UNIVERSITÁ DI PISA
DOTTORATO DI RICERCA IN INGEGNERIA DELL'INFORMAZIONE

ENHANCING AUTHOR NAME DISAMBIGUATION WORKFLOWS IN BIG DATA SCHOLARLY KNOWLEDGE GRAPHS

DOCTORAL THESIS

Author
Michele De Bonis

Tutor (s)
Dr. Paolo Manghi
Dr. Fabrizio Falchi
Prof. Marco Avvenuti

Reviewer (s)
Dr. Francesco Osborne
Dr. Markus Stocker

The Coordinator of the PhD Program
Prof. Fulvio Gini

Pisa, 1 2024

XXXVI

To my naivety,
which magnificently underestimated the stress factor in going for a Ph.D.

Acknowledgements

My journey in these 3 years of the Ph.D. has been a rollercoaster of stress, pain, and frustration but, of course, it had also its advantages. One of the biggest lessons I learned is the importance of having a solid team by your side. And by "team", I don't just mean supervisors to guide you or colleagues to work with, but also friends and family to support you in moments of discouragement.

This is why I want to acknowledge the people who mostly contributed to this work, starting with Dr. Paolo Manghi and Dr. Fabrizio Falchi: they have always been an inspiration and a guiding light for me. Their precious suggestions got me through all the challenges I faced and gave me the courage to proceed on this difficult path. Many thanks go also to my supervisor Prof. Marco Avvenuti for accepting my Ph.D. project and for the advice and support over these three years.

I would also like to say many thanks to my awesome colleagues at the Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo" in the CNR. My gratitude goes especially to the SCI-Lab crew, which includes Michele Artini, Claudio Atzori, Miriam Baglioni, Alessia Bardi, Giambattista Bloisi, Sandro La Bruzzo, Andrea Mannocci, and Gina Pavone. They have been always ready to help me in every possible way. They taught me everything I know, starting from tech stuff, going to paper writing and even laughing at my bad jokes.

My friends have played an important role in this path. Some of them are in Pisa (Pietro, Noemy, Leonardo, Diletta, Chiara, Chiara, Claudia, Ilenia, and Marco), and some of them are somewhere else (Alex, Filippo, Sfetano, Annuccia, Fosia). They have always been like my family and I am pretty sure that without them I would not have been able to stay focused and sane. A special mention goes to my two roommates, Francesco (aka Fofino) and Ivan (aka Ivano), who gave me good times at home after the grind at the office. I want to include into this also Miriam T. (aka Mirima). Spending time with the three of you during the COVID lockdown has been one of the luckiest things that ever happened in my life. I would like to express my gratitude also to the "unKnown friend", whose impact has been palpable even in their physical absence. I would like to tell them Ευχαριστώ πολύ, Μου αρέσει το σουβλάκι, στρίψτε δεξιά, στρίψτε αριστερά, χρόνια πολλά. Don't worry if you don't understand, neither do I.

Last but not least, many thanks to my family. They check in on me every day with

their “how are you?” and regular concerns about meals, washing machines, and football plans. Dad, Mom, brothers, sisters, nieces, nephews, you have always been important for me even if I never told you. Of course, I consider part of my family also Alessandro (aka Sale), as he is and has always been my friend for life and my reliable refuge in times of need.

Looking back at my life, I recognize that I’ve never experienced loneliness. The credit for this goes to the incredible squad I’ve been lucky to have – a team that selflessly supported me without expecting anything in return.

Summary

Open Science, defined by its commitment to transparency, collaboration, openness, and accessibility, has deeply affected scientific research. Following this new paradigm, scientists produce and publish research data and software alongside research publications to enable reproducibility, monitoring, and assessment of science. In this context, Scholarly Knowledge Graphs (SKGs) are "big data" metadata collections, playing a crucial role in research discovery and assessment by aggregating bibliographic metadata records and semantic relationships describing research products and their associations between them (e.g., citations, versions) and with other entities, such as organizations, authors, funders, etc. Examples of SKGs are the OpenAIRE Graph, Google Scholar, OpenAlex, Semantic Scholar, OpenCitations, and ResearchGraph.org. However, constructing and maintaining SKGs demands innovative solutions to address the inherent scalability, heterogeneity, duplication, inconsistency, and incompleteness challenges introduced by the metadata sources to be aggregated.

Motivated by the urge of Open Science and the challenges posed by SKG construction, this Ph.D. thesis makes pioneering contributions to the field of Author Name Disambiguation (AND). This perennial issue addresses the challenge of identifying and removing duplicate author nodes representing the same author in the SKG. Acknowledging the pivotal role of AND, the thesis discerns two main interwoven imperatives in the duplicate resolution processes: mitigating the *efficiency challenge* derived by the inherent quadratic complexity in comparing hundreds of millions of author nodes; and the *effectiveness challenge* introduced by the efficiency optimization strategies, which renounce parts of the matches, and affected by the poverty of metadata used to compare author nodes, which is often limited to the name's string.

To address the efficiency challenge, the thesis introduces FDup, a groundbreaking framework meticulously designed to reimagine and enhance the traditional disambiguation workflow. At its core, FDup prioritizes the optimization of the similarity match phase. This optimization is achieved through the incorporation of a decision tree-based comparison technique. This innovative approach ensures a customizable and efficient disambiguation workflow and enables parallelization, a crucial aspect in handling the substantial datasets inherent in Scholarly Knowledge Graphs.

To address the effectiveness challenge, the thesis leverages Graph Neural Networks

(GNNs), which have been recently successfully applied to perform innovative research on node classification, graph classification, and link prediction. The proposed contributions manifest in two dedicated GNN architectures to enhance the effectiveness of Author Name Disambiguation via an evaluation of the outputs of a disambiguation algorithm: the first technique evaluates similarity relationships with an attentive neural network integrating GraphSAGE models; the second technique evaluates groups of duplicates with a combination of Graph Attention Network (GAT) and Long Short Term Memory (LSTM) components.

In summary, this thesis is a responsive and forward-thinking contribution within the landscape of Open Science and Scholarly Knowledge Graphs. By introducing novel frameworks and harnessing advanced techniques like Graph Neural Networks, the thesis not only addresses the current challenges but also lays the groundwork for the continual evolution of Open Science practices and the optimal utilization of Scholarly Knowledge Graphs in the ever-expanding realm of scientific knowledge.

Sommario

La Scienza Aperta, col suo impegno per la trasparenza, la collaborazione, e l'accessibilità, ha profondamente influenzato la ricerca scientifica. Seguendo questo nuovo paradigma, gli scienziati producono e pubblicano articoli scientifici, dati e software derivanti dalla loro ricerca con lo scopo di consentire la riproducibilità, il monitoraggio e la valutazione della scienza. In questo contesto, i Grafi di Conoscenza Scientifici (SKG), sono collezioni di metadati che svolgono un ruolo cruciale nella scoperta e nella valutazione della ricerca, aggregando record di metadati bibliografici e relazioni semantiche che descrivono i prodotti di ricerca e le loro associazioni (*e.g.* citazioni, versioni) con altre entità, come organizzazioni, autori, finanziatori, ecc. Alcuni esempi popolari di SKG sono l'OpenAIRE Graph, Google Scholar, OpenAlex, Semantic Scholar, OpenCitations e ResearchGraph.org. Costruire e mantenere i SKG richiede soluzioni innovative per affrontare sfide intrinseche di scalabilità, eterogeneità, duplicazione, inconsistenza e incompletezza introdotte dalle fonti di metadati da aggregare.

Questa tesi di dottorato si pone l'obiettivo di risolvere alcune delle sfide più comuni che si presentano durante la costruzione di tali grafi e propone importanti contributi riguardanti il campo della disambiguazione degli autori (AND). Questa operazione consiste nell'identificazione di nodi di tipo "autore" duplicati, ovvero che rappresentano lo stesso autore all'interno del SKG. Dopo uno studio della letteratura nel campo della disambiguazione degli autori, la tesi individua due strade per migliorare la qualità dei risultati. La prima consiste nel migliorare l'efficienza del processo, afflitta dalla complessità quadratica necessaria a confrontare tra loro centinaia di milioni di nodi di tipo autore; la seconda consiste nel migliorare l'efficacia del processo, spesso minata proprio dai tentativi di migliorare l'efficienza, che spesso sono basati sulla rinuncia ad alcuni confronti, e dalla scarsità dei metadati, spesso limitati al singolo nome e cognome.

Per migliorare l'efficienza del processo di disambiguazione, la tesi presenta un framework innovativo in grado di ridefinire ed ottimizzare il tradizionale workflow della disambiguazione: FDup. Tale framework pone l'accento sull'ottimizzazione della fase di comparazione a coppie, ottenuta attraverso una tecnica di confronto basata su alberi decisionali. Questo approccio innovativo assicura un workflow di disambiguazione

personalizzabile ed efficiente che consente anche la parallelizzazione, da sempre un aspetto cruciale nella gestione dei dati nei SKG.

Per migliorare l'efficacia del processo di disambiguazione, la tesi presenta soluzioni basate sulle Reti Neurali per Grafi (GNN), una tecnologia recentemente applicati con successo per condurre ricerche innovative sulla classificazione dei nodi, la classificazione dei grafi e la previsione dei collegamenti. I contributi proposti si manifestano in due architetture dedicate di GNN per migliorare l'efficacia della AND attraverso la valutazione degli output di un algoritmo di disambiguazione: la prima tecnica valuta le relazioni di similarità con una rete neurale "attenta" che integra modelli GraphSAGE; la seconda tecnica valuta gruppi di duplicati con una combinazione dei componenti come Graph Attention Network (GAT) e Long Short-Term Memory (LSTM).

In sintesi, questa tesi rappresenta un contributo importante e innovativo nel panorama della Scienza Aperta e dei SKG, perché affronta non solo le sfide attuali, ma getta le basi per l'evoluzione continua delle pratiche della Scienza Aperta e per l'utilizzo dei SKG.

List of publications

International Journals

1. **De Bonis, M.**, Manghi, P., and Atzori, C. (2022). FDup: a framework for general-purpose and efficient entity deduplication of record collections. PeerJ Computer Science, 8, e1058. https://doi.org/10.7717/peerj-cs.1058

2. **De Bonis, M.**, Falchi, F., and Manghi, P. (2023). Graph-based methods for Author Name Disambiguation: a survey. PeerJ Computer Science, 9, e1536. https://doi.org/10.7717/peerj-cs.1536

International Conferences/Workshops with Peer Review

1. Minutella, F., Falchi, F., Manghi, P., **De Bonis, M.**, and Messina, N. (2022). Towards Unsupervised Machine Learning Approaches for Knowledge Graphs. In 18th Italian Research Conference on Digital Libraries. https://ceur-ws.org/Vol-3160/short12.pdf

2. Vichos, K., **De Bonis, M.**, Kanellos, I., Chatzopoulos, S., Atzori, C., Manola, N., Manghi, P., and Vergoulis, T. (2022). A Preliminary Assessment of the Article Deduplication Algorithm Used for the OpenAIRE Research Graph. In 18th Italian Research Conference on Digital Library Management Systems. https://ceur-ws.org/Vol-3160/short16.pdf

3. **De Bonis, M.**, Minutella, F., Falchi, F., and Manghi, P. (2023, September). A Graph Neural Network Approach for Evaluating Correctness of Groups of Duplicates. In International Conference on Theory and Practice of Digital Libraries (pp. 207-219). Cham: Springer Nature Switzerland. https://doi.org/10.1007/978-3-031-43849-3_18

4. Baglioni, M., Mannocci, A., Pavone, G., **De Bonis, M.**, and Manghi, P. (2023). (Semi) automated disambiguation of scholarly repositories. In 19th Conference on Information and Research science Connecting to Digital and Library science 2023. arXiv preprint arXiv:2307.02647. https://ceur-ws.org/Vol-3365/paper2.pdf

Others

1. Manghi, P., Atzori, C., Bardi, A., Baglioni, M., Schirrwagen, J., Dimitropoulos, H., La Bruzzo, S., Foufoulas, I., Mannocci, A., Horst, M., Czerniak, A., Kiatropoulou, K., Kokogiannaki, A., **De Bonis, M.**, Artini, M., Lempesis, A., Ioannidis, A., Vergoulis, T., Chatzopoulos, S., and Pierrakos, D. (2023). OpenAIRE Graph: Dataset for research communities and initiatives (7.0.0). Zenodo. https://doi.org/10.5281/zenodo.10521976

2. Bardi, A., Kuchma, I., Pavone, G., Artini, M., Atzori, C., Bäcker, A., Baglioni, M., Czerniak, A., De Bonis, M., Dimitropoulos, H., Foufoulas, I., Horst, M., Iatropoulou, K., Kokogiannaki, A., La Bruzzo, S., Lazzeri, E., Manghi, P., Mannocci, A., Manola, N., **De Bonis, M.**, et al. (2023). OpenAIRE Covid-19 publications, datasets, software and projects metadata (6.0.0). Zenodo. https://doi.org/10.5281/zenodo.8221703

3. Manghi, P., Atzori, C., Bardi, A., Baglioni, M., Schirrwagen, J., Dimitropoulos, H., La Bruzzo, S., **De Bonis, M.**, et al. (2023). OpenAIRE Graph Dataset. OpenAIRE. https://doi.org/10.5281/zenodo.8217359

4. Baglioni, M., Atzori, C., Bardi, A., Bloisi, G., La Bruzzo, S., Manghi, P., Dimitropoulos, H., **De Bonis, M.**, et al. (2023). OpenAIRE Graph Beginner's Kit Dataset. OpenAIRE. https://doi.org/10.5281/zenodo.8223812

5. Baglioni, M., Atzori, C., Bardi, A., Manghi, P., Dimitropoulos, H., La Bruzzo, S., Foufoulas, I., **De Bonis, M.**, et al. (2022). OpenAIRE Graph Beginner's Kit. OpenAIRE Nexus. https://doi.org/10.5281/zenodo.7490192

6. Manghi, P., Atzori, C., Bardi, A., Baglioni, M., Schirrwagen, J., Dimitropoulos, H., La Bruzzo, S., **De Bonis, M.**, et al. (2022). OpenAIRE Research Graph Dump. Bielefeld University. https://doi.org/10.5281/zenodo.7488618

7. Manghi, P., Atzori, C., Bardi, A., Baglioni, M., Schirrwagen, J., Dimitropoulos, H., La Bruzzo, S., **De Bonis, M.**, et al. (2022). OpenAIRE Research Graph: Dumps for research communities and initiatives. Bielefeld University. https://doi.org/10.5281/zenodo.6638478

List of Abbreviations

AI	Artificial Intelligence.
AND	Author Name Disambiguation.
BERT	Bidirectional Encoder Representations from Transformers.
DGL	Deep Graph Library.
FDup	Flat Collections Deduper.
FNR	False Negative Rate.
FPR	False Positive Rate.
GAT	Graph Attention Network.
GDup	Graph Deduper.
GNN	Graph Neural Network.
HDFS	Hadoop Distributed File System.
LDA	Latent Dirichlet Allocation.
LNFI	Last Name First Initial.
LSTM	Long Short Term Memory.
ML	Machine Learning.
NLP	Natural Language Processing.
ORCID	Open Researcher and Contributor ID.
RNN	Recurrent Neural Network.

SKG	Scholarly Knowledge Graph.
TNR	True Negative Rate.
TPR	True Positive Rate.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

Open Science is a movement focused on openness, reproducibility, transparency, and multi-disciplinarity of scientific results. It fosters the principle to make science *"as open as possible, as closed as necessary"*[1] to respect the limits of sensitive and industrial data. It promotes collaboration and sharing of research methodologies, data, and software to help prevent "scientific fraud", i.e., the manipulation or misrepresentation of scientific findings, and reduce the cost of science. By promoting inclusiveness, it aims to embrace diversity in scientific contributions and foster interdisciplinary collaboration. Lastly, Open Science recognizes the need for an all-encompassing science assessment, valuing contributions from various disciplines, methodologies, and research outputs.

Open Science publishing principles include Open Access [1], FAIR Data [2, 3], and Open Source [4] practices to ensure that all the outcomes of an experiment, e.g., article, data, and software, are shared with the community. According to these practices, which today become mandates for many funders, scientists are demanded to publish, in digital form, under an Open Access license their *research products*, providing the metadata required to ensure the discovery, reuse, and assessment of their experiments. In practice, authors publish via cross-disciplinary (e.g., National, institutional) or disciplinary repositories, archives, or databases, the metadata and files of their *research publications* (e.g., article, book chapter, thesis, report), *research data* (e.g., benchmarks used for the experiments, set of images), and *research software* (e.g., code in programming languages, methods, containers). Metadata *properties* represent scientific attribution (e.g., authors, affiliations), and product access (e.g., persistent identifier, URL, license), and product description (e.g., title, abstract); while metadata *relationships* describe the semantic links to other related products (e.g., data and software "supplementing" the

[1] according to Guidelines on FAIR Data Management in Horizon 2020

article, articles "citing" the data, documentation "describing" software).

1.1 Scholarly Knowledge Graphs as maps of Open Science

In the last decade, to support the demand for discovery and assessment of results under the Open Science paradigm, research, and industry have invested resources in the design, implementation, and operation of so-called *Scholarly Knowledge Graphs* (SKGs) [5], which are a specific class within the broader concept of Knowledge Graphs [6, 7]. With some meaningful exceptions [8, 9], SKGs (see Figure 1.1) are metadata graphs modeling a map of the evolution of science in one or more disciplines, where *nodes* represent the metadata records of research products and their scientific context (e.g., *research organizations*, *scientific disciplines*, *funders*, *projects*), and *edges* representing the semantic relationships between nodes (e.g., *cites*, *fundedBy*, *producedBy*, *authoredBy*). SKGs are the result of a continuous and costly aggregation of metadata records (and files to refine metadata with inference techniques) collected from thousands of sources used by scientists to publish and share their products.

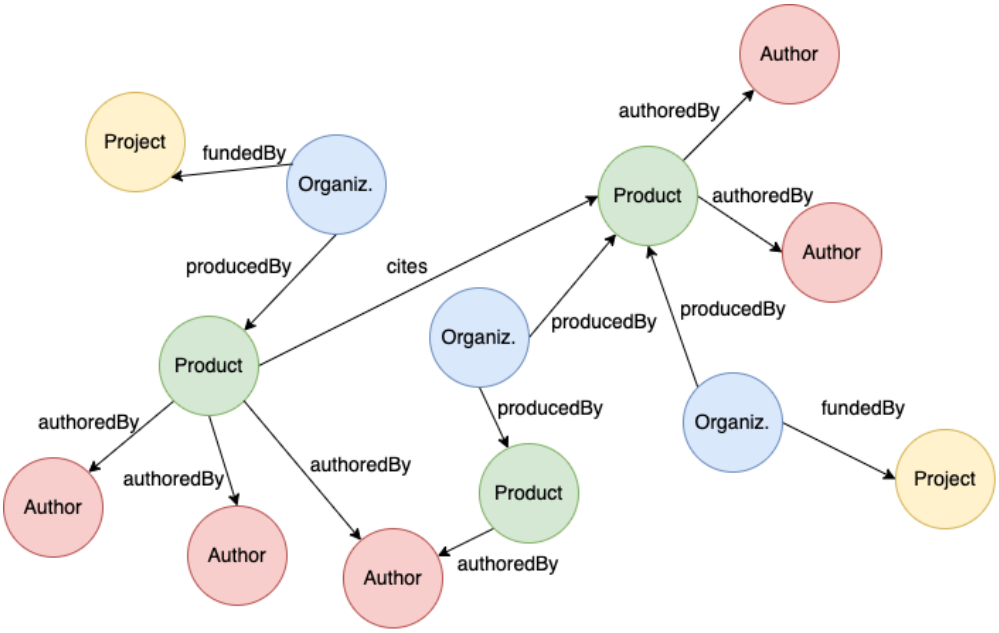

Figure 1.1: *Example of Scholarly Knowledge Graph: a heterogeneous graph of research actors and their semantic relationships.*

Relevant examples of SKGs are Google Scholar[2] [10], OpenAlex[3] [11], Semantic Scholar[4] [12], OpenAIRE Graph[5] [13], OpenCitations[6] [14], SemOpenAlex [15], AIDA-KG [16], CS-KG [17], Nanopublications [18], SoftwareKG [19], Aminer [20], and CORE [21], etc. The consumers of these graphs include researchers, research organizations, funders, and ministries, interested in discovering and reusing science, monitoring research and Open Science trends, and assessing the impact of science in

[2]Google Scholar – `http://scholar.google.com`
[3]OpenAlex – `http://openalex.org`
[4]Semantic Scholar – `http://semanticscholar.org`
[5]OpenAIRE Graph – `http://graph.openaire.eu`
[6]OpenCitations – `http://opencitations.net`

society as a whole (research, industry, education). However, constructing consistent and complete SKGs is far from trivial and presents a series of interesting scientific challenges:

Heterogeneity: the metadata sources aggregated by SKGs describe their research products using different vocabularies, ontologies, exchange formats, etc.;

Duplication: the same research product is often published by co-authors in different sources, in turn, aggregated by SKGs, leading to multiple nodes representing the same product;

Inconsistency: metadata and relationships collected or inferred by the SKG may be incorrect and introduce inconsistencies in the graph;

Incompleteness: sources may expose metadata of different quality and completeness, in some cases lacking key properties.

Due to such reasons, creating an SKG is generally the result of a complex processing phase aiming to address heterogeneity via harmonization (e.g., metadata crosswalks), completeness, and inconsistencies via enrichment strategies (e.g., full-text mining, natural language processing, AI, anomaly detection), and duplication via disambiguation (e.g., entity linking, data disambiguation).

1.2 Author Name Disambiguation challenges in SKGs

This thesis is focused on SKG disambiguation, which involves the identification and consolidation of duplicate nodes into a single "representative" node. As previously stated, metadata duplication occurs when numerous metadata records describing the same real-world entity are present within the same collection. Duplicates in SKGs cannot be tolerated, as ambiguity and redundancy compromise the graph's utility for discovery and research evaluation.

This phenomenon can arise with all node entities in SKGs, including research products, authors, organizations, and others. Due to its particularly challenging scenario, this work focuses on the task of Author Name Disambiguation (AND). First, duplication rates are particularly high for author entities, which are created by the extraction of the author's name strings (referred to as *author names* in the following) from the metadata properties of the research product. Secondly, the name of the author is in most cases the only metadata property available, definitely not enough to establish the equivalence of two SKG author nodes. An example of author name duplication is depicted in Figure 1.2.

AND is the process of identifying groups of equivalent author names and merging them into a single node representing the actual real-life author. This process traditionally comprises four main stages (depicted in Figure 1.3):

Characterization: Author nodes are enriched with information derived from their "neighbor" that characterizes the identity of the author; for instance, metadata of the related publication (e.g., ORCID persistent identifiers, co-author names, topics);

Blocking: Author nodes are grouped into "potentially equivalent" nodes to limit the number of pair-wise comparisons;

Figure 1.2: *Example of AND: underlined authors represent the same person in multiple instances.*

Similarity match: The metadata characterizations of author nodes in the same block are pair-wise matched to check for equivalence and identify pairs of duplicates; successful matches draw a *similarity relationship* between the two nodes;

Disambiguation: Identification of groups of duplicates by "closing the meshes" of nodes linked by similarity relationships resulting from the previous phase.

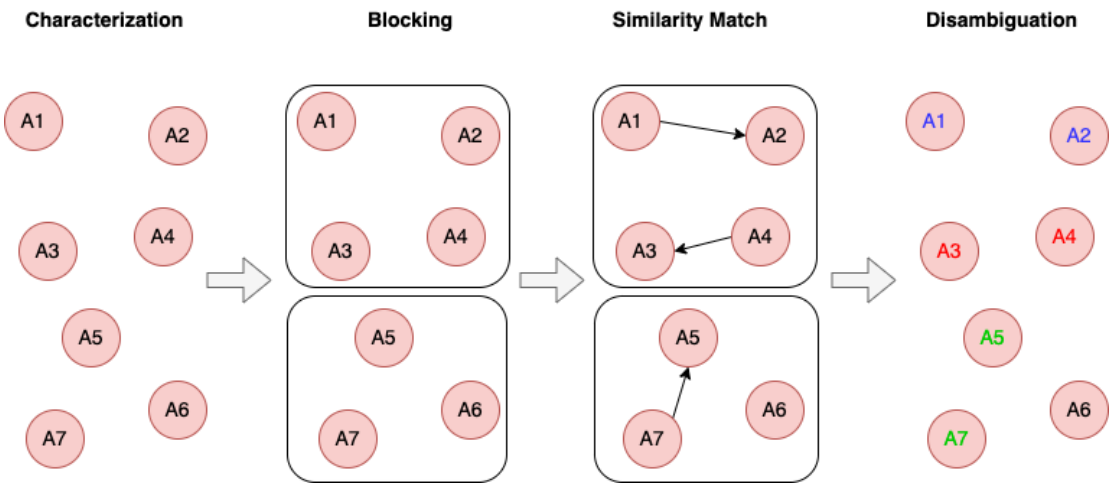

Figure 1.3: *The traditional disambiguation workflow: authors are pair-wise compared within blocks of potentially equivalent to prepare a graph for the transitive closure aimed to create groups of duplicate authors.*

This process entails known issues, due to a trade-off between *efficiency* and *effectiveness*. On the one hand, blocking is introduced to avoid the quadratic complexity of "big data" SKGs, derived from the need for comparing every node with all the nodes of the same entity in the graph. On the other hand, the accuracy of the process is affected by false positives and false negatives, introduced by blocking itself and by the intrinsic limits of the characterization and the similarity match phases. The former is the common cause that mainly affects the *precision*, and the latter mainly affects the *recall*.

