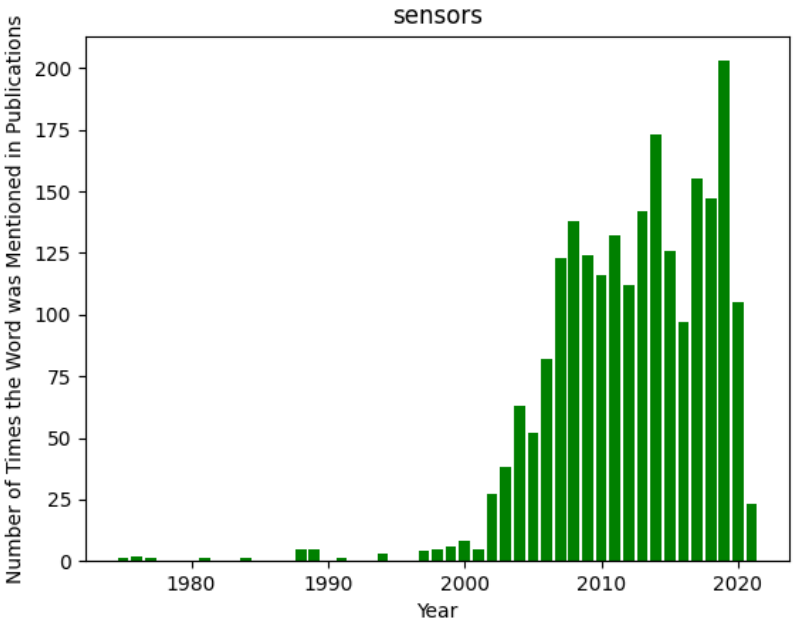

Figure D.15: Number of occurrences of the word 'sensors' over the past years.

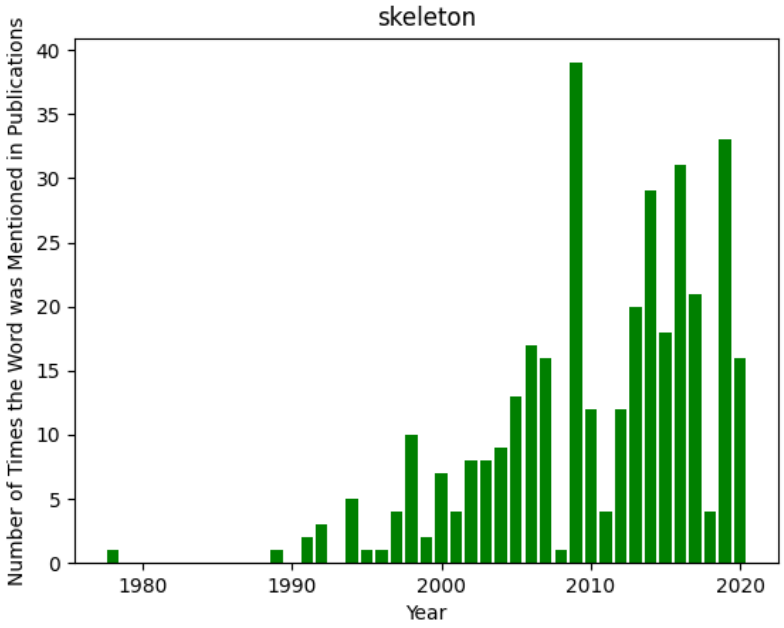

Figure D.16: Number of occurrences of the word 'skeleton' over the past years.

