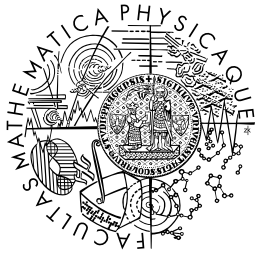

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Jindřich Bär

Social network analysis in academic environment

Department of Software Engineering

Supervisor of the master thesis: prof. RNDr. Tomáš Skopal, Ph.D.
Study programme: Computer Science
Study branch: Software and Data Engineering

Prague 2024

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

I would like to express my gratitude to prof. RNDr. Tomáš Skopal, PhD., for his scholarly leadership and for the time and advice he has given me during the writing of this thesis. I would also like to extend my thanks to my family and friends, who have supported and encouraged me greatly during my studies.

Title: Social network analysis in academic environment

Author: Jindřich Bär

Department: Department of Software Engineering

Supervisor: prof. RNDr. Tomáš Skopal, Ph.D., Department of Software Engineering

Abstract: While university information systems usually have detailed data about courses, publications and lecturers, they rarely mine the relationships between these entities to provide additional value to the users. This thesis aims to improve the Charles Explorer web application by utilizing the synthesized academic social network from the existing relational data. We propose a pipeline for transforming relational data from the university systems into a graph representation of the academic social network. We explore the network and propose various ways to infer missing data using the graph model. Later, we benchmark different re-ranking strategies using social network metrics against existing academic search engines and show that social network-based re-ranking can improve search results ranking. Lastly, we reimplement the tool for visualising the academic social network in the Charles Explorer application for a better user experience.

Keywords: social network, academia, information retrieval, reranking

Contents

Introduction

A **social network** is a theoretical construct describing relations between entities that are usually homogenous in nature. This definition formed over years mostly from social sciences and psychology, where it was used to create an abstraction of the real-world social interactions between people. In those fields, the social networks were used to study - among others - social interactions between pairs of people and their influence on the overall structure of the society.

Later on, the social network theory slowly permeated into other fields of study, including discrete mathematics and computer science. The computing performance of modern computers and the theoretical advances in the field of graph theory allowed for the analysis of large datasets, which in turn allowed for the study of social networks on a larger scale. This led to the creation of the field of *social network analysis*.

While the theory behind social network analysis is already quite mature, its practical applications are still being explored. The most prominent use cases include the analysis of social media platforms, where the social network is formed by the users of the platform and their interactions. Unfortunately, perhaps to keep the competitive advantage, the social media platforms seldom share any details about their social network analysis methods.

Without the access to the user data of commercial social media platforms, we are left with the analysis of social networks that are publicly available. One such example is the social network of academic researchers - individuals - and their publications - which represent their interactions between each other. The realm of academic researchers and collaboration on different publications opens a multitude of opportunities for research - since both the "nodes" (the researchers) and the "edges" (publications) of the social network have interesting data attributes available for them.

For the researchers, this can be e.g. affiliation with parts of the university, their academic title or their role within the university - e.g. are they lecturers, postdoc researchers, or graduate students helping out on one or two publications etc.

For the publications, the data can include the authors, the publication's affiliation with faculties, year of publishing or publication keywords.

This thesis demonstrates practical use of social network theory in the context of academic research and explores the usability of the social network metrics for re-ranking of document retrieval results.

Lately, the term "social network" has been popularized as a synonym to the term **social media**. These are online platforms that allow users to create a profile, share content and interact with other users. In the rest of this thesis, the term "social network" will be used to refer strictly to the theoretical concept or it's concrete representation in the form of a graph, while the term "social media" will be preferred for the online platforms.

Related work

This section talks about the related work in the field of social network analysis, named entity recognition, graph visualization and academic search engines.

Note that more related work is mentioned in the respective chapters of this thesis, especially in cases where we compare our proposed methods with the existing solutions.

Social network analysis and ranking

The social network analysis of research groups has been a topic of interest for various publications. Ordoobadi et al. [2019] and Cimenler et al. [2014] explore the social network of researchers and their publications, and they try to infer the social network structure from the data about the co-authorship of the publications. Later on they try to compare the social network metrics with the academic performance - citation counts - of the researchers.

Haddad and Dridi [2013] explore the use of social relevance data for improving the search result ranking in the context of information retrieval. By using data from the *Umaps Knowtex* dataset (no longer available) they show that the social relevance metrics - like share count, comment count or like count - can be used to improve the ranking of the search results.

Jabeur et al. [2010] introduce a social model for academic literature access systems, where metrics on the social network of researchers, their publications and the users of the system is used to improve the search results ranking. In this paper, the authors explore weighted approach with multiple measures like betweeness, closeness or PageRank.

Note that while this is quite similar to one of the topics of this thesis, the authors of this paper only evaluate the social model on a small dataset of publications in the ACM SIGIR conference. Because of this, the publications are tightly connected in topic and the social network is quite dense, unlike in our case.

Beel et al. [2010] describe inner workings of academic search engines on the example of Google Scholar and discuss search engine optimization in the context of academic search engines. The reviewers of this paper mention the potentially harmful effect of academic search engine optimization (ASEO) when the authors artificially increase the relevance of their publications to the search queries by deliberately overusing certain keywords.

Named entity recognition

The second chapter of this thesis is dedicated to the transformation of the relational data model into the graph data model and inferring the missing identity data from the social network.

Identity inference is a problem that has been extensively explored in the context of *named entity recognition* in the natural language processing. *Named entity recognition* is a subfield of the natural language processing that deals with the identification of named entities in the text. This can prove useful e.g. in the

semantization of plain text, where the named entities are enhanced with links to the actual referenced entities in the knowledge base.

Mansouri et al. [2008] explore different approaches to the named entity recognition. Note that the common ground for all the mentioned approaches is extensive use of NLP methods acquiring the context for each occurence of the named entity in the text. While we briefly explore similar approach (normalizing the entity names to the canonical form) in the second chapter of this thesis, the main focus is on the possibility of inferring the missing identity data from the *social network*.

Graph visualization

The last chapter of this thesis explores the visualization of the academic social network.

Many (e.g. Bennett et al. [2007] or Tominski et al. [2009]) studied the problem of graph visualization and the usability of the different graph layouts, with respect to the usability, aesthetics and the performance of the visualization.

Our use case adds the additional layer of complexity, due to the requirement of deploying the visualization in the web application. While many have proposed own libraries for the graph visualization (e.g. Gretarsson et al. [2010] or Franz et al. [2016]), a recent research by Greif and Burel [2024] suggests that the developer public is largely skewed towards the use of the *D3.js*[1] library for the any visualization (including graph).

Few [2012] lists a large number of different techniques and guidelines regarding data presentation in a visual form, including the graph visualizations. We will refer to this book later on when discussing design decisions for the social network visualization.

Academic search engines

Due to the goals of this thesis, we briefly explore the search engines used for academic results retrieval and their features.

Charles Explorer is an academic open-source search engine developed for the Charles University in Prague. The system indexes the publications, classes, researchers and study programmes affiliated with the university and allows for exploring the entities and relations between them.

As we mention in the next section, this thesis aims to improve the user experience and data quality of the Charles Explorer application by data mining the social network of researchers and their publications.

Google Scholar is a popular academic search engine maintained by Google LLC IPA. According to the documentation[2], the system crawls the Internet for well-formed academic publications and indexes them automatically. Orduna-Malea et al. [2015] estimated the size of the Google Scholar index to be around 99.8

[1] https://d3js.org/

[2] https://scholar.google.com/intl/en/scholar/inclusion.html

million documents in 2014. Later estimates by Gusenbauer [2019] put the size of the Google Scholar index at around 389 million documents in 2019.

While the automated approach to the indexing of the documents allows for the inclusion of a large number of documents, Wijewickrema [2024] claims it might lead to the inclusion of low-quality or predatory publications.

Because of the automatic indexing approach - and the possibility of irregular schema of the indexed documents stemming from this - the Google Scholar index also does not feature search on metadata like author / organization affiliations. Many publications are also missing the links to the author profiles (authors are often only mentioned by their name), which makes the exploration of the social network of researchers difficult.

Scopus is a subscription-based academic search engine maintained by Elsevier. Orduna-Malea et al. [2015] estimated the size of the Scopus index to be around 53.4 million documents in 2014.

According to the documentation[3], contributions to the Scopus index are actively curated by a team of field experts, which allows for the inclusion of high-quality publications. Despite this effort, Macháček and Srholec [2022] and others claim that the Scopus index still includes a significant number of predatory publications.

Because of the manual curation of the documents, the Scopus web application features a faceted search on various publication metadata and relations between the authors, their organizations and the publications themselves.

Web of Science is a subscription-based academic search engine maintained by Clarivate Plc. Orduna-Malea et al. [2015] estimated the size of the Scopus index to be around 56.9 million documents in 2014. Similarly to Scopus, indexing of new publications is actively curated by a team of field experts.

Thanks to the manual curation of the documents, the Web of Science web application also features a faceted search on publication metadata.

Tennant [2020] compares the coverage of the Web of Science and Scopus indexes and points out that neither of the indexes might not fairly represent the global scientific output. According to the claims, the indices are biased towards the publications from the English-speaking countries, written in English and concerning several specific fields of study.

Goals of the thesis

The main goal of this thesis is to explore the practical applications of social network analysis in the context of academic research. The overarching goal is to improve the usability of the Charles Explorer application by using the social network analysis.

The first goal is to create a social network of researchers and their publications from the data available in the university's information system. In this part, we want to devise an effective transformation of the relational data model into a

[3] https://www.elsevier.com/products/scopus/content/content-policy-and-selection

graph data model, which will allow us to use the graph algorithms for the social network analysis.

The second goal is to explore the practical applications of the social network analysis in the context of academic research. In this part, we evaluate the usability of the social network metrics for the re-ranking of the document retrieval results.

The final goal is to improve the visualization of the academic social network in the Charles Explorer application. While the current implementation is sufficient for the basic exploration of the social network, it lacks the advanced features that would allow for the more detailed analysis of the social network. The current state of the visualization also suffers from performance and UX issues, which we want to address in this thesis.

Experimental setup

In various parts of this thesis, we are running different benchmarks and experiments. While most of those do not measure the computational performance of the algorithms, we still want to provide the reader with the information about the hardware and software used in the experiments for full reproducibility.

If not stated otherwise, the experiments were run on a machine with the following specifications:

- **APU**: AMD Ryzen 7 PRO 2700U
- **RAM**: 24 GB
- **OS**: Linux Mint 21 (5.15.0-112-generic)

Additionally, the following software versions were used in the experiments:

- **Python** 3.10.12
- **SQLite** 3.45.1
- **Memgraph** v2.18.0

Where important (due to breaking changes across versions or the use of specific features), the versions of the libraries used in the experiments are listed in the `README` files of the respective repositories.

Blog links

As the experiments evaluated in this thesis are quite extensive, the thesis itself contains only the most vital details about the experimental setup, outcomes and the conclusions for brevity.

For the full details about the experiments, the reader is encouraged to visit the blog posts that accompany this thesis. These blog posts contain the full implementation details, the code snippets, the intermediate results of the experiments or reasoning behind some of the less important design decisions that were made during the implementation.

In parts with relevant blog posts available, the links to the blog posts are provided with the following notation:[blog].

Note that the links to the blog posts are only available in the digital version of the thesis and the URLs are not visible in the printed version. In case of reading the printed version of the thesis, the reader is encouraged to visit the blog at https://jindrich.bar/edu/thesis-blog/.

1. Definitions and notation

The initial chapter lays out the definitions of the most essential social network-related terms and concepts. The chapter also introduces the notation used throughout the thesis.

1.1 Discrete graphs

In discrete mathematics, *graph* is a mathematical structure that consists of a set of *nodes* (often denoted as V for *vertices*) and a set of *edges* (often denoted as E) that connect pairs of the nodes.

This section introduces some of the thesis's less common graph-related terms and concepts.

Definition 1.1.1 (Adjacency matrix)**.** The *adjacency matrix* of a graph $G = (V, E)$ is a square matrix A of size $|V| \times |V|$ where $A_{uv} = 1$ if there is an edge between nodes u and v, and $A_{uv} = 0$ otherwise.

Different matrix operations can be used to calculate various properties of the graph. For example, the degree of a node can be calculated as the sum of the elements in the row of the adjacency matrix corresponding to the node.

Definition 1.1.2 (Distance matrix)**.** The *distance matrix* of a graph $G = (V, E)$ is a square matrix D of size $|V| \times |V|$ where D_{uv} is the length of the shortest path between nodes u and v.

The distance matrix can be calculated using Floyd-Warshall algorithm or repeated Dijkstra's algorithm.

Definition 1.1.3 (Node neighborhood)**.** The *neighborhood* of a node v in a graph G is the set of all nodes that are connected to v by an edge.

$$N(v) = \{u \in V \mid \text{there exists an edge between } v \text{ and } u\}$$

Definition 1.1.4 (k-hop neighborhood)**.** The *k-hop neighborhood* of a node v in a graph G is the set of all nodes that are reachable from v by traversing at most k edges.

$$N_k(v) = \{u \in V \mid \text{there exists a path of length at most } k \text{ from } v \text{ to } u\}$$

This can be useful to capture the local structure of a graph - e.g., Nikolentzos et al. [2019] use the concept of k-hop neighborhoods to enhance graph embeddings in graph neural network (GNN).

Definition 1.1.5 (Subgraph)**.** A *subgraph* $G' = (V', E')$ of a graph $G = (V, E)$ is a graph where $V' \subseteq V$ and $E' \subseteq E$.

Definition 1.1.6 (Induced subgraph)**.** An *induced subgraph* $G' = (V', E')$ of a graph $G = (V, E)$ is a subgraph where $V' \subseteq V$ and $E' = \{(u, v) \in E \mid u, v \in V'\}$.

In simpler terms, an *induced subgraph* is a subgraph that contains all the edges between the nodes in the subgraph.

Definition 1.1.7 (Bipartite graph)**.** A *bipartite graph* is a graph where the nodes can be divided into two disjoint sets V_1 and V_2 such that all edges connect nodes from V_1 to nodes from V_2.

Formally, a graph $G = (V, E)$ is bipartite if there exists a partition of V into two sets V_1 and V_2 such that

$$E \subseteq \{(u, v) \mid u \in V_1, v \in V_2\}$$

Definition 1.1.8 (Monopartite projection)**.** The *monopartite projection* of a bipartite graph $G = (V_1 \cup V_2, E)$ onto a set of nodes V_1 is a graph $G' = (V_1, E')$, where the nodes are the nodes in V_1, and the edges are between the nodes in V_1 that share a common neighbor in V_2.

Formally defined, the monopartite projection is a graph $G' = (V_1, E')$ where

$$E' = \{(u, v) \mid \exists w \in V_2 \text{ such that } (u, w) \in E \text{ and } (v, w) \in E\}$$

Projection algorithms can produce multiple edges between the nodes in the projection. The resolution of this problem largely depends on the specific use case of the projection — e.g., G' can be a multigraph, or the edges can be weighted by the number of common neighbors from V_2.

Definition 1.1.9 (Connected component)**.** A *connected component* of a graph $G = (V, E)$ is a subgraph $G' = (V', E')$ where all nodes in V' are reachable from each other by traversing the edges in E'.

1.2 Social networks

While the social sciences have been studying social networks for decades, not all of the terminology is relevant to this thesis. This section introduces the most essential computation-related social network terms and concepts used in the thesis.

For this thesis, a *social network* is a graph where the nodes represent real-world entities (people, their publications) and the edges represent relationships between the entities (only authorship in this thesis).

Definition 1.2.1 (Ego network)**.** The *ego network* of a node v in a social network G is the induced subgraph of G that contains v and its (1-hop) neighborhood.

$$E(v) = (N(v), \{(v, u) \mid u \in N(v)\})$$

Definition 1.2.2 (Ego (vertex))**.** The *ego* of a node v in an ego network $E(v)$ is the node v itself.

Definition 1.2.3 (Alter (vertex))**.** An *alter* of a node v in an ego network $E(v)$ is any node u that is not the ego v.

1.3 Classifier evaluation metrics

In the context of this thesis, the social network metrics are used to evaluate the performance of the classifiers that are run on the social network data.

This section introduces the most common classifier evaluation metrics used in the thesis.

Definition 1.3.1 (Precision)**.** The *precision* of a classifier is the ratio of the true positive predictions to the total number of positive predictions.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Definition 1.3.2 (Recall)**.** The *recall* of a classifier is the ratio of the true positive predictions to the total number of actual positive instances.

$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Definition 1.3.3 (F1 score)**.** The *F1 score* of a classifier is the harmonic mean of the precision and recall.