1.2.1 Efficiency

The efficiency challenge is traditionally tackled in the *blocking* and in the *similarity match* stage by adopting a combination of heuristics devised to limit the number of comparisons: *block purging*, *block filtering*, and *sliding windows*. Creating blocks of "potentially equivalent" nodes aims to optimize time consumption by reducing the number of pair-wise comparisons and enabling parallel execution across different blocks. This operation can be further enhanced by eliminating redundant blocks through block purging and removing redundant pairs of nodes through block filtering. The sliding window technique further optimizes the number of matches within individual blocks by (i) sorting the records so that similar records are likely kept close to each other and (ii) matching each record with the "k" following records ("K-length window").

These heuristics guarantee optimized computation time but may introduce errors due to the maximum block size, chosen to optimize memory usage, or the sliding window size, empirically determined to limit the number of comparisons within each block. Such limitations, combined with the blocking stage, may likely fail in comparing all potential duplicates and can therefore lead to false negatives, i.e., duplicate nodes are mistakenly identified as non-duplicates.

In this context, the research question to which the thesis aims to respond is:

"How can disambiguation efficiency be enhanced without losing precision and recall?"

1.2.2 Effectiveness

The main issue with the effectiveness of the disambiguation result mostly occurs because author names are simple strings provided as properties of bibliographic metadata. The disambiguation based on such basic information may cause the following:

Homonymy: different real-life persons sharing the same author name;

Synonymy: the same real-life person having different author names in bibliographic records, hence hard to detect by similarity matching (e.g., pseudonyms, language, string differences).

Typically, the characterization stage of the disambiguation process addresses the issue of scarcity of information by enriching author nodes with contextual information extracted or inferred from the bibliographic metadata records where the name originally occurred, e.g., identifiers, titles, co-authors, abstracts, subjects, dates, topics. Other techniques enrich author nodes with information from nodes in the neighborhood by taking advantage of SKG relationships. Nonetheless, the characterization of author nodes is still prone to errors due to the unbalanced topology of SKGs, which gives rise to nodes with different degrees of richness:

Over-description: authors that published a high number of research products are described by a variety of properties that tend to over-match many other under-described authors;

Under-description: authors that published a low number of research products are encoded with low-descriptive characteristics, which endanger the veridicity of similarity matches against other under-described and over-described authors.

The consequences of undetected synonyms and homonyms can be dramatic in terms of disambiguation. A similarity relationship between two homonym nodes will most likely create a false positive, hence a "bridge" between otherwise distinct groups of duplicate author nodes. Consequently, the disambiguation phase will erroneously merge the two groups, causing the SKG to mix the outcome of two real-life persons into one. On the contrary, it would be difficult to draw a similarity relationship between two synonym nodes, because the traditional disambiguation workflow may not be able to capture the equivalence, therefore resulting in a false negative. On the other side, an over-described author may also lead to false positives as it could match with too many other under-described authors. In contrast, an under-described author may lead to false negatives as it could not match with others due to the impossibility of capturing any equivalence due to the scarce amount of information describing it.

With recent advancements in Artificial Intelligence (AI) and, in particular, Machine Learning (ML) [22], characterization challenges took advantage of deep learning solutions, where metadata properties were replaced by *embeddings*. In particular, the graph structure of SKGs suggests that using Graph Neural Network (GNN) may empower traditional techniques to overcome their limitations. For example, GNN architectures can bypass both the homonymy/synonymy problem and the under-description/over-description problem as they can take advantage of the relationships in the graph. Such architectures use nodes in the neighborhood by properly weighing them to find a better description of the data not affected by the aforementioned problems. This feature of the GNNs makes every node comparable, decreasing the chances of false negatives.

In this context, the research question to which the thesis aims to respond is:

"How can disambiguation effectiveness be enhanced using novel Graph Neural Networks?"

1.3 Research contributions

This thesis defines and tests novel solutions to the issues of efficiency and effectiveness in AND by extending traditional blocking-similarity matching solutions and advancing AI-based techniques in the domain.

In particular, traditional approaches address efficiency optimization by reducing the number of pair-wise comparisons in the blocking and similarity match phases using clustering and sliding window techniques. Given the big data scenario, this thesis proposes a complementary approach where an extra optimization step is introduced in the similarity match phase by limiting the number of similarity checks to be performed.

Concerning effectiveness, the thesis proposes a GNN approach that takes advantage of the graph structure of a SKG. Firstly, a general framework describing AND approaches adopting GNN has been derived; secondly, two solutions to improve precision and recall and evaluate the quality of the latter have been developed by empowering state-of-the-art approaches.

1.3.1 Efficiency

The approach to reducing time consumption and optimizing efficiency was never explored in existing state-of-the-art methods and involves optimizing the similarity match phase by fragmenting the pair-wise comparison process between two nodes into a set

of interrelated smaller predicates. More specifically, the thesis introduces a novel approach to improve computation time, beyond the known techniques of "blocking" and "sliding window" by introducing a smart similarity matching function T-match engineered as a decision tree that drives the comparisons of the fields of two records as branches of predicates and allows for successful or unsuccessful early-exit strategies. The approach significantly enhances performance in big data scenarios by purging a considerable number of useless checks.

As a result of this investigation, the framework FDup (*Flat Collections Deduper*) [23] has been engineered to provide a general-purpose open-source software framework supporting a complete disambiguation workflow over big data record collections based on the Apache Spark Hadoop framework. FDup supports a customizable data model, adaptable to any node characterizations, tools for the configuration of blocking, and a highly configurable similarity match function. FDup delivers a full traditional disambiguation framework in a single easy-to-use software package, where developers can customize the optimal and parallel workflow steps of blocking, sliding windows, and the similarity matching function T-match via an intuitive configuration file.

1.3.2 Effectiveness

The approach to improving effectiveness relies on GNNs, as a growing interest in those kinds of solutions to the task of AND has been observed. As a first contribution to the field, to understand trends and open challenges in the domain, and to compare results in this area, the thesis presents a review of the state-of-the-art GNN-based workflows for AND. The exercise resulted in a general framework and ontology that models the methods applied in the literature in terms of a sequence of processing steps [24]. Inspecting solutions via the framework, two areas of exploration have been identified, leading to two main contributions to improving the effectiveness. First, traditional pairwise similarity matching is not effective in bypassing homonymy/synonymy and under-description/over-description problems. Second, traditional disambiguation workflows do not include a post-processing phase to correct errors generated by the various workflow stages. In general, solutions are missing to perform a quality evaluation of the results of the various disambiguation stages to purge out similarity relationships and groups of duplicates.

The analysis made in this thesis has shown that disambiguation may benefit from GNN solutions to solve those open challenges, which are the primary cause of false positives and false negatives, hence lower accuracy of the disambiguation result. As a contribution in this direction, two dedicated GNN architectures have been proposed:

Evaluation of similarity relationships This contribution consists of a novel approach to entity disambiguation, focused on the enhancement of the accuracy of the disambiguation using a GNN with 4 GraphSAGE and a metapath attention mechanism. The architecture can assign a quality score to each similarity relationship, giving the possibility to inspect potentially wrong results of the similarity match stage. Such evaluation may be subsequently used to prune the similarity relationships to fine-tune the number of groups of duplicates.

Evaluation of groups of duplicates This contribution consists of a novel approach to entity disambiguation, focused on the enhancement of the accuracy of "groups

of duplicates" using a GNN with a Graph Attention Network (GAT) and a Long Short-Term Memory (LSTM) [25]. The architecture can assign a quality score to each group of duplicates, giving the possibility to evaluate the reliability of the information produced by the disambiguation stage. Such evaluation may be subsequently used to fine-tune the result of the whole disambiguation process by intervening in potentially wrong groups of duplicates.

1.4 Thesis outline

The remainder of this thesis is organized as follows. Chapter 2 presents a review of the literature to highlight how the challenges faced in the thesis have been treated by the research community. The chapter also presents a conceptual framework designed after a wide literature review of graph-based Author Name Disambiguation methods. Chapter 3 introduces the OpenAIRE Graph, the real-case Scholarly Knowledge Graph used as a reference benchmark in the rest of the thesis to discuss efficiency challenges and classes of metadata anomalies that undermined the effectiveness of the AND process. Chapter 4 describes the solution adopted to address efficiency issues in AND by presenting FDup, a general-purpose framework and tool developed to perform and optimize the whole traditional disambiguation workflow, with a special focus on the pair-wise comparisons in the similarity match stage. Chapter 5 describes the methodology adopted to face the challenges of enhancing the effectiveness in Author Name Disambiguation. This is done through the so-called quality evaluation by presenting specialized GNN architectures to enable the evaluation of similarity relationships and groups of duplicates to fine-tune the disambiguation result. Finally, Chapter 6 summarizes and discusses the contributions of the thesis and presents possible future research directions of the work.

CHAPTER 2

Background

This chapter analyzes the state-of-the-art methods and approaches to face the AND efficiency and effectiveness challenges identified in this thesis. On the topic of efficiency, the analysis focuses on existing solutions to improve performance in a trade-off between recall and precision, identifies gaps and open challenges in the literature, and eventually advocates for the innovation brought by the solution proposed in this thesis. On effectiveness, to motivate and justify the adoption of GNN for the AND evaluation, the analysis first delves into possible solutions to this task, then digs into graph-based AND solutions.

The chapter is organized as follows: Section 2.1 highlights how disambiguation efficiency issues have been faced in the literature by analyzing the state-of-the-art; concerning disambiguation effectiveness, the state-of-the-art touches upon three main areas of investigation: Section 2.2 presents the literature methods and approaches used for the disambiguation evaluation that could possibly be used for the effectiveness enhancement; Section 2.3 presents the theoretical aspects of Graph Neural Networks by providing a brief description of their most common applications; Section 2.4 focuses on the AND topic by presenting a deep study of the literature surveys, describes the most recent AND graph-based methods, and identifies a generic workflow and taxonomy for such methods;

2.1 Disambiguation methods

Many tools and frameworks contributed in different ways to the general task of "entity linking" of which disambiguation is a specific application. Complete surveys of such approaches can be found in [26] and [27]. This class of problems has been deeply studied in the literature, and many solutions were proposed to specifically tackle the

usability and efficiency of the approaches. Solutions focus mainly on the optimization of the quadratic complexity by accurately selecting (e.g., heuristics) the pairs of entities to be matched and by parallelizing the actual match operations [28]. Interestingly, the survey also analyzes the approaches used for similarity matching of a pair of records. Record matching is in general driven by similarity measures that are computed atomically but never mentioned as a phase where further optimization of the overall process can take place.

The work in [29] proposes a clustering algorithm that tolerates errors and catalogs variations by using a search engine and an approximate string-matching algorithm. The approach proved to be effective as it identifies more than 90 percent of the related records and includes incorrect records in less than 1 percent of the clusters.

In the research presented in [30], authors describe a solution based on a graph partitioning formulation that improves the accuracy of entity resolution by incrementally revising results whenever new information about the input entities is provided. The approach improves accuracy and optimizes the process by reducing the number of comparisons required in subsequent rounds of disambiguation. The GDup framework described in [31], offers an out-of-the-box solution to a complete workflow of disambiguation for SKG. The framework includes ground truth management, candidate identification via blocking and sliding windows, identification and merging of duplicates, and graph topology consolidation. The Spark-based implementation drives the parallelization process, further boosting the performance introduced by clustering. The similarity match function can be flexibly configured but does not support any optimization option. A novel contribution to performance optimization is offered by [32], where "block purging" and "block filtering" techniques are adopted to further reduce the number of records in a block, hence the number of matches. These techniques take advantage of the frequency to which a pair of entities appear in the same group to avoid redundant and rare comparisons. The approach shows better performance than traditional blocking/sliding window techniques, but it is not recommended when a high recall is required.

In most approaches, the similarity match phase is performed via a similarity function provided as a weighted mean of the sum of comparators applied to the pair of records without performing any optimization process. For example, in [33], the authors present a six-step disambiguation process in which the comparison between two entities is driven by a similarity vector. Such vector is represented as an aggregation of single similarity scores between attributes and it is subsequently used to apply rules defining threshold-based conditions for the equivalence of the entities. Similarly, in the context of data association, the research in [34] proposed a solution to link criminal records that possibly refer to the same suspect. This method is based on the calculation of a total similarity measure as a weighted sum of the similarity measures of all corresponding feature values. Moreover, [35] proposes a record linkage algorithm for detecting deceptive identities by combining personal attribute scores into an overall similarity score. To conclude, it establishes a threshold for match decisions using a set of identity pairs labeled by an expert.

In summary, existing approaches tackle efficiency enhancement by optimizing the disambiguation workflow stages that precede the similarity match, in some cases renouncing precision due to low recall. The FDup contribution of this thesis (see Chap-

ter 4) follows the same approach by providing an easy-to-use Apache Spark-based framework for disambiguation but further improves performance by introducing a similarity function T-match, capable of further reducing execution time without renouncing recall.

2.2 Disambiguation quality evaluation methods

The existing literature falls short in addressing the issue of evaluating the quality of "groups of duplicates" created by a disambiguation process. These groups essentially correspond to clusters of equivalent data. To bridge this gap, researchers turn to *clustering evaluation metrics*, which emerge as a promising solution. These metrics fall into two categories: "extrinsic measures", which necessitate a ground truth label, and "intrinsic measures", which operate without relying on a ground truth label.

In the realm of intrinsic measures, popular metrics include the *Rand Index*, the *Mutual Information*, the *V-measure*, and the *Fowlkes-Mallow score*. The *Rand Index* quantifies the similarity between the pairs of data points in the true clustering and the clustering obtained from the algorithm. The *Mutual Information* measures the mutual dependence between the true and predicted cluster assignments, offering insights into the information shared between them. The *V-measure* combines precision and recall, providing a balanced evaluation of the clustering's completeness and homogeneity. The *Fowlkes-Mallow score* calculates the geometric mean of precision and recall, offering a measure of the algorithm's accuracy in identifying true positive pairs.

On the extrinsic side, metrics such as the *Silhouette Coefficient*, the *Calinski-Harabasz Index*, and the *Davies-Bouldin Index* are commonly employed, requiring a ground truth label for evaluation. The *Silhouette Coefficient* measures how well-defined the clusters are within the data. It is calculated as the difference between the average intra-cluster distance and the nearest-cluster distance, normalized by the maximum of the two. The *Calinski-Harabasz Index* assesses the ratio of between-cluster variance to within-cluster variance, providing a measure of cluster compactness. The *Davies-Bouldin Index* quantifies the average similarity between each cluster and its most similar cluster, aiding in the evaluation of cluster separation.

To apply these metrics effectively, conceptualizing the group of duplicates as a set of points in an n-dimensional space becomes crucial. In some instances, defining a measure of distance between these points is essential for accurate assessment. However, a notable limitation arises in that the evaluation of deduplication using these metrics does not permit the independent evaluation of each group. The final score generated by the formulas of these metrics is either impractical to compute or provides a holistic view of the entire disambiguation result.

Alternative approaches included as a contribution in this thesis (see Section 3.3.5), is based on evaluating a disambiguation result and involve leveraging persistent identifiers of entities, such as Digital Object Identifiers (DOIs) [36, 37]. While this method is reliable when persistent identifiers are available for all entities in a real-case scenario, the lack of universal identifier coverage may compromise the trustworthiness of the evaluation measure.

The literature on the graph classification problem offers insights into potential solutions [38, 39]. Existing surveys summarize various methods, particularly in the context of classifying molecules and proteins using supervised techniques with acceptable ac-

curacy ranges. However, these methods are not directly applicable to groups of duplicates due to the unique and diverse nature of such groups. The distribution of relations within these groups can be either dense or sparse, and this distribution is not necessarily indicative of the correctness of the group.

One notable approach for graph classification is Graphormer [40], which introduces the transformer concept into Graph Neural Networks (GNN). The authors claim that Graphormer represents the state-of-the-art in graph classification. However, the practicality of Graphormer is hindered by the inherent characteristics of transformers, making both training and inference processes slow and feasible only with substantial computational power—a resource often lacking in typical scenarios.

In conclusion, while various metrics and approaches exist for assessing the quality of groups of duplicate entities, each comes with its set of challenges and limitations. As the field progresses, addressing these challenges and exploring alternative methods will be pivotal for advancing the effectiveness of deduplication processes and enhancing the reliability of evaluation metrics.

2.3 Graph Neural Networks

Since this thesis focuses on the AND task for Scholarly Knowledge Graphs, the effectiveness enhancement through disambiguation quality evaluation can unleash its full power by using AI architectures to process such kind of data structures: the GNNs. As described in [41], one of the recent advancements in AI technologies led to the adaptation of Neural Networks to leverage the structure and properties of a graph, able to represent relationships (i.e., edges) between collections of entities (i.e., nodes). To describe each node, edge, or the entire graph, information can be stored in each of the pieces forming it. Such information (i.e., the embedding) is in the form of a scalar or vector and it is used to encode the information of the entities in a machine-friendly format. A GNN is an optimizable transformation on the attributes of the graph that preserves graph symmetries. The purpose is to learn embeddings in a way that they are aware of the graph connectivity information. To do this, a GNN implements the *message-passing* protocol: nodes in the neighbors exchange information and influence each other's updated embeddings. The general idea of the *message-passing* can be summarized in three steps (see Figure 2.1 for a reference). In particular, each node in the graph:

- collects all the node embeddings in the neighborhood;
- aggregates all the node embeddings it received via an aggregation function (e.g., mean, sum, max, min);
- updates the aggregation via a learned neural network.

This sequence of operations is key for leveraging the connectivity of graphs and, when applied once, is the simplest type of *message-passing* GNN layer. This is no more than a standard convolution on images: both the *message-passing* and the convolution operation are operations used to aggregate and process the information of an element's neighbors to update the element's value, as described in [42]. In graphs, the element is the node, and in images, the element is a pixel. By stacking multiple GNN layers

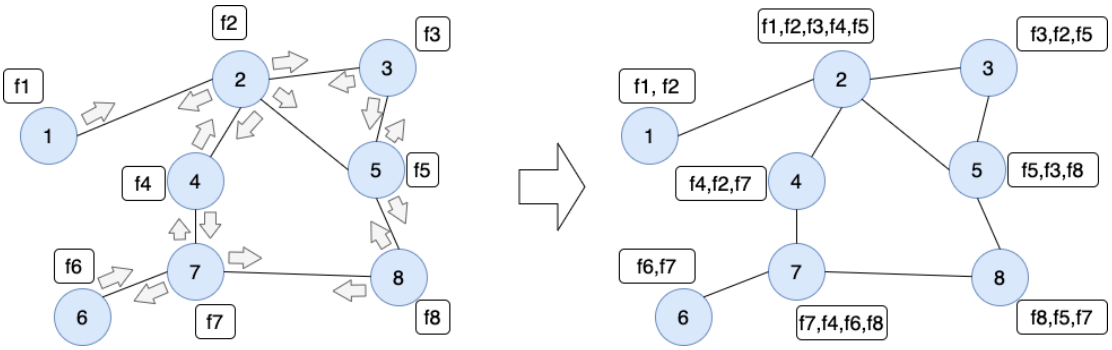

Figure 2.1: *Example of Graph Neural Network: the message passing protocol enriches every node of the network with information about the topology of the graph. In each layer of the GNN a node sends its features to the neighbors and aggregates the features it received.*

together, a node becomes able to include information from even further nodes (i.e., with 4 layers, a node collects information about nodes that are four steps away from it). As all the other AI networks, GNN may be trained in a supervised, semi-supervised, and unsupervised way based on the availability of labeled data. In the supervised training, the network is trained with a set of labeled data and it learns how to classify unseen nodes based on what is in the training set. In the semi-supervised training, the base network model is trained with available labeled data, such a model is consequently used to predict labels for unlabeled data and this new data is subsequently used to train again the network. In unsupervised training, the network discovers hidden patterns or data groupings without the need for human intervention. The information encoded in the node embedding depends on the type of training and on the computation of the loss function, which calculates the error between the actual result and the expected result to backpropagate the error and adjust the network weights accordingly. In this thesis, the GNN have been used by configuring the challenges presented in Chapter 1 into one of the three most popular operations in the literature: the link prediction (i.e., the operation of predicting the existence of a link between two nodes of the graph), the node classification (i.e., the operation of classifying each node of a graph in a predefined set of classes), and the graph classification (i.e., the operation of classify a whole graph based on its typology and its nodes).

Node classification The purpose of models for node classification is to predict non-existing node properties (e.g., the class) based on other existing node properties and the relationships with the neighbors in the graph. The learned node embedding is meant to be closer to nodes belonging to the same class and further from nodes belonging to a different class. An example of a node classification task is presented in [43], where the dataset is composed of people and their relationships in a Karate Club, and the purpose is to classify whether a given person becomes loyal to one or the other master after their feud. In this case, the distance between a node and the master is crucial. In images, those problems are analogous to image segmentation (i.e., predict the role of each pixel). In texts, those problems are analogous to the prediction of the role of each word in a line (e.g., noun, subject, verb, adverb). The research in [44] presents a survey and an evaluation of existing GNNs for node classification in the literature. Figure 2.2 presents a visualization of this task.

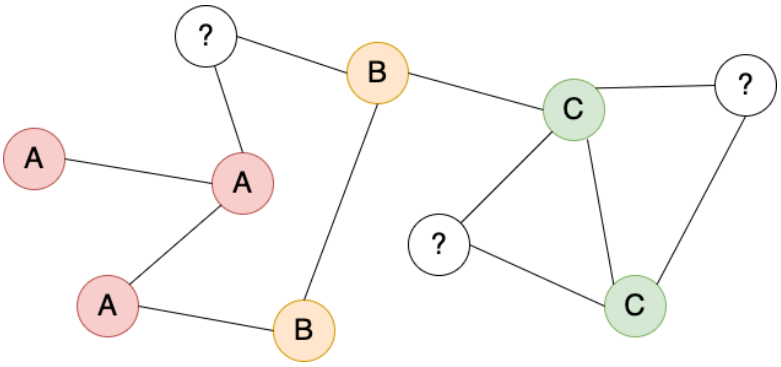

Figure 2.2: *Example of node classification: the node embedding is used to predict the class of the node based on the classes of similar nodes.*

Link prediction The purpose of models for link prediction is to infer which links are most likely to be added or missing based on the observed connections and the structure of the network. Therefore, giving a partially observed network, link prediction models may be useful to increase the completeness of a graph. In this scenario, the training is often performed by providing a set of positive and negative links, obtained with negative sampling (i.e., creating a graph with non-existent links). An example of a link prediction task is in image scene understanding: given nodes that represent the objects in an image, the purpose is to predict which of these nodes shares an edge with one other. It is possible also to predict the meaning of that edge for those two nodes. Prominent examples in the literature are presented in [45] and [46], with the research in [47] specifically focused on SKGs. Figure 2.3 presents a visualization of this task.

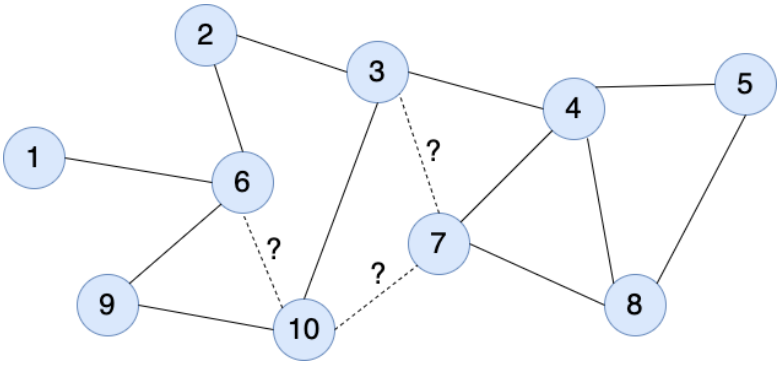

Figure 2.3: *Example of link prediction: the node embedding is used to predict the existence of a link based on other links in the graph.*

Graph classification The purpose of models for graph classification is to classify the whole graph into different categories. In such an architecture, the node embeddings are computed in the traditional method and the graph embedding is obtained with an aggregation of all the node embeddings of the graph performed by the readout operation, properly configured depending on the activity needs. An example of a graph classification task is molecule prediction: given a graph representing a molecule (i.e., nodes as atoms and edges as bonds), the purpose is to predict what the molecule smells like or whether it will bind to a receptor implicated in a disease. In images, those problems are

analogous to image classification problems where the objective is to associate a label to a full image. In texts, a similar problem is sentiment analysis, where the objective is to identify the mood of a full text. The research in [48] presents a fair comparison of existing GNNs for graph classification in the literature. Figure 2.4 presents a visualization of this task.