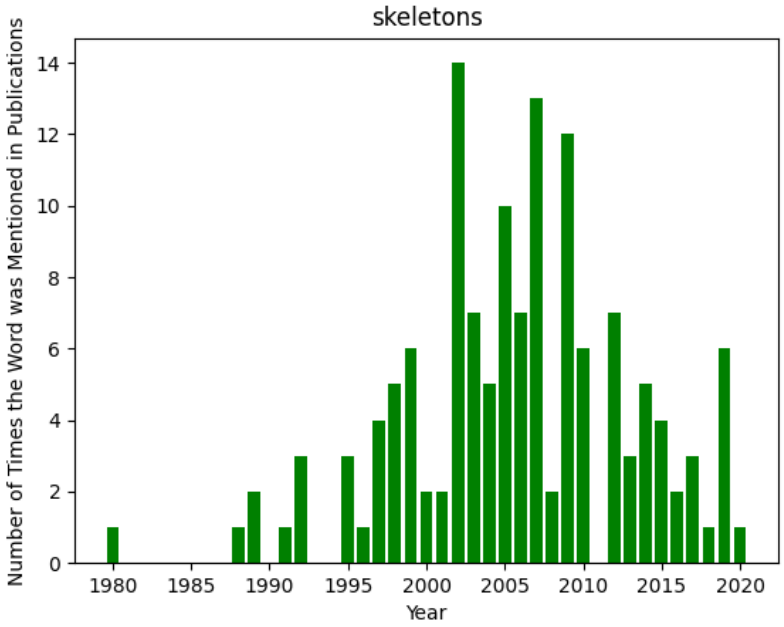

Figure D.17: Number of occurrences of the word 'skeletons' over the past years.

E Contributions

E.1 Individual Contributions to the Project

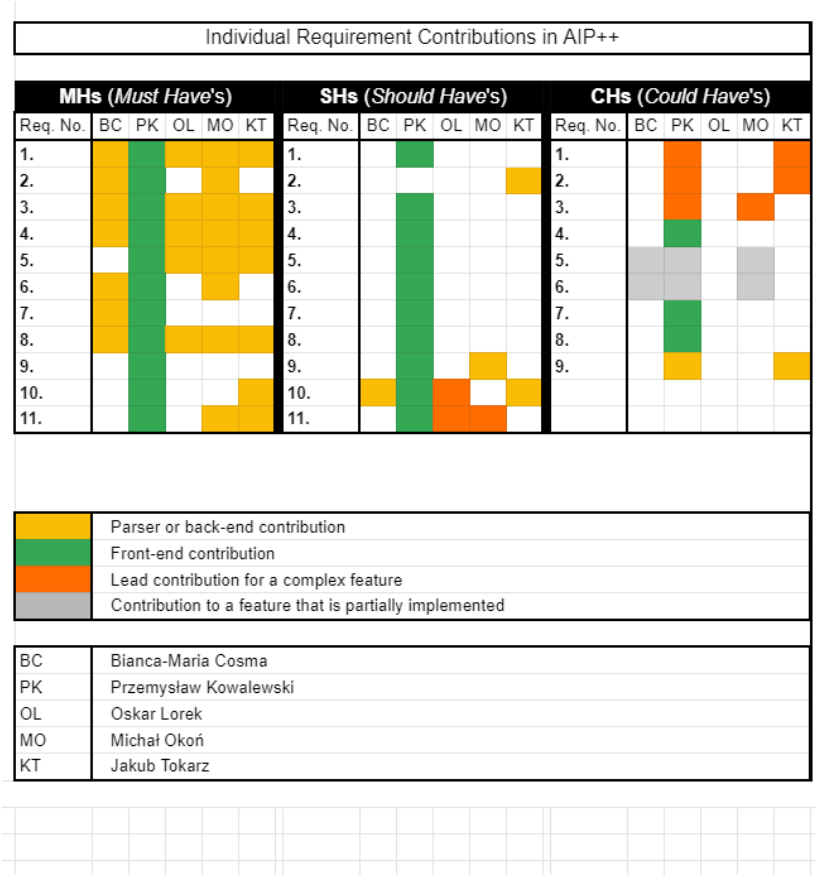

Figure E.1: Individual contributions of the team members to each requirement developed in the final application. It is very important to highlight that this chart does not fully reflect the effort put into this project by each member, as some of the most time-demanding activities are not included (e.g. creating the report and other documentation, research, and working on features that did not come to fruition, such as adding support for the graph database).