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Definition 1.3.4 (Macro-averaged F1 score)**.** For a multi-class classification task, the *macro-averaged F1 score* is the average of the F1 scores for each class.

$$\text{macro-F1} = \frac{1}{n} \sum_{i=1}^{n} \text{F1}_i$$

Definition 1.3.5 (Micro-averaged F1 score)**.** For multi-class classification tasks, the *micro-averaged F1 score* is the F1 score calculated from the total number of true positives, false positives, and false negatives.

$$\text{micro-F1} = 2 \cdot \frac{\sum_{i=1}^{n} \text{TP}_i}{\sum_{i=1}^{n} \text{TP}_i + \sum_{i=1}^{n} \text{FP}_i + \sum_{i=1}^{n} \text{FN}_i}$$

This metric is less susceptible to class imbalance than the macro-averaged F1 score. Each class contributes to the micro-averaged F1 score proportionally to the number of instances in the class.

2. Data models and transformations

The data about the academic researchers and their publications at Charles University (CUNI) is internally stored in a relational database. The first chapter of this thesis explores the transformation of the relational data model into a graph data model, which will allow us to use the graph algorithms for social network analysis.

We also address pitfalls and challenges of the transformation stemming from the specific nature of the data and the technical limitations of the source systems.

2.1 Input data format

The Charles University information system comprises a set of separate systems and applications.

The *Studium* information system[1] contains the data about the researchers, teachers, courses, and study programs available at the university.

The *Věda* information system[2] aggregates the data about the creative activities of the researchers, the research projects, and inter-university mobility programs. For this thesis, we are primarily interested in the Osobní bibliografická databáze (Personal Bibliographic Database) (OBD) (or Verso) module, containing data about academic publications.

The *Whois* staff information system[3] contains the data about the employees of the university, their affiliations with the faculties and departments and their academic titles.

While none of the systems offer public APIs, a data import pipeline has been set up by Ústav výpočetní techniky (Computer Science Centre) (ÚVT) (the university's IT department) in advance for the Charles Explorer application. This pipeline consists of a set of database views tracking the changes in the data and a script[4] that exports the data to an SQLite database.

2.2 Target data model

We aim to transform the relational data model into a graph data model, allowing us to use the graph algorithms for the social network analysis.

While the available data would allow us to create a multi-relational graph model with different types of nodes and edges (e.g., *Person*, *Publication*, *Course*, *Study*

[1] Available at https://is.cuni.cz/studium/.

[2] Available at https://is.cuni.cz/veda/, login only.

[3] Available at https://is.cuni.cz/webapps/whois2.

[4] https://gitlab.mff.cuni.cz/barj/charles-explorer/-/blob/master/scripts/export.sh

program), we limit the scope of this thesis to the social network between the **academic researchers** and their **publications**.

From the view of the input data, this is the most significant and most interconnected part of the data, which will allow us to demonstrate the capabilities of the graph algorithms. The connections in the resulting graph are also easily interpretable (e.g. the coauthorship relations between the researchers, the authorship relations between the researchers and the publications).

The graph nodes will represent the academic researchers and the publications. In contrast, the edges will represent the coauthorship relations between the researchers and the authorship relations between the researchers and the publications.

We can notice that the graph is a *bipartite graph*, as the nodes can be divided into two disjoint sets - the set of the researchers and the set of the publications.

2.2.1 Exploring the schema

We provide an example of the schema of the relational database together with example data to illustrate the structure of the input data. Since this thesis focuses on the social network between academic researchers and their publications, we limit this example only to the relevant parts of the schema.

The social network data is accessible in the following three database views:

```
PERSON (
    PERSON_ID, - UKČO personal number
    PERSON_NAME, - Full name of the person, incl. the academic titles
    PERSON_WEBSITE, - Personal website of the person
    PERSON_WHOIS_ID, - ID of the person in the Whois system
    TYPE - Person type - teacher(U), external employee(E), or other(O)
)
```

The `Person` view contains the data about the academic researchers and teachers at the university. While the `TYPE` column might suggest that the table includes records about external people as well (e.g., guest co-authors of publications published by the CUNI researchers), it only contains the data about the people affiliated with the university. The `TYPE` column is only used to distinguish between the different employment types at the university.

```
PUBLICATION_KEYWORDS (
    PUBLICATION_ID, - internal ID of the publication
    PUB_YEAR, - year of the publication
    TITLE, - title of the publication
    ABSTRACT, - abstract of the publication
    KEYWORDS, - keywords of the publication
    LANGUAGUE(sic!), - ISO 639-2 language code of the TITLE,
                                ABSTRACT and KEYWORD columns
    ORIGINAL - whether the LANGUAGUE column is the original
                                 language of the publication
)
```

This schema shows some more issues with the data - the PUBLICATION_KEYWORDS is missing a single primary key attribute, as the PUBLICATION_ID is not unique across the table. This is because the same publication can have multiple records in the table, each representing a different language version of the title, abstract, and keywords.

Moreover, the PUBLICATION_KEYWORDS view does not contain a universally accepted publication identifier (e.g., a DOI or an ISBN) that would allow us to link the publication to the external sources of the publication data. This is because of a technical limitation of the information systems, as the OBD module is not fully interoperable with the *Studium* module we are consuming the data from.

```
PUBLICATION_AUTHOR_ALL (
    PUBLICATION_ID, - internal ID of the publication
    PERSON_ID, - UKČO personal number of the author, if available
    PERSON_NAME, - Full name of the author, incl. the academic titles
)
```

The relational database view PUBLICATION_AUTHOR_ALL contains the links between the publications and the authors from the previous views.

With the most naive approach, the transformation of the relational data model into a graph data model could be done by transforming the records of the PERSON and (deduplicated) PUBLICATION_KEYWORDS views into the nodes of the graph, and the records of the PUBLICATION_AUTHOR_ALL view into the edges of the graph.

2.2.2 Missing identities

In the comment for the PUBLICATION_AUTHOR_ALL.PERSON_ID column, we note that the UKČO personal number is not always available. This is because of external authors who are not affiliated with the university and do not have a UKČO personal number. What is worse, such authors are only identified by their names, which can be inconsistent across the publications. This is again caused by the limited interoperability of the data source modules - however, note that this problem is present in larger academic search engines (e.g., Google Scholar) too.

Aside from the obvious implications of this issue - i.e., user confusion and potential performance issues, this also poses a challenge for the social network analysis, as the graph algorithms might not be able to correctly identify the external authors.

In the aforementioned PUBLICATION_AUTHOR_ALL view, counts of the relations mentioning the authors with/without the UKČO personal number are as follows:

Type	Count	Distinct
PERSON_ID present	808467	39523
PERSON_ID missing	671332	?

Figure 2.1: Counts of the relations in the PUBLICATION_AUTHOR_ALL view.

We see that the no-identifier authors take up to 45% of the total count of the relations. This explains the importance of inferring missing identities.

2.3 Identity inference - Naïve approach

The naïve solution to this problem is using the *names* for the identity inference in case of missing identifiers. While this is a simple transformation, it has obvious drawbacks.

Firstly, the names are not guaranteed to be *unique* across the dataset.

Secondly, the names are not guaranteed to be *consistent* across the dataset - due to (e.g., marital) name changes, typos, or different conventions in the academic titles. See the example of a search for the name "Jaroslav Peška" in the `PUBLICATION_AUTHOR_ALL` view:

COUNT(*)	PERSON_NAME	PERSON_ID
2	doc. PhDr. Jaroslav Peška Ph.D.	14124313...[5]
5	doc. PhDr. Jaroslav Peška Ph.D.	null
2	Doc. PhDr. Jaroslav Peška Ph.D.	null
1	doc. PhDr. Jaroslav Peška PhD.	null
4	Jaroslav Peška	null

Figure 2.2: Search for the name "Jaroslav Peška" in the `PUBLICATION_AUTHOR_ALL` view.

While there are five variants of the same name in the dataset, only one is correctly linked to the UKČO personal number. This could have been caused by human error in data input or by the limitations of the source systems. Either way, this creates an unrecoverable loss of information in the dataset.

2.3.1 Algorithm

In the SQL database export, we can "merge" the records using the naïve approach efficiently using a many-to-one (non-injective) mapping on the `PERSON_NAME` column.

Let us define a function f:

$$f : \text{PERSON\_NAME} \to \text{NORMALIZED\_NAME}$$

We require f to map all person names to a *normalized form*, which is defined as follows:

- The academic titles are stripped from the name.
- The name is converted to lowercase.
- The name is stripped of any diacritics.
- The name is stripped of any non-alphabetic characters.
- The whitespace characters are normalized to a single space.
- The name is stripped of any leading or trailing whitespace.

[5] PERSON_ID redacted for privacy reasons, replaced by truncated PERSON_WHOIS_ID.

We can see that in the case of Jaroslav Peška, the normalized version for all the variants is the same (i.e., $f(\texttt{PERSON\_NAME}) = \texttt{jaroslav peska}$).

If we decide to add the output of this normalization function as an attribute in the `PUBLICATION_AUTHOR_ALL` view, We can proceed with merging the records using SQL GROUP BY and the aggregation functions.

2.3.2 Implementation

For the implementation of the name normalization function, we use the SQLite loadable extensions mechanism. The transformation itself is done in three steps:

1. Using the `glib` function `g_utf8_normalize`, the string representation of the name is normalized to the canonical Unicode NFD form. This step ensures that the byte representation of diacritized characters is now composed of separate characters for the base character and the diacritics (e.g. `ě(U+011B)` is decomposed to `e(U+0065)` and `˘(U+02C7)`).
2. We scan the normalized string and remove all the characters that are not in the range of the Latin alphabet. We convert all the alphabetic characters to lowercase and replace all the whitespace characters with a single space.
3. Using regular expressions, we remove the academic titles from the name. We construct the expression by defining a list of the academic titles and concatenating them using the alternative separator `|` (e.g., `PhDr.|Ph.D.`). We then use this regex to find the start index of the actual person name in the string and use the `glib` function `g_strndup` to extract the substring.

The full implementation of the normalization function is available in the GitHub repository of this thesis.[6].

> **SQLite Loadable Extensions**
>
> While it would be possible to implement the normalization function using inbuilt scalar SQLite functions (namely `lower`, `trim`, and repeated use of `replace`), the developer experience of such a solution is suboptimal, as it requires a lot of nested function calls in the `SELECT` clause.
>
> Because of the way the scalar functions are defined in SQLite - and their implementation[a], the repeated calls to `replace` also result in repeated string allocations and deallocations - which can be a performance bottleneck.
>
> Fortunately, SQLite allows users to define their own loadable extensions[b] in C. This allows us to define the normalization function in C and load it as a scalar function in the SQLite database.
>
> [a] https://sqlite.org/src/file?name=src/func.c
> [b] https://www.sqlite.org/loadext.html

For reference purposes, we also implement the same normalization function using Python. This implementation loads the data from the database, normalizes the

[6] https://github.com/barjin/master-thesis/tree/main/examples/sqlite-normalize-ext

names using a Python function, and writes the normalized names into a new table. The implementation can also be found in the GitHub repository[7].

2.3.3 Performance

As mentioned in the infobox above, the SQLite extension for normalizing the names should improve performance over the composed scalar functions. To test this hypothesis, we repeatedly construct a temporary table with the normalized names using both scalar functions and the extension.

```
CREATE TABLE test AS
    SELECT normalize_name(PERSON_NAME) -- or the scalar functions
        AS NORMALIZED
    FROM PUBLICATION_AUTHOR_ALL;
```

Figure 2.3: The SQL query for the performance test of different normalization methods.

SQLite engine has a default limit on the maximum expression tree depth of 1000[8]. This limit is relatively low and can be easily reached using the composed scalar functions.

Because of this limitation, we are not replacing all of the academic titles in the *composed scalar functions* case of the experiment. Even with the reduced number of operations, the composed scalar functions seem to be much slower than the SQLite extension:

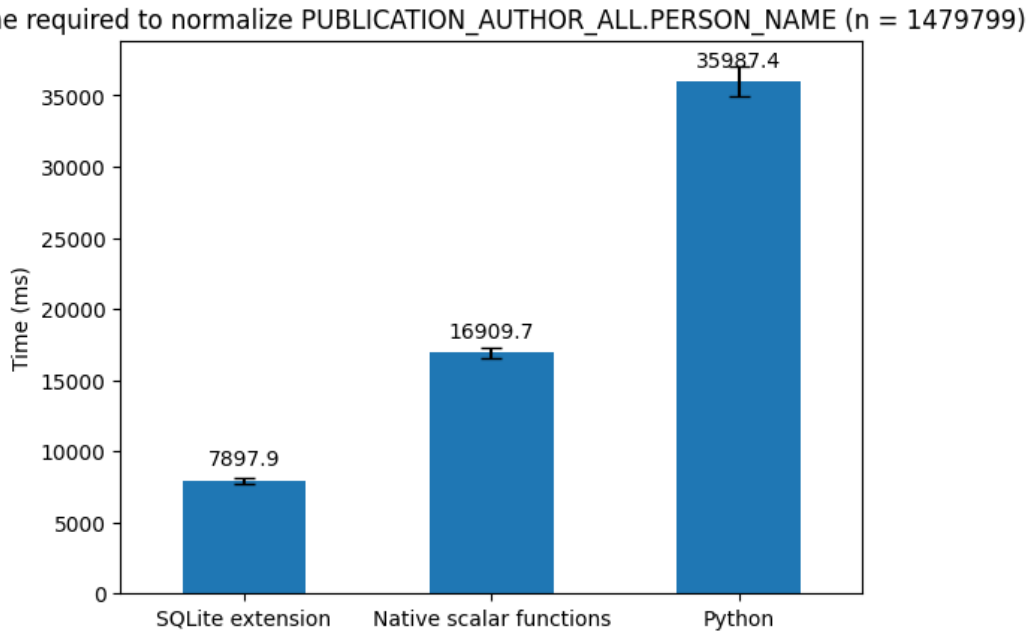

Figure 2.4: The performance comparison of the SQLite extension and the composed scalar functions (mean over 10 runs).

[7] https://github.com/barjin/master-thesis/tree/main/examples/sqlite-normalize-python
[8] https://www.sqlite.org/limits.html#max_expr_depth

The Python reference implementation is expectedly much slower than both the SQLite extension and the composed scalar functions. This is likely caused by the interface read-write overhead of the Python `sqlite3` library - compared to the native data access of the SQLite extension.

2.3.4 Results evaluation

Even though the missing identifiers of the external authors make actual evaluation of this method impossible, we can assess the effectiveness of the naïve merging approach on the set of the **internal authors** (i.e., authors with existing explicit identifiers). By comparing the identities inferred by the proposed method with the existing identifiers, we evaluate the method's accuracy.

Note that this only works under the assumption that the names of the external (no-identifier) authors follow a similar distribution as the names of the internal authors. While this might not be completely true—as external authors might, for example, be more likely to be foreign nationals, perhaps with different naming conventions—we consider this assumption a reasonable simplification.

To define metrics for the evaluation, let us first reformulate the identity inference *on the labeled data* as a classification problem:

For a normalized variant of a name n, the predictor f returns (inferred) identifier i, such that:

$$f(n) = MIN(\{\text{r.PERSON\_ID} \mid \text{r.PERSON\_NORM\_NAME} = n\})$$

- The target classes for the classification are the *internal author identifiers* (identities).
- For a given normalized name, the predictor returns the *minimal* author identifier with the corresponding normalized name (using e.g. the lexicographical order).
  - While sampling the identifiers from a probability distribution based on the number of publications of the given person would be more accurate, this would require the knowledge of the publication counts of the external authors - which is not available to us (each external author is connected to only one publication).

    We also assume that the number of different external authors of the same name is small - and that the authors with the same name are likely to be the same person.

Note that this proposed classifier simply groups all the records with the same normalized name under one deterministically selected identifier.

As we now define the algorithm as a classification problem, we can use the standard classification metrics to evaluate the method's performance. Since the task is a multi-class classification, we calculate the *macro* and *micro-*F_1 *score*.

For the dataset of 39523 known-distinct internal authors, the grouping by the normalized names results in 35610 merged records. As predicted in the previous

section, the method likely merges people with the same name together. The evaluation metrics of the naïve merging method are as follows:

Method	Value
Macro-averaged F_1 score	0.874959
Micro-averaged F_1 score	0.900994

Figure 2.5: The evaluation metrics of the naïve merging method[9].

2.3.5 Issues with the naïve approach

While the name-based merging approach is simple, it poses several issues. Firstly, name-based merging can cause over-merging in the case of authors with common names.