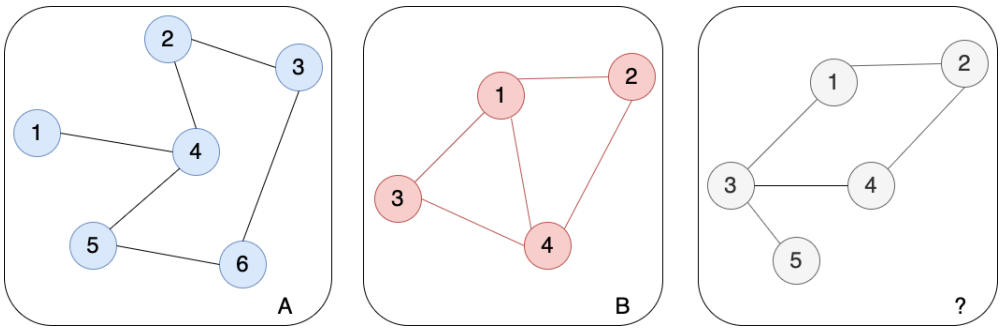

Figure 2.4: *Example of graph classification: the node embeddings of each graph are aggregated to create a graph embedding to be used to predict the class of the graph based on the classes of similar graphs.*

2.4 Author Name Disambiguation methods

The AND task has been faced in several different ways in the literature. To highlight the most popular types, it may be useful to study how such approaches have been categorized in the existing surveys on the topic.

The survey in [49] presents a list of methods for the AND created between 2004 and the beginning of 2010. The authors highlight a list of challenges that an AND approach has to face (i.e., homonymy, name changes for marriage, spelling variations, incomplete metadata, and the impossibility of manually tagging all the authors) and classify methods in two subcategories: manual and automatic disambiguation. The two classes engage in a trade-off between precision and scalability: manual approaches are more precise but not scalable, while automatic ones show a higher error rate but can be applied to very large data collections. The reported automatic disambiguation methods are based on clustering and supervised learning. Clustering methods use publication attributes to create embeddings and cluster them to identify the work of a specific author; supervised learning methods can be categorized as naive Bayes models to calculate the probability of a pair of authors being the same person, and SVM models trained to discriminate authors. Since the survey is one of the first on this topic, graph-based methods have not been deeply studied, and no specific method has been developed in that direction.

The survey in [50] proposes a taxonomy for characterizing the current AND methods described in the literature. The survey categorizes automatic methods based on two features: the evidence explored in the disambiguation task (web information, citation relationships, etc.) and the main type of exploited approach (author *grouping methods*, and author *assignment methods*). Author grouping methods aim at identifying groups of author names in a set of references based on properties of the publication nodes and potential relations with other nodes. They are based on similarity functions that compare

two author names by using predefined techniques (Levenshtein, Jaccard, etc), learned from ground truth data (providing pairs of equivalent/different authors), or graph-based techniques (similarity degree of two authors based on co-authors or properties of the related publications). From the resulting equivalent name pairs, such methods identify the groups of equivalent author names. Author assignment methods are instead based on the assumption that a set of references with disambiguated author names exist, i.e., *classification* strategies, and/or a mathematical representation of the authors exist, i.e., *clustering* strategies. Classification strategies, given a set of references, predict the author of the references among a set of predefined authors. Clustering strategies attempt to directly assign references to authors' work by optimizing the fit between a set of references to an author and mathematical models used to represent that author. The survey touches on graph-based methods but only from the point of view of graph-based similarity functions.

The survey in [51] divides AND methods into five categories: supervised, unsupervised, semi-supervised, graph-based, and heuristic-based. Supervised techniques in this domain are based on labeled training data that associates the corresponding author record class to the author representations (e.g., embeddings, vectors). Unsupervised techniques are those adopted when labeled data is not available, but the idea is similar to the supervised methods because the method has to classify authors into a pre-defined set of classes. Heuristic-based AND techniques are used when scalability issues occur. Such methods approximate the solution giving a result that is as close as possible to reality. As for the Graph-based AND techniques, the focus is on methods relying on graphs in which author names are nodes, and edges identify the co-authors' relations (when two author names occur in the same publication). Similarity measures or deep learning are then applied to such graphs to identify groups of equivalent author names.

The survey in [52] provides a generic five-step framework to handle AND issues. Such steps are (i) dataset preparation, (ii) publication attributes selection, (iii) similarity metrics selection, (iv) model selection, and (v) clustering performance evaluation. An important contribution of this survey is the definition of a set of common challenges in AND to be faced when developing a framework. The methods in this survey are categorized into supervised, unsupervised, semi-supervised, graph-based, and ontology-based. Classes defined in the taxonomy resulting from this survey have already been described by other surveys reviewed in this paragraph. As for ontology-based classification, the survey defines ontology as the knowledge of concepts and their relationships within a domain, i.e., the knowledge representation of a domain. Methods included in this category use such representation to identify groups of equivalent author names.

The survey in [53] focuses on AND challenges in PubMed[1], the publication repository of the Life Science community. This work surveys a set of solutions considering as input data the citation graph formed by PubMed where the author set is not known a priori. The outcome is a general framework composed of four stages: (i) citation extraction, (ii) LN-FI blocks creation, (iii) similarity profile creation (creation of similarity for each pair of citations), and (iv) author-individual clusters creation based on similarity profiles. The survey proposes a taxonomy that classifies methods based on evidence explored (only co-authorship information or multiple metadata), and techniques used to generate similarity profiles (supervised, graph-based, and heuristic-based). Only one

[1] PubMed – `https://pubmed.ncbi.nlm.nih.gov`

graph-based method is described in this review, named GHOST (GrapHical framewOrk for name diSambiguaTion), also included in [51] above, which implements a similarity measure based on the graph composed of co-authors' relations and applies a clustering method to create groups of equivalent author names.

A recap of the characteristics of each survey described in this section is depicted in Section 2.4. Clearly, in recent years, graph-based approaches are regarded as relevant but only as one of the potential classes of AND solutions; no specific investigation digs into the features of this class of problems and methods.

Survey	Taxonomy
[49]	manual and automatic disambiguation
[50]	evidence explored (web information, citation information, and implicit evidence) or exploited approach (author grouping methods, and author assignment methods).
[51]	supervised, unsupervised, semi-supervised, graph-based, and heuristic-based
[52]	supervised, unsupervised, semi-supervised, graph-based, and ontology-based
[53]	based on evidence explored or techniques used to generate similarity profiles (graph-based, heuristic-based, supervised)

Table 2.1: *Summary of existing surveys on AND methods.*

2.4.1 Detailed review of graph-based AND methods

Graph-based AND methods have been proven to be very effective and are becoming the most popular in this topic, as SKGs semantic relationships have the power to boost the performances of every graph-related task. For this reason, it becomes crucial to provide an exhaustive review of the methods in the literature. The articles included in the review have been identified by searching Google Scholar for the keywords *"graph based author name disambiguation"*. To prioritize the latest research trends in the domain, the candidates have been limited to work published after 1/1/2021. Since it became clear that after 100 search results, topics tended to diverge from the focus, the investigation has been interrupted there.

To ease the investigation process (and for the reproducibility of the Google Scholar search), the search was performed using a Python script[2] from [54] with the following command:

```
$ python sortgs.py --kw "graph based author name disambiguation" --startyear 2021
```

The command returns a CSV file that contains the first 94 publications matching the query (articles with corrupted metadata have been excluded), each with metadata about *Title*, *Number of Citations*, and *Rank*; by default, the script sorts the list by number of citations. The CSV has been explored to identify work relevant to the survey: papers

[2]Python script to query Google Scholar – `https://github.com/WittmannF/sort-google-scholar`

were downloaded, first selected based on the relevance of the abstract, and then studied; the number of citations and the query rank were used as an indicator of quality but not as selection criteria; the reference list of selected articles was also analyzed to identify further relevant titles published before 2021. The full list of articles returned by the query on the 1st of July 2023 is available as a CSV in [55]. The CSV includes a column to show the ones selected for this survey (including 1 article published in 2019 identified exploring the bibliographies); note that three articles are marked as "N.A.", as they matched the relevance criteria defined above but are written in Chinese, and no English version could be found.

The methods included in the review are listed below:

LAND [56] proposes a framework called Literally AND (*LAND*), a representation learning method without training data. Such framework utilizes multimodal literal information generated from the Scholarly Knowledge Graphto create node embeddings for the graph (so-called Knowledge Graph Embeddings KGEs). *LAND* is based on three components:

- Multimodal embeddings: learn representative features of entities and relations in the graph by using a multimodal extension of a semantic matching model called *DistMult*. The extension is based on literals of publication attributes (e.g., publication title encoded with *SPECTER* - a pre-trained BERT model, and date encoded as described in *LiteralE*) which are used to modify the scoring function to maximize the score for existing triples and minimizing the scores for non-existing triples;
- A blocking procedure: divides authors into groups with LNFI blocking to reduce the number of pairwise comparisons required by the AND task;
- Clustering: Hierarchical Agglomerative Clustering (*HAC*) presented in [57], used to split embeddings in the same block into k-clusters that identify each unique author.

HGCN [58] proposes a novel, efficient, re-trainable, and incremental AND framework based on unsupervised learning since it does not need labeled data. The idea is to construct a publication heterogeneous network for each ambiguous name using the meta-path approach. Such publication network is consequently processed by a custom heterogeneous graph convolutional network (so-called *HGCN*) that calculates the embeddings for each node encoding both graph structure and node attribute information. Once all publication embeddings have been computed, authors use a graph-enhanced clustering method for name disambiguation that can significantly accelerate the clustering process without the specification of the number of distinct authors. The proposed method is based on two components:

- Publication Heterogeneous Network (*PHNet*) Embedding: each publication is vectorized using *Doc2Vec*, subsequently the *HGCN* aggregates the publication vectors to create the publication final embedding. The research tests different GCNs to perform the aggregation (i.e., DeepWalk, LINE, meta-path2Vec, Hin2Vec, and GraphSAGE);

- Clustering: uses Graph-enhanced *HAC* (*GHAC*) over the publication graph for the ambiguous name.

AE [59] proposes an unsupervised representation learning framework based on 4 modules to bridge the gap between semantic and relational embeddings. The goal of the research is to jointly encode both semantic and relations information into a common low-dimensional space for AND task. The 4 modules of the framework are:

- Semantic embedding module: publications are processed with *Word2Vec* using *TF-IDF* weighting to represent content;
- Relationship embedding module: the framework constructs the homogeneous network applying three meta-paths (the result is a unique homogeneous network where the weight of each relationship between two publications is given by the number of meta-paths connecting those publications). Each node is embedded with a MultiLayer Perceptron (*MLP*) which uses triplet loss to train;
- Semantic and relationship joint embedding module: a variational autoencoder is used to learn the joint embedding by minimizing reconstruction loss;
- Clustering: *HAC* to create publications clusters inside a group of ambiguous authors.

jGAT [60] presents a solution that considers both content and relational information to disambiguate. In the research, authors construct a heterogeneous graph based on meta-information of publications (e.g. collaborators, institutions, and venues). The heterogeneous graph is subsequently transformed into 3 homogeneous graphs using 3 meta-paths (co-author, co-organization, and co-venue meta-path). Graph Attention Networks (*GAT*) is used to jointly learn content (abstract information and title) and relational information by optimizing an embedding vector: each node (publication) of the graph is vectorized (using *Word2Vec*), subsequently, an embedding is computed for each meta-path, and a concatenation of the 3 embeddings is inputted to a fully connected network to create the k-dimensional vector representing the final embedding of the publication. Finally, a clustering algorithm is presented to gather author names most likely representing the same person (spectral clustering algorithm to learn the embedding vectors which have been learned by *GAT*).

RF-LRC [61] develops a robust supervised machine learning approach in combination with graph community detection methods to disambiguate author names in the Web of Science publication database. The framework uses publication pairs to train a Random Forest and a Logistic Regression Classifier. The labeled data is given by the ResearcherID, through which a pair can be identified as equal or not, and the features are properties of the pair (i.e., the result of the comparisons of publication fields). The classifier is used to create a graph which is consequently inputted to the infomap graph community detection algorithm to identify all publications belonging to the same author. The distinction is always performed in a subset of publications with ambiguous authors, obtained using the LNFI blocking strategy.

sGCN [62] provides a disambiguation model based on *GCN* that combines both attribute features and linkage information. The first step consists of computing the embeddings of the publications using *Word2Vec*. Then 3 different graphs are built:

- a paper-to-paper graph: nodes of the graph are publications and an edge is drawn whenever the similarity of the attributes exceeds a threshold;
- a co-author graph: all authors are represented as nodes and an edge indicates that there is a cooperative relationship between the authors;
- a paper-to-author graph: publications and authors represented as nodes and edges representing relations between publication and author.

Each graph is fed to a specialized *GCN* and the final output is a hybrid feature computed following the following steps: (i) embeddings of publications and authors are obtained respectively from AuthorGCN and PaperGCN, (ii) triples samples from the paper-to-paper graph to minimize the error, (iii) triples sample from the co-author graph to minimize the error, and (iv) triple samples from paper-to-author graph to minimize the error and update network weights at the same time. The PaperGCN output is the final embedding of the publication. Finally, the *HAC* algorithm is applied to divide publications into disjoint clusters of authors.

LP [63] presents a semi-supervised algorithm to disambiguate authorship pairs: the method consists of various nonlinear tree-based classifiers trained to classify pairs of authors to construct a graph, which is subsequently processed with label propagation to cluster group of authors. The LNFI blocking strategy is applied to create groups of publications with ambiguous authors, subsequently, a probabilistic classification model is trained to decide whether two publications within a given block belong to the same author. The classifier resulting from the training is used to create authorship graphs as follows: publications are represented as nodes, while an edge is drawn between two nodes if the classifier predicts that both are authored by the same person. The classifier's class probabilities are used as edge weights to obtain a labeled graph. Finally, a clustering algorithm based on the label propagation algorithm is applied to the constructed graph. The label propagation algorithm works as follows: each node of the graph is initialized with a random unique label, then the process starts and each node is labeled iteratively with the label shared by the majority of its neighbors until an equilibrium is reached. Experiments have been performed over a dataset for author disambiguation taken from ADS.

DND [64] presents a supervised Distributed Framework for Name Disambiguation (so-called *DND*), developed as a linkage prediction task to overcome the limitations of knowing the number of clusters a priori. Authors of the framework train a robust function to measure similarities between publications to determine whether they belong to the same author. Publications features are transformed into vectors using *Word2Vec*, such publications are subsequently used as nodes in a fully connected publication network where dashed lines denote ambiguity relationships between two authors in a publication pair. Each pair of publications that have an ambiguity relation is processed by a classification task, which returns 1 if the same author writes the pair and 0 otherwise.

Finally, *DND* merges initial partitions by a rule-based algorithm to get the disambiguation result.

MFAND [65] presents a framework called Multiple Features Driven Author Name Disambiguation (so-called *MFAND*). The authors construct six similarity graphs (using the raw document and fusion feature) for each ambiguous author name. The structural information (global and local) extracted from these graphs is inputted into a novel encoder called *R3JG*, which integrates and reconstructs the information for an author. An author is therefore associated with four types of information: the raw document feature, the publication embedding based on the raw feature, the local structural information from the neighborhood, and the global structural information of the graph. Each node is embedded by using the Random Walk on the fusion feature graph. The goal of the framework is to learn the latent information to enhance the generalization ability of the *MFAND*. Then, the integrated and reconstructed information is fed into a binary classification model for disambiguation.

DHGN [66] proposes a Dual-Channel Heterogeneous Graph Network (*DHGN*) to solve the name disambiguation task. In the research, authors use the heterogeneous graph network to capture various node information to ensure the learning of more accurate data structure information. *FastText* is used to extract the semantic information of the data through the textual information, which generates a vector representing each publication. Then the semantic similarity matrix of the publications is obtained, by computing the cosine similarity between such vectors. On the other side, the meta-path Random Walk algorithm is used to extract the features from the publications, especially from the relationships, by computing their feature vectors, and their similarity matrix. Once both the semantic and the relationship features have been exploited, the similarity matrixes are merged to compute the similarity matrix fusion. Such matrix is clustered using *DBSCAN*, an unsupervised clustering algorithm.

SA [67] proposes an approach that uses attention-based graph convolution over a multi-hop neighborhood of a heterogeneous graph of the documents for learning representations of the nodes. The approach consists of an AutoEncoder-based representation learning method divided into an encoder and a decoder. The encoder performs the following operations:

- generates the initial vectors representing the nodes;
- generates node representations based on attention over neighbor types;
- fine-tunes the node representations based on attention over different relation types.

The decoder takes the output of the encoder to generate a homogeneous graph without considering relation types. Finally, vectors coming from the decoder are clustered by using *HAC*.

SSP [68] proposes a method based on representation learning for heterogeneous networks and clustering, and exploits the self-attention technology. The method can capture both structural and semantic features of a publication and uses the weighted sum

of those two embeddings to cluster publications written by the same author with *HAC*. The structural features of a publication are extracted by using meta-paths (in particular *Paper-Author-Paper*, *Paper-Organization-Paper*, *Paper-Venue-Paper*, *Paper-Year-Paper*, and *Paper-Word-Paper*). The representations of publications are subsequently learned by a skip-gram model. The semantic features of a publication are extracted from the title, the abstract, and the keywords by using *Doc2Vec*.

2.4.2 Taxonomy and general workflow of graph-based AND methods

The literature analysis revealed that graph-based AND methods could be represented employing the framework depicted in Figure 2.5. The framework describes different methods as instances of the same workflow template, featuring some or all of the identified steps.

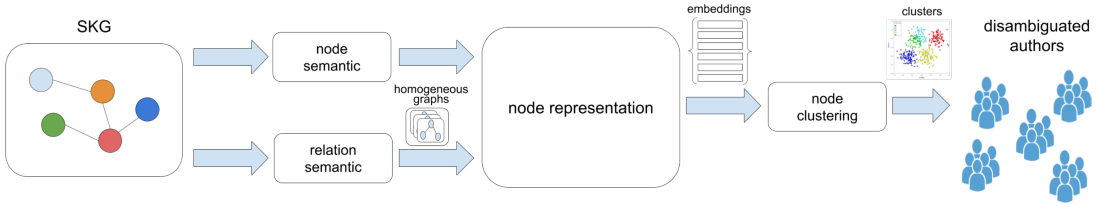

Figure 2.5: *General framework for graph-based AND methods in this survey.*

SKG are characterized by *semantic* features, i.e., a set of publication nodes with title, abstract, author names, venue, and publishing date, and by *semantic* features, i.e., the set of relationships between publication and authors, citation relationships, or other contextual information, such as relationships of publications to organizations, topics, etc. As shown in Figure 2.5, the SKG is the source of information necessary to produce homogeneous *node representations* that will be in turn input to a *node clustering module* that will identify the groups of equivalent author names. Node representations are generated by the *node representation module*, exploiting the semantic and relational features extracted from the input Scholarly Knowledge Graph: via the *node semantic module*, i.e., extracting semantic information from publication metadata, and the *relation semantic module*, i.e., generating *homogeneous graphs* to capture semantic information from the topology of several graph views to provide some context to the next module. *Node representations* can be in the form of embeddings, generated via graph-based methods, or similarity vectors, in turn forming node similarity matrices; in other words, the essence of nodes are captured via *node neighborhood* strategies, with the number of hops determined by the number of layers of the network for the computation, or via *similarity degrees* of the node with all the nodes in the same cluster. Finally, once the node representations have been computed, they are fed to a *node clustering module* whose purpose is to group equivalent author names using strategies that depend on the node representation nature, e.g., embeddings distance, similarity matrix, graph cliques. Typically, clustering is limited to a set of candidate author names, identified by a blocking method that groups all publications related with names of potentially equivalent authors; the majority of methods adopt an LN-FI strategy, e.g., *"John Smith"* generates a key *"smithj"*.

In a more detailed view, the modules can be summarized into:

- The *node semantic module* processes the input SKG to compute vector representations for publications in the SKG to be provided as input to the node representation module. This module produces the so-called "raw embeddings", calculated using the node attributes without considering relations. Algorithms applied within the module could be used to simply clean the data to prepare them for the pair-wise comparison or to convert string attributes to vectors easily comparable with rather simple mathematical operations;
- The *relation semantic module* processes the input SKG to extract homogeneous graphs capturing a specific relational interpretation of authors or publications in the graph. A known approach is the one of "meta-paths" (exploited in [56], [58], [59], [60], [62], [65], [66], and [68]) which creates homogeneous graphs from the input SKG from which meaningful embeddings can be subsequently computed; for example, the application of the "co-author" meta-path or the "co-venue" meta-path approach generates respectively a graph where authors are nodes linked by relationships if they co-authored one publication or a graph where publications are related if they share the same venue. Such graphs can be used to generate different author representations, capturing relational features of the graph and the authors therein. The module can be used to generate one or more graphs and potentially these can be merged to produce compound views;
- The *node representation module* is the core of the framework as it processes the input of the node semantic and relation semantic modules (i.e., homogeneous graphs and/or the publication vector representations) to generate node representations to support the subsequent clustering of publications written by the same authors; a node representation captures the semantic features of an individual publication and in some cases include features of a given author; node representations can be: (i) similarity vectors obtained by similarity comparisons between pairs of publications potentially written by the same author, identified via LN-FI pre-clustering strategies on author names; (ii) node embeddings obtained via networks applied to nodes in the neighborhood of the graphs resulting by merging node and relation semantic information;
- The *node clustering module* applies a clustering method to blocks of publications potentially written by the same author, identified via LN-FI pre-clustering strategies on author names. Clustering algorithms differ depending on the nature of node representations, i.e., clustering functions acting on graphs for similarity vectors or clustering functions on an embedding vector space.

Section 2.4.2 sums up the results of the analysis to better highlight the differences between the methods reviewed in this survey. The table indicates how each module specifically fulfills the workflow modules described above for each method in the review.

Following the comparison of the different methods, Figure 2.6 identifies a taxonomy that classifies AND approaches in three macro features. A method can be characterized concerning the *learning strategy*, the *evidence explored*, i.e., the type of information used to create the author representations, and the *node representation strategy*. A recap

method	article	semantic module	relation module	node representation module	clustering module
LAND	[56]	SPECTER, LiteralE	N.A.	DistMult multimodel extension	HAC
HGCN	[58]	Doc2Vec	meta-path	HGCN	Graph-enhanced HAC
AE	[59]	Word2Vec	meta-path	Variational AutoEncoder	HAC
jGAT	[60]	Word2Vec	meta-path	GAT	spectral clustering
RF-LRC	[61]	N.A.	N.A	Random Forest & Logistic Regression Classifier	infomap algorithm
sGCN	[62]	Word2Vec	meta-path	specialized GCNs	HAC
LP	[63]	N.A.	N.A.	tree-based classifier	label propagation
DND	[64]	Word2Vec	N.A.	N.A.	rule-based algorithm
MFAND	[65]	Random Walk	N.A.	R3JG	binary classification
DHGN	[66]	FastText	N.A.	Random Walk	DBSCAN
SA	[67]	Word2Vec	N.A.	Spectral GCN & Dense Network	HAC
SSP	[68]	Doc2Vec	meta-path	skip-gram	HAC

Table 2.2: *Recap of AND methods modules.*

of the surveyed graph-based AND methods concerning this taxonomy is depicted in Section 2.4.2.

The first and most common feature in deep learning surveys is the *learning strategy*, which defines the approach used to train the graph-based network. Depending on the training methodology, a method may be supervised [61, 64], unsupervised [56, 58–60, 62, 65–68] and semi-supervised [63]. Usually, unsupervised methods include a preliminary blocking stage and subsequently an approach based on *Graph Convolutional Networks (GCN)* and/or *AutoEncoders* to vectorize graph elements by creating an embedding of the publication node. In GCN methods the embedding of a node is usually created by aggregating information from the node and its related neighbors, while AutoEncoders exploit an artificial neural network to generate efficient encodings of nodes. Supervised methods sometimes include a preliminary stage of blocking, followed by pair-wise comparisons exploiting either a network or a classifier trained to recognize whether a pair of author names is equivalent. Semi-supervised approaches tend to be a mixture of the above-mentioned methods. A known challenge of such non-unsupervised approaches is the cost of producing a well-defined training set of data.