Secondly, the proposed algorithm does not support the existing identifiers from the dataset. By only inferring the identifier from the name, we willingly discard the existing information about the person.

The latter problem could be partially solved by coalescing the inferred identifiers with the existing ones. Using the SQL `COALESCE` function, we can create a new column in the `PUBLICATION_AUTHOR_ALL` view that would contain the inferred identifier only if the original one is missing.

However, such an approach would only fully work in the case of only internal - or only external authors. In the case of the internal authors with few records with missing identifiers (e.g., Jaroslav Peška), this would still result in the creation of up to two nodes for the same person (i.e., `14124313...` for the "internal" records and `jaroslav-peska` for the "external" ones).

"On-demand" naïve identity inference

In an attempt to address the first mentioned issue - the over-merging of the authors with common names - we slightly modify the algorithm.

Instead of merging the records directly in the database, we merge the records on-demand in the application's visualization layer. If we e.g., restrict the direct views only to internal authors - and show the external authors only as collaborators of those, we can reuse the naïve approach quite effectively.

By restricting the domain of the mergeable records to a smaller, tighter subcommunity of the graph (i.e., collaborators of one internal author), we reduce the number of possible false positives in the merging.

Obviously, this approach still carries the risk of false positives in the case of different external authors with the same name in the same subcommunity.

Also, note that this approach only improves the user experience with the application. Using this, every external author's coauthorship is stored as a separate node in the graph, which can negatively impact the performance of the graph algorithms.

[9]The SQL implementation of both evaluation metrics is available in the GitHub repository.

2.4 Identity inference - Hierarchical clustering

To solve the issues of the naïve approach, we can try to use the graph structure of the data to infer the missing identities. In the following subsection, we propose an algorithm using the *hierarchical clustering* methods to group the nodes from similar parts of the graph together. At the end of the subsection, we evaluate the performance of the proposed method on the labeled data.

In the realm of data mining and statistics, *hierarchical clustering* is an umbrella term for a set of unsupervised learning algorithms for grouping given data points into a hierarchy of clusters. Typically, these algorithms iteratively merge the closest clusters together, until only one cluster remains.[10] The distance metric used for the clustering can be any metric that defines the similarity between the data points.

2.4.1 Algorithm

In the case of our dataset, we can use the shortest path distance between the nodes in the graph as the distance metric.

We define the merging process as follows:

1. Start with a graph with nodes only merged based on explicit identifiers. `PUBLICATION_AUTHOR_ALL` records without an explicit identifier are represented as separate nodes.
2. Select an arbitrary unmerged node without an identifier.
   (a) Using the naïve approach, we find all merge candidates for the selected node. Note that we are using the normalized name equality as the requirement for the merge.
   (b) We calculate the distance matrix between all the merge candidates.
   (c) Using a hierarchical clustering algorithm, we cluster the data points based on the distance matrix. We use a static preselected threshold condition to determine the cluster boundaries (or mergeability of the node groups).
   (d) We use the clustering results to merge all nodes belonging to the same clusters.
3. Go to step 2 until all the non-identifier nodes are merged.

We can make a few observations in the internal loop of the second step. As mentioned in 2.2, the graph containing the nodes with the explicit identifiers is a *bipartite graph*. Because of this, any path between two person-type nodes is even-lengthed. Therefore, the shortest path distance between two person-type nodes is 2.

Additionally, the shortest path distance between two *mergeable* nodes is 4. This is based on an observation that two-person nodes in the distance 2 are connected

[10] In the case of top-down clustering, the process is reversed.

to the same publication node. In case there are two merge candidates (based on the naïve mergeability requirement) connected to the same publication node, these should not be considered *mergeable*, as the publication is expected to list unique authors - i.e., the candidates are different people with the same name.

2.4.2 Implementation

The proposed algorithm is computationally expensive. The distance matrix calculation has a time complexity of $O(n^2)$ and the standard hierarchical agglomerative clustering algorithm has a time complexity of $O(n^3)$.

Distance matrix calculation[blog]

Loading the social network into Memgraph[11] , we quite easily implement the calculation of the distance matrix between the merge candidates using the Cypher graph query language.

```
MATCH path=(p1: Person)-[*BFS]-(p2: Person)
WHERE
    p1.PERSON_ID IN $queryCandidates
    AND p2.PERSON_ID IN $queryCandidates
    AND p1.PERSON_ID < p2.PERSON_ID
RETURN
    p1.PERSON_ID,
    p2.PERSON_ID,
    size(path) AS distance
```

Figure 2.6: The Cypher query for the distance matrix calculation.[12]

In this query, `$queryCandidates` is an array containing the node identifiers of the merge candidates for a given normalized name group. Note that the query results can also be paginated using the `SKIP` and `LIMIT` clauses.

Preliminary experiments show that the query execution time is around `5.37` ms per pair of nodes. Calculating the matrix of, e.g., `490000` pairs (for an external author consisting of 700 merge candidates) would take around 43 minutes. While this is a one-time operation - and the distances cannot change, as the external author nodes are always connected only to the one publication they are attributed to - this still leaves space for optimization.

Optimization: Monopartite projection

The run time of the distance matrix calculation is artificially increased by the *bipartitedness* of the graph. During the calculation of the shortest path distance between two person nodes, the breadth-first search algorithm is forced to explore the intermediate publication nodes - even though those have no possibility of being the target node. This can inflate the size of the

[11] An open-source in-memory graph database. Available at https://memgraph.com/.

[12] Rather curiously, passing the IDs array as a parameter results in a highly inefficient execution plan in Memgraph, causing a full scan of the nodes followed by a filter on the PERSON_ID. Inlining the parameter array into the query causes Memgraph to use faster indexed access.

internal BFS queue and the number of the visited nodes.

To mitigate this issue, we can calculate the distance matrix in `2(b)` on the monopartite projection of the social network on the set of the *person nodes*. By replacing the publication nodes with edges on the person nodes, we reduce the lengths of the shortest paths in half.

While the monopartite projection of the social network can be calculated using the Cypher query language in Memgraph, we transform the data straight in the source SQL database. Using a self `INNER JOIN` on the `PUBLICATION_AUTHOR_ALL` view over the `PUBLICATION_ID` column, we receive the monopartite projection of the social network on the set of the person nodes. Projecting the data onto the person identifiers leaves us with a table containing the pairs of the person identifiers connected by an edge.

```
SELECT DISTINCT X.PERSON_ID, Y.PERSON_ID
   FROM (
      SELECT PERSON_ID, PUBLICATION_ID FROM PUBLICATION_AUTHOR_ALL
   ) AS X
   INNER JOIN (
      SELECT PERSON_ID, PUBLICATION_ID FROM PUBLICATION_AUTHOR_ALL
   ) AS Y
USING(PUBLICATION_ID)
WHERE X.PERSON_ID < Y.PERSON_ID;
```

Figure 2.7: The SQL query calculating the monopartite projection of the social network.

Note that the `PERSON_ID` in the query above is either the internal person identifier or a *generated unique identifier* for the external authors.

Unfortunately, even with the projected graph, the performance of the shortest path distance calculation in Memgraph is still suboptimal. With the mean time of `5.68` ms per pair of nodes, this approach does not provide any significant performance improvement over the original one.

This might be caused by the specific structure of the academic social network, where the person nodes are connected by a much smaller number of publication nodes. This is especially true with the graph with external authors, where the external author nodes are connected only to the one publication node.

Note that the BFS algorithm has the complexity of $\mathcal{O}(V + E)$, where V is the number of nodes and E is the number of edges. While the monopartite projection reduces the number of the nodes in the graph, it does not reduce the number of the edges - as the publication nodes are replaced by the edges between all the coauthoring person nodes.

In case of the process described with the Figure 2.6, each publication node is replaces by k edges, where

$$k = \binom{deg(v)}{2}$$

for $deg(v)$ being the degree of the publication node v.

Optimization: BFS "vectorization"

Memgraph resolves the Cypher query for calculating the distance matrix with an execution plan that includes a shortest path operation for *each node pair*.

While Memgraph uses a slightly optimized algorithm[a] for calculating the shortest path distance between the two nodes, the distance matrix calculation can still be optimized by using a *vectorized* approach.

During a regular BFS, the algorithm visits the growing k-hop neighborhoods of the start node until it finds the target node. Since in our case, we are calculating the entire distance matrix (for a set of nodes), we can optimize the process by searching for all the target nodes in the same BFS run. This allows us to acquire the shortest path distances between the start node and all the target nodes in a single BFS run.

[a]Expanding the BFS fringe from both ends (see source on GitHub).

The algorithm itself is quite simple and differs from the regular BFS approach only in a handful of details. The following pseudocode example describes the vectorized BFS algorithm:

```
1: function VectorizedBFS(graph, start, targets)
2:     visited, queue, distances ← ∅
3:     queue.push(start, 0)
4:     while not queue.empty() and not targets.empty() do
5:         (node, depth) ← queue.pop()
6:         if node in targets then
7:             distances[node] ← depth
8:             targets.remove(node)
9:         end if
10:        for neighbor in graph[node] do
11:            if neighbor not in visited then
12:                queue.push((neighbor, depth + 1))
13:                visited.add(neighbor)
14:            end if
15:        end for
16:    end while
17:    return distances
18: end function
```

Notice the underlined parts of the algorithm - these are the parts that differ from the regular breadth-first search for a single target node. Unlike the single target BFS, the vectorized version only terminates once all the target nodes are visited.

Note that - given we consider the *targets* list constant-sized - the algorithm has the same asymptotic complexity as the regular BFS - i.e., $\mathcal{O}(V + E)$. This is obvious if we compare this vectorized approach to the regular single-target BFS for the *furthest* target node. Both algorithms visit the same number of nodes

and edges, the only difference is in the output - the vectorized BFS outputs the distances to all the target nodes.

At the time of writing this thesis (July 18, 2024), the vectorized BFS algorithm is not implemented in Memgraph or any other major graph database (Neo4j, OrientDB, ArangoDB). While with the vertex identifiers inlined in the query, the Cypher query planner might be able to automatically recognize the need for the vectorized BFS, the obscure nature of the use case is likely the reason for the lack of the implementation.

Because of this, we implement our own distance matrix calculator based on the vectorized BFS algorithm in C++[13]. This program takes the graph data - i.e., the output of Figure 2.7 - in CSV format and parses the data into an in-memory graph representation. It then iterates over a list of the user-specified target nodes, taking each node as the start node for the *VectorizedBFS* algorithm. The rest of the list is used as the target nodes for the algorithm.

By iterating through the list of the target nodes, we calculate the distance matrix in k BFS runs, where k is the number of the target nodes. This is a significant improvement over the k^2 BFS runs in the original Memgraph approach.

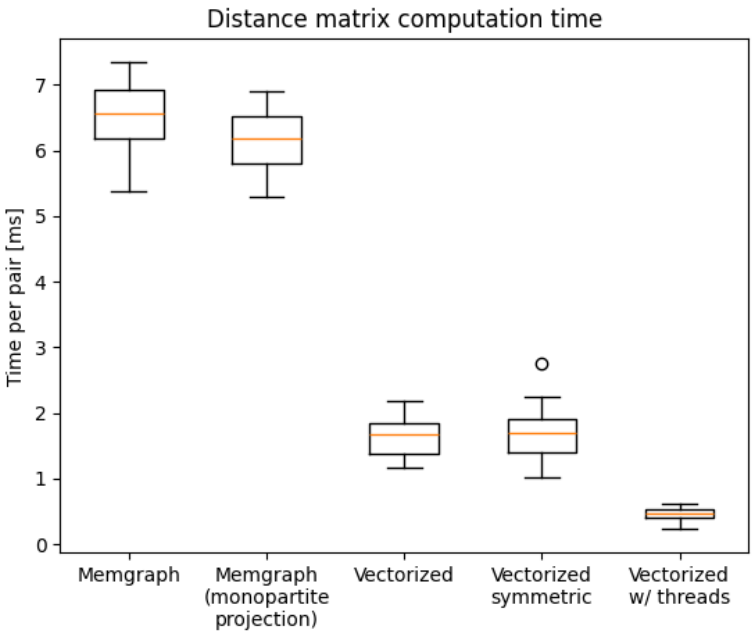

Figure 2.8: The performance comparison of different distance matrix calculation methods ($n = 10$ runs for each method).

Benchmarks show that the vectorized BFS algorithm is with a mean time of 1.65 ms per pair of nodes significantly faster than the regular BFS algorithm.

Figure 2.8 compares the results to two more possible optimizations of the vectorized approach.

The **Vectorized symmetric** approach only calculates the upper triangle of the distance matrix and mirrors the results to the lower triangle. This utilizes the

[13]Implementation at https://github.com/barjin/master-thesis/tree/main/examples/distance-matrix

fact that the social network graph we are using is undirected - i.e., the shortest path distance between the nodes u and v is the same as the distance between the nodes v and u. This makes the distance matrix symmetric and allows us to save half of the calculations by only calculating the upper triangle.

This approach brings next to no performance improvement over the regular vectorized BFS. The reason becomes apparent when we consider the fact that the vectorized BFS's complexity is $\mathcal{O}(V + E)$ - i.e. does not depend on the number of the target nodes.

The approach labeled **Vectorized w/ threads** parallelizes the vectorized BFS algorithm using multithreading. Since the BFS algorithm is inherently sequential, parallelization is done on the outer loop iterating through the start nodes. The parallelization itself is quite straightforward, as *VectorizedBFS* is a pure function with no side effects.

This approach comes out as the most performant, with the mean time of 0.45 ms per pair of nodes.

2.4.3 Clustering process

After calculating the distance matrix, we can proceed with the hierarchical clustering algorithm. For the purpose of this thesis, we will be using specifically the *bottom-up agglomerative clustering* algorithm.

Unlike the naïve approach, the hierarchical clustering algorithm is parametrizable. Note that in the step $2(c)$ of the proposed algorithm, we are mentioning a *static preselected threshold condition* for the cluster boundaries.

Any bottom-up hierarchical clustering algorithm starts with original data points as separate clusters and iteratively merges the closest clusters together - until there is only one cluster left. We need to break down the hierarchy of recursively merging clusters into disjoint groups of nodes with the same inferred identifier.

Sokal and Rohlf [1962] define the *cophenetic value* (or *cophenetic distance*) of two clusters as the distance at which the two clusters are merged. Considering a maximum cophenetic value $c_{\max}$, we can define a stopping criterion for the hierarchical clustering algorithm.

If the two clusters we are about to merge are at a distance $c_{\max}$ or further apart, we stop the merging process and return the currently existing clusters as the final data point grouping.

As mentioned earlier, the closest distance between two mergeable candidates in our graph is 4. This is because two-person nodes we can consider to be mergeable have to be connected to two different publication nodes. These publication nodes need to be connected by - at least - one other person node.

Since the best choice for the cutoff distance is unclear, we will later evaluate the performance of the hierarchical clustering algorithm for different threshold values.

2.4.4 Results evaluation[blog]

As with the naïve approach, we evaluate the performance of the hierarchical clustering merge on the set of the *internal authors*, as we have the ground truth for the identities of these authors.

However, since this method uses the graph data structure, we need to slightly edit the graph on the labeled data, so it resembles the data with the missing identifiers better.

Firstly, the external (no-identifier) authors are always associated with a single publication only. Therefore, for evaluation, we split the internal authors into multiple nodes - each representing a single "authorship" of a publication. We then connect these nodes to the respective publication node.

Note that only splitting the internal authors into multiple nodes would result in a severely disconnected graph - each publication would be a separate connected component. This would effectively disable the hierarchical clustering algorithm based on the path lengths between individual merge candidates.

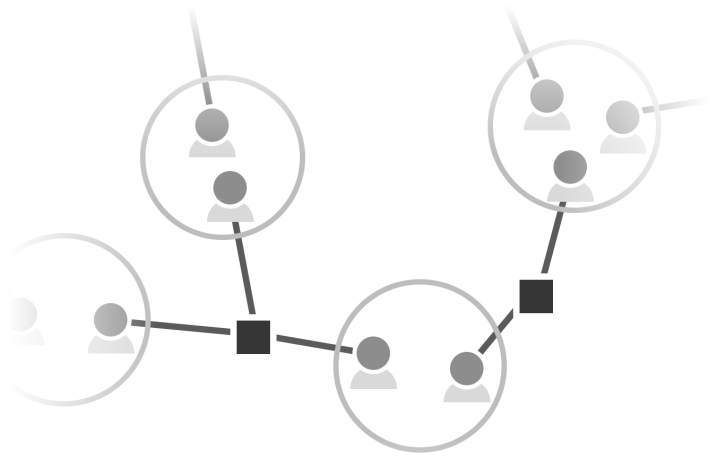

Figure 2.9: The splitting of the internal authors into multiple nodes. Encircled nodes represent the original author identities. Note that the resulting graph is disconnected.

We can see that for a single set of merge candidates - i.e., nodes with the same normalized name - most nodes would end up in separate connected components after the splitting (with the exception of two different people with the same normalized name collaborating on the same publication).

We solve this issue by reintroducing the original internal author nodes into the graph. This operation requires attention, too, though. Consider a merge candidate set $\mathbf{C}$ consisting solely of the "authorship" nodes for a single internal author. After returning the original author node to the graph, the distance matrix for $\mathbf{C}$

would degenerate to

$$D_{\mathbf{C}} = \mathbf{4} - 4\mathbf{I_n} = \begin{pmatrix} 0 & 4 & 4 & \dots \\ 4 & 0 & 4 & \dots \\ 4 & 4 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

This is because one of the shortest paths between the authorship nodes would always lead through the **original author node**, which is connected to both ends' publications.

During the distance matrix calculation in the vectorized BFS algorithm, we solve this by introducing a *forbidden* set of nodes. This set contains the nodes that are not allowed to be visited during the BFS run. For each BFS run, we add the original author node to the forbidden set, effectively preventing the BFS algorithm from visiting the node.

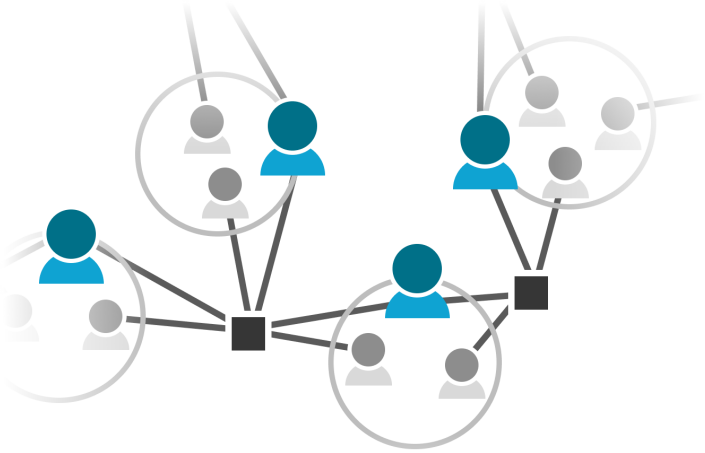

Figure 2.10: Graph after reintroducing the original author nodes (blue). The graph is now connected. Note that to calculate the distance between two split "authorship" nodes, we forbid the BFS algorithm from visiting their original author node.

Evaluation metrics

The *merging algorithm to multi-class classifier* mapping introduced in the naïve approach evaluation is still valid, with slight differences. For the purpose of the evaluation, we again reformulate the identity inference as a classification problem:

For a node n without an identifier and a (generated) cluster C_n containing n (and some other nodes with the same normalized name), the predictor f returns (an inferred) identifier i, such that:

$$f(n, C_n) = \min(\{\text{r.PERSON\_ID} \mid r \in C_n\})$$

Just like in the naïve case, we simply group all the nodes in the cluster under the minimal identifier - but now only in the context of the cluster. Note that in the case of a degenerated clustering algorithm always returning a single cluster containing all nodes with the same normalized name, the proposed algorithm would yield the same results as the naïve approach.

As we proposed in the Subsection 2.4.3, we evaluate the performance of the hierarchical clustering algorithm for different threshold values to determine the optimal cutoff distance.

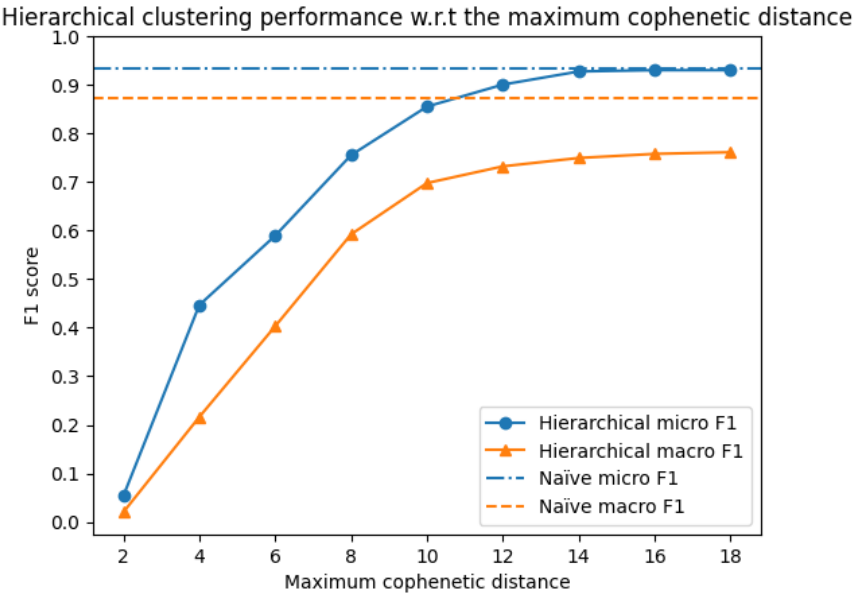

Figure 2.11: The F1 scores for the hierarchical clustering algorithm for different threshold values.[14]

We notice that for the maximum cophenetic distance $c_{\max} = 2$, the algorithm performance is very poor. This is because in such case, the algorithm cannot merge any two merge candidates - as the closest distance between them is 4. In such case the algorithm only correctly classifies the first node in the cluster - while the rest of the nodes are (incorrectly) classified as separate entities.

We also see that the performance of the hierarchical clustering algorithm is dominated by the naïve approach, regardless of the parameters. Unsurprisingly, both methods perform more similarly with the increasing cutoff cophenetic distance. This is because the naïve approach is essentially a special case of the hierarchical clustering algorithm with a fixed (infinite) cutoff distance - merging all the nodes with the same normalized name in one cluster.

The superior performance of the naïve approach is likely caused by the nature of the input data and the variance in the normalized names. While the possibility of splitting the over-merged groups into multiple clusters with the hierarchical clustering algorithm might seem advantageous, We see that this is not the case in the context of academic social network data.

[14]Both methods have been reevaluated on a smaller representative subset of data ($n = 500$). For this reason, the naïve approach performance slightly differs from the previous evaluation.

Note that the macro F1 score for hierarchical merging continues to grow with the increasing cutoff distance, meeting the performance of the naïve approach at 1000. This is because the distance matrix computation tool denotes "infinite" distance (e.g., between two nodes in two disconnected components of the graph) as 999.

3. Social network boosted search ranking

In the previous chapter, we explored the process of transforming the relational academic data into a true graph representation, including using graph metrics for missing data inference.

In this chapter, we focus on the *search engine* part of the Charles Explorer application. Using the graph data representation from the previous chapter, we explore the possible issues with the classic document-based search engine and experiment with the social network data for the search results re-ranking.

3.1 Full-text search

Nowadays, full-text search is an essential part of any information retrieval system. Many search engines - including Apache Solr in Charles Explorer - implement the full text search by utilizing the TF-IDF algorithm or similar. This is a simple and efficient way to rank the search results based on the relevance of the documents to the search query.

The TF-IDF algorithm is based on the term frequency and inverse document frequency of the terms in the documents - a *document* is, in general, unstructured free text content. Some entities in our academic search engine map well to this notion of a *document* - e.g., a `publication` or `class` both have inherent textual content (titles, abstracts, or class syllabi). Unfortunately, this does not hold for all the entities in the system.

As an example, a `person` entity usually does not have any explicit textual content associated with it. When searching for a person interested in a particular topic, the search engine has to rely on the textual content of the publications, classes, or other entities associated with the person, i.e. traverse - at least implicitly - the knowledge graph (or the social network of the person).

A simple solution to this problem would be to represent every person as a document, concatenating all the textual content of the entities associated with the person. This can be further refined by assigning different weights - e.g. to the different types of entities (a `class` might be more important than a `publication`), or different concatenated parts of the documents (e.g. the publication title and class name are more important than the abstracts and syllabi).

Regarding adding new entities related to a given person, the concatenation also serves us well - we simply append the new entity's content to the person's document and reindex it.

However, this approach also has several drawbacks. First of all, assuming we're building a general academic search engine allowing for search in publications, classes and people, we would be indexing the same content multiple times. This is not only inefficient in terms of storage, but also prone to update errors - there

is multiple copies of the same content, which have to be updated separately. The second issue arises from the concatenation - if any of the person's associated entities changes, the whole document has to be reindexed. However, this might be less of a problem than the first issue, as it's not too common for academic records to get updated or removed - at least in comparison to the number of new records being added.

3.1.1 TD-IDF ranking issues

Result ranking in information retrieval refers to the ordering of the search results when presented to the end user. This is often based on the relevance of the documents to the current search query. The relevance-based ranking is often enough for the basic use case - the user is presented with the most relevant documents first, and can further explore the less relevant ones if needed.

While it might seem a bit superficial, the ranking is, in fact, still part of the information retrieval process. Glick et al. [2014] showed that the ranking of the search results positively correlates with the click-through rate of the results - likely because of the typical top-left to bottom-right reading pattern of the users. This can be further affected by other, more technical factors - such as the need for an additional user action like scrolling or pagination to see the results further down the list.

Considering a simple tf-idf-based search engine, the ranking of the search results is based on the relevance of the documents to the search query. This is directly related to the term frequency of the query in the document - for a fixed query and document collection, we can forget about the inverse document frequency, as it's constant over all the documents. Ranking the documents solely based on the term frequency might however lead to unfavourable results - especially in the case of a proxy-representation of a given entity.

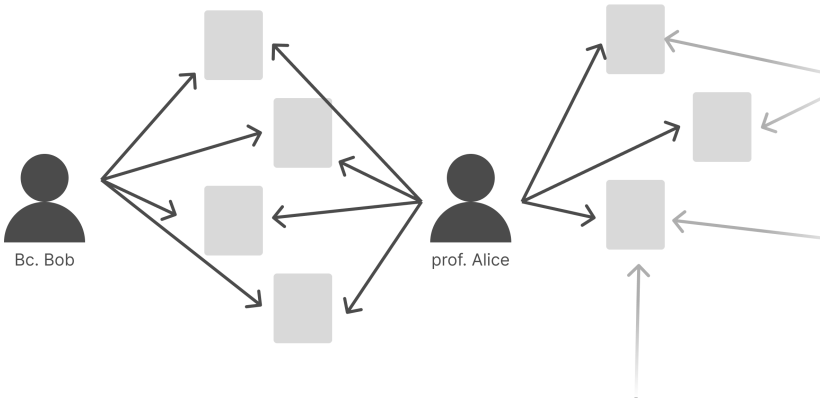

Figure 3.1: Simple representation of the social network of Alice and Bob.

Let us explore the issues in an example where we represent people as documents, concatenating the textual content of the entities associated with them. Consider two academic researchers in our system - *prof. Alice* and *Bc. Bob*. Bob is a student of Alice and has published several papers on *Information retrieval* with her. Aside from those, Bob has not published any other papers. On the other

hand, Alice has published a lot of papers on various topics - related to IR, but also to other similar fields. See a simple representation of their social network above.

Note that aside from the common publications, Bob has no other entities associated with him, while Alice has other publications with other co-authors.

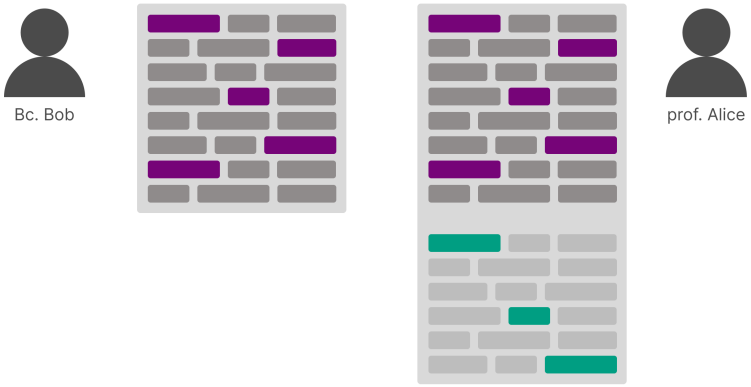

Figure 3.2: Concatenating representations of Alice and Bob as documents.

We see that the document of Bob is shorter than the one of Alice, because he has less associated entities. We also notice that the Alice's document fully contains the Bob's document.

The colored terms represent the current search query - if Alice only published papers on the topic of the query with Bob, the term frequency of the query in Bob's document would be strictly higher, because of the shorter document length. In case Alice also publishes on the topic with other co-authors, it gets harder to reason about which of the term frequencies is higher.

The relation between the tf-idf scores of the documents of Alice and Bob can be expressed as follows:

$$\mathrm{tf}(q, d_{\mathrm{Bob}}) \times \mathrm{idf}(q, D) \overset{?}{>} \mathrm{tf}(q, d_{\mathrm{Alice}}) \times \mathrm{idf}(q, D) \tag{3.1a}$$

$$\mathrm{tf}(q, d_{\mathrm{Bob}}) \overset{?}{>} \mathrm{tf}(q, d_{\mathrm{Alice}}) \tag{3.1b}$$

Normalized term frequency $\mathrm{tf}_{q,d}$ of a term q in a document d is defined as the number of times the term q appears in the document d (denoted by $f_{q,d}$), divided by the total number of terms in the document d (the document's length).

$$\frac{f_{q,d_{\mathrm{Bob}}}}{\mathrm{len}(d_{\mathrm{Bob}})} \overset{?}{>} \frac{f_{q,d_{\mathrm{Alice}}}}{\mathrm{len}(d_{\mathrm{Alice}})} \Big/ \ {}^{-1} \tag{3.2a}$$

$$\frac{\mathrm{len}(d_{\mathrm{Bob}})}{f_{q,d_{\mathrm{Bob}}}} \overset{?}{<} \frac{\mathrm{len}(d_{\mathrm{Alice}})}{f_{q,d_{\mathrm{Alice}}}} \Big/ \ \textit{assuming that } f_{q,d_{\mathrm{Bob}}} = f_{q,d_{\mathrm{Alice}}} \tag{3.2b}$$

$$\mathrm{len}(d_{\mathrm{Bob}}) \ < \ \mathrm{len}(d_{\mathrm{Alice}}) \tag{3.2c}$$

Since we have assumed that Alice's document is a superset of Bob's document, the result inequality holds - and therefore, the tf-idf score of Bob's document is higher than the one of Alice's document.

Note that the assumption in the equation 3.2b is only true for the cases when Alice does not publish any other papers on the topic of the query with other co-authors.

Either way, this approach most likely will not provide the desired outcome - while Bob has published papers only on the given topic - and therefore has higher tf-idf relevance to the query, Alice is likely the more relevant person for the searching user since she is more experienced in the topic.

3.2 Re-ranking

Knowing the theoretical issues of the current Charles Explorer search engine implementation, we try to address them by proposing a system for re-ranking the search results. In the following sections, we go over different existing re-ranking strategies utilized in other systems and explore the possibilities of using social network data for the re-ranking in Charles Explorer.

Even though we have only talked about person entities in the previous sections, we will conduct all the experiments on the ranking of the search results for **publication** records. This is because the publications are uniquely identifiable (unlike people - see 2.2.2 for more details) and they have implicit textual content associated with them - the title, abstract, and keywords.

Note that the issues from 3.1.1 still apply for the publications - Beel et al. [2010] mention different approaches to ASEO. By overusing certain keywords in the publication titles and abstracts, the authors can artificially increase the relevance of their publications to the search queries. Reranking the publications by a less volatile and less falsifiable metric - like the social network data - might help to mitigate this issue.

3.2.1 Existing re-ranking strategies

In the current (2024) commercial search engines, there are often multiple re-ranking strategies available to the users. Unfortunately, only a small portion of the engines actually discloses the details of the re-ranking algorithms used - perhaps to protect the intellectual property and the competitive advantage.

To have a look at the existing re-ranking strategies, we examine the related academic literature on the topic.

Böhm et al. [2011] propose a reranking algorithm for the Wikipedia search engine, ranking the search results based on a current Wikipedia page and its linkage (paths in the web graph) to the search results. Bendersky and Kurland [2008] propose a reranking algorithm for document search using metrics on a *passage graph* - a graph representation of the documents and the passages within them with the edges representing the similarity between the passages.

Famously, Brin and Page [1998] proposed the PageRank algorithm for the Google search engine. The PageRank algorithm itself does not rerank the search results - it is used to produce scoring of the web pages based on the link structure of the web. The ranking of the search results is then based on multiple factors, including the PageRank score of the pages.