Graph-based AND approaches can be classified according to the *explored evidence*, intended as the type of information used to generate the embeddings. In the case of a method based on *publication features*, the procedure relies on publication attributes such as title, date, and abstract to represent a node. Instead, a method based on *relations and publication features* takes advantage of the relations between publications and,

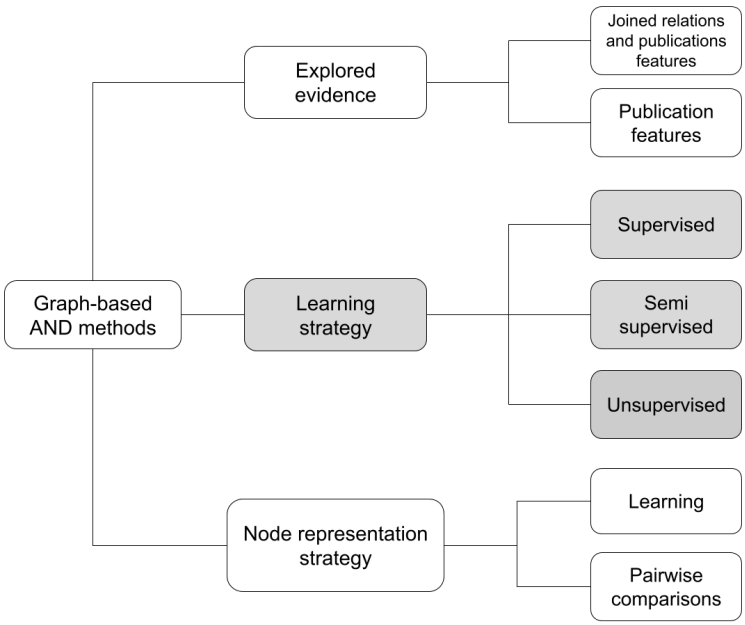

Figure 2.6: *Proposed taxonomy for graph-based AND methods.*

in general, nodes of the SKG. As highlighted in many AND surveys, such methods, capturing both topological and semantic representations of a node, turn out to be the most promising in the literature.

The *node representation strategy* feature describes the strategy used to compute node representations. *Learning* methods compute embeddings of the nodes by aggregating information from their neighborhood. They are typically applied on homogeneous graphs as returned from the graph processing module (e.g., the meta-path approach described above). *Pairwise comparison* methods [61, 63, 64] compare nodes of the graph with others to generate similarity vectors in which each element indicates the similarity degree between the target node and the others. Depending on the method, the vector may contain 0 (different author) or 1 (same author), or a similarity degree. Such representations can be used to create a similarity graph (in some cases called "ambiguity graph") in which clustering algorithms can identify "cliques" of nodes, e.g., groups of equivalent author names.

2.4.3 Main lacks of graph-based AND

Graph-based methods have not only demonstrated their effectiveness in comparison to traditional approaches within the existing literature but have also shown to be the most important trend in contemporary research on AND. Since the accuracies of the methods presented in this brief survey were comparable, the selection of a specific approach depends on the context to which the AND is applied. An important consideration is made over the use case scenario: the SKG. In these contexts, users have access to a wide range of information collected from the various relationships (e.g., *cites*, *fundedBy*, *producedBy*). The increasing availability of such kind of information suggests that using techniques that take advantage of them is often essential for achieving optimal outcomes.

method	article	learning strategy	node representation strategy	explored evidence
LAND	[56]	unsupervised	learning	joint relation and publication feature
HGCN	[58]	unsupervised	learning	joint relation and publication feature
AE	[59]	unsupervised	learning	joint relation and publication feature
jGAT	[60]	unsupervised	learning	joint relation and publication feature
RF-LRC	[61]	supervised	pairwise comparisons	publication features
sGCN	[62]	unsupervised	learning	joint relation and publication feature
LP	[63]	semi-supervised	pairwise comparisons	publication features
DND	[64]	supervised	pairwise comparisons	publication features
MFAND	[65]	unsupervised	learning	joint relation and publication feature
DHGN	[66]	unsupervised	learning	joint relation and publication feature
SA	[67]	unsupervised	learning	joint relation and publication feature
SSP	[68]	unsupervised	learning	joint relation and publication feature

Table 2.3: *Recap of graph-based AND methods.*

On the other hand, when dealing with flat entity collections (i.e., when relationships between entities are not available), the construction of a graph formed by edges representing similarity relationships between nodes emerges as the most viable strategy. This consideration clearly shows the importance of taking advantage of the graph structure, because it can provide more flexibility and adaptability to every scenario.

In these types of approaches, unsupervised learning methods have high popularity in the literature because they can imitate the complexities of real-world AND scenarios where authors are frequently scarcely described. Nonetheless, a semi-supervised approach is still a viable solution as it can mitigate the challenges of unsupervised learning by taking advantage of the known information available.

The synthesis of insights learned from an exhaustive literature review and a comprehensive study of graph-based AND methodologies lead to several observations. The

framework depicted in Figure 2.5 can correctly encapsulate the characteristics of state-of-the-art disambiguation approaches, grouping stages under common blocks that describe the activities of different methods. Nonetheless, some critical aspects remain unsolved. For example, those concerning the evaluation of disambiguation results.

To enhance the effectiveness of the disambiguation process, the integration of traditional techniques with AI-based solutions, particularly those grounded in GNN, takes center stage. A GNN-based approach is aligned with the requirements of AND tasks, exhibiting the capacity to address challenges stemming from incomplete attribute information and elevating the overall quality of disambiguation outcomes. A hybrid model, fusing the strengths of traditional methods and AI-based solutions, emerges as a promising strategy to face the AND tasks.

CHAPTER 3

The motivating scenario

This chapter presents the real-case scenario of the OpenAIRE Graph, which has been used to test and validate the frameworks and the AI architectures that are contributions of this thesis. The OpenAIRE Graph is a production service, co-funded by the European Commission, which ensures the operation and maintenance of an extensive Scholarly Knowledge Graph modeling scientific publications, research data, research software, authors, and other related entities in support of several worldwide discovery and research monitoring services. The provisioning workflow guaranteeing the OpenAIRE Graph is currently operated by the Institute of Information Science and Technologies of the Italian National Research Council (CNR), where many of the underlying technologies and solutions have been devised and engineered. In particular, its AND challenges inspired the solutions discussed in the thesis whose results have been integrated as part of the service (FDup framework) or are being experimented with within its BETA environment.

The chapter is organized as follows: Section 3.1 describes the OpenAIRE Graph, which will be used as a benchmark for all the contributions of the thesis; Section 3.2 presents the OpenAIRE data model, i.e., the entities and relationships populating the OpenAIRE Graph; Section 3.3 details the process engineered to create the OpenAIRE Graph; Section 3.4 discusses about the current techniques to perform the AND task in the OpenAIRE Graph.

3.1 The OpenAIRE Graph

The OpenAIRE AMKE[1] is a not-for-profit legal entity co-funded by the European Commission to foster and support Open Science publishing workflows in Europe and glob-

[1] OpenAIRE – `http://www.openaire.eu`

ally. Its members, more than 50 research organizations in Europe, establish collaborations via a strong network of Open Science stakeholders to align policies, practices, guidelines, and tools at the global level, across countries and disciplines. To this aim, it supports and operates a rich portfolio of services in support of Open Science publishing, discovery, and monitoring [2]. The core service of its portfolio is the OpenAIRE Graph, one of the largest SKGs worldwide, representing the overall publishing record, including research products (publications, data, software), authors, funders, projects, and organizations. The Graph aggregates millions of metadata records from thousands of scholarly data sources, including Open Access journals (from DOAJ registry), institutional and thematic repositories (from OpenDOAR, re3data, FAIRSharing registries, e.g., UKPubMed [69], ArXiv [70], HAL, Zenodo, Figshare [71], Dryad, and Repec), national or disciplinary aggregators, but also large persistent identifiers databases such as Crossref [72], ORCID, ROR.org, and Datacite [73].

After metadata cleaning, disambiguation, enrichment, and full-text mining processes, the Graph is made available to all, via dumps and APIs[3], for research purposes or the operation of discovery and monitoring services. For example, the OpenAIRE EXPLORE[4] and the OpenAIRE CONNECT[5] offer discovery gateways over the OpenAIRE Graph APIs; the OpenAIRE MONITOR[6] and the Open Science Observatory[7] offer access to statistics and numbers regarding Open Science trends.

3.2 Graph data model

The OpenAIRE Graph comprises several types of nodes and relationships among them. Their structure and semantics are illustrated in Figure 3.1.

3.2.1 Node types

The main node types are described in brief as follows:

Results represent the outcome (or products) of research activities and include research publications, research data, research software, and other kinds of products;

Data Sources are the sources from which the metadata of graph objects are collected;

Organizations correspond to companies or research institutions involved in projects, responsible for operating data sources or consisting of the affiliations of product creators;

Projects are research grants awarded by a funding agency, such as a National or international funder or a foundation;

Authors are authors of research products with an ORCID identifier.

[2] OpenAIRE Catalogue – `http://catalogue.openaire.eu`
[3] OpenAIRE Public APIs – `https://develop.openaire.eu`
[4] OpenAIRE EXPLORE, `https://explore.openaire.eu`
[5] OpenAIRE CONNECT, `https://connect.openaire.eu`
[6] OpenAIRE MONITOR, `https://monitor.openaire.eu/`
[7] Open Science Observatory, `https://osobservatory.openaire.eu/home`

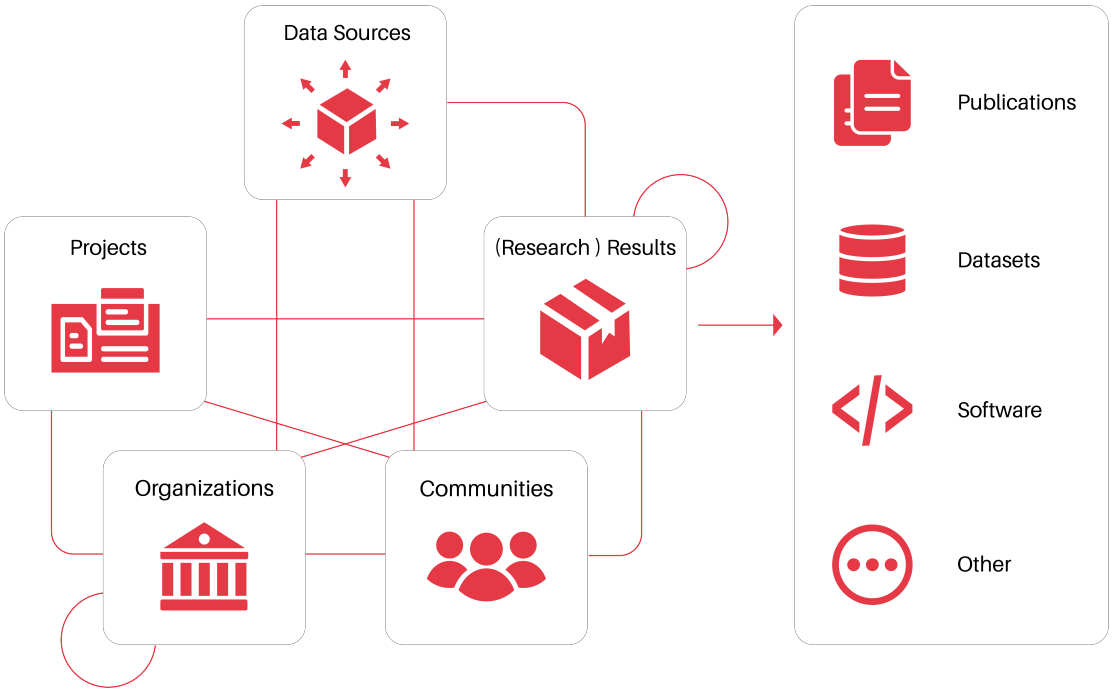

Figure 3.1: *OpenAIRE Graph data model.*

Results

Results are intended as digital objects, described by metadata, resulting from a scientific process. They are described by a common set of attributes that includes the OpenAIRE identifier (i.e., that identifies the node univocally in the OpenAIRE context), the main title, the authors' list, the description (i.e., the abstract or in general a brief description of the resource and the context in which the resource was created), and the persistent identifiers (i.e., original identifiers for the result such as DOIs). The list of authors is provided as a set of strings, each representing an author's name. As explained below, an edge between the *Result* and the *Author* is provided when the author is also a node in the graph (i.e., it has an ORCID identifier). Results are further categorized into four sub-types, which inherit all their properties and extend them when needed. Sub-types include *Publications* (i.e., the research literature, including articles, presentations, books, thesis), *Dataset* (i.e., the research data, including images, sounds, datasets), *Software* (i.e., the research software), and *Other Research Products* (i.e., the research products that cannot be classified as research literature, data or software, including lectures, annotations, collections).

Data Sources

Data sources export information packages (e.g., XML records, HTTP responses, RDF data, JSON) that may contain information on one or more of the graph nodes. OpenAIRE Graph nodes are created from data collected from various data sources, such as publication repositories, dataset archives, CRIS systems, funder databases, etc.

For example, a metadata record about a project carries information for the creation of a *Project* node and its participants (as *Organization* nodes). It is important, once each

piece of information is extracted from such packages and inserted into the OpenAIRE Graph as a node, to keep provenance information relative to the originating data source. This is to give visibility to the data source, but also to enable the reconstruction of the very same piece of information if problems arise. The attributes of a *Data Source* include the OpenAIRE identifier and the original name of the data source.

Organizations

Organizations include companies, research centers, or institutions involved as project partners or as responsible for operating data sources. Information about organizations is collected from funder databases like CORDA, registries of data sources like OpenDOAR and re3Data, and CRIS systems, as being related to projects or data sources. The attributes of an *Organization* include the OpenAIRE identifier and the legal name of the organization.

Projects

Projects are characterized by a list of funding streams (e.g., FP7, H2020 for the EC), which identify the strands of funding since the identification of such pieces of information is of crucial interest to OpenAIRE. Funding streams can be nested to form a tree of sub-funding streams. The attributes of a *Project* include the OpenAIRE identifier, the title and the funding information.

Authors

Authors are characterized by an ORCID identifier, a name, a surname, and a full name. It should be noticed that authors appear in the graph both as strings, hence properties of Results (e.g., the author name of an article), and graph nodes, hence author entities with ORCID identifiers. In many cases, metadata records of Results feature hybrid scenarios, with part of their authors provided as nodes and part as strings. This is because bibliographic metadata is often provided with partial information about the authors.

Table 3.1 presents statistics on the number of node types in the latest version of the OpenAIRE Graph. Such information is kept up to date and made available in the OpenAIRE Graph website[8].

Node type	Node subtype	Number of nodes
Result	Publication	175M
	Dataset	60M
	Software	364K
	Other	21M
Data Source	-	130K
Organization	-	312K
Project	-	3.5M
Author	-	2.1B

Table 3.1: *Approximate statistics for node types in the OpenAIRE Graph.*

[8]OpenAIRE Graph statistics – `https://graph.openaire.eu/statistics`

3.2.2 Edge types

The semantic relationships between nodes in the graph, are modeled by several edge types, described in brief as follows:

Project to Result edges indicate that a Result has been co-funded by a given research grant or Project and that a Project co-funded a list of Results (i.e., *produces/isProducedBy* relationships);

Project to Organization edges indicate the Organizations participating in a Project and the list of Projects an Organization has participated in (i.e., *hasParticipant/isParticipant* relationships);

Result to Result edges are the most common edges in the graph, describing semantic relationships such as citations, similarity, versioning, and supplementary material;

Result to Author edges link a Result to its authors when these provide an ORCID identifier;

Result to Organization edges relate Results and Organizations when an author of the Result is affiliated to the Organization (i.e., *hasAuthorInstitution/isAuthorInstitutionOf* relationships);

Result to Data Source edges indicate the provenance of the metadata record of the Result (i.e., *isProvidedBy/provides* relationships);

Data Source to Organization edges indicate when a data source is operated by an Organization (i.e., *isProvidedBy/provides* relationships).

Table 3.2 presents statistics on the number of edge types in the latest version of the OpenAIRE Graph.

Edge type	Edge subtype	Number of edges
Result edges	Publication to other nodes	150M
	Dataset to other nodes	150M
	Software to other nodes	413K
Affiliation	Publication	153M
	Dataset	1.2M
	Software	15K
	Other	5M

Table 3.2: *Approximate statistics for edge types in the OpenAIRE Graph.*

3.3 Graph provision workflow

The OpenAIRE Graph is the result of collecting and aggregating metadata records from more than $100,000$ scholarly communication sources from all over the world, including open-access institutional repositories, data archives, journals, and entity registries for people, data sources, organizations, projects, and funders. Dedicated harmonization methods are applied to the records, to generate a uniform SKG, which is further enriched by inferring metadata properties and links via full-text mining of the PDFs of $22Mi$ (and growing) Open Access publications. Enrichments complete the resulting

SKG with links between results and projects, author affiliations, subject classification, and links to entries of domain-specific scientific databases (e.g., ProtDB). Finally, the resulting Graph is disambiguated, to group information about equivalent *Results*, *Organizations*, and *Data Sources*. Equivalent nodes for such entities are identified and merged to obtain an open, trusted, public SKG enabling unprecedented explorations and analysis of the scholarly communication landscape.

3.3.1 Aggregation

The OpenAIRE Graph is materialized as an open knowledge graph where products of the research life-cycle are semantically linked to each other, carry information about their access rights (i.e., if they are Open Access, Restricted, Embargoed, or Closed), and are associated with the sources in which they are hosted. This is done via a set of autonomic, orchestrated workflows operating in a regimen of continuous data aggregation and integration [74, 75]. Such workflows harvest "raw" metadata records via standard protocols (e.g., OAI-PMH, REST APIs, FTP) from thousands of data sources and harmonize them to converge to the OpenAIRE data model structure and semantics. Such mapping effort is far from trivial and requires ad-hoc conversions from data source metadata exchange formats to the OpenAIRE Graph model. Such process is facilitated by the fact that many of the data sources comply with interoperability standards defined by the OpenAIRE Graph, called OpenAIRE Guidelines for Content Providerds[9]. On the other hand, dedicated workflows must be defined for key data sources such as Crossref and DataCite registries, the ORCID registry of scientists, the ROR.org registry for research organizations, the DOAJ, OpenDOAR, re3data, and FAIRSharing registries for data sources.

The initial aggregation graph, shown in Figure 3.2, ensures that nodes obey common structure and vocabularies. However, this process cannot ensure the completeness and correctness of the data, whose accuracy depends on the data acquisition at the local data source sites. For example, the OpenAIRE Graph collects 450 million records, of which the 30% does not provide an abstract for products and the 45% does not include an ORCID identifier.

3.3.2 Enrichment

The OpenAIRE Graph exploits information inference to compensate for the lack or incompleteness of information in the "aggregation graph". In particular, two kinds of inference processes are applied: via full-text and metadata.

In the first case, the aggregation workflows collect, when the license and the proper URLs are provided, the full texts of the publications, to apply text mining algorithms. Such algorithms are intended to extract properties and relationships and inject them into the graph. The PDF Aggregation Service, based on the publications' URLs found in the metadata, crawls the web to locate and download the full texts of the Open Access publications; as of today, more than $22M$ PDFs/XMLs are downloaded and their full-texts extracted and provided as input for mining.

Several mining modules are applied:

[9] OpenAIRE Guidelines – `http://guidelines.openaire.eu`

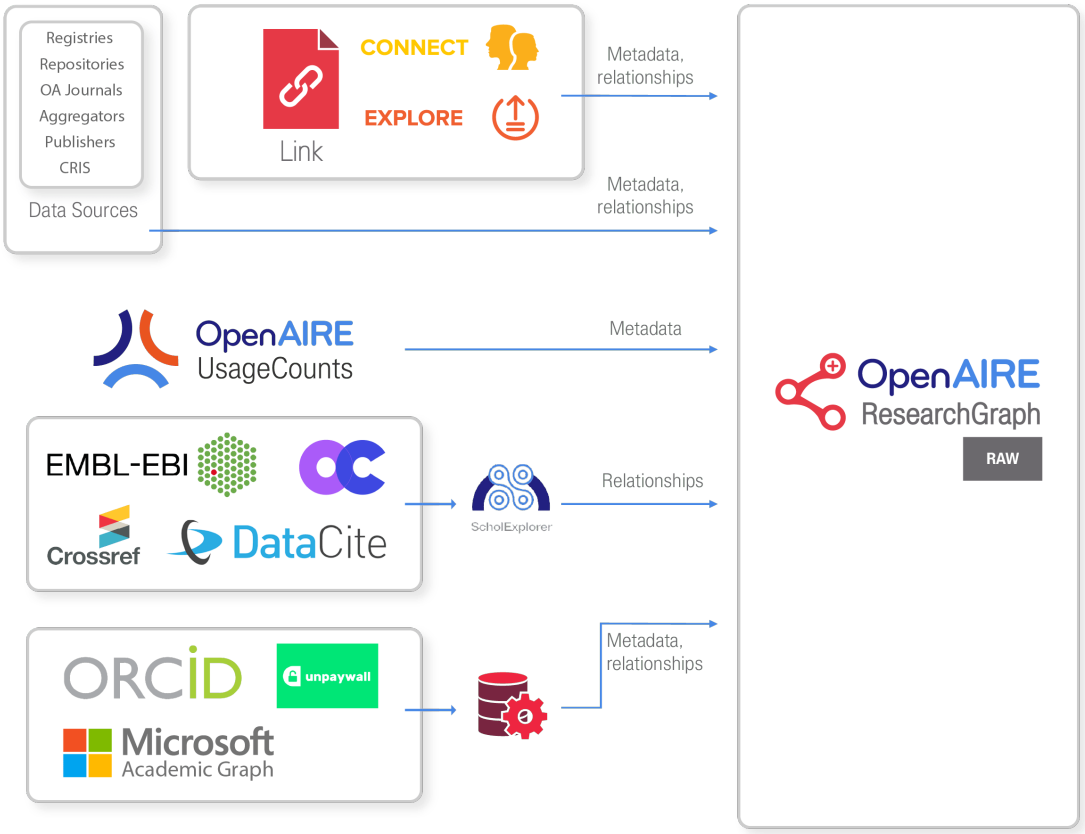

Figure 3.2: *The OpenAIRE Graph aggregation process.*

- the *Affiliation matching* module has the goal of matching affiliations (Result-Organization edges) extracted from the PDF and XML documents with organizations from the OpenAIRE organization database;
- the *Citation matching* module has the goal of discovering links between documents to create citation edges;
- the *Document classification* module has the goal of assigning a scientific text to one or more predefined topics from known vocabularies, such as Sustainable Development Goals, Fields of Science, Mesh classification, etc.;
- the *Documents similarity* module has the goal of finding similar documents among the ones available in the OpenAIRE Graph and produces similarity edges;
- the *Acknowledgement extraction* module has the goal of scanning the plain text of the document to extract grant identifiers of funders to link a research product to a research community;
- the *Metadata extraction* module aims to extract metadata information such as abstracts, titles, keywords, etc..

In the second case of inference, the OpenAIRE Graph is enriched by *deduction* and *propagation* processes.

The deduction process (also known as "bulk tagging") enriches each record with new information that can be derived from the existing property values. For example, the deduction can be used to assign a Result to a given disciplinary vocabulary term starting from existing keywords or other subjects.

The propagation process is instead used to migrate information from one node to another node, thanks to the existence of an edge whose semantics strongly relate to the two nodes. For example, when a node A *isSupplementOf* a node B, the semantics claims that A comes as a supplementary product of B; e.g., research data or software that comes as a supplement of a scientific article. In such a case, the semantics allows, if B is funded by a given project, to propagate the link to A, adding an edge *producedBy* between A and the project.

3.3.3 Disambiguation

The graph resulting from the aggregation and enrichment phases still suffers from a high duplication rate. Indeed, the OpenAIRE Graph is populated by aggregating metadata records from distinct data sources whose content typically overlaps. For example, the collection of article metadata records from the publisher' archives (e.g., Frontiers, Elsevier, Copernicus) and from pre-print platforms (e.g., ArXiv.org, UKPubMed, BioarXiv.org). To support the monitoring of science, the OpenAIRE Graph implements record deduplication and merge strategies so that scientific production can be consistently statistically represented. Such strategies reflect the following intuition behind OpenAIRE monitoring: *"Two metadata records are equivalent when they describe the same research product, they feature compatible resource types, have the same title, the same authors, or the same PID"*. Finally, groups of duplicates can be whitelisted or blacklisted, to manually refine the quality of this strategy.

It should be noted that publication dates do not make a difference, as different versions of the same product can be published at different times; e.g., the pre-print and a published version of a scientific article, which should be counted as one object; abstracts, subjects, and other possible related fields, are not used to strengthen similarity, due to their heterogeneity or absence across different data sources. Moreover, when a product is indicated as a new version of the other (same title, same abstract, etc.), the presence of different authors will not bring them into the same group to avoid the unfair distribution of scientific rewards.