Note that reranking is not only applicable in the context of search engines - recommender systems use similar strategies of reordering the presented items based on the user's preferences or other factors. Pei et al. [2019] propose a personalized reranking algorithm for the recommendation systems, which takes into account the user's preferences and the item's popularity. See that unlike in the search engines, the reranking in the recommendation systems is based on the user's interactions with the system, rather on the "global" statistics of the items.

3.2.2 Algorithm

In the previous subsection, we have listed several existing reranking strategies. The common base for all these systems is the **two-stage search pipeline**. In the first stage, a traditional search engine (e.g. TF-IDF based) is used to retrieve the initial set of results. Then, these results are re-ranked using a different algorithm.

While this might seem redundant, this approach allows the second algorithm to focus only on the more relevant results, and be perhaps more computationally expensive. Unfortunately, this also means that the reranking algorithm is not able to affect the initial search results; it can only change the order of the results. It also brings in the issue of pagination - if the user has to go through multiple pages of the search results, the reranking might not be as effective, as it only affects the current page of the results.

Based on this framework, we propose a reranking algorithm for the Charles Explorer search engine:

For a query q, a set of all publications D and a social network G of all the people and publications in the system, we propose the following reranking algorithm:

1. Retrieve the initial list R_q of search results relevant to q, $R_q \subset D$ using the traditional search engine. R_q is an ordered list of publications, ranked by the relevance to the query.
2. For every publication $p \in R_q$, get the auxiliary relevance score $s(p)$ **based on the social network data**.
3. Construct a new list Rank_q with items from R_q and ranking based both on the original relevance scores and the auxiliary relevance scores.

We notice that this algorithm definition is quite abstract - and aside from the mention of the social network data, it does not differ from the definition of the general two-stage reranking algorithm.

However, the third step of the algorithm is not trivial either. The balance of the original relevance scores and the auxiliary relevance scores in the final ranking is a task of *multiobjective optimization* and might be hard to solve in a general

case. Later on, we try to experimentally evaluate the performance of different ranking-combination strategies.

In the following sections, we explore different social network metrics that can be used for auxiliary relevance score $s(p)$ calculation.

3.2.3 Social network metrics for reranking

The academic social network graph is often large and sparse, as academic systems often accumulate many records over their lifetime. This poses a challenge for the social network metrics calculation - the computational complexity of any global graph metric grows with the number of nodes in the graph.

This combined with the need for the real-time response of the search engine makes the calculation of the global graph metrics infeasible. In the following sections, we propose some of the measures defined on close neighborhoods of the nodes in the graph, the performance of which we will later evaluate in the experiments.

Node degree

The *degree* of a node v in a graph G is the number of edges incident to v.

$$\deg(v) = |N(v)|$$

In the case of the publication nodes, the node degree is the number of people that have collaborated on the publication. The possible benefit of using this metric is rather clear - we expect that there might be a correlation between the number of collaborators on a publication and the global relevance of a publication.

Neighborhood-separating node cut size

Similarly to the *node degree*, we define the *neighborhood-separating node cut size* measure. For a given node n and its neighbourhood $N(n)$, we define the *neighborhood-separating node cut size* as the number of nodes (outside of $N(n)$) that have to be removed to separate $N(n)$ from the rest of the graph.

In the context of the academic social network, for a publication p, this counts the number of other publications the authors of p have authored. Note that in case the authors of p have collaborated on some other publications, those still only contribute to the cut size once.

We can also calculate this measure as

$$\text{cut}(n) = |N_2(n)| - |N_1(n)|$$

where $N_k(n)$ is the k-hop neighbourhood of the node n.

Ego betweenness centrality

Betweenness centrality of a node v in a graph G is the fraction of all shortest paths that pass through v.

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

where σ_{st} is the number of shortest paths between nodes s and t, and $\sigma_{st}(v)$ is the number of those paths that pass through v.

While this is usually calculated in the context of the entire graph, it is an useful measure for ego-networks too, as it can help us quantify the importance of a node in its local neighbourhood. Everett and Borgatti [2005] have shown that for real-life networks, the ego betweenness centrality often correlates with the actual global betweenness of a node in the graph.

In our data, the collaboration graph is bipartite - the nodes are either publications or people, and there are no edges between the nodes of the same type. This means that the ego betweenness centrality of a publication is in fact proportional to the number of people that have collaborated on the publication.

2-hop betweenness centrality

Similar to the ego betweenness centrality, we calculate the *2-hop betweenness centrality* as the betweenness centrality of a node in a subgraph induced by the node and its 2-hop neighborhood.

In our case of the bipartite collaboration graph, the *2-hop betweenness centrality* of a publication is no longer proportional only to the number of people that have collaborated on the publication, but also to the *number of other publications that the people have collaborated on.*

Note that this concept can be further extended to the k-hop betweenness centrality, but the computational complexity of the centrality calculation grows exponentially with the k. Materializing the induced subgraphs for the k-hop betweenness centrality calculation also poses a challenge in regard to memory consumption.

In our experiments, we only use the $1-$ and $2-$ *hop* neighborhoods for the betweenness centrality calculation due to the fast growth of the computational complexity with larger k.

Eigenvector centrality

The *eigenvector centrality* of a node v_i in a graph G is the sum of the centrality scores of the nodes that are connected to v.

$$\text{eig}(v_i) = \frac{1}{\lambda} \sum_{v_j \in N(v)} \text{eig}(v_j)$$

where λ is a constant. Note that the definition can also be rewritten as

$$\text{eig}(v_i) = \frac{1}{\lambda} \sum_{v_j \in V} a_{v_i,v_j} \text{eig}(v_i)$$

with a_{vu} being the elements of the adjacency matrix of the graph G.

Denoting $\mathbf{eig}(\mathbf{v})$ as a vector of eigenvector centralities for all nodes in the graph, the equation can be rewritten as

$$\lambda \, \mathbf{eig}(\mathbf{v}) = A \, \mathbf{eig}(\mathbf{v})$$

where A is the adjacency matrix of the graph G. This shows the reasoning behind the name of the centrality measure - the centrality scores are the eigenvectors of the adjacency matrix.

Note that we are mentioning this measure only because of the following mentioned measure. We do not calculate the eigenvector centrality in our experiments, as the computational complexity of the centrality calculation is far too high for the large graphs.

Katz centrality

Katz centrality is a special case of the eigenvector centrality.

Katz [1953] defines the *Katz centrality* of a node v_i in a graph G as

$$\mathrm{katz}(v_i) = \sum_{k=1}^{\infty} \sum_{j=1}^{n} \alpha^k (A^k)_{ij}$$

where $\alpha \in (0, 1/\lambda_{\max})$ is a constant, and $\lambda_{\max}$ is the largest eigenvalue of the adjacency matrix A.

The Katz centrality counts the number of paths between the central node and all other nodes in the graph. For a path of length k is additionally discounted by the *attenuation* factor α^k.

The above formula uses the fact that the power of the adjacency matrix A^k counts the number of paths of length k between nodes i and j.

> As follows from the definitions, the betweenness centrality of a node v is computationally expensive to calculate for large graphs, as it considers all pairs of nodes in the graph.
>
> While e.g. Brandes [2004] offers an algorithm that reduces the computational complexity of the betweenness centrality calculation, the calculation still poses a significant performance bottleneck for large graphs.
>
> On the other hand, the Katz centrality can be calculated more efficiently - since the definition introduces the attenuation factor α, the importance of the nodes that are further away from the central node is discounted and the centrality measure can be approximated by truncating the sum at a certain small value of k.

3.2.4 Combining the metrics

In the third step of our proposed reranking algorithm, we want to combine the original relevance scores of the publications with the auxiliary relevance scores based on the social network metrics.

In our experiments, we compare two different strategies for combining the scores:

1. **Linear combination** - the final relevance score of a publication is a linear combination of the original relevance score and the auxiliary relevance score.

   This includes edge cases of 100% weight on the original relevance score or the auxiliary relevance score.

$$\text{final\_score}(p) = \alpha \cdot \text{original\_score}(p) + (1 - \alpha) \cdot \text{auxiliary\_score}(p)$$

2. **Neural networks** - we train a neural network to predict the final relevance score of a publication based on the original relevance score and the auxiliary relevance score.

   Note that this approach is more complex and computationally expensive than the linear combination, but it might provide better results, as it allows for the non-linear combination of the scores.

3.3 Benchmarking setup

To determine the performance of our proposed re-ranking solution, we establish benchmarks to compare the results of the traditional tf-idf based search engine with the social network enhanced search engine.

3.3.1 Sourcing the golden data

The common denominator for many of the ranking measures - like discounted cumulative gain (DCG) or mean reciprocal rank (MRR) - is the *user interaction*. The user is presented with the search results and picks the most relevant one or scores the results based on the relevance to the query.

Unfortunately, this is not applicable to our case - while we are tracking the user interactions in the Charles Explorer web appplication, the amount of collected data is far too low to be statistically significant.

To establish a gold standard for the search results relevance, we do not have to rely solely on human interactions.

Elsevier Scopus[1] is a large academic database that provides a search engine for academic publications. Aside from the web application, it also provides a REST API for consuming the data programmatically.

We use the Scopus API to retrieve ranked lists of publications for different queries and then use the ranking of the publications as the source of the "global relevance" for the search query in the benchmark. Simply put, by comparing the (ranking of the) search results in our search engine to the results of the Scopus search engine, we determine the relevance of the search results.

We are expecting the search results of the Scopus search engine to be more precise and relevant than the ones of the Charles Explorer search engine - Scopus is a

[1] https://www.elsevier.com/products/scopus

commercial product with a large team of developers and researchers, while Charles Explorer is a small academic project. The data available to Scopus also contain details about the citations of the publications and author profiles, which can be used to further improve the search results ranking.

To partially mitigate the possible bias of the Scopus search engine, we also evaluate the search results of the graph-enhanced reranking against ranking based on the citation count.

We can consider the citation count as a proxy for the global relevance of the publication - the more citations a publication has, the more relevant it is to the academic community.

3.3.2 Sampling the search query set[blog]

As the first step, we need to sample the *search query set* for the benchmark. Since we want to rule out possible biases - or at least mitigate their impact - we need a large and diverse enough set of queries to compare the search engines on.

Generating these manually would be time-consuming and error-prone. Therefore, to solve this issue, we use a *wordnet* - a lexical language database of English. We use it to generate a large set of diverse queries, perhaps less biased than a manually generated set.

We start by selecting a set of *seed words* - in our experimental case, those were the words *"field of study"* and *"medicine"*. Then, we traverse the wordnet to recursively find the hyponyms of those seed words, up to a certain depth.

Running this process for the seed words *"field of study"* and *"medicine"* with a depth of 4, we get a set of `915` search queries.

While this approach gives us a sizable set of queries, we have no guarantee of the quality of the queries - they might be too general or too specific, or not relevant to the academic domain at all. One of our goals was also to ensure the fairness of the query set - this is not guaranteed by the wordnet traversal either, as the queries might be too similar to each other (or target the similar topics in the publications).

Ensuring the query set fairness

While *fairness* is a largely subjective measure, we let the available data guide us in this case. For the *academic publications*, we have their *titles*, *abstracts*, *keywords*, *faculty affiliations*, and *authors* available in our system.

Since titles, abstracts, and keywords are free text fields, we omit them from our analysis - the preprocessing of the text data is a complex task on its own. Given the nature of our experiment - i.e. measuring the impact of using the social network data for the search results ranking - we have to leave the authorship information out as well.

This process leaves us with the faculty affiliations.

Charles University has *17* faculties, each with a different focus and research areas. Each publication in our data is attributed to exactly one faculty. This allows us to use the faculty affiliations as a proxy for the fairness of the search queries.

Kullback-Leibler divergence

As the fairness measure, we compare the distribution of the faculty affiliations in the search query results to the distribution of the faculty affiliations of all the publications in the system.

The standard way of comparing probability distributions is the *Kullback-Leibler divergence* - a measure of how one probability distribution diverges from a second, expected probability distribution.

For discrete probability distributions P and Q defined on the same sample space Ω, the Kullback-Leibler divergence from Q to P is defined as

$$D_{KL}(P||Q) = \sum_{\omega \in \Omega} P(\omega) \log \frac{P(\omega)}{Q(\omega)}$$

The KL-divergence is always non-negative, and is zero if and only if P and Q are the same distribution.

With the measure of the fairness of the search query set established, we now proceed to the benchmarking of the search results ranking in Charles Explorer.

By sampling up to 30 results for each search query from the Charles Explorer search engine, we acquire the faculty distribution for the entire search query set ($N = 915$). We then compare this distribution to the distribution of the faculty affiliations of all the publications in the system.

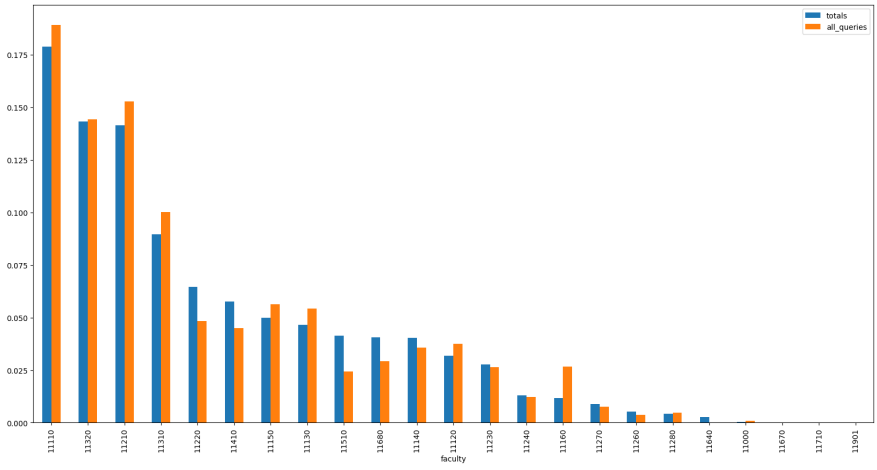

Figure 3.3: Comparing the faculty distribution of the search query results *(orange, right)* to the distribution of all the publications in the system *(blue, left)*.

The Kullback-Leibler divergence of the faculty distribution of the **search query results** from the distribution of **all the publications** in the system is approximately `0.0471`.

Optimizing the KL-divergence

With the defined measure of the fairness of the search query set, we now try to optimize it. In our case, optimizing the KL-divergence means *finding a subset* of search queries which would minimize the divergence of the faculty distribution of the search query results from the distribution of all the publications in the system.

Unfortunately, this poses serious challenges. Finding a subset with an optimal aggregate property is a well-known NP-hard problem - often referred to as the *0-1 knapsack problem* or the *subset sum problem*. Even worse, we cannot simply reuse some of the existing algorithms for these problems, as those rely on the distributivity and associativity of the sum operation. This is, however, not the case for the KL divergence.

Similarly to the sum of the item values (in the Knapsack problem), the KL divergence is evaluated on the entire set, but unlike the sum, the items themselves do not have any "value" - and their contribution to the KL divergence changes depending on the other items in the set. This leaves us with a limited choice of algorithms to solve the problem. Because of the complexity of the problem and its smallish role in this work, we use a simple random search.

This approach works in two steps:

1. Repeatedly sample a random subset of size k of the search queries, keeping track of the subset with the lowest KL divergence. After a fixed number of iterations, take the subset with the lowest KL divergence.
2. From the k-sized subset, repeatedly remove the search query with the highest contribution to the divergence. Stop when the KL-divergence stops decreasing, i.e. we cannot remove any more search queries to decrease the divergence, or we've reached the minimum subset size l.

This is a simple and computationally cheap approach, but it might not always find the optimal solution. Furthermore, since the KL-divergence is a non-convex function, the second step of the algorithm might get stuck in a suboptimal solution. While this could be mitigated using a more complex optimization algorithm (e.g., simulated annealing, etc.), experimental results show that the simple random search is sufficient for our purposes.

Running the optimization algorithm with $n = 10000$ initial random samples of size $k = 300$ on the search query set, we arrive at a set of 174 queries with the search result KL-divergence of `0.00317`. This is a significant improvement over the original divergence of `0.0471`, and we will consider this a fairer subset of the search queries for the benchmarking.

3.3.3 Collecting the data[blog]

Since our proposed benchmark only evaluates the search results ranking, we collect the search results for the benchmark queries in advance. Similarly, we also collect the data from the Scopus API for the same queries, as the automated relevance feedback.

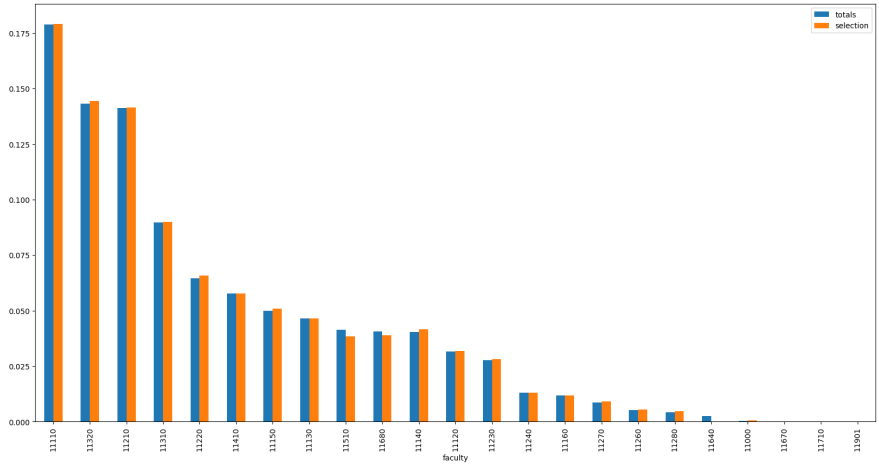

Figure 3.4: Comparing the faculty distribution of the search query results *(orange, right)* to the distribution of all the publications in the system *(blue, left) after running the optimization algorithm.*

While collecting the search results from the Charles Explorer search engine is straightforward since the API is available to us, the Scopus Advanced search feature requires us to use a special query language[2] to submit the search queries. This query language offers a set of Prolog-like functors, each related to a specific field - or a set of fields - of the publication record. The attributes of these functors are used in a *substring search* on the specified fields.

Apart from this, the query language also supports logical operators, such as `AND`, `OR`, and `AND NOT`.

We will use two of the available functors: `TITLE-ABS-KEY` and `AF-ID`:

- `TITLE-ABS-KEY` searches for the specified substring in the title, abstract, and keywords of the publication record. In this regard, it is similar to the full-text search in Charles Explorer, which searches in the same fields.
- `AF-ID` filters the search results by the organization affiliation of the publication. This is useful for filtering the search results to only those publications where at least one of the authors is affiliated with Charles University. Since Elsevier Scopus contains many records not affiliated with Charles University (but Charles Explorer only contains such records), this will help us to get a more comparable set of search results.

By calling the Scopus API, we get the search results in JSON format, which we then process and store in our database for benchmarking.

We see that the `ranking` column - the position of a publication in the search results - is only very weakly correlated with the other numeric attributes of the search results. This suggests that the default Scopus ranking is mostly influenced by full-text search relevance and does not take any further attributes into account. The strongest correlated attribute is the `pubYear` - this suggests that the older publications are ranked higher in the search results. However, the absolute value of the correlation coefficient (`0.109848`) is still very low.

[2] https://schema.elsevier.com/dtds/document/bkapi/search/SCOPUSSearchTips.htm

	ranking	totalAuthors	scopusId	pubYear	citationCount	referenceCount
ranking	1.000000	0.038005	0.081229	0.109848	0.062467	0.053487
totalAuthors	0.038005	1.000000	0.033948	0.040538	0.113336	0.094358
scopusId	0.081229	0.033948	1.000000	0.806411	0.015393	0.243830
pubYear	0.109848	0.040538	0.806411	1.000000	0.033019	0.283521
citationCount	0.062467	0.113336	0.015393	0.033019	1.000000	0.218415
referenceCount	0.053487	0.094358	0.243830	0.283521	0.218415	1.000000

Figure 3.5: Correlation matrix of the Elsevier Scopus search results numeric attributes.

It also suggests that the Scopus result ranking might not be too dependent on the social network measures and that we might not be able to improve the ranking by using the social network data (as the "explicit" social network data like the citation count or the reference count are already not correlated with the ranking).

3.3.4 Simulating relevance feedback[blog]

With the data collected, we now proceed with the actual analysis of the search results ranking in Charles Explorer.

Considering the Scopus search results as the gold standard, we calculate the per-query precision, recall, and F_1 score for the search results of Charles Explorer.

Query	**Precision**	**Recall**	**F_1 score**
physics	0.043011	0.040000	0.041451
bolus	0.125000	0.121212	0.123077
draft	0.010870	0.010753	0.010811
...	...	...	...

Figure 3.6: Per-query precision, recall and F_1 score for the search results of Charles Explorer.

After aggregation over all the queries, this gives us the following unfavorable statistics:

Mean	0.208727
Standard deviation	0.211699
Minimum	0.010101
25%	0.074786
50%	0.137028
75%	0.265263
Maximum	1.000000

Figure 3.7: Aggregated statistics of the F_1 score for the search results of Charles Explorer.

We see that the current Charles Explorer search results differ quite a lot from the Scopus search results. This can be caused by mutiple reasons - either the publications are not present in the Scopus database, or the queries are not specific enough and the search results are returning partially disjoint sets of publications.

Note that this is an issue that goes beyond re-ranking the Charles Explorer search results. We cannot quantify the benefit of reordering the search results if we consider all the search results irrelevant. This hinders our ability to use the Scopus search results ranking as the proxy for the relevance feedback.

Since this thesis is focused on the search result ranking algorithms, we will proceed with the benchmarking as planned. However, to improve the relevance score assignment, we add a *similarity search* step.

In the F_1 score calculation, we are currently only matching the Charles Explorer search results with the Scopus search results by the *publication title* (case-insensitive). This matching criterion is prone to even the slightest variations in the publication titles, which can lead to false negatives.

Inferring the publication relevance with semantic search[blog]

In the proposed *similarity search* step, we use the similarity of LLM (Large Language Model) embeddings to match the publication titles. This should help us to relate the publications missing from the Scopus search results to the ones present there and assign them a relevance score.

> **LLM embeddings**
>
> *LLM embeddings* are vector representations of a given text, generated by a large language model. While those can be arbitrary vectors, embeddings are usually optimized to capture the semantic meaning of the text.
>
> This means that texts with similar meanings should have similar embeddings - i.e. the (cosine) similarity of the embedding vectors should be high.

We enhance the relevance calculation with the similarity search process as follows:

1. By means of an *LLM embedding model*, we precalculate the embeddings for the publication titles of the Elsevier Scopus search results. We store these embeddings in a vector database.
2. For each publication title in the Charles Explorer search results, we calculate its embedding. In the database, we search for the nearest embedding among Scopus search results embeddings. Furthermore, we require the retrieved document to be a result of the same query (in Elsevier Scopus) as the Charles Explorer search result.
3. We calculate the Charles Explorer document's inferred relevance from the most similar document's attributes - e.g., its position in Scopus search.

For the document embedding, we use the `all-MiniLM-L6-v2`[3] sentence - transformer model. This is a general-purpose English embedding language model designed for running on consumer-grade hardware. Due to its small size and competitive performance, it's often used for the real-time use-cases, like semantic search or RAG (Retrieval-Augmented Generation).

[3] https://www.sbert.net/docs/sentence_transformer/pretrained_models.html

For the similarity search on the embeddings, we use the ChromaDB database[4]. ChromaDB is a vector database designed for the similarity search on the embeddings, with support for enhancing the search results with the additional metadata attributes of the documents.

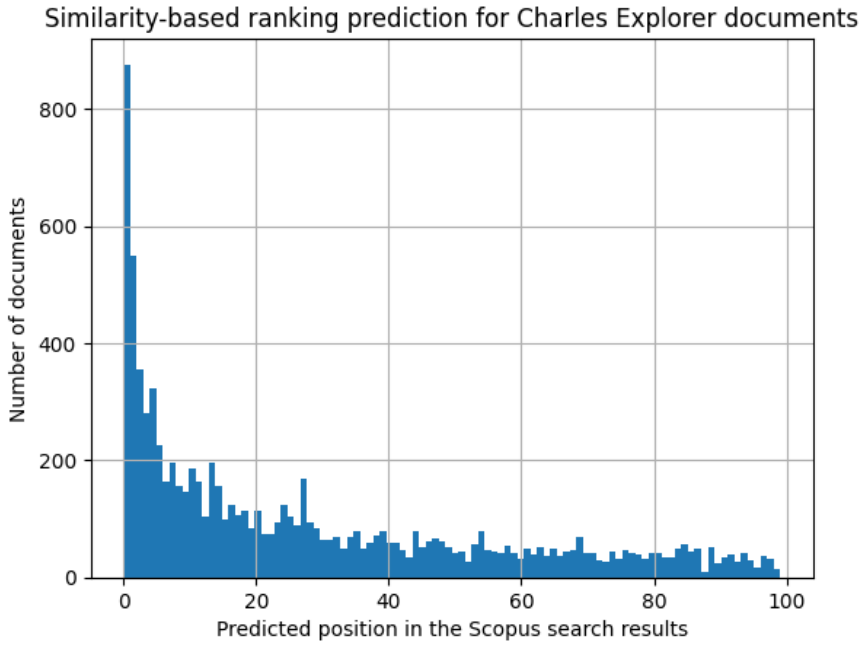

Figure 3.8: Histogram of inferred positions of the Charles Explorer search results in the Scopus search results.

This process gives us the predicted rankings, which follow a rather skewed distribution. However, this does not pose a serious problem to our benchmark.

Firstly, we are not trying to predict the exact ranking of the search results, but rather to assign a relevance score to each search result. The peak of the distribution is at the top of the rankings, which is in line with the well-known tendency of human users to have a much clearer opinion about the few top results than the long tail of the search results (as described by, e.g., Su et al. [2021]).

Secondly, the left-skewed distribution might be caused by the non-uniform lengths of the search result lists. Since for some of the queries, Scopus returns only a few relevant search results (100 is only the maximum limit), the resulting predicted rankings will be skewed towards the top of the list for these queries.

3.4 Evaluation

Using the original ranking positions and the predicted ranking positions as the source for the relevance feedback, we calculate the *nDCG* (Normalized Discounted Cumulative Gain) score for the search result ranking.

DCG score for a single search result list is calculated as the sum of the relevance scores of the search results, discounted by their position in the ranking.

[4] https://www.trychroma.com/

$$DCG_{\text{list}} = \sum_{i=1}^{N} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

The IDCG score is the DCG score of the ideal ranking of the search results, i.e. the items in the list are sorted in descending order by rel_i.

The normalized DCG score is then calculated as the ratio of the DCG score to the IDCG score.

$$nDCG_{\text{list}} = \frac{DCG_{\text{list}}}{IDCG_{\text{list}}}$$

To transform the predicted Scopus rankings from 3.3.4 into relevance feedback, we introduce a new function $rel_q(d)$.

For a given query q, the document d is considered to have relevance of $rel_q(d)$, which is *inversely proportional* to its predicted ranking. This is necessary for the *nDCG* score calculation, which requires more relevant documents to have higher relevance scores.

The inverse proportionality is achieved by the following formula:

$$rel_q(d) = \frac{a}{\text{rank}_q(d) + 1}$$

where a is a constant that scales the relevance scores and can help achieve better stability of the nDCG score with respect to rounding errors.

For the purpose of the experiments in this thesis, we set $a = 5$. The $+1$ in the denominator is necessary to avoid division by zero, as our rankings are $0-$ based.

While it would be possible to achieve the ranking $\rightarrow$ relevance transformation via e.g. subtracting the predicted ranking from the total number of search results, our proposed method with $rel_q(d)$ introduces a *non-linear* transformation of the predicted rankings. This differentiates better between the search results that are ranked higher in the Scopus search results. This is again in line with the aforementioned tendency of human users to have clearer opinions about the top search results.

3.4.1 Baseline benchmark

Calculating `nDCG` over search results for individual queries in Charles Explorer with the predicted relevance scores from 3.3.4 gives us the following results:

The mean nDCG score of 0.761607 suggests that the search result ranking in Charles Explorer already works well - and that the relevance feedback based on the predicted Scopus rankings gives us a good approximation of the relevance of the search results.

	dcg	idcg	ndcg
mean	14.919819	19.167405	0.761607
std	16.810894	17.665142	0.180979
min	0.094340	0.094340	0.405669
25%	5.250473	7.704989	0.627563
50%	9.527864	14.840570	0.736246
75%	18.064385	24.112511	0.934206
max	104.693354	104.693354	1.000000

Figure 3.9: Aggregated statistics of the nDCG score for the original search results of Charles Explorer (*query count* = 149).

3.4.2 Using graph metrics for re-ranking

With the relevance feedback and baseline benchmark values established, we can now proceed with the re-ranking of the search results in Charles Explorer using the social network data. After retrieving the metrics listed in Subsection 3.2.3, we train two models for aggregating the social network metrics and the original relevance scores to predict the final relevance scores.

As mentioned in Subsection 3.2.4, we use two different approaches for the aggregation.

Linear combination: The first proposed solution suggests a linear combination of the original relevance scores and the auxiliary relevance scores. To find the best coefficients of the linear combination, we use a simple least squares regression model.

By training a regularized linear regression model on the training data ($n = 9600$) (i.e. original relevance-based ranking and the social network metrics), we find the coefficients that minimize the mean squared error (MSE) of the predicted relevance scores.

Evaluation of this model on the test data ($n = 2265$) gives us a MSE of 2.1016. While this is not interpretable as is - since we are learning an arbitrary relevance score, not a ranking - we can use the predicted relevance scores to rerank the search results and calculate the nDCG score.

Furthermore, the simplicity of the linear regression model allows us to inspect the internal coefficients learned by the model and to interpret the importance of the individual features for the final relevance score.

From the coefficients, we can see that the most important feature for the final relevance score is the original relevance score - i.e. the ranking of the search results in Charles Explorer. This seemingly confirms our hypothesis that the Scopus search result ranking is mainly influenced by the relevance of the search results themselves, rather than any other - more global - publication measures.

Neural networks: The second proposed solution suggests training a neural network to predict the final relevance scores based on the original relevance scores and the social network metrics. As mentioned before, this might help with finding the non-linear relationships between the features and the final relevance scores.

Feature	Coefficient
charles_explorer_ranking	-0.23068
centrality_1	0.08317
centrality_2	0.01018
degree	-0.08756
katz_centrality	-0.00491
node_cut	-0.02733

Figure 3.10: Coefficients of the linear regression model for the linear combination of the original relevance scores and the social network metrics.

On the other side, such a model is less interpretable than the linear regression model.

Training a neural network (2 hidden layers, 100 neurons each) on the same training dataset ($n = 9600$) as the linear regression model gives us a MSE of 2.0498 on the test data ($n = 2265$). We see that in this regard, the neural network model outperforms the linear regression model only marginally. This further suggests that the relationships between the features and the final relevance scores are rather simple and can be captured by a linear model.

Using both models to aggregate the new "predicted" relevance scores, we calculate the nDCG score for the re-ranked search results. We compare the NDCG scores of the re-ranked search results to the baseline benchmark to determine the performance of the social network-enhanced search engine.

	Baseline nDCG	Linear model	Neural network
count	149.000000	149.000000	149.000000
mean	0.761607	0.746176	0.770519
std	0.180979	0.172590	0.178775
min	0.405669	0.404108	0.425830
25%	0.627563	0.618646	0.615640
50%	0.736246	0.735163	0.759933
75%	0.934206	0.904977	0.955842
max	1.000000	1.000000	1.000000

Figure 3.11: Comparison of the nDCG scores of the baseline search results and the search results re-ranked using the linear regression and neural network models.

We see that the nDCG score for both the models utilizing the graph measures is lower - or comparable - to the nDCG score of the original search results.

This shows that the graph measures we have collected do not provide much useful information for the reranking of the search results.

Note that while the evaluation of the reranking performance is done on the entire set (including the linear regression / neural network training set), the results are still not any better than the original search results. This hints at the high dimensionality of the problem and the lack of generalization of either of the models.

As mentioned before, the Scopus search result ranking is likely mainly influenced by the relevance of the search results themselves, rather than any other - more global - publication measures. In a way, the fact that the graph measures do not help with the reranking does not come as a surprise. This might also be partially caused by the title similarity search step, which helps with the missing publications problem.

This shows that the Scopus search result ranking is likely not a good proxy for the relevance feedback, if searching for a global relevance measure of the publications.

3.5 Predicting the citation count[blog]

As we have noticed before, the search results ranking we have acquired from Scopus is mostly based on raw query relevance. Because of this, it might also be riddled with the problems we have discussed in Subsection 3.1.1 - such as the susceptibility to (potentially malicious) search engine optimization.

In search of a better ranking system for Charles Explorer, we try to predict the *citation count* of the papers based on the local graph measures we have calculated in the previous section. This decision comes from the idea that the *citation count* is a good indicator of the importance of the paper that cannot be easily interfered with by the authors.