Groups of duplicates are finally merged into a new record that embeds all properties of the merged records and carries provenance information about the data sources and the relative "instances" (i.e., manifestations of the products, together with their resource type, access rights, and publishing date). The disambiguation of the various nodes in the graph is already implemented employing the FDup framework, whose description is left to Chapter 4.

It is important to mention that edges involved in nodes identified as duplicated are eventually marked as virtually deleted and used as a template for creating a new relation pointing to the new representative record, as shown in Figure 3.3. Nodes and relationships marked as virtually deleted are not exported via APIs or data dumps.

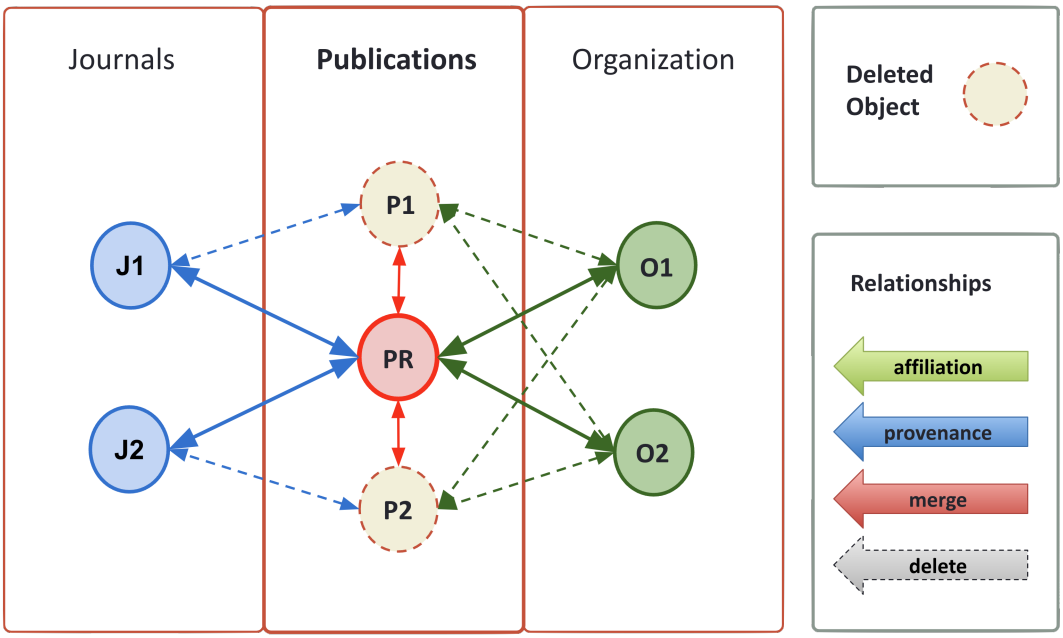

Figure 3.3: *The re-distribution of edges in the OpenAIRE Graph.*

3.3.4 Indexing

The last step of the graph provision workflow is to finalize the graph by ensuring an overall minimal quality degree. Bibliographic records that do not meet the minimal completeness and consistency requirements for being part of the OpenAIRE Graph are purged during this phase. Such requirements can vary over time and currently include the mandatory presence of title, date, and authors for Result nodes.

The resulting final version of the OpenAIRE Graph is indexed on a high-throughput Solr server whose APIs are used by the OpenAIRE portals EXPLORE[10] and CONNECT[11]. The APIs are also heavily used by several third-party applications and organizations, such as:

- The OpenAIRE Graph APIs and Portals offer to the EOSC (European Open Science Cloud) an Open Science Resource Catalogue, keeping an up-to-date map of all research results (publications, datasets, software), services, organizations, projects, funders in Europe and beyond.
- EC participant portal (Sygma - System for Grant Management) uses the OpenAIRE API in the "Continuous Reporting" section. Sygma automatically fetches from the OpenAIRE Search API the list of publications and datasets in the OpenAIRE Graph that are linked to the project. The user can select the research products from the list and easily compile the continuous reporting data of the project.
- ScholExplorer is used by different players of the scholarly communication ecosystem. For example, Elsevier uses its API to make the links between publications and datasets automatically appear on ScienceDirect.

[10] OpenAIRE EXPLORE – `http://explore.openaire.eu`
[11] OpenAIRE CONNECT – `http://connect.openaire.eu`

- DSpace & EPrints repositories can install the OpenAIRE plugin to expose metadata records in compliance with OpenAIRE via their OAI-PMH endpoint and offer researchers the possibility to link their depositions to the funding project, by selecting it from the list of projects provided by OpenAIRE.

The OpenAIRE Graph is also processed by a pipeline for extracting the statistics and producing the charts for funders, research initiatives, research infrastructures, and policymakers available via the OpenAIRE MONITOR[12]. Based on the information available on the graph, OpenAIRE MONITOR provides a set of indicators for monitoring the funding and research impact and the uptake of Open Science publishing practices, such as FAIRness indicators, Open Access publishing of publications and datasets, availability of interlinks between research products, availability of post-print versions in institutional or thematic Open Access repositories, etc.

3.3.5 Evaluation

The operations leading to the creation of the OpenAIRE Graph are evaluated by employing persistent identifiers. Those are used to establish whether the disambiguation process produced a correct result and to verify the duplication of the repositories from which the data to form the OpenAIRE Graph has been collected.

Evaluation of the repositories duplication

Scholarly repositories are pivotal to Open Science, as they serve as the main source for accessing research products and ensure their preservation. Such repositories are collected from data registries, which are authoritative sources including information about them. However, the existence of multiple registries may lead to the duplication of scholarly repositories, resulting in an undesired adversary for the disambiguation task. Therefore, evaluating the overlapping of the repositories inside the registries becomes crucial to lighten the disambiguation complexity. This task has been studied as a lateral contribution in this thesis [76], and it focuses on the investigation of four prominent scholarly registries: FAIRsharing[13], re3data[14], OpenDOAR[15], and ROAR[16].

The methodology used for this aim incorporates registry claims, such as "same-as" equivalences among repository profiles registered across different registries, to generate initial duplicate sets. These sets are subsequently expanded by integrating them with the groups obtained via the FDup framework to enrich the information to be used for the understanding of the repository landscape. For a visual representation of the methodology, refer to Figure 3.4.

Evaluation of the publication disambiguation

Similarly, evaluating the accuracy of the entity disambiguation becomes crucial to understanding the reliability of the result produced by the disambiguation algorithm. This task has been studied as a lateral contribution of this thesis as well [77], and it focuses

[12] OpenAIRE MONITOR – `http://monitor.openaire.eu`
[13] FAIRsharing – `https://fairsharing.org`
[14] re3data registry – `https://re3data.org`
[15] OpenDOAR registry – `https://v2.sherpa.ac.uk/opendoar`
[16] ROAR registry – `http://roar.eprints.org/information.html`

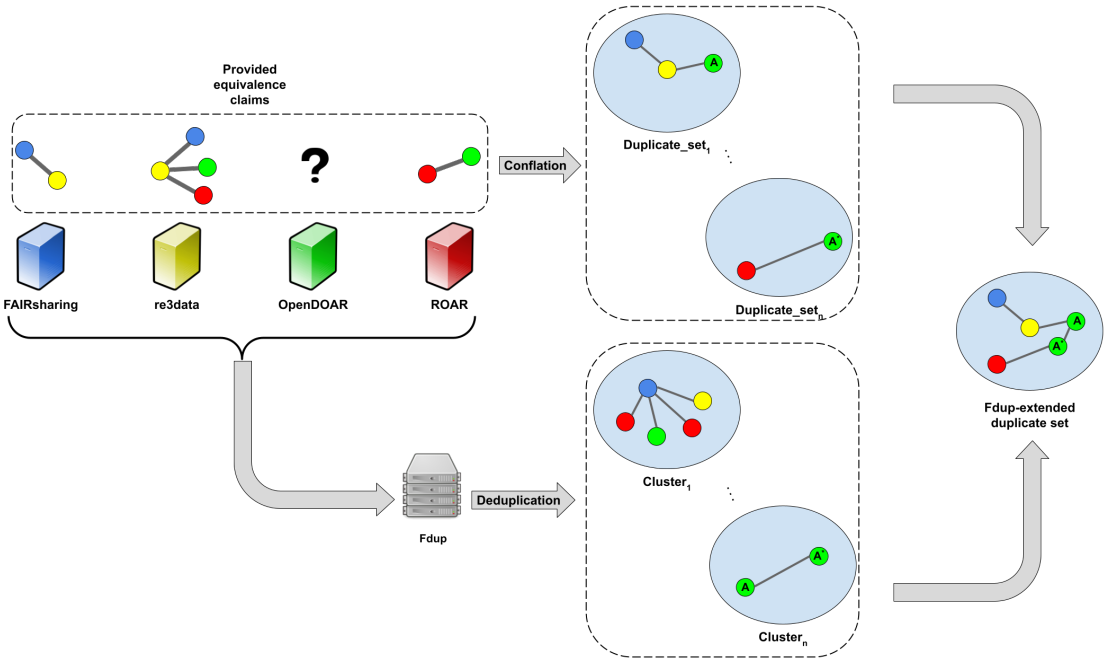

Figure 3.4: *Overview of the methodology. Firstly, the claims provided by the registries are conflated to derive duplicate sets; then, the identified duplicate sets are extended with the groups of duplicates provided by the FDup framework running against the content of the registries. In this example, FDup discovers an equivalence between repository profiles A and A^*, and this triggers a merger between the two duplicate sets represented above.*

on the evaluation of the research publication disambiguation. In this context, experiments can be divided into two groups: first, the comparison of the disambiguation results to sets of known DOI aliases, and then the investigation of equivalent objects that do not correspond to known aliases.

Quantifying disambiguation false negatives using DOI aliases To perform a preliminary analysis on the quality of the disambiguation result, information from doi.org's API regarding DOI aliases has been used. Reporting aliases is the default mechanism for registrants of DOIs to report duplicate DOIs. Since not all duplicates are reported by the respective registrants, DOI aliases cannot be used to quantify the false positives that DOI disambiguation algorithms produce. However, any DOIs that have been reported as aliases are guaranteed to refer to equivalent objects, hence they can be used as a ground truth to quantify false negatives and this is how we leveraged them in this experiment. Since gathering the aliases for all distinct DOIs is a time-consuming process (especially, if the implemented process makes responsible usage of the API respecting request limits), the analysis has been restricted only to those DOIs that are reported to have at least one equivalent DOI according to the disambiguation strategy used for the OpenAIRE Graph. The snapshot of the graph (produced on October 26th, 2021) contained $112,216,333$ distinct disambiguated entities in total, $5,885,861$ of which contained at least two equivalent DOIs. Using doi.org's API, aliases of respective distinct DOIs have been gathered ($14,427,982$ in total) and used to generate the corresponding groups of DOI aliases. Subsequently, these sets of aliases have been compared with

the sets of equivalent entities provided by the disambiguation algorithm. During this comparison, all unresolvable DOIs have been ignored (i.e., the analysis was performed using the rest). A summarization of the results of this phase is depicted in Table 3.3.

Types of duplicated entities	Number of entities
Completely matching sets of aliases (true positives)	32,476
Involving unreported aliases (false negatives)	1,100
Not compliant to aliases (false positives or missing aliases)	5,852,174
Containing only unresolvable DOIs	111

Table 3.3: *Statistics for the various types of disambiguated entities.*

In particular, it may be worth noticing that a lot of groups of duplicates found by the FDup framework ($32,476$) were completely compliant with the list of known aliases (i.e., confirmed true positives). Also, $1,100$ of the entries could be considered false negatives since they did not contain even one known alias. However, the vast majority of the groups of duplicates contained groups of known aliases (hence, implying missing aliases or false positives). Finally, only a negligible number of groups of duplicates contained only unresolvable DOIs. After this analysis, it is evident that the OpenAIRE disambiguation produced a very small number of confirmed false negatives (they account for less than 0.02% of the examined entities). In addition, it seems that the algorithm identifies many equivalent DOIs that are not reported as aliases in doi.org.

Investigating reported equivalent objects with no DOI aliases The second step of the analysis consists of the determination of whether the result can be mainly attributed to a large number of false positives, or if a huge number of equivalent DOIs are not reported as aliases. The main objective was to get insights into the scale of false positives in the disambiguation result. The only way to fulfill this objective is to have expert judgments on the sets of equivalent objects that the deduplication algorithm produces. The experts use DOI-related metadata and the corresponding content (e.g., the manuscript in case of publication) and provide judgments regarding the correctness of the disambiguation result (i.e., if the reported DOIs correspond to equivalent objects or not). To this aim, the respective tasks have been assigned to 4 experts (computer engineers, two of them PhDs). However, since the manual inspection is time-consuming and the data to be examined is immense (more than $5.8M$ entries), a sample of 300 randomly selected entries per expert has been assigned, resulting in a dataset of $1,200$ entries. Each expert was given the task of assigning each set of equivalent DOIs with one of 8 predetermined class labels (Table 3.4).

Each of the classes has particular semantics, explained in the "Interpretation" column. These semantics determine whether the objects in the respective group are equivalent or not ("Judgement" column). Figure 3.5 illustrates the proportion of the groups of duplicates that have been annotated with each of the classes, and summarises the proportion of true and false positives; due to the existence of the "Ambiguous" class, there were also a lot of entries for which it was not possible to provide a judgment (denoted by "N/A"). It is evident that the majority of the groups of duplicates (64.9%) produced by the algorithm are correct; most of them contain different versions of the same object and extensions of older works. The false positives, on the other hand, correspond to a significantly smaller percentage (23%).

Name	Interpretation	Judgment
Ambiguous	At least one DOI is invalid (no metadata are available)	N/A
Deleted duplicates	DOIs once pointing to the same research object, currently deleted	True
Multi-published	Publication published in more than one location (full or abstract)	True
Versions	Multiple versions of the same research object (e.g., pre-prints, post-prints)	True
Erroneous	Unrelated set of objects	False
Paper-extensions	Extended version of a conference paper in a journal	False
Part-of-a-group	Multiple parts of the same research object (e.g., photos of the same collection)	False
Supplementary	Publication and its supplementary material (including errata)	False

Table 3.4: *Statistics for the various types of groups of duplicates.*

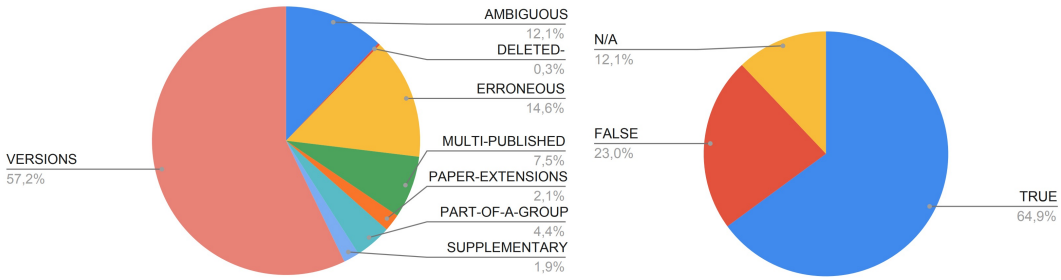

Figure 3.5: *Expert evaluation results.*

3.4 AND in the OpenAIRE Graph

AND in the OpenAIRE Graph is applied by exploiting the presence of ORCID identifiers. This means that whenever a *Result* record provides the ORCID associated with one of its authors, the OpenAIRE Graph will feature the corresponding *Author* node and *Result* node with an *hasAuthor* edge between the two. This policy ensures that all records that refer to the same ORCID identifier will be linked with an edge to the same *Author* node independently from the original *Data Source*. This configuration of the data provides a "safe" AND because it is sufficient to use the edges between the *Author* and the *Result* to immediately derive the list of research products written by the same author. In case an *Author* has multiple ORCIDs because they are associated to its instance in different institutions, the OpenAIRE Graph will feature only one ORCID to discourage duplication. Having said that, multiple ORCIDs are not popular in the data collected in the graph. It is important to mention that the ORCID-based disambiguation also guarantees a precision close to 100% because ORCID can be considered to be a trusted source of information with an extremely low error rate, however dependent on the correctness of information inserted in the ORCID profile. On the other hand, this way of disambiguating authors suffers from a low recall because it is not guaranteed that ORCID identifiers are provided for each *Author* node in the Graph.

As things stand in the OpenAIRE Graph approach, no methods are implemented to identify duplicate authors across author names (i.e., extracted from the *Results*) and authors with ORCID identifiers. In the state-of-the-art, author names are not fully represented in the Graph with a unique node unless the authors keep an extremely clean and up-to-date ORCID profile; in this case, since the OpenAIRE Graph includes the whole ORCID dataset of *Authors* and related *Results*, the disambiguation of Results will deliver a consistent and potentially richer *Author* record in the Graph because all the duplicate *Results* can inherit the *hasAuthor* edges of a richer *Result*.

The reason behind the adoption of an ORCID-based approach is twofold. On the one hand, the computational complexity of the problem is brought by the fact that OpenAIRE Graph counts more than $2B$ author names. On the other hand, the precision and recall trade-off is complicated by the lack of author name characterization. As mentioned above, high precision is guaranteed by the ORCIDs while the recall can be possibly improved by implementing a way of linking author names with ORCID authors. The problems in this context are those already described in 1.2.

The outcomes of this thesis are intended to mitigate the challenges of performance, precision, and recall opening the way to implementing a full AND strategy for the OpenAIRE Graph. In particular, this thesis faces efficiency and effectiveness by providing the following methods:

Efficiency A general-purpose framework and tool developed to perform and optimize the whole traditional disambiguation workflow (see Chapter 4);

Effectiveness Two GNN-based architectures leveraging evaluation of disambiguation outcomes as a tool to fine-tune disambiguation methods (see Chapter 5).

CHAPTER 4

Enhancing efficiency via computational complexity reduction

This chapter presents a novel solution to the enhancement of the efficiency of the disambiguation process. FDup (Flat Collections Deduper) is a software framework based on the Apache Spark Framework that was conceived as an enhancement of the GDup framework (Graph Deduper) [78]. GDup implements a full disambiguation process that identifies duplicates and delivers strategies to resolve duplication while preserving the topology of the graph. In particular, efficiency is tackled in the initial stages of disambiguation via techniques such as parallel blocking and sliding windows. FDup further enhances the efficiency of the process by extending the framework with the T-match function, which pioneers an approach to efficiency optimization focusing on the final stage of similarity match. The framework can disambiguate node collections regardless of their structure and can be easily used to disambiguate authors. The resulting software modules have been published as a stand-alone software package [79], which is today in use in the production system of the OpenAIRE infrastructure to disambiguate the entities of the OpenAIRE Graph[1] [80].
The chapter is organized as follows: Section 4.1 formally presents the functional architecture of FDup; Section 4.2 focuses on the model requirements, the model adopted, and the technical implementation of the framework providing an example of its usage in the OpenAIRE infrastructure; Section 4.3 provides experimental results highlighting how FDup overcomes traditional approaches in terms of time consumption.

[1] OpenAIRE Graph – `http://graph.openaire.eu`

4.1 FDup architecture

FDup realizes the disambiguation workflow shown in Figure 4.1, which mainly includes the stages already described in Section 1.2. The workflow is intended to disambiguate very large collections of nodes (i.e., research publications, research data, research software, authors), processing them in four sequential stages:

Collection import: to set the node collection ready to process by defining the attributes to be used by the disambiguation (i.e., the *characterization* stage);

Candidate identification: to cluster the nodes to be matched into blocks of potentially equivalent (i.e., the *blocking* stage);

Duplicates identification: to efficiently identify pairs of equivalent nodes via the T-match function, which draws "similarity relationships" (i.e., the *similarity match* stage);

Duplicates grouping: to identify groups of equivalent nodes via transitive closure, which creates "groups of duplicates" (i.e., the *disambiguation* stage).

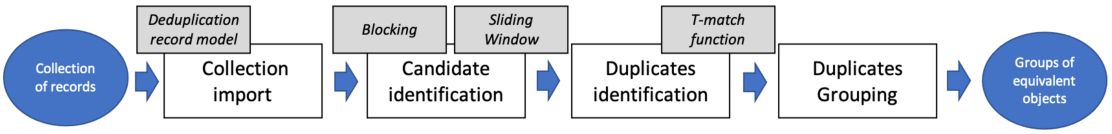

Figure 4.1: *FDup disambiguation workflow.*

4.1.1 Collection import

FDup operates over SKGs nodes presented as a set of "flat" records, whose structure consists of a set of labels (custom name) and values (extracted by the original record). Therefore, the first step of the workflow consists of mapping the target collection of records, which may not be flat, onto an FDup flat collection of records with labels $[l_1, \ldots, l_k]$, which will be used as the template for the configuration of the disambiguation stages.

To better highlight the effects of the framework, experiments have been conducted over a collection of bibliographic records for research publications as provided by the OpenAIRE Graph, featuring the following flat structure:

- *PIDs*: a sequence of values denoting persistent identifiers of the records, i.e., unique identifiers; each record may have more than one PID, released by different agencies at the moment of depositing the article, such as DOIs in Crossref, ArXiv identifiers, PubMed identifiers, etc.;
- *title*: the title of the article;
- *abstract*: the abstract of the article;
- *authors*: the list of authors and contributors of the article, provided as a list of strings typically following different formats, e.g., *"J. Smith", "Smith, J.", "Smith, John H.", "J.H. Smith"*;

- *date*: the date of publication of the article, harmonized to a common format dd/mm/yyyy;
- *venue*: the venue of publication, such as the conference or a journal, typically comes as a string of free text.

4.1.2 Candidate identification

The ideal similarity-matching process, where every pair of records is confronted to yield an equivalence score, features quadratic complexity and known performance issues. As mentioned in Chapter 1, two common solutions are the techniques of blocking and sliding windows. Both methods apply heuristics to identify, before record matching, a selection of record pairs in the collection that are candidates for equivalence.

Blocking functions (referred to as *clustering functions* in the following) are of the form $clusteringKey([l_1, \ldots, l_k])$ and are applied to all records in the initial collection to produce one or more "keys" per record. Functions should be smart enough to ensure that potentially equivalent records likely return the same key. Records are then grouped by key into "blocks", and pair-wise comparisons are executed within such blocks. For the scientific publication records above, a reasonable clustering function generates n-grams (fragments of n characters of a string) from the title words or one that generates the PIDs (especially the DOI) when these are available.

Sliding window methods introduce a further optimization of the number of comparisons. Records in one block are sorted by a key (typically obtained from a value of the record's field) ($orderField$) to obtain an array that is visited from the first element to the last. At every iteration, the pivot record is matched with the subsequent ones in a limited K interval (window). The key used for the sorting should be generated so that similar records are likely closer in the ordering, possibly within the window range K.

FDup offers the possibility to easily configure every single stage previously described via a single configuration profile, including a pre-defined and extendable set of $clusteringKey$ functions, the $orderField$, and the length K of the sliding window.

4.1.3 Duplicate identification: T-match function

Duplicate identification consists of a similarity function that matches the equivalence conditions between two records in a block. As such, it is defined by matching the values of the records fields so that domain-specific conditions of equivalence are met.

For research publication records, OpenAIRE defines two records as equivalent when they describe the same scientific work, hence one object to measure impact. For example, different depositions of the same article in distinct repositories (e.g., ArXiv, Zenodo) are to be considered equivalent, i.e., cannot be counted as two independent scientific results. For the same reason, two different versions of a document have to be considered different, as they denote different efforts. For example, version 1 and version 2 of a project deliverable.

In real case scenarios, like the one of OpenAIRE, where metadata is harvested from highly heterogeneous data sources, many of the field values in the records are not and cannot be sufficiently harmonized to the degree of uniformity required to make them reliable in the process of equivalence check. This is the case for $abstract$, for which establishing a distance and weight for it is not trivial, and $venue$ whose values can

hardly be harmonized into comparable values. Moreover, the $date$ field cannot be used as a discriminator, as different versions of the article, published at different times, may indeed be regarded as equivalent.

Record equivalence is assessed by relying on three fields, PID, $title$, and $authors$, when matching the following considerations:

- **Equivalence by identity**: when two records have one PID in common, they are equivalent;
- **Equivalence by value**: when $PIDs$ are not matching, the records may still be equivalent (e.g., deposited in different repositories), hence a field value equivalence matching is required.

While equivalence by identity is rather straightforward, equivalence by value requires context-driven rules, which in the case of publication match in OpenAIRE are: (i) a rather high $title$ equivalence confidence of 99%, as indeed typos may occur; (ii) ensuring that the $0,01\%$ of difference in the titles is not due to a number or a Roman number denoting different versions of the publication; and, (iii) making sure the records have corresponding authors, by checking their names.

The majority of disambiguation frameworks in the literature encode record similarity match conditions via a similarity function of the form:

$$f([v_1, \ldots, v_k], [v'_1, \ldots, v'_k]) = \sum_{i:0\ldots k} f_i(v_i, v'_i) \times w_i$$

where the v_i's are the values of field l_i, $f_i(v_i, v'_i)$ are *comparators*, functions measuring the "distance" of v_i and v'_i for the field l_i, and w_i's are the weights assigned to the comparators f_i's, such that $\sum_{i:0\ldots k} w_i = 1$.

As a result, f returns a value in a given range, e.g., $[0 \ldots 1]$, scoring the "distance" between two records. The records are regarded as equivalent if the distance measure is greater than a given threshold.

For the example above, the similarity function $PublicationWeightedMatch$, created using the GDup framework in OpenAIRE, encodes both equivalence by identity and by value as follows:

$$\begin{aligned} PublicationWeightedMatch(r, r') = {} & jsonListMatch(r.PIDs, r'.PIDs) \times 0.5 + \\ & TitleVersionMatch(r.title, r'.title) \times 0.1 + \\ & AuthorsMatch(r.authors, r'.authors) \times 0.2 + \\ & LevenshteinTitle(r.title, r'.title) \times 0.2 \end{aligned}$$

where $jsonListMatch$, applied to the field PID, returns 1 if there is at least one PID in common in the two records; $TitleVersionMatch$, applied to the titles, returns 1 if the two titles contain identical numbers or Roman numbers; $LevenshteinTitle$ returns 1 if the two (normalized) titles have a Levenshtein distance greater than 90%, and $AuthorsMatch$ performs a "smart" matching of two lists of author name strings and returns 1 if they are 90% similar. The threshold has been computed by manually validating a set of equivalent records, hence measuring the minimal distance value between the authors' lists of two equivalent records. All comparators return 0 if their condition

is not met. The minimal threshold for two records to be equivalent is 0.5, the threshold that can be reached by $jsonListMatch$ alone or by combining the positive results of the three functions $TitleVersionMatch$, $AuthorsMatch$, and $LevenshteinTitle$.

All f_i's in $PublicationWeightedMatch$ are computed at the same time and averagely require a constant execution time, despite the successful or unsuccessful match that those may feature. Motivated by such observation, FDup introduces a similarity match function T-match that returns an equivalence match exploiting a decision tree, nesting the comparator functions. Each tree node verifies a condition, which can be the result of combining one or more comparators, and introduces a positive (*MATCH*) or negative (*NO_MATCH*) *exit strategy*. If the exit strategy is not fired, T-match heads to the next node. An early exit skips the full traversal of the tree and can turn the result into a *MATCH*, i.e., a *simRel* relationship between the two records is drawn, or into a *NO_MATCH*, i.e., no similarity relationship is drawn.

A T-match decision is formed by a tree of *named nodes* with outgoing *edges*. The core elements of a T-match node are the *aggregation* function, the list of *comparators*, and a *threshold* value. The aggregation function collects the output of the comparators and delivers an "aggregated" result based on one of the following functions: maximum, minimum, average, and weighted mean. Each comparator in a node accepts two values of the input records for a given field and returns a value in the range $0 \ldots 1$. Notably, different comparators in an aggregation can refer to different fields, giving a high degree of customization to end-users (in the following, one node will encode a weighted mean function as the one described above). The execution of a T-match node must end with a decision, which may be:

- *positive* if the result of the aggregation function is greater than or equal to the threshold value;
- *negative* if the result of the aggregation function is lower than the threshold;
- *undefined*, if one of the comparators cannot be computed (e.g., absence of values); a node also bears a flag *ignoreUndefined* that ignores the *undefined* edge even if one of the values is absent.

For each decision, the node provides the *name* of the next node to be executed. By default, T-match provides two nodes *MATCH* and *NO_MATCH* to be used to force a successful or unsuccessful early exit from the tree.

The example in Figure 4.2 shows the function $PublicationTreeMatch$, which uses the same comparators but exploits a T-match decision tree, therefore implementing early-exit conditions. The individual matches are lined up by introducing *MATCH* conditions early in the process, i.e., equivalence by identity via $PIDMatch$, and then ordering *NO_MATCH* conditions by ascendant execution time, i.e., equivalence by value via $versionMatch$, $titleMatch$, and $authorsMatch$.

Independently of the domain, smart identification of exit strategies becomes a means for developers to reduce the overall disambiguation time. Moreover, T-match allows for the definition of multiple paths, hence the simultaneous application of alternative similarity match strategies in one single function. The experiments described in later sections will show that when the number of records is very large, T-match significantly improves the overall performance of the whole disambiguation process.

Figure 4.2: *T-match's decision tree for* $PublicationTreeMatch$.

4.1.4 Duplicates Grouping

The outcome of the duplicate identification stage is the graph resulting from combining the input collection of records with the set of *simRel* relationships between them. The duplicate grouping stage first finds the sets of equivalent records by calculating the connected components in the graph via the transitive closure of the *simRel* relationships; for example, $A\ simRel\ B$ and $B\ simRel\ C$ identify the group A, B, C. Secondly, it generates a new graph, where the groups of equivalent records are all linked with a *mergeRel* relationship to a *representative record*, created by the process to identify the groups; for example, the group of nodes A, B, C will deliver the graph of four nodes A, B, C, R with the relationships $A\ mergeRel\ R$, $B\ mergeRel\ R$, $C\ mergeRel\ R$.

4.2 Software implementation

FDup's software[2] [79] has three modules, `Pace_Core`, `Dedup_Workflow`, and `Configuration file` depicted in Figure 4.3. The framework is implemented in Java and Scala, and grounds on the Apache Spark Framework, an open-source distributed general-purpose cluster-computing framework. FDup exploits Apache Spark to define record collection parallel processing strategies that distribute the computation workload and reduce the execution time of the entire workflow. Scala is instead required to exploit the out-of-the-box library for the calculation of a "closed mesh" in GraphX[3].

[2]GitHub project – `https://github.com/miconis/fdup`
[3]Apache Spark GraphX – `https://spark.apache.org/graphx/`

The three modules implement the following aspects of FDup's architecture:

- `Pace_Core` includes the functions implementing the candidate identification stage (blocking and sliding window) and the T-match function, as well as the (extensible) libraries of comparators and clustering functions.
- `Dedup_Workflow` is the code required to build a disambiguation workflow in the Apache Spark Framework by assembling the functions in `Pace_core` according to the comparators, clustering functions, and parameters specified in the `Configuration file`.
- `Configuration file` sets the parameters to configure the disambiguation workflow stages, including record data model, blocking and clustering conditions, and T-match function strategy.

In the following sections, the three modules are described in detail.

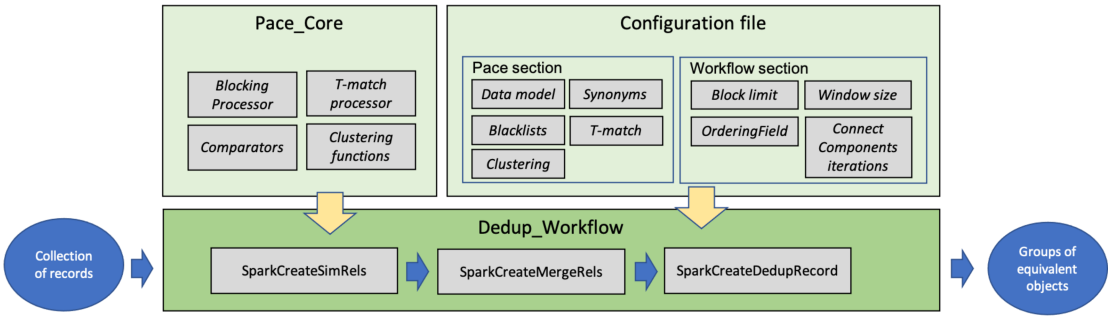

Figure 4.3: *FDup software modules.*

4.2.1 The configuration file

The FDup's configuration file is expressed in JSON format and consists of two different sections: `pace`, which defines the T-match function parameters; and `workflow`, which defines the disambiguation workflow parameters.

Pace section

The section specifies the configuration for the pair-wise comparisons, including the *data model*, the record pairs' *blacklist*, the *synonyms*, and the *decision tree* of the T-match function.

FDup operates on a collection of flat records with the same structure. As the original collection of JSON records may not be flat, FDup introduces the mechanism of the data model. The *data model* subsection of the configuration drives the transformation of the original JSON records onto a flat record with labels and values as depicted in Figure 4.4. The mapping is defined employing JSON paths, whose result is implicitly assigned to a given field of the FDup's record data model. Additional parameters such as length and size can be used to limit the value to take from the original JSON entity.

The *clustering* subsection includes the list of clustering functions (more than one can be used) to be used for the key extraction. Each clustering function specifies the list of record fields to which the function should be applied. The user can also specify

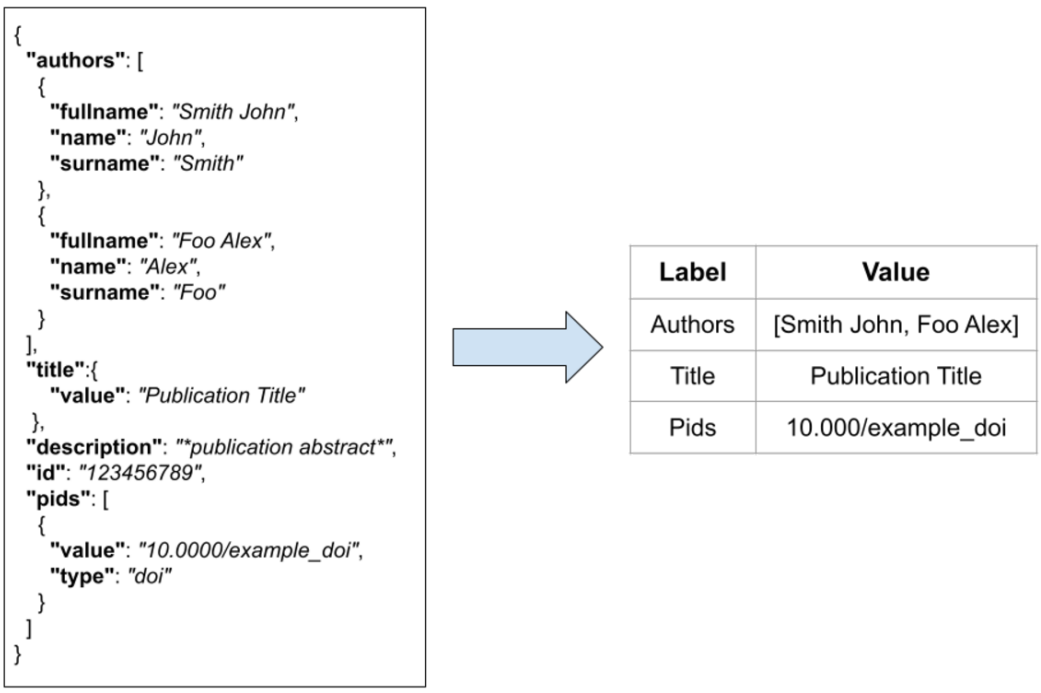

Figure 4.4: *The transformation of an original JSON record into a flat record.*

parameters for the clustering function such as key length, maximum number of keys to extract, and other configurations. The structure of a clustering function is depicted in Section 4.2.1.

The *blacklist* is a list of field values to be excluded from the computation. The mechanism is useful to exclude false positives from subsequent rounds of disambiguation. It is expressed as a map in which the key is the field name while the value is the list of strings to exclude. When the process is running, entities with one of those values for the specified field will be ruled out of the comparisons.

The *synonyms* are lists of equivalent terms. They are typically used to encode semantic equivalence across different vocabularies (e.g., replacing terms with a common code). For example, synonyms are exploited to address translations of terms across different languages and therefore capture their equivalence.

The *decisionTree* section sets the configuration of T-match's pair-wise comparison algorithm. T-match's tree can be shaped by creating different nodes, each composed of one or more comparators. Each comparator acts on a pair of records and over a given pair of fields, to return a number that reflects the similarity of two fields. The results of the comparators in one node are then aggregated to return an overall similarity score. The user can define the aggregation function to be used (e.g., weighted mean, average, maximum, minimum). The structure of a comparator and a tree node are depicted in Section 4.2.1 and Section 4.2.1.

Workflow section

The section specifies parameters for the whole disambiguation workflow. Such parameters are $groupMaxSize$ (i.e., the limit for the block size), $slidingWindowSize$ (i.e., the size of the sliding window), $orderField$ (i.e., the field to be used when sorting the sliding window in a block), and $maxIterations$ (i.e., the maximum number of iterations when the connected components are computed via the close mesh operation).

Field	Description
name	the name of the clustering function
fields	the list of fields to which the clustering function should be applied
params	the list of parameters to configure the clustering function. Every parameter has a name and can assume a number as a value. Those parameters are accessible from the clustering function.

Table 4.1: *Definition of a clustering function.*

Field	Description
field	the field to compare (it must be defined in the model)
comparator	the comparator to use for the comparison
weight	the weight of the score for the comparator. The value is used when the comparator's scores are aggregated to give weight to the comparator.
countIfUndefined	boolean value that specifies if the score should be considered in the aggregation also when the result of the comparison is undefined (i.e., missing field)
params	the list of parameters for the comparator. Every parameter has a name and can assume a string as a value. Those parameters are accessible from the comparator.

Table 4.2: *Definition of a comparator.*

4.2.2 Core modules

The `BlockProcessor` is the Java class that provides functionalities to process a block (e.g., entity sorting), it implements the sliding window mechanism and it invokes the `T-match Processor` to perform the record pair-wise comparisons. The core of this class is the function that implements queue management when blocks of records are sorted. Records in a block are queued in alphabetical order (of the $orderField$) and subsequently, the queue is processed and pair-wise comparisons are computed.

The `T-match Processor` is the Java class that navigates the decision tree by calculating the result for every node according to the configuration provided in the `Configuration file`. Such class is invoked by the `BlockProcessor` and invokes each comparator involved in the node to collect and aggregate their scores (as specified by the user). At the end of the aggregation, a threshold is applied to the final score, and the next tree node to process is chosen.

For the example in Figure 4.2, when the `T-match Processor` is computing the score of the second node of the tree, it firstly calculates the scores for the two

Field	Description
fields	the list of comparators contributing to the computation of the final score
threshold	the threshold for the final score of the node. If the final score is greater than the threshold, the execution will follow the positive edge; otherwise, it will follow the negative edge. The execution will follow the undefined edge when the ignoreUndefined is not enabled and one of the comparators returned an undefined result.
aggregation	the aggregation function to use for the computation of the final score (e.g., maximum, minimum, average, weighted mean, etc.)
positive	the name of the next node to compute if the final score of this node is greater than or equal to the threshold.
negative	the name of the next node to compute if the final score of this node is lower than the threshold.
undefined	the name of the next node to compute if the result of the node is undefined.
ignoreUndefined	boolean value that specifies if the undefined tree edge should be ignored or not.

Table 4.3: *Definition of the tree node.*

comparators (i.e., $titleVersionMatch$ and $sizeMatch$) and then aggregates them with an AND operation (i.e., both conditions must be verified).

4.2.3 Libraries

As stated before, FDup offers a set of predefined and extendable libraries of comparators and clustering functions. They are implemented as Java classes inside the framework package.

The user can customize the framework by implementing new components depending on its needs. To implement a new comparator, it is sufficient to implement a Java class that extends the proper interface defined in the package.

Comparators

In the context of the pair-wise comparison, the comparator specifies the logic that computes the score measuring the similarity degree of two fields of the pair, one for each record. Such a score is then aggregated with others in the same tree node to compute the overall similarity score of the node.

Comparators are Java classes that implement the comparison between two fields `a` and `b` of a record, whose Java interface is:

```
public interface Comparator {
   public double compare(Field a, Field b, Config conf);
}
```

The only method to be implemented within the interface is `compare`. The method has three parameters: the two fields to be compared and the `Configuration file`, which drives the disambiguation and may include comparators' parameters. `compare` yields a `double` value in the range $0, \ldots, 1$ (0 different, 1 identical) indicating the similarity between the two field values. In particular, it returns 0 when the comparator

cannot produce a result (e.g., because one of the two labels is empty or missing) and a value in the range $0, \ldots, 1$ otherwise.
Section 4.2.3 describes the set of pre-defined comparators available in FDup today, which can be extended to address new application needs. In general, such a comparator relies on the assumption that values are available via record fields. Hence, specific comparators can for example accept as input fields whose values are the result of smart pre-processing of the record collection, e.g., machine learning embeddings, full-text extraction, and topic modeling.

Clustering functions

Clustering functions are Java classes that implement the key extraction from a flat record. In the context of blocking, the clustering function specifies the logic that extracts the keys from the value of a certain label, e.g., by computing ngrams, extracting the domain from an URL, etc. Such keys are subsequently used to group similar records into the same cluster and therefore limit the number of pair-wise comparisons.

To create a new clustering function, the following Java interface must be implemented:

```
public interface ClusteringFunction {
    public Collection<String> apply(Config c, List<Field> f);
    public Map<String, Integer> getParams();
}
```

The interface provides two methods:

- `getParams`: to access the list of parameters of the clustering function indicated in the configuration;
- `apply`: to produce the list of string keys extracted from the labels (e.g., ngrams). The `config` parameter gives the user the possibility to implement comparators with access to the `Configuration file`.

FDup offers a list of pre-defined clustering functions, listed in Section 4.2.3. Such a list can be extended to introduce new approaches and strategies.

4.2.4 Disambiguation workflow

This module implements the workflow depicted in Figure 4.1 by building on the Apache Spark Framework, which allows defining and configuring applications with high performance for batch and streaming data. The framework can be used with different programming languages (Java and Scala in FDup) and offers over 80 high-level operators to realize parallel applications.

The disambiguation workflow is implemented as an Oozie workflow that incapsulates jobs executing the three steps depicted in Figure 4.3, to compute: (i) the similarity relations ($SparkCreateSimRels$), (ii) the merge relations ($SparkCreateMergeRels$), and (iii) the groups of duplicates ($SparkCreateDedupEntity$). More specifically:

- $SparkCreateSimRels$: uses classes in the `Pace_Core` module to divide entities into blocks (clusters) and subsequently computes ***simRels*** (i.e., similarity relationships) according to the `Configuration file` settings for T-match;

name	description
AuthorsMatch	performs a "smart" comparison between two lists of authors; author names are matched by considering a custom similarity threshold and the result is the percentage of common elements between the two lists
CityMatch	extracts city names from the field and returns the percentage of names in common; city names are matched by considering translations in different languages of the most important cities in the world
ContainsMatch	searches for a given string in the input fields; logic operators (i.e., AND, OR, XOR) can be used
ExactMatch	returns 1 if the two fields are exactly the same. There are also other implementations of exact matches specific for a particular scope (i.e., *DoiExactMatch* specific for DOIs, *DomainExactMatch* specific for URLs, and *ExactMatchIgnoreCase* to perform a case insensitive comparison)
JaroWinkler	computes the JaroWinkler similarity function between two fields; the comparator can be specialized to address special cases, e.g., *JaroWinklerNormalizedName* to compare two fields after removing city names and keywords, *JaroWinklerTitle* to compare fields containing titles, *SortedJaroWinkler* to a sorted version of the algorithm), *Level2JaroWinkler*, *Level2JaroWinklerTitle* and *SortedLevel2JaroWinkler*
JsonListMatch	returns the common element percentage between two JSON lists extracted from the input fields
KeywordMatch	extracts keywords from string fields and returns the common element percentage; the keywords are compared by considering translations in different languages provided via an input CSV; the list of keywords is customizable and located in the classpath
Levenshtein	computes the Levenshtein distance measures between the two strings in the fields; possible specializations are *LevenshteinTitle* to compare specific title fields, *LevenshteinTitleIgnoreVersion* to compare titles and removing versions, and *SubStringLevenshtein* to compute the distance on substrings of the field, and *Level2Levenshtein*
MustBeDifferent	returns 1 if the two fields are different
NumbersMatch	extracts numbers from the input fields and returns 1 if they are equal
RomansMatch	extracts Roman numbers from the input fields and returns 1 if they are equal
SizeMatch	specific for lists, returns 1 if the size of two lists is equal
StringListMatch	returns the percentage of common elements in two lists of strings
TitleVersionMatch	specific for title fields, performs a normalization and returns 1 if numbers in the title are equal
UrlMatcher	specific for URLs, performs a normalization of the URLs and returns 1 if they are equal
YearMatch	extracts the year from the input fields and returns 1 if they are equal

Table 4.4: *List of FDup comparators.*

- $SparkCreateMergeRels$: uses GraphX library to process the ***simRels*** and close meshes they form; for each connected component, a master record ID is chosen

name	description
Acronyms	creates a number of acronyms out of the words in the input field
KeywordsClustering	creates keys by extracting keywords, out of a customizable list provided in the classpath, from the field's value
LowercaseClustering	creates keys by lowercasing the field's value
Ngrams	creates ngrams from the field's value; the number of ngrams and the length is indicated via parameters
PersonClustering	specific for Person names, uses name and surname to create keys
PersonHash	creates an hash of the Person name
RandomClusteringFunction	creates random keys from the field's value
SortedNgramPairs	creates ngrams from the field's value and then combines them in pairs
SpaceTrimmingFieldValue	creates keys by trimming spaces in the field's value
SuffixPrefix	creates keys by concatenating suffixes and prefixes from words in the field's value
UrlClustering	creates keys for a URL field by extracting the domain
WordsStatsSuffixPrefixChain	creates keys containing concatenated statistics of the field, i.e., number of words, number of letters, and a chain of suffixes and prefixes of the words

Table 4.5: *List of FDup clustering functions.*

and *mergeRels* relationships are drawn between the master record and the connected records;

- $SparkCreateDedupEntity$: uses *mergeRels* to group connected records and create representative objects (i.e., groups of duplicates).

4.3 Efficiency evaluation

This section describes the methods employed to assess the efficiency of FDup when T-match can implement exit strategies, compared to a traditional approach where this technique is not exploited. The evaluation has been carried out using the metadata record collections used to populate the OpenAIRE Graph, where FDup is used as the core disambiguation component.

The evaluation aims to assess the efficiency, not the effectiveness of the disambiguation results, which instead depends on the clustering functions, sliding windows, comparators, similarity thresholds, quality of metadata, and ultimately AI techniques (see Chapter 5). Therefore, the analysis is aimed to verify if FDup can obtain the same disambiguation result by taking less execution time.

4.3.1 Experiment settings and methodology

The experiment aims to show the time gain yielded by the proper configuration of T-match in a disambiguation workflow for the publication similarity match example presented in Figure 4.2. To this aim, the experiment sets two disambiguation workflows with identical blocking and sliding window settings but distinct T-match configurations. Both configurations address the similarity criteria but in opposite ways:

- $PublicationTreeMatch$ **configuration**: a configuration that implements the decision tree illustrated in Figure 4.2, taking advantage of early exits;

- $PublicationWeightedMatch$ **configuration**: a configuration that implements the similarity match as the GDup (average mean) function described in Section 4.1 by combining all comparators in one node, whose final result is a *MATCH* or *NO_MATCH* decision.

Both configurations are based on the same settings for candidate identification and duplicate identification. In particular:

- the clustering functions used to extract keys from publication records are the $LowercaseClustering$ on the DOI (e.g., a record produces a key equal to the lowercase DOI, the result is a set of clusters composed by publications with the same DOI) and the $SuffixPrefix$ on the publication title (e.g., a record entitled *"Framework for general-purpose disambiguation"* produces the key *"orkgen"*, the result is a set of clusters composed by publications with potentially equivalent titles); both functions are described in Section 4.2.3
- the $groupMaxSize$ is set to 200 (empirically) to avoid the creation of big clusters requiring long execution time;
- the $slidingWindowSize$ to limit the number of comparisons inside a block is set to 100 (empirically).

Both the $PublicationTreeMatch$ and the $PublicationWeightedMatch$ configurations were performed over the research publication record collection[4] published in [81]. The collection contains a set of 10M publications[5] represented in JSON records extracted from the OpenAIRE Graph Dump [80]. In particular, publications have been selected from the Dump to form a dataset with a real-case duplication ratio of around 30% and a size that is appropriate to prove the substantial performance improvement yielded by the early exit approach.

I performed two different tests, comparing the performance of the configurations described above over the $10M$ and the $230M$ collections respectively. The tests are intended to measure the added value of T-match in terms of performance gain, i.e., $PublicationTreeMatch$ vs $PublicationWeightedMatch$ execution times.

The tests were performed with a driver memory set to 4 Gb, the number of executors to 32, the executor cores to 4, and the executor memory to 12 Gb. The Spark dynamic allocation has been disabled to ensure a fixed amount of executors in the Spark environment, to avoid aleatory behavior. Moreover, since Spark's parallelization shows different execution times, depending on both the distribution of the records in the executors and the cutting operations on the blocking phase, each test has been executed 10 times and the average time has been calculated.