Unlike the search result ranking optimization from the previous sections, the citation count inference is a popular research topic in the field of bibliometrics. Notably, Abbasi et al. [2011] explores the influence of small-scale graph neighborhoods of authors on their academic performance and the citation count of their papers.

3.5.1 Sourcing the data

Unfortunately, in the data collected for the previous sections' benchmarking, we do not have the citation count of the publications. Reusing the Scopus API access from Subsection 3.3.3, we can retrieve the citation count for the publications in our system.

This shrinks the dataset to $n = 962$ publications, as the Scopus API does not contain all the publications in our system. While this is a significant reduction in the dataset size, we still consider this representative and large enough for the purposes of this thesis.

3.5.2 Training the models

As in the previous section, we train two models for predicting the citation count of the publications based on the local graph measures.

Linear regression: The first model is a simple linear regression model, which tries to predict the citation count of the publications based on the local graph measures. The model is trained on the training dataset ($n = 768$) and evaluated

on the test dataset ($n = 194$). On the test dataset, the model achieves a MSE of 6725.05077.

As in the previous experiment, the model allows us to inspect the coefficients of the individual features and interpret their importance for the prediction of the citation count.

Feature	Coefficient
centrality_1	13.60649
centrality_2	-12.04390
degree	35.42298
katz_centrality	-5.46456
node_cut	3.06015

Figure 3.12: Coefficients of the linear regression model for predicting the publication citation count.

Note that from the linear regression coefficients, it seems that the `degree` of the publication is the most important feature for the prediction of the citation count (all the features are normalized to the same scale before training the regression). This is in line with the findings of Abbasi et al. [2011], which - while inspecting slightly different graph measures - also found the `degree` of the publications to be the most important feature for the prediction of the citation count.

Neural network: The second model is a neural network model, which tries to predict the citation count of the publications based on the local graph measures. Training a neural network (2 hidden layers, 100 neurons each) on the training dataset ($n = 768$) gives us a MSE of 7425.57273 on the test data ($n = 194$).

Note that in terms of the MSE, neither of the models is performing especially well - in the absolute numbers of root mean squared error (RMSE), the models are off by approximately 80 citations on average. This is a significant error, considering that the average citation count in the dataset is 30.

3.5.3 Citation-based ranking[blog]

Note that despite the unfavorable results of the previous experiments - i.e., predicting the absolute value of the citation count - we might still be able to use the citation count as a proxy for the global relevance of the publications.

This is beause the ranking benchmarks (and human users of search engines) are usually not be interested in the exact citation count, but rather in the relative importance of the publications. Both benchmarks and users also tend to discount the publications further down in the search result list.

Because of our original goal, i.e. reorder the search results to set the more "globally" important papers higher, we might try to use the predicted citation count as the relevance score for the search results ranking.

In the last experiment, we calculate the nDCG score of the search results in Charles Explorer with the true citation count as the relevance feedback and the predicted citation count based on the graph metrics as the ordering.

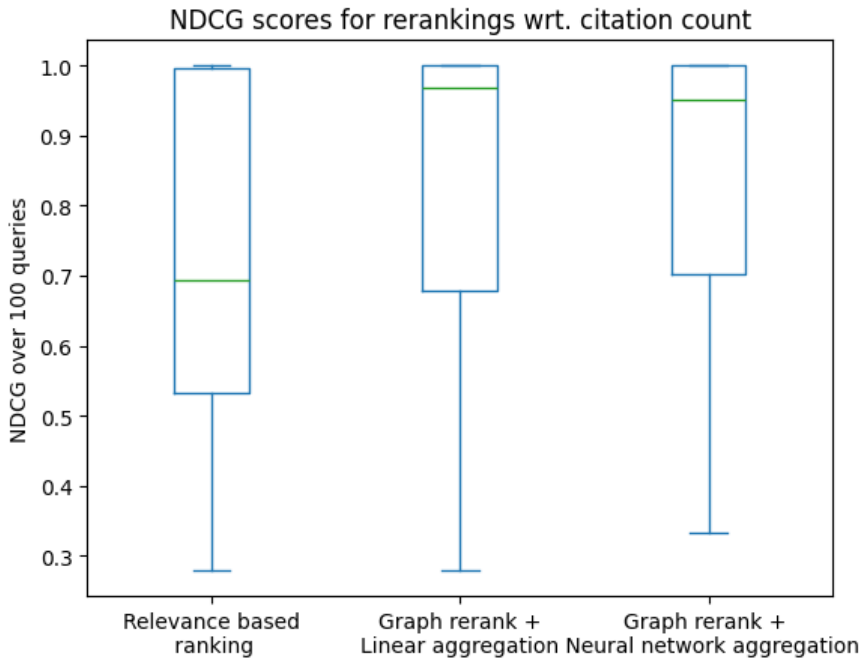

Figure 3.13: nDCG scores of 100 queries with different citation count prediction methods.

We see that both the methods utilizing the graph measures for the citation count prediction are performing much better than the baseline benchmark. This confirms our hypothesis from the beginning of this subsection - while the graph measures might not provide enough data for the absolute prediction of the citation count, they can still be utilized to rank the publications.

4. Visualising networks on the Web

Even though the data mining processes described in the previous chapters give us valuable insights into the structure of social networks, they are not necessarily easy to interpret for laymen. One way to make the results of data mining more accessible is to create data visualization, that is present the data with their visual representation, while using different visual cues to guide the viewers' attention towards different qualities.

In this part of the thesis, we will assess the current state of the visualizations in the Charles Explorer application, will propose some improvements to make the visualization more accessible to the users and will implement those.

At the moment of publishing this thesis, the person search mode in the Charles Explorer application[1] already contains the experimental prototype of the changes proposed in this chapter. For consistency, we will still consider the "current state" of the visualization the one *without* the proposed changes.

The implementation of the visualization components is available in the GitLab repository[2].

> **Visual decoding** Also called *preattentive processing* or *preattentive vision*, visual decoding is the instantaneous perception of the visual field that comes without apparent mental effort. (Cleveland and Mcgill [1985])

4.1 Assessing the current state

The current state of the Charles Explorer visualization views is quite simple.

In the *Person* search mode, the user can search for people inside Charles University. When accessing a person's profile, the application shows the person's *ego network* with the main person and their direct collaborators as nodes and their common publications aggregated to the edges.

The graph is displayed with a force-directed layout. The edge thickness is proportional to the number of common publications between the two people and the colors of the nodes represent the person's faculty association.

This approach has multiple drawbacks which we will now discuss.

4.1.1 Problems with color coding

Firstly, the color coding of the nodes does not prove useful, as it hinders the *visual decoding* of the graph. The user spends attention on reading the legend rather than interpreting the graph subconsciously.

[1] Available, e.g., at https://explorer.cuni.cz/person/1732969562160398

[2] At https://gitlab.mff.cuni.cz/barj/charles-explorer/

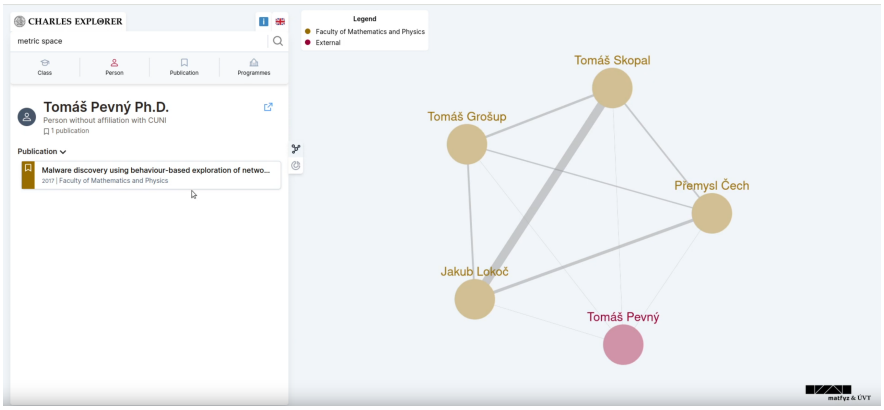

Figure 4.1: Charles Explorer showing the *ego network* of a person.

This is especially true for larger ego networks with many nodes with different faculty affiliations. Additionaly, the application does not provide any alternative visual cue for color vision deficient users.

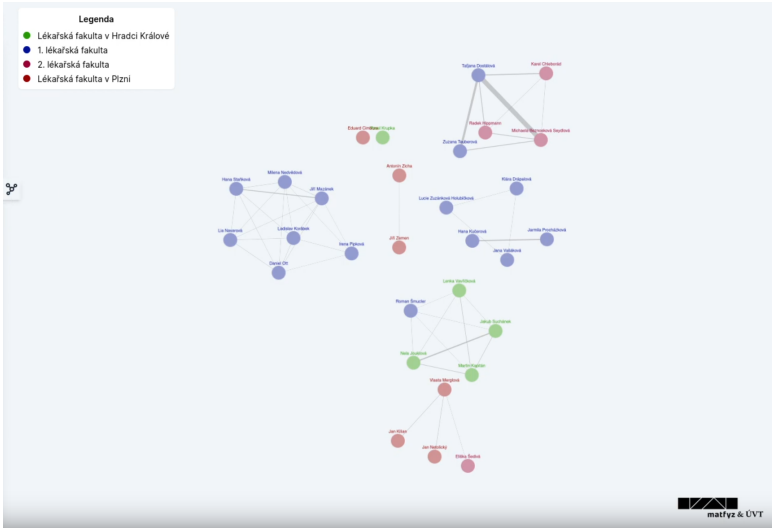

Figure 4.2: Graph view for query *'dentistry'* shows nodes with various faculty affiliations.

According to Cleveland and Mcgill [1985], the upper bound on color discrimination in one figure is 5–6 colors for a healthy viewer. This is not enough for the 17 faculties and departments of Charles University. With faculties, there is also little room for a meaningful aggregation (of more faculties into one color), as the faculty structure is not hierarchical.

4.1.2 Layouting problems

The arbitrary positions of the nodes based on the physical simulation layout increase the cognitive load of the viewer and contribute to the graph's worse readability.

Munzner [2015] states that the position of data points on a common scale is the most effective way to communicate the data to the viewer. By using the physical

simulation layout, we are willingly giving up this visual channel (the `x` and `y` position of the nodes in the screen space).

In case the user is looking for a specific person in the graph, they have to scan the whole graph to find the person, as the node position does not code any inherent information about the person.

4.1.3 Contracting publication nodes into edges

The current data visualization effectively presents a monopartite projection of the social network - uses authors as the nodes and contracts publications as the edges. The number of common publications is aggregated to the edge thickness. Few [2012] considers thickness a visual channel with a limited quantitative resolution.

Setting the edge width in proportion to the number of common publications might also cause confusion, as the *area* of the edge depends both on the width and the length of the edge. Longer edges might appear more important than the shorter ones, even though they might denote the same number of common publications.

This is undesirable, as it goes against the underlying idea of the physical simulation layout, which places the nodes closer to each other if they are more connected.

Furthermore, the contraction of the publication nodes into edges poses another problem. A publication with n authors contributes to $\frac{n(n-1)}{2}$ edges between its authors.

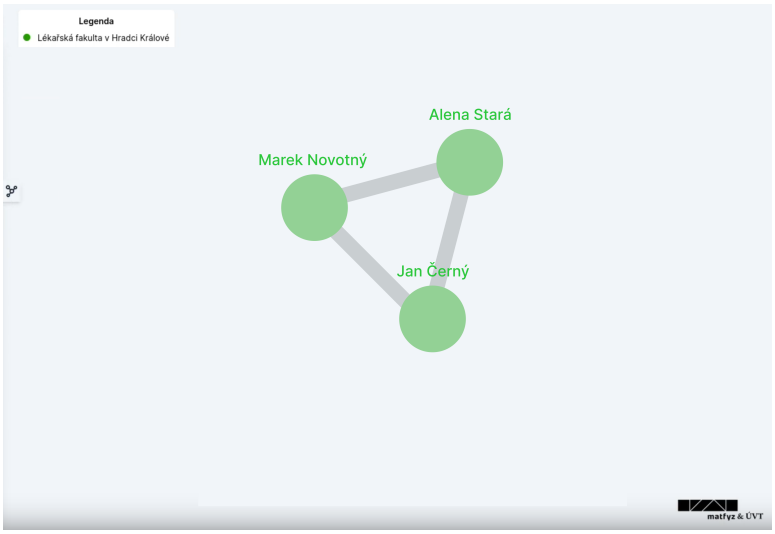

Figure 4.3: The monopartite projection correctly shows only coauthorship between pairs of authors. Larger communities of coauthors (# coauthors > 2) are not displayed correctly in the graph.

This means that publications with one author are not represented in the graph at all, as there are no edges to draw between the nodes.

For publications with more than two authors, one publication is represented by multiple edges between all the pairs of the authors. This clutters the graph with edges and makes the publication-edge mapping less readable. For correct

representation of those publications, we would need to draw hyperedges between the nodes, which might cause the readability of the graph to decrease even further.

4.2 Addressing the issues

To address the problems from the section 4.1 and improve the visualization of the ego networks, we propose changes to the current visualization. We also implement an experimental prototype visualization of the proposed changes.

4.2.1 Ego-network visualization

To start, we can address the problem of the *publication-edge contraction* by displaying the publications as nodes in the graph.

This way, both the entity types (authors and publications) are displayed as *nodes* in the visualization. This is perhaps easier to understand for a layman, as both authors and publications are real-life entities. The edges between the nodes now represent an incidence relation between the two entities.

The resulting graph is a bipartite graph, with two types of nodes - authors and publications, and edges connecting the authors to the publications they have co-authored.

To support the *visual decoding* of the graph, we distinguish the two types of nodes by their shape. This way, the user can easily distinguish between the authors and the publications, even if they are color-vision deficient or are viewing the visualization reproduced using a monochrome display medium (for example a print from a black-and-white printer).

The main node (the ego) is highlighted with color fill - since it is the only node of this type in the graph, it is easy to distinguish from the other nodes - even with monochrome display mediums or color vision impairments.

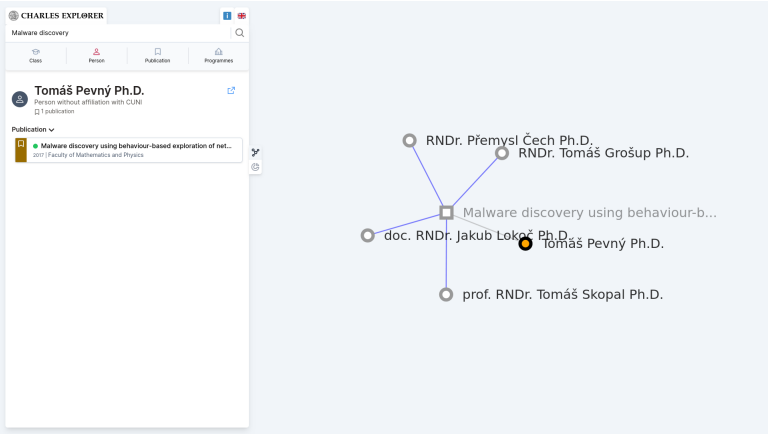

Figure 4.4: In the proposed visualization, the publications are displayed as nodes in the graph. *Larger-than-binary* coauthorships are now represented correctly.

This approach also removes the need for the edge-width encoding of the number of common publications.

To avoid the color coding problems from 4.1.1, we defer the faculty affiliation information to the node tooltip. This is only visible after the user hovers over the node with the mouse cursor, so it does not clutter the graph view.

4.2.2 Node locality and layouting

As mentioned in 4.1.2, the arbitrary positions of the nodes in the force-directed layout increase the cognitive load of the viewer. For larger graphs, the viewer might not be able to find the node they are looking for, as the node position does not code any inherent information about the person.

To address this, we propose a search tool that highlights the search results in the graph.