The execution time has been measured in terms of processing time required by the $SparkCreateSimRels$, where the pair-wise comparisons are performed, and by the $SparkCreateMergeRels$, where groups of duplicates are generated. It was observed that the $SparkCreateSimRels$ is dominant taking 70% of the overall processing time. As a consequence, for the sake of experiment evaluation, it has been: (i) reported and confronted the time consumed by the $SparkCreateSimRels$ under different tests to showcase the performance gain of T-match, and (ii) reported the results

[4]OpenAIRE publications sample collection – `https://doi.org/10.5281/zenodo.5347803`
[5]OpenAIRE Graph Dump – `https://doi.org/10.5281/zenodo.4707307`

of the $SparkCreateMergeRels$ to ensure that the tests are sound, i.e., yield the same number of groups.

4.3.2 Experimental results

The results of the tests on the $10M$ publication records dataset and the $230M$ full publication datasets are depicted in Figure 4.5 and Figure 4.6, respectively. The graphs show the average time consumption of the $SparkCreateSimRels$ phase for each execution of the test.

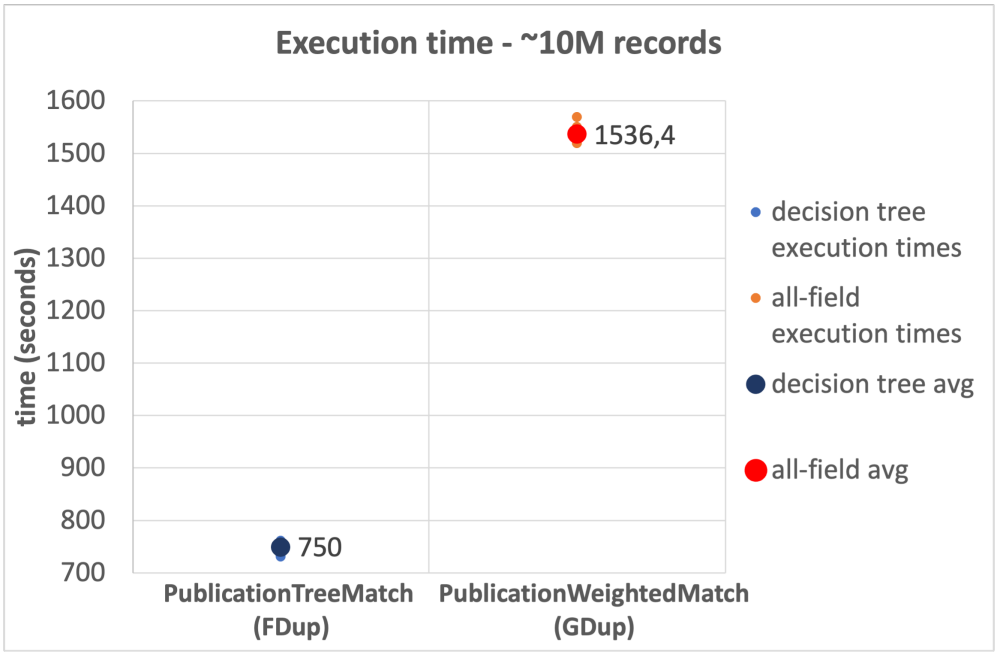

Figure 4.5: *Disambiguation test on 10M records.*

The average time of the $SparkCreateSimRels$ stage in the test performed over 10M records dataset with the $PublicationTreeMatch$ configuration is 750 seconds, while the $PublicationWeightedMatch$ configuration consumes $1,536.4$ seconds. The $SparkCreateSimRels$ test on the $230M$ records dataset features an average time of $9,637.6$ seconds for $PublicationTreeMatch$ and an average time of $15,224.5$ seconds for $PublicationWeightedMatch$.
The results reported in Section 4.3.2 show that the two scenarios produced a comparable but not identical amount of *simRels*, *mergeRels*, and *connectedComponent*. Differences are due to two main aspects: the size of the datasets, which required us to impose a limit to the block size to avoid uncontrolled execution time, and the Apache Spark behavior, which introduces a non-deterministic degree in the way blocks are formed (i.e., keys are randomly distributed in parallel across blocks). These factors may introduce slight differences between the blocks resulting from different runs over the same input set. However, for both input datasets, the differences of *simRels* and *mergeRels* across different runs are limited to a range of $1,000 \ldots 2,000$ and are therefore not influential to the validation of the experiment. The differences between the two configurations are measured using the relative change, e.g., the variation between the number of relations

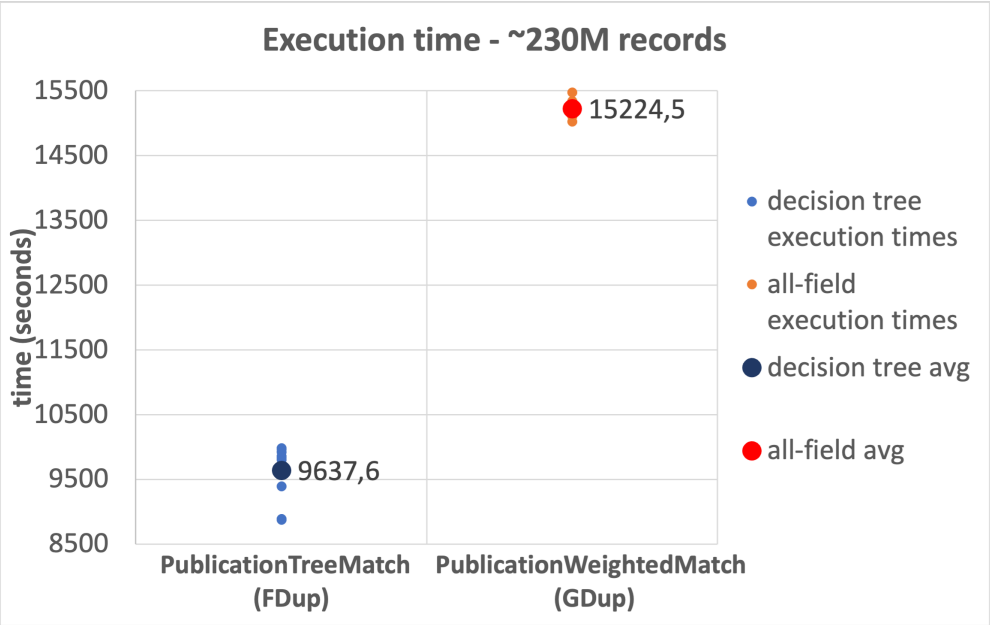

Figure 4.6: *Disambiguation test on 230M records.*

size	relation type	TreeMatch	WeightedMatch	relative change (%)
10M	$simRels$	13,865,552	13,866,320	0.000055
	$mergeRels$	5,247,252	5,247,585	0.000063
	$connectedComponents$	1,890,012	1,890,148	0.000071
	$pairwiseComparisons$	255,772,628	255,772,628	0.0
230M	$simRels$	172,510,072	172,511,772	0.0000098
	$mergeRels$	69,974,139	69,974,155	0.00000022
	$connectedComponents$	25,250,036	25,250,143	0.0000042
	$pairwiseComparisons$	3,650,733,202	3,650,733,202	0.0

Table 4.6: *Average number of relations drawn by the disambiguation workflow on 10M and 230M publication records.*

in terms of percentage.

Based on such results, it can be stated that the $PublicationTreeMatch$ configuration overtakes the $PublicationWeightedMatch$ configuration in terms of time consumption, by improving performance up to 50% in the first test and up to 37% in the second test. The tests show a significant performance improvement, which suggests that the performance gain does not depend on the size of the dataset but improves with the number of early exits. It is also important to mention that the time measured in our tests includes the blocking stage. This may suggest that the key generation process consumes a notable amount of time, especially when the input dataset is larger.

CHAPTER 5

Enhancing effectiveness via Graph Neural Networks

This chapter presents novel solutions to the enhancement of the effectiveness of the disambiguation process. Firstly, it describes the research methodology adopted to create the benchmarks used to train the models and conduct the experiments. Secondly, after providing some information about theoretical aspects and the employed frameworks and tools, it presents two Graph Neural Networks architectures for the evaluation of similarity relationships and groups of duplicates. The former employs the use of four different GraphSAGE models to compute node embeddings from different homogeneous graphs, which are subsequently aggregated via a "metapath attention" mechanism. The latter exploits and combines the concept of Graph Attention and Long Short Term Memory (LSTM). Both models have been trained in a supervised way and their overall accuracy resulted in about 90%.

The chapter is organized as follows: Section 5.1 presents the set of frameworks and tools utilized to conduct the research; Section 5.2 describes the process leading to the creation of the benchmark used for the training and testing of the architectures and the validation of the findings, providing information on the creation of author nodes; Section 5.3 presents the novel GNN architecture for the evaluation of similarity relationships; Section 5.4 presents the novel GNN architecture for the evaluation of groups of duplicates.

5.1 Frameworks and Tools

The work in this thesis has been developed by relying on multiple frameworks and tools. The contributions combine both Java and Python code, relying on tools and frameworks provided by user-developed libraries. Given the "Big Data" nature of the

graph used to conduct the experiments (i.e., the OpenAIRE Graph described in Chapter 3), the processing of the data has been conducted using the Hadoop Distributed File System (HDFS), one of the major components of Apache Hadoop which can handle large data sets optimizing the memory consumption. HDFS is used to scale a single Apache Hadoop cluster to thousands of nodes. All the data used for the research in this thesis is stored in different records organized in JSON lines. The data has been processed by using the Apache Spark[1] framework, an open-source distributed general-purpose cluster-computing framework that implements and optimizes the MapReduce paradigm. This framework allows us to shorten the computation time by providing a set of tools for feature extraction and text processing. To what concerns the GNN models presented as contributes of this thesis, the training, testing, and validation have been made by utilizing the Deep Graph Library[2] (DGL), a framework agnostic library that allows the implementation of fast and memory-efficient message passing primitives for training Graph Neural Network. The library can scale to giant graphs via multi-GPU acceleration and distributed training infrastructure. The whole library provides a big set of implemented models and functionality that immediately mirrors the best state-of-the-art models allowing the users to test every model with no effort and by allowing them to customize the network layers. DGL also empowers a variety of domain-specific projects including DGL-KE for learning large-scale knowledge graph embeddings, DGL-LifeSci for bioinformatics and cheminformatics, and many others. All the GNN models presented in the thesis have been trained using an *NVIDIA GeForce RTX 3060 Laptop GPU*. The evaluation of the models and the computations of the statistics have been made by utilizing the Scikit Learn Library[3], an open-source set of simple and efficient tools for predictive data analysis, accessible to everybody and reusable in various contexts.

5.2 Benchmark preparation

To perform the training and the testing of the GNN architectures presented in this chapter, and subsequently evaluate the quality of the findings, a graph benchmark has been created to highlight the main benefits of the architectures. The benchmark aims to reproduce the real-case scenario of the OpenAIRE Graph (see Chapter 3) and includes a small portion of it. The idea of using only a subset of the entire OpenAIRE Graph comes after this intuition: to make the operations feasible, distributed training based on node sampling is necessary to fit with the size of the input data. This will require a high computation time which can be shortened by having the training process running on multiple GPUs. Since GNN makes only neighboring nodes exchange information, it is meaningful to apply the approach to a small portion of the graph to evaluate its behavior when the training set is larger. When the set of publications to be used has been obtained, their semantic relationships are collected. Subsequently, raw author names are extracted to form author nodes, and a new heterogeneous sub-graph with publications and authors have been obtained introducing noised nodes to imitate the real-case scenario. Finally, a disambiguation using the FDup framework (see Chapter 4) has been made on the author set to draw similarity relationships and create groups of duplicates.

[1] Apache Spark – `https://spark.apache.org/`
[2] Deep Graph Library – `https://www.dgl.ai/`
[3] Scikit-Learn Library – `https://scikit-learn.org/`

Figure 5.1 depicts the various stages of the benchmark preparation pipeline.

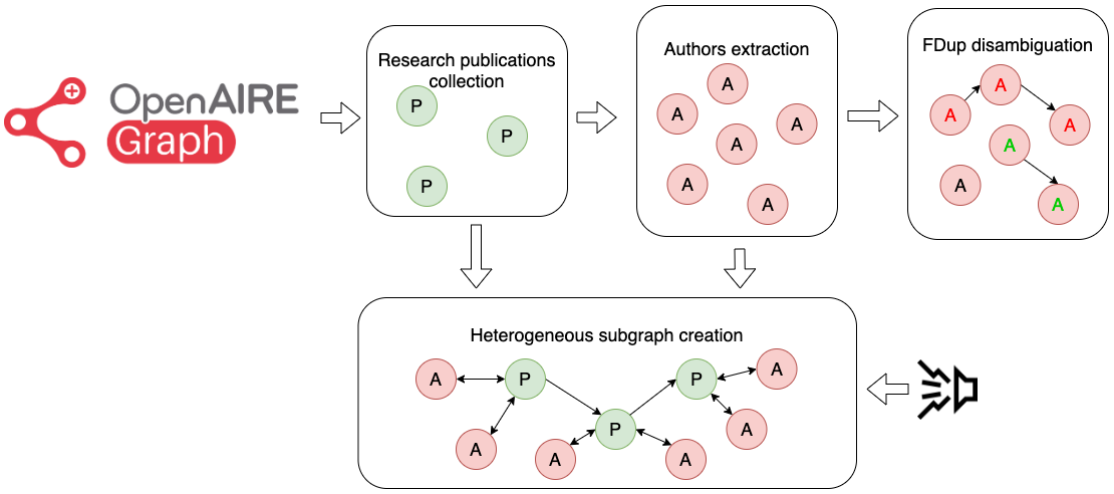

Figure 5.1: *Benchmark preparation pipeline: (i) filtering of publications and their semantic relationships from the OpenAIRE Graph, (ii) extraction of authors with ORCID identifiers from publications, (iii) creation of a heterogeneous sub-graph with publications and authors adding noise to imitate real-case scenarios, and (iv) AND using the FDup framework.*

5.2.1 Research publications collection

The research publications used to create this benchmark have been filtered from the OpenAIRE Graph following two criteria. In particular, to be included in the benchmark, a research publication must:

- be collected from PubMed[4], to include records of the same research topic in order to have a higher chance of duplicate authors;
- include at least one author with an ORCID identifier in their authors' list.

The choice of such publications is necessary because the ORCID identifier is a good ally when the purpose is to create a ground truth aimed at defining the correctness of the AND task.

To build the subgraph, it is important to collect also semantic information. Those are in the form of relationships to be filtered from the OpenAIRE Graph as well. In particular, all the relationships in the graph involving one of the research publications collected by the previous filtering have been included. To this aim, not only relationships between two publications have been collected. Also, relationships between a publication and a different node type have been taken into account. In this way, the additional node type will be considered as a "bridge" between two publications and considered as a direct relationship between them. More precisely, the following relationships have been included:

- cites relationships, i.e., all those relationships between two publications indicating when a publication cites another;

[4]PubMed – `https://pubmed.ncbi.nlm.nih.gov/`

- co-produced relationships, i.e., relationships between two publications indicating when they are produced by the same research project.

5.2.2 Authors extraction: creation of raw author nodes

The purpose of the AND in the OpenAIRE context can be formulated as the disambiguation of raw author names indicated as properties of research publications. Having said that, it becomes a mandate to extract those names from the original research publication to create separate nodes describing the authors. To do this, it is necessary to describe each author with a set of attributes that may be used by the disambiguation algorithm for blocking, similarity matching, and finally disambiguation (in this case of study, the FDup framework described in Chapter 4). Usually, each author inherits its attributes from the research publication from which it has been extracted. Those attributes are:

- the ORCID[5] identifier: a persistent identifier that a researcher owns and controls, and that distinguishes him from every other researcher. This identifier is fundamental to create ground truth and to evaluate the quality of the approaches, as it acts as a unique ID with no duplicates;
- the author name: a string that indicates information about the name of an author, which can be well-formed (i.e., it provides a name and a surname, possibly divided by a comma) or not, e.g., *"Surname, Name"* or *"Surname, N."*;
- the co-authors list: a list of names indicating the authors belonging to the same research publication;
- the research publication abstract: the text describing the topic and the field of study of the research publication itself.

Since the Graph Neural Networks involve mathematical operations, a preliminary process on the nodes and the edges of the graph is necessary to convert their attributes into machine-friendly vectors. Therefore, a preliminary feature extraction operation is needed to make the data compliant with the architectures. Feature extraction is a process of transforming raw data into numerical features preserving the information in the original data set. This process yields better results than applying machine learning techniques directly to the raw data. This operation can be performed manually (i.e., identify and describe the relevant features for a given problem) or automatically (i.e., use a specialized algorithm or deep networks to extract features without human intervention). Vectors resulting from a feature extraction operation (the initial *node embedding*, also called *node feature*) must encode all the information of the original attributes by encapsulating all the characteristics that make two attributes similar. In other words, the idea behind the feature extraction is to plot node attributes in a n-dimensional space where vectors close to each other represent similar attributes based on some of their characteristics. In the use case of this thesis (the Scholarly Knowledge Graphs), nodes and edges describe research products and their semantic relationships in the scientific community, therefore their attributes are in the form of text (i.e., author names, titles, publication abstracts). The processing and the transformation of such type of node attributes lie

[5]Open Researcher and Contributor ID – `https://orcid.org/`

in the concept of text mining and Natural Language Processing (NLP), where machine learning is used to reveal the structure and the meaning of a text [82–84]. With those types of algorithms, attributes of a research product are easily analyzable and their equivalences may also be caught when they are not directly comparable in their original form. For every contribution of this thesis, features have been extracted manually and automatically. For manually extracted features, a 55-dimensional vector has been extracted from author names and acts as an encoding of the letters inspired by the "bag of words" model [85]. For automatically extracted features, have been used two popular algorithms in the literature for the topic modeling: a short encoding made by using the Latent Dirichlet Allocation (LDA) [86], and a long encoding made by using the BERT sentence embedding [87].

Author Name encoding: the "bag of letters" model

The author name is the first attribute that describes an author node in a Scholarly Knowledge Graph, therefore it is the first hint of the disambiguation as its value determines the preliminary blocking stage of the whole disambiguation process. Having said that, it is important to provide an effective encoding of it to give the GNN the possibility to learn from its representation. The purpose of the encoding of this attribute is to have an n-dimensional vector that puts "close" to each other similar author names in a typo-tolerant setting. To this aim, it becomes crucial to encode each letter in order not to lose information when the author's name is translated into a machine-friendly format. The intuition behind the encoding used for author names is inspired by the most popular "bag of words" model, early described in a linguistic context in [88]. Such a model uses a representation of text that is based on an unordered collection (or "bag") of words, and it is widely used in NLP and information retrieval. The main characteristic of the "bag of words" lies in the fact that it disregards word order (and thus any non-trivial notion of grammar) but captures multiplicity. In particular, in the "bag of words" model, every word is a token and the encoding of a document is a vector counting the number of occurrences of each word in it. Similarly, in the "bag of letters" model, letters are tokens and the encoding of a name is a vector counting the number of occurrences of each letter in it. Therefore, after this analysis, every author name is firstly transliterated to the English alphabet (including accents when necessary) and secondly encoded in a 55-dimensional feature vector in which each element indicates the frequency of a specific letter in the name (the size of 55 indicates the number of characters in the alphabet used as a dictionary, which includes punctuation marks and letters of the alphabet with different accents). Such kind of encoding is sufficient to achieve good results since it guarantees a good representation of typos, which may be present in author names but are still coded in similar vectors (in case of typing errors leading to letters swapped in positions, the process results exactly in the same encoding). An example of how an author name is encoded with the "bag of letters" model is shown in Figure 5.2.

Research publication abstract encoding: exploring topic modeling techniques

The research publication abstract is a good ally in disambiguation, as it can add information to an under-described author which provides few attributes (e.g., no co-authors, no identifiers) and is therefore prone to disambiguation errors. For the AND, compar-

Figure 5.2: *Simplified example of the encoding computed with the "bag of letters" model (in this case the dictionary includes only letters of the alphabet for better visualization).*

ing the authors based on their "field of study" is proved to be effective as researchers tend to work on the same research topic in their academic careers. Nonetheless, it is unfeasible to directly compare abstracts because string similarity measures are not effective for long strings. Therefore, the encoding of research publication abstracts is fundamental to (i) make the fields "comparable", and (ii) mitigate both homonymy and synonymy, as it may group or split similar authors based on their research topic. To this aim, 2 different encodings for the research publication abstract have been created. The first is a short encoding, to be used by FDup, and therefore aimed to save further computation time; the second is a long encoding, to be used by the GNN, and therefore aimed to provide as much information as possible.

Latent Dirichlet Allocation (LDA) [86] is a generative statistical model (in the form of a Bayesian Network) for Natural Language Processing that models automatically extracted topics in textual corpora. In this context, words are grouped into research publication abstracts (i.e., the documents), and the presence of each word is attributable to one of the topics of the document. The intuition behind this algorithm is that each document contains only a small number of topics and can be therefore encoded in a relatively small N-dimensional vector where each element indicates the degree of belonging to that specific topic. Because of its nature, this encoding is widely used in machine learning, specifically for topic modeling, whose purpose is to discover topics in a collection of documents, and then automatically classify any individual document within the collection in terms of how "relevant" it is to each of the discovered topics. For example, in the document collections of this use case (i.e., research publication abstracts), the terms *"informatics", "computer", and "processor"* would suggest a topic related to the discipline of informatics, while terms like *"mathematics", "algorithm", and "multiplication"* would suggest a topic related to the discipline of mathematics. If the document collection is sufficiently large, LDA will discover such topics based on the co-occurrences of individual terms, but the task of assigning a meaningful label to each topic is left to another activity and it is not trivial. The LDA approach works on the following assumptions:

- The semantic content of a document is composed by combining one or more terms from one or more topics;
- Ambiguous terms may belong to more than one topic with different probability, e.g., the term "algorithm" may be applied to both "informatic" and "mathematics", but is more likely to refer to mathematics;
- Most documents contain a small number of topics;
- Certain terms may be used much more frequently than others.

The last two assumptions may be controlled with two parameters: the document-topic density and topic-word density. A high value for the former (also known as "alpha") means that documents are composed of more topics, while a higher value for the latter (also known as "beta") indicates that topics are composed of a large number of words in the corpus.

To train the LDA model for topic modeling, a pre-processing of the research publication abstracts has been applied. This process consists of the following sequential operations:

- Tokenization: abstract split in tokens, e.g., the words;
- Stop-words removal: tokens cleaned up by removing the stop-words to reduce the noise, e.g., terms as "the", "or", etc.
- Preliminary vectorization: cleaned tokens transformed into vectors using the previously described "bag of words" model with the Dewey Decimal Classification [89] as a dictionary.

Figure 5.3 depicts a visual representation of the words in the research publication abstracts. As the picture suggests, it is immediately notable how the removal of the stop words is fundamental to getting rid of the noise in the data.

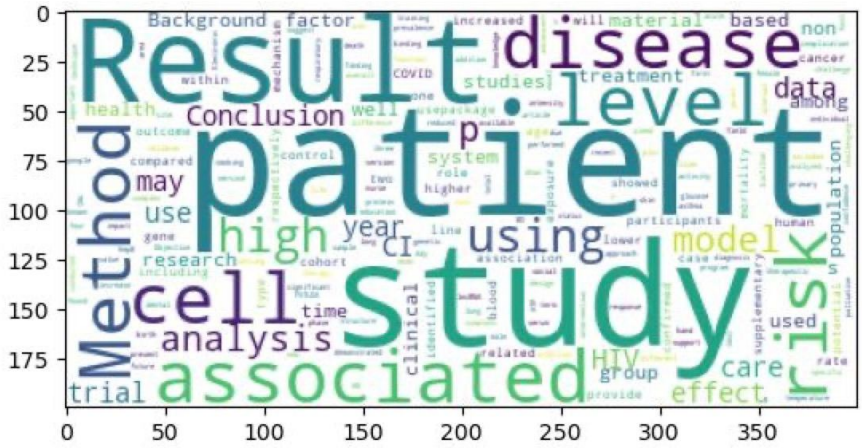

Figure 5.3: *Word cloud of the research publication abstracts: the bigger a word is, the more present it is in the texts.*

Once the preliminary operations have been completed and the "bag of words" vectors have been obtained, the document collection has been divided into training and testing sets, by taking 50% of the documents for each set. The LDA models have been consequently trained on the training set by varying the number of topics (also indicated as "k") and evaluated over the testing set to choose the best model in terms of the best perplexity score, a standard that evaluates how well a probability model can predict a sample. Such evaluation metric represents the exponentiated average negative log-likelihood of a sequence, therefore a lower perplexity score suggests that the model has a higher certainty in its predictions. The alpha and beta parameters of the LDA model have been optimized by the algorithm implementation provided by the Apache Spark

framework. A new model for "k" varying from 5 to 100 has been trained and their perplexity scores are depicted in Figure 5.4.