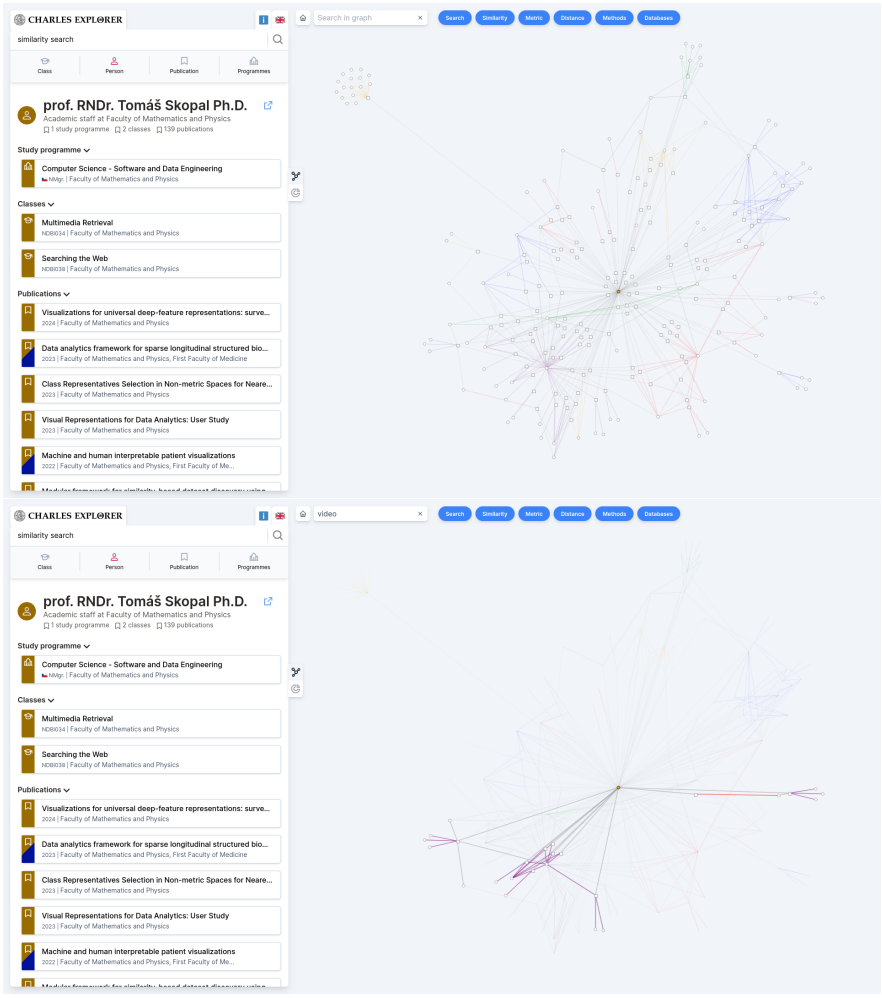

Figure 4.5: The first picture shows a large ego network with many nodes. The second picture shows the same network with entities highlighted by the search tool (search for *video* shows related publications and co-authors).

To further improve the graph readability and the usability of the tool, we propose a *query suggester*. This is a user interface element consisting of multiple

buttons with predefined queries, which the user can click to search for the query in the graph. The suggested queries are based on tf-idf analysis of the text data connected to the ego network (publication titles and abstracts).

The tokens (unigrams and bigrams) with the highest tf-idf score are selected as the suggested queries.

4.2.3 Faculty affiliation

While we have already addressed the color coding issues from 4.1.1 by introducing the on-hover tooltip, this has removed some of the information from the graph view.

In the original visualization, the faculty affiliation for people was displayed as the node color. This has helped the user to quickly identify the faculty affiliation of the ego's collaborators, but also to see the distribution of the faculty affiliations in the ego network, and identify interesting collaboration patterns. This is no longer possible with the new tooltip-based approach.

To address this, we propose a new visualization of the faculty affiliation data. Each co-author of the ego has a faculty affiliation assigned to them, along with the number of common publications with the ego.

For visualizing the distribution of faculty affiliations in the ego network, we can use a *pie chart*.

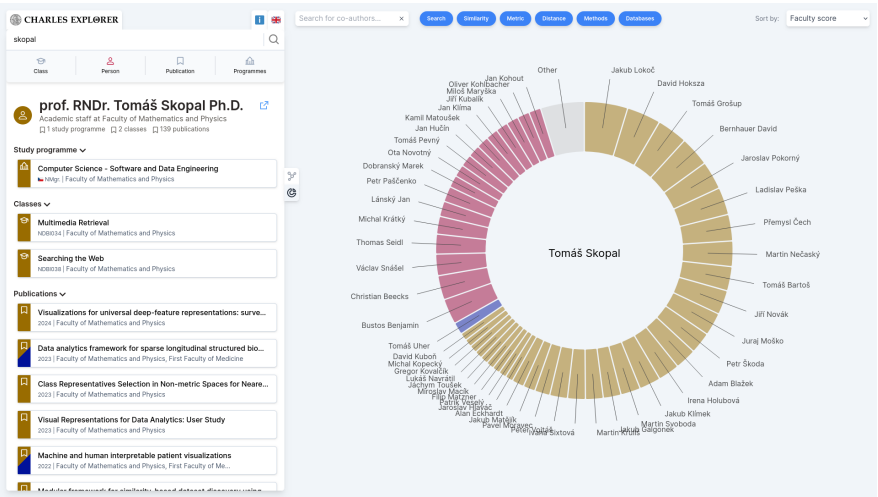

Figure 4.6: The *pie chart* view shows the distribution of faculty affiliations in the ego network, as well as the most frequent collaborators of the ego.

This visualization is still suffering from some of the problems of the original one.

A publication with n authors still contributes to $(n-1)$ pie chart arcs. This causes the solo publications to not be represented in the pie chart at all, and the publications with more than two authors to be overrepresented.

The color coding of the pie chart arcs (denoting the faculty affiliation) is also not ideal since it still poses a problem for color-vision deficient users.

To address these issues, we add a few more features to the pie chart view.

Firstly, we reuse the tooltip user interface element from the graph view. When the user hovers over the pie chart arc, they can see the faculty affiliation name and the number of co-authored publications.

This helps both the color vision deficient users with identifying the faculty affiliation, and the general users with understanding the scale of the collaboration.

Secondly, we add an on-click boolean `AND` filter. When the user clicks on the pie chart arc, the view filters the underlying data to only represent co-authors of both the ego and the clicked-on co-author.

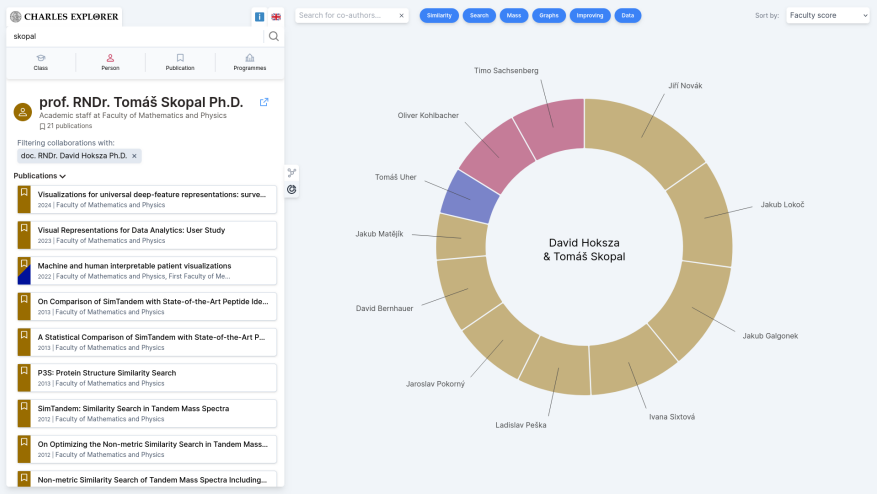

Figure 4.7: Right-clicking on the pie chart arc shows the intersection of the ego's and the clicked-on person's collaborators.

The `AND` filter can be used repeatedly, adding new "pinned" collaborators one at a time. Focusing on a smaller subset of the ego network can help to mitigate the problems with the overrepresentation of the publications with more than two authors helping the user to understand the smaller-scale collaboration patterns better.

The filter also applies to the left side of the view, where the user can see the actual publications of the selected co-authors. This further clears up the multiple co-authorship problem.

Conclusion

The goal of this thesis was to improve the Charles Explorer web application - both in terms of user experience and the search engine - using the synthesized *academic social network* from the existing relational data.

We have devised a highly performant pipeline for transforming the relational data into a graph representation of the academic social network. After the transformation, we explored the network and proposed various ways of utilizing the network metrics for inferring missing data.

Surprisingly, the baseline naïve methods dominated the network-based methods in the task of inferring missing data. This is likely due to the specific distribution of person names in our testing dataset. The improved (context-aware) naïve approach, showed even more promising results and is now a part of the Charles Explorer application.

With the synthesized social network in place, we expressed concerns about the quality of full-text search-based ranking, and we explored the possibility of using the network for re-ranking the search results in the Charles Explorer application.

Deeper comparison with search results of the Elsevier Scopus academic search engine revealed that professional search engines seemingly use only full-text search based ranking too.

By retrieving *citation counts* from the Scopus database, we were able to benchmark different re-ranking strategies using the social network metrics against an objective measure of academic impact.

While these experiments yielded promising results - as the search result reranking based on the social network metrics did outperform the full-text search based ranking - the computational performance of the re-ranking strategies did not satisfy the requirements for the usage in the production environment.

Lastly, we have assessed the user experience of the Charles Explorer application and identified the problematic parts of the existing visualization tool for the academic social network.

We reimplemented the tool for visualizing the academic social network in the Charles Explorer application. The tool now better adheres to the visualization principles of graph data, it is more user-friendly and performant. By simplifying certain parts of the visualization, we improve the interpretability of the visualized data.

This visualization tool became a part of the Charles Explorer application and is now available to users.

Bibliography

Alireza Abbasi, Jörn Altmann, and Liaquat Hossain. Identifying the effects of co-authorship networks on the performance of scholars: A correlation and regression analysis of performance measures and social network analysis measures. *Journal of Informetrics*, 5(4):594–607, 2011. ISSN 1751-1577. doi: https://doi.org/10.1016/j.joi.2011.05.007. URL https://www.sciencedirect.com/science/article/pii/S1751157711000630.

Joeran Beel, Bela Gipp, and Erik Wilde. Academic search engine optimization (aseo). *Journal of Scholarly Publishing*, 41:176–190, 01 2010. doi: 10.3138/jsp.41.2.176.

Michael Bendersky and Oren Kurland. Re-ranking search results using document-passage graphs. pages 853–854, 07 2008. doi: 10.1145/1390334.1390539.

Chris Bennett, Jody Ryall, Leo Spalteholz, and Amy Gooch. The aesthetics of graph visualization. In *CAe*, pages 57–64, 2007.

Ulrik Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25, 03 2004. doi: 10.1080/0022250X.2001.9990249.

Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. volume 30, 11 1998. doi: 10.1016/j.comnet.2012.10.007.

Christoph Böhm, Eyk Kny, Benjamin Emde, Ziawasch Abedjan, and Felix Naumann. Sprint: ranking search results by paths. pages 546–549, 03 2011. doi: 10.1145/1951365.1951437.

Oguz Cimenler, Kingsley A. Reeves, and John Skvoretz. A regression analysis of researchers' social network metrics on their citation performance in a college of engineering. *Journal of Informetrics*, 8(3):667–682, 2014. ISSN 1751-1577. doi: https://doi.org/10.1016/j.joi.2014.06.004. URL https://www.sciencedirect.com/science/article/pii/S1751157714000571.

William Cleveland and Ron Mcgill. Graphical perception and graphical methods for analyzing scientific data. *Science (New York, N.Y.)*, 229:828–33, 09 1985. doi: 10.1126/science.229.4716.828.

Martin Everett and Stephen Borgatti. Ego network betweenness. *Social Networks*, 27:31–38, 01 2005. doi: 10.1016/j.socnet.2004.11.007.

Stephen Few. *Show Me the Numbers: Designing Tables and Graphs to Enlighten*. Analytics Press, Oakland, CA, USA, 2nd edition, 2012. ISBN 0970601972.

Max Franz, Christian T Lopes, Gerardo Huck, Yue Dong, Onur Sumer, and Gary D Bader. Cytoscape.js: a graph theory library for visualisation and analysis. *Bioinformatics*, 32(2):309–311, 2016.

Mark Glick, Greg Richards, Margarita Sapozhnikov, and Paul Seabright. How does ranking affect user choice in online search? *Review of Industrial Organization*, 45(2):99–119, Sep 2014. ISSN 1573-7160. doi: 10.1007/s11151-014-9435-y. URL https://doi.org/10.1007/s11151-014-9435-y.

Sacha Greif and Eric Burel. State of javascript 2023, other tools - graphics and animation, 2024. URL https://2023.stateofjs.com/en-US/other-tools/#graphics_animation.

Brynjar Gretarsson, Svetlin Bostandjiev, John O'Donovan, and Tobias Höllerer. Wigis: A framework for scalable web-based interactive graph visualizations. In *Graph Drawing: 17th International Symposium, GD 2009, Chicago, IL, USA, September 22-25, 2009. Revised Papers 17*, pages 119–134. Springer, 2010.

Michael Gusenbauer. Google scholar to overshadow them all? comparing the sizes of 12 academic search engines and bibliographic databases. *Scientometrics*, 118: 177–214, 01 2019. doi: 10.1007/s11192-018-2958-5.

Hatem Haddad and Amna Dridi. Social relevance for re-ranking documents of search engines results. 06 2013.

Lamjed Jabeur, Lynda Tamine, and Mohand Boughanem. A social model for literature access: Towards a weighted social network of authors. 01 2010.

Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, Mar 1953. ISSN 1860-0980. doi: 10.1007/BF02289026. URL https://doi.org/10.1007/BF02289026.

Vít Macháček and Martin Srholec. Predatory publishing in Scopus: Evidence on cross-country differences. *Quantitative Science Studies*, 3(3):859–887, 11 2022. ISSN 2641-3337. doi: 10.1162/qss_a_00213. URL https://doi.org/10.1162/qss_a_00213.

Alireza Mansouri, Lilly Suriani Affendey, and Ali Mamat. Named entity recognition approaches. *International Journal of Computer Science and Network Security*, 8(2):339–344, 2008.

T. Munzner. *Visualization Analysis and Design*. AK Peters Visualization Series. CRC Press, 2015. ISBN 9781498759717. URL https://books.google.de/books?id=NfkYCwAAQBAJ.

Giannis Nikolentzos, George Dasoulas, and Michalis Vazirgiannis. k-hop graph neural networks, 2019.

Alexander J. Ordoobadi, Numa Perez, Maggie Westfal, David C. Chang, and Cassandra Kelleher. Social network analysis of authors in scientific journals. *Journal of the American College of Surgeons*, 229(4, Supplement 1):S164, 2019. ISSN 1072-7515. doi: https://doi.org/10.1016/j.jamcollsurg.2019.08.362. URL https://www.sciencedirect.com/science/article/pii/S107275151930818X. Scientific Forum Abstracts: 2019 Clinical Congress.

Enrique Orduna-Malea, Juan Ayllón, Alberto Martín-Martín, and Emilio Delgado López-Cózar. Methods for estimating the size of google scholar. *Scientometrics*, 104, 06 2015. doi: 10.1007/s11192-015-1614-6.

Changhua Pei, Wenwu Ou, Dan Pei, Yi Zhang, Yongfeng Zhang, Fei Sun, Xiao Lin, Hanxiao Sun, Jian Wu, Peng Jiang, and Junfeng Ge. Personalized re-ranking for recommendation. pages 3–11, 09 2019. ISBN 978-1-4503-6243-6. doi: 10.1145/3298689.3347000.

Robert Sokal and F. Rohlf. The comparison of dendrograms by objective methods. taxon 11: 33-40. *Taxon*, 11:33–40, 02 1962. doi: 10.2307/1217208.

Zhan Su, Zuyi Lin, Jun Ai, and Hui Li. Rating prediction in recommender systems based on user behavior probability and complex network modeling. *IEEE Access*, 9:30739–30749, 2021. doi: 10.1109/ACCESS.2021.3060016.

Jonathan Tennant. Web of science and scopus are not global databases of knowledge. *European Science Editing*, 46, 10 2020. doi: 10.3897/ese.2020.e51987.

Christian Tominski, James Abello, and Heidrun Schumann. Cgv—an interactive graph visualization system. *Computers & Graphics*, 33(6):660–678, 2009.

Manjula Wijewickrema. Reality or illusion: Comparing google scholar and scopus data for predatory journals. *portal: Libraries and the Academy*, 24:35–58, 01 2024. doi: 10.1353/pla.2024.a916989.

List of Abbreviations

ACM Association for Computing Machinery

ASEO academic search engine optimization

CUNI Charles University

DCG discounted cumulative gain

DOI Digital Object Identifier

GNN graph neural network

ISBN International Standard Book Number

MRR mean reciprocal rank

MSE mean squared error

NFD Normalization Form D

NLP natural language processing

OBD Osobní bibliografická databáze (Personal Bibliographic Database)

RMSE root mean squared error

SIGIR Special Interest Group on Information Retrieval

UKČO Číslo osoby (Person ID in the CU information system)

ÚVT Ústav výpočetní techniky (Computer Science Centre)