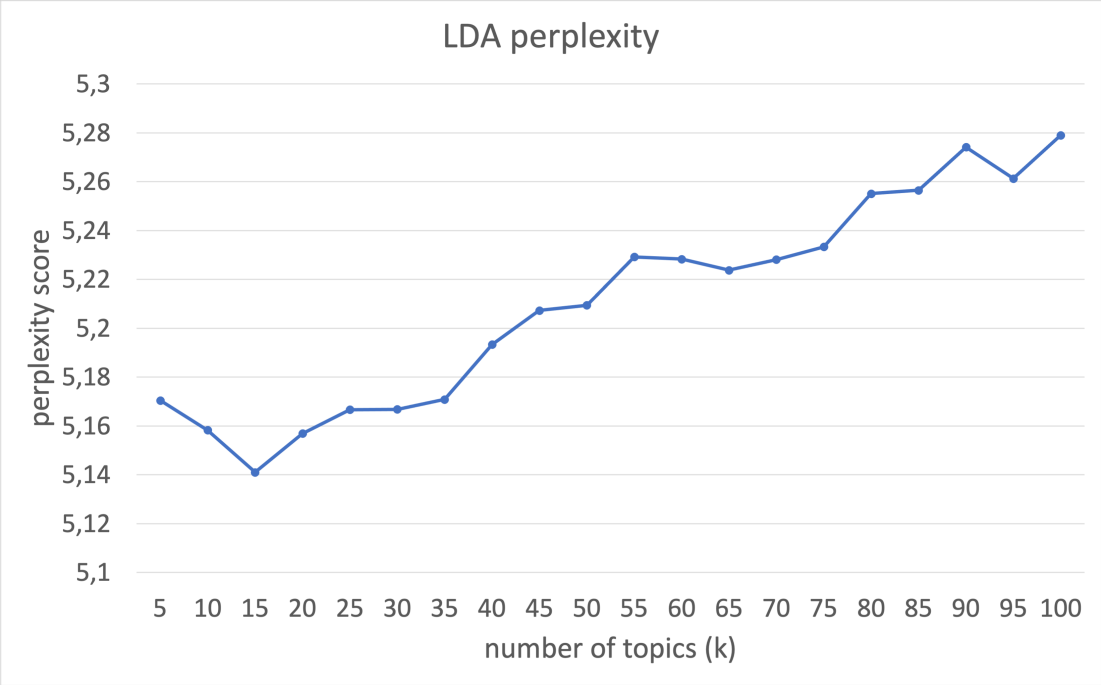

Figure 5.4: *LDA perplexity score varying the number of topics.*

As it is immediately noticeable from the graph, the best model in terms of perplexity for the chosen document collection suggests that 15 topics are sufficient to properly describe each research publication abstract. Once the optimal LDA model in terms of perplexity is obtained, every abstract has been processed to produce a 15-dimensional topic vector.

Bidirectional Encoder Representation from Transformers (BERT) [90] is a family of language models introduced by researchers at Google. BERT has become a ubiquitous baseline in NLP as it counts over 150 research publications analyzing and improving the model. BERT is engineered as an "encoder-only" transformer architecture consisting of three modules: (i) the embedding module, to convert an array of one-hot encoded tokens into an array of vectors representing the tokens, (ii) a stack of encoders, to perform transformations over the array of representation vectors, and (iii) the un-embedding module, to convert the final representation vectors into one-hot encoded tokens again. The last module is necessary only for pre-training as the embeddings of the processed text are taken by the penultimate layer of the architecture. The pre-trained architecture used to encode the publication abstract is a BERT Sentence Embedding model called *bert-base-multilingual-cased* [91]. The training data has been collected based on the top 104 languages with the largest Wikipedia using a masked language modeling objective. In particular, the model was pre-trained on the raw texts only, with no human labeling and an automatic process to generate inputs and labels from the texts. To be more precise, the pre-training had the following objectives:

- Masked Language Modeling (MLM): having a sentence, the model trains itself by

randomly masking 15% of the words in the input. The entire masked sentence is processed by the model which has the objective of predicting the masked words. Note that this method allows the model to learn a bidirectional representation of the sentence;

- Next Sentence Prediction (NSP): having two masked sentences as inputs during pre-training, the model trains itself intending to predict whether the two sentences were following each other in the original text or not;

This way, the model learns an inner representation of the languages in the training set that can then be used to extract features useful for downstream tasks. In the case of this thesis, it will be useful to process the research publication abstract by capturing differences in the language used and possibly the style of the writer. Once the research publication abstracts have been processed by the aforementioned transformer, every abstract becomes a 768-dimensional embedding vector describing it.

5.2.3 Heterogeneous subgraph creation

In order to provide additional information to the graph, more semantic relationships have been computed and included. The purpose of these newly created relationships is to link author nodes and publication nodes which, as things stand, are still separated in the heterogeneous graph. In particular, those relationships are:

- writes relationships, i.e., relationships between authors and publications indicating the writer of the publication;
- collaborates relationships, i.e., relationships between two authors indicating when they collaborated in the writing of a publication;
- potentially equates relationships, i.e., relationships between two authors having the same LNFI key to link author sharing the surname and the first letter of the name;
- equates relationships, i.e., relationships between two authors having the same ORCID identifier to capture their equivalence.

The objective of the benchmark is to imitate as much as possible a real-case scenario, which is often characterized by noise and typos in the node attributes since the information is collected from various sources. To this aim, a random noise has been added in author nodes by including a set of common errors to the 45% of the entities. In particular, errors leading to the impossibility of the creation of "potentially equates" relationships have been included since those types of errors are the most important information to derive the similarity between two nodes. More precisely, the set of errors is the following:

- missing name and surname (i.e., author nodes with no name and surname), to imitate the common case where authors are not well-formed and their names are not immediately comparable;
- typo (i.e., author nodes attributes with errors in the letters), to imitate the common case where an author name is misspelled;

- transliteration (i.e., author nodes with names written in different alphabets), to imitate the common case where authors are collected from various sources which can have different languages and alphabets.

Table 5.1 reports some statistics about the number of nodes for each type included in the subgraph to be used as a benchmark. Note that the number of author nodes is significantly higher than the number of publications, and this is coherent with the fact there is at least one author for each one of them. The ratio between authors per publication is about 2, meaning that each publication has a sufficient amount of relationships to be defined in the subgraph.

node type	number
author	714,880
publication	358,432

Table 5.1: *Number of nodes for each type in the heterogeneous subgraph.*

Table 5.2 reports some statistics about the number of edges, including both inferred and filtered ones. Note that the number of edges between authors almost equates to the number of edges between publications making the scenario well-balanced and suitable to the training process, which requires information to be not skewed.

edge type	source node type	target node type	number
collaborates	author	author	6,150,040
equates	author	author	1,909,878
potentially equates	author	author	11,496,638
writes	author	publication	714,931
isWrittenBy	publication	author	714,931
cites	publication	publication	39,037
co-produced	publication	publication	17,973,875

Table 5.2: *Number of edges for each type in the heterogeneous subgraph.*

5.2.4 AND using the FDup framework

Once the subset of author nodes has been extracted from the publications, the next step is to create similarity relationships and groups of duplicates as the benchmark is meant to be used as the base dataset to which train and test GNN models. The FDup framework has been used to reach this purpose and to perform the full disambiguation process. In particular, the disambiguation has been configured to perform:

- a preliminary Last Name First Initial (LNFI) blocking stage to identify potentially equivalent authors as authors sharing the surname and the first letter of the first name; in particular, authors having the same surname and the same initial letter of the name are considered potentially equivalent and therefore processed by the similarity matching function (i.e., *"Sandra Smith"* and *"Steven Smith"* will end up in the same block as they share the same blocking key - *"smiths"*);
- a pair-wise similarity matching based on comparing the co-authors' lists and LDA topic vectors. The similarity on the co-authors' lists is measured by counting

the number of common names among the lists (i.e., number of similar names), while the cosine similarity measures the similarity on the topic vectors. Note that the threshold of the co-authors similarity has been set empirically to 2, while the threshold of the topic vectors similarity has been set to 0.5 after a False Positive - False Negative analysis varying the threshold on all the possible comparisons. The aforementioned decision tree is depicted in Figure 5.5.

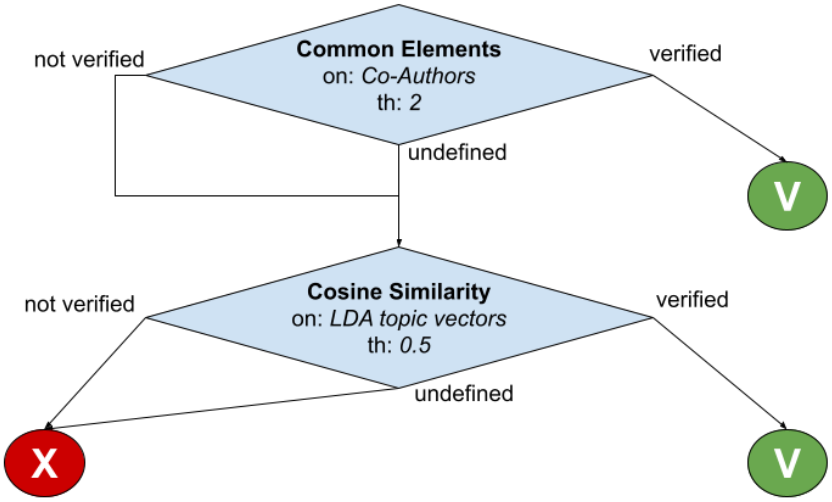

Figure 5.5: *Decision tree for the creation of evaluation benchmark for similarity relationships and groups of duplicates.*

The full disambiguation process configured with FDup in the aforementioned method creates groups of authors sharing at least 2 co-authors and/or having a cosine similarity of their topic vectors greater than 0.5. The idea behind this configuration is that authors tend to work with the same colleagues, therefore whenever two author instances have at least 2 co-authors it is more likely those two authors describe the same real person, considering that only authors with the same first letter of the name and the same surname are compared. On the contrary, when the list of co-authors is not provided, or it cannot determine a match due to a publication written by less than 2 people, the comparison takes advantage of the topic vector. The last assumption considers that authors tend to work in the same research field, therefore two raw authors having a similar topic vector are more likely to represent the same real-world person. The configuration utilized to process the subset of authors guarantees the creation of common errors in a disambiguation algorithm because the use of a topic similarity whenever other attributes are not available is prone to errors. Once the FDup disambiguation has produced a result, the dataset has been processed to become the proper training set for the Graph Neural Networks. To this aim, a manual labeling of the groups of duplicates has been performed. The labeling is meant to tag the groups in two classes, in particular:

- positive groups, i.e., when all the authors in the group have the same ORCID identifier;

- negative groups, i.e., when at least one author in the group has a different ORCID identifier;

Subsequently, groups of duplicates with less than 2 authors have been removed as they are not meant to be considered as groups (but pairs), and the dataset has been balanced to have the same number of positive and negative samples. Statistics on the groups are reported in Table 5.3. The dataset is available on Zenodo.org [92]. Note that the total number of positive and negative groups has been balanced, but the dataset reflects the common situation in real-case scenarios where the number of wrong groups increases with the size of the groups.

	positive	**negative**
global	25,450	25,450
groups of 3	12,291	6,699
groups of 4 to 10	11,882	12,107
groups of more than 10	1,277	6,644
total	50,900	

Table 5.3: *Statistics of benchmark for groups of duplicates evaluation.*

In addition, also similarity relationships produced by the FDup framework have been collected and manually labeled. In this case, the labeling is meant to tag the similarity relationships in two classes, in particular:

- positive similarity relationship, i.e., when the source node and the target node have the same ORCID identifiers;
- negative similarity relationship, i.e., when the source node and the target node have different ORCID identifiers.

Statistics on the similarity relationships are reported in Table 5.4. The dataset is available on Zenodo.org [93]. In this case, none of the similarity relationships has been excluded as they have the same relevance.

	number
positive	271,805
negative	324,752
total	596,557

Table 5.4: *Statistics of benchmark for similarity relationships evaluation.*

It is important to mention that the FDup configuration employed to create the benchmark does not reflect a "final" disambiguation configuration that would be used in a real-case scenario because it does not take into account ORCID identifiers. The efficiency of the FDup approach has already been demonstrated in the previous chapter, and it becomes more evident when the configuration is more complex and provides many early exits. In this case, the purpose of the disambiguation is to imitate a scenario where ORCID identifiers are not available, therefore the disambiguation is prone to errors. Whenever ORCID identifiers are available, they can be used for both training and drawing similarity relationships and the final result can be further optimized by the application of GNN based methods trained over labeled data.

5.3 Evaluation of similarity relationships

The first method applicable to enhance the effectiveness of the AND consists of the evaluation of similarity relationships produced by the disambiguation algorithm to identify potentially wrong links. This operation can be used to give each similarity relationship a "quality score" which indicates its likelihood to be a "bridge", i.e., wrong connection between two different authors. One possible way to perform such an evaluation is to use persistent identifiers (see Section 3.3.5), e.g., ORCID identifiers when the scenario is AND). Nonetheless, such identifiers are not always provided and, when provided, they can measure only an exact equivalence, therefore a similarity relationship can be marked as *True* or *False* with no fuzziness in the outcome. A possible way to introduce fuzziness in the quality score is in the use of similarity metrics to measure the distance between the topic vectors (e.g., cosine similarity, euclidean distance, dot product) but these methods are not able to capture information in the graph, which have been proved to be fundamental in this scenario. The approach described in this section is based on a novel Graph Neural Network that takes advantage of ORCID identifiers when available to train the model and subsequently uses the model to give a "quality score" even to similarity relationships where the ORCID identifier is not provided. The quality score computed by the model can measure the correctness of a similarity relationship by giving a percentage degree where 0% stands for completely wrong and 100% stands for perfectly correct.

The idea behind this novel approach is to configure the task as a link prediction, therefore computing node embeddings using labeled similarity relationships drawn by the disambiguation algorithm to train the network in a supervised way. Once the node embeddings have been computed, they are concatenated to form the edge embedding which is subsequently classified by a standard Neural Network that computes the score based on the label.

In particular, the general setting of the problem is depicted in Figure 5.6 and includes the following components:

- the Metapath module: responsible for the extraction of several homogeneous graphs from the heterogeneous input graph. The objective of the module is to prepare author-centric homogeneous graphs to be inputted to the Node Embedding module. This operation is fundamental to capturing different topologies based on different semantic relationships;

- the Node Embedding module: responsible for the implementation of the message passing and the computation of the node embeddings for the nodes in the inputted homogeneous graphs. With this operation, every node has several embeddings which are subsequently aggregated via a "metapath attention" mechanism aimed to compute a weighted mean between the node embeddings;

- the Edge Scorer module: responsible for the computation of the quality score given to the inputted edges, whose embedding is a concatenation of source and target node embeddings. The last layer of this module consists of a sigmoid function that normalizes the score to fit in the interval between 0 and 1, where 0 stands for a completely wrong similarity relation while 1 stands for a perfect similarity relation.

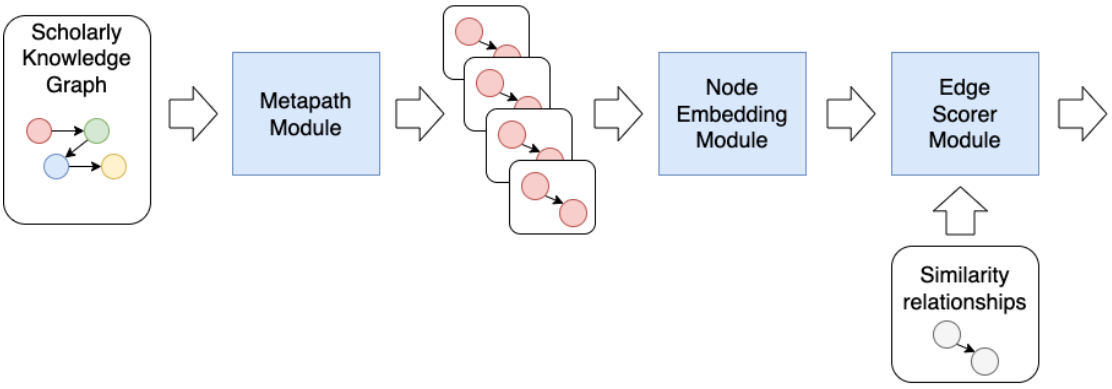

Figure 5.6: *General setting of similarity relationship evaluation model.*

The whole architecture has been trained with a loss function that forces the classifier to give the correct similarity relationships a score as close as possible to 1, while to the wrong similarity relationships a score as close as possible to 0. Such loss value is consequently back propagated to the previous network layers which adapt their weights accordingly. The training process was stopped when the overfitting condition was verified (i.e., the loss on the validation set failed to decrease for more than 20 epochs). The code is available on GitHub[6]. The following sections describe each module of the architecture in detail.

Metapath Module

The extraction of several homogeneous graphs from the inputted heterogeneous graph (i.e., the SKG, or the subgraph prepared in Section 5.2) has been made by exploiting the so-called "metapath approach". This approach consists of replacing the link chain between two entities of the same type with a direct link. For example, in a graph with *Authors* and *Results* as nodes in which an edge between the nodes indicates that the *Result* is written by the *Author* (i.e., writtenBy relationship), the *Author-Result-Author* metapath extracts the homogeneous form that contains only *Author* nodes, with edges encoding they collaborated in the writing of a *Result*. An example of how the metapath approach extracts a homogeneous graph from a heterogeneous graph is depicted in Figure 5.7.

Figure 5.7: *Metapath example: the heterogeneous graph with publications and authors has been transformed into a homogeneous graph with only author node types.*

In the use case of this thesis, consisting of the benchmark presented in Section 5.2.3, two node types can have different types of relationships, e.g., two research publications

[6]GitHub project – `https://github.com/miconis/gnn-simrels-evaluator`

can be linked via a *co-produced* or a *cites* relationship. For this reason, the metapath applied indicates the edge types they cross, rather than node types. Having said that, the Metapath module extracts 4 different homogeneous graphs from the input heterogeneous graph. Those homogeneous graphs are meant to provide meaningful information, in terms of semantic relationships, to the Node Embedding module, which has to compute node embeddings to be inputted to the classifier which has to derive the correctness of the edge. This way, the Node Embedding module can take advantage of the neighborhood of each node in each homogeneous graph, with the result of encoding different flavors of the community an *Author* belongs to. In particular, the following author-centric homogeneous graphs have been extracted:

- The collaboration graph: obtained with the application of the *writes-isWrittenBy* metapath. The graph links two *Author* nodes when they collaborated in the writing of a research publication;
- The citation graph: obtained with the application of the *writes-cites-isWrittenBy* metapath. The graph links two *Author* nodes when one of them wrote a research publication that cites a research publication written by the other;
- The colleague graph: obtained with the application of the *writes-coproduced-isWrittenBy* metapath. The graph links two *Author* nodes when they wrote a research publication produced by the same research project;
- The potentially equivalent graph: obtained with the application of the *potentially equates* metapath. The graph links two *Author* nodes when they share the same LNFI key;

Node Embeddings module

The computation of the node embeddings has been made by processing the 4 homogeneous graphs extracted by the Metapath module. This operation consists of two steps, described as follows:

- computation of node embeddings for each homogeneous graph. After this step, each node will have 4 embeddings, each one of them encapsulating the information of the specific homogeneous graph;
- aggregation of node embeddings. After this step, each node will have only one embedding which aggregates the embeddings of each homogeneous graph.

In particular, the computation of the node embeddings is performed via 4 different GraphSAGE [94] architectures with 2 layers of SAGE convolution implementing the message passing. The operation produces a set of 4 100-dimensional node embeddings for each author node. The GraphSAGE architecture is a popular method for inductive representation learning on large graphs that can be used to generate low-dimensional vector representations for nodes. The patterns learned by the model have a stronger ability to generalize on unseen data, therefore the architecture is often referred to as leveraging inductive learning as opposed to transductive learning. To do this, the architecture performs a sampling of the node embeddings in the neighborhood of each node in the graph and learns how to aggregate the information each node receives in a

permutation-invariant setting. For this reason, the resulting node embeddings are more transferable than transductive modeling approaches.

Once the node embeddings have been computed for each input homogeneous graph, they are aggregated with a mechanism called "metapath attention". This is a submodule of the Node Embeddings module which is composed of a set of Linear layers (i.e., fully connected, FC) returning an attentive value for each of the examined node embeddings resulting from different homogeneous graphs. In other words, the metapath attention submodule gives each node embedding a degree of relevance with respect to other node embeddings computed for the same node. To this aim, the attentive values computed by the Linear layers are processed by a softmax function, which makes sure that their sum is equal to 1. This value will be consequently used as the weight in the weighted mean of the vectors, resulting in the aggregation of the embeddings that determines the final node embedding. It is important to mention that the insertion of the metapath attention submodule can be used for explainability in the context of explainable AI. It is sufficient to process the weights calculated by the submodule to have information on which homogeneous graph contributes the most to the evaluation of the correctness of the similarity relationship. This could also overcome the limitations due to nodes poorly described in a specific context because the submodule would weigh less than the node embedding coming from that homogeneous graph to give more importance to the others. The architecture of the node embedding module is depicted in Figure 5.8.

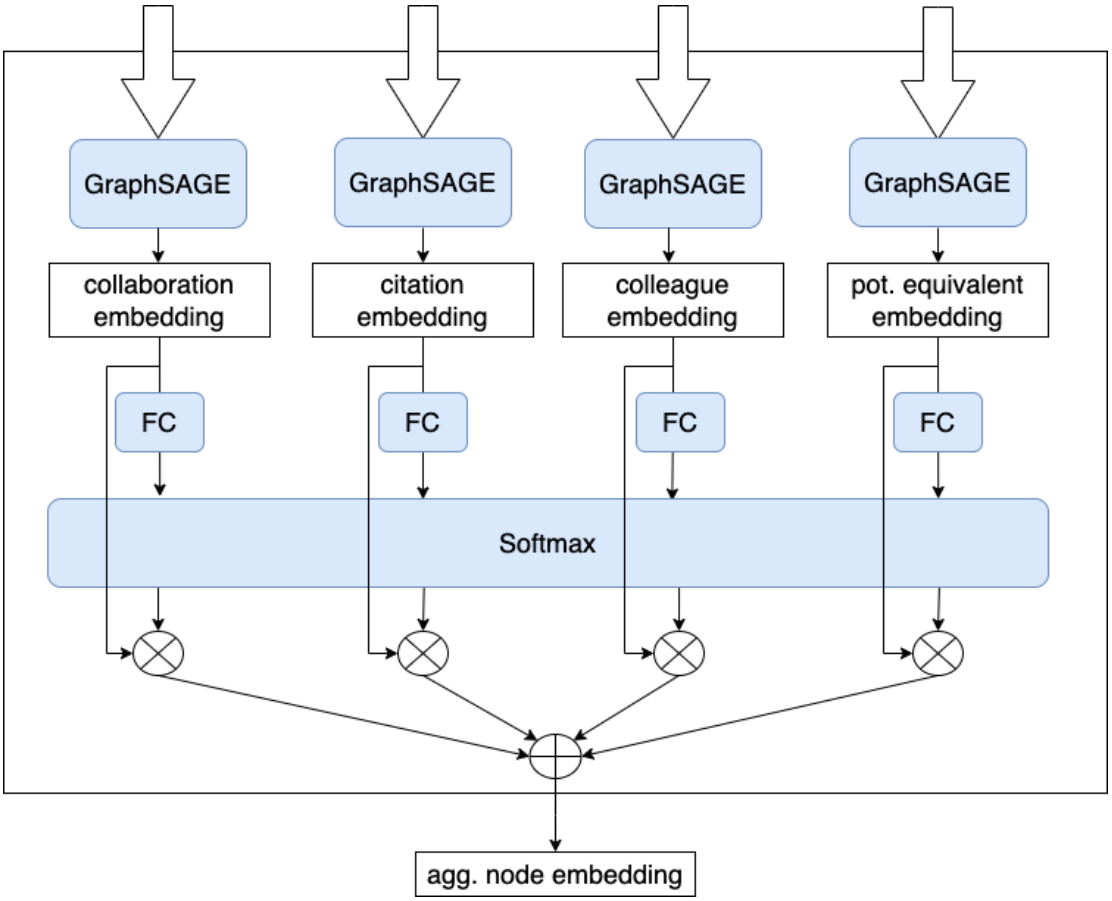

Figure 5.8: *Architecture of the node embedding module.*

Edge Scorer module

Once the node embeddings have been computed, the information is processed by the Edge Scorer module. Such a module performs the following operations:

- concatenates the source and target node embeddings, i.e., it creates the edge embedding to be used for the classification;
- classifies the edge embedding, i.e., it assigns a quality score to each edge.

The module architecture is depicted in Figure 5.9. In particular, it is composed of 2 stacked Linear layers that process the edge embeddings and derive their class, being able to separate them also when the separation is not linear. Note that a sigmoid function is attached to the last layer to squeeze the value between 0 (i.e., completely wrong similarity relationship) and 1 (i.e., perfectly correct similarity relationship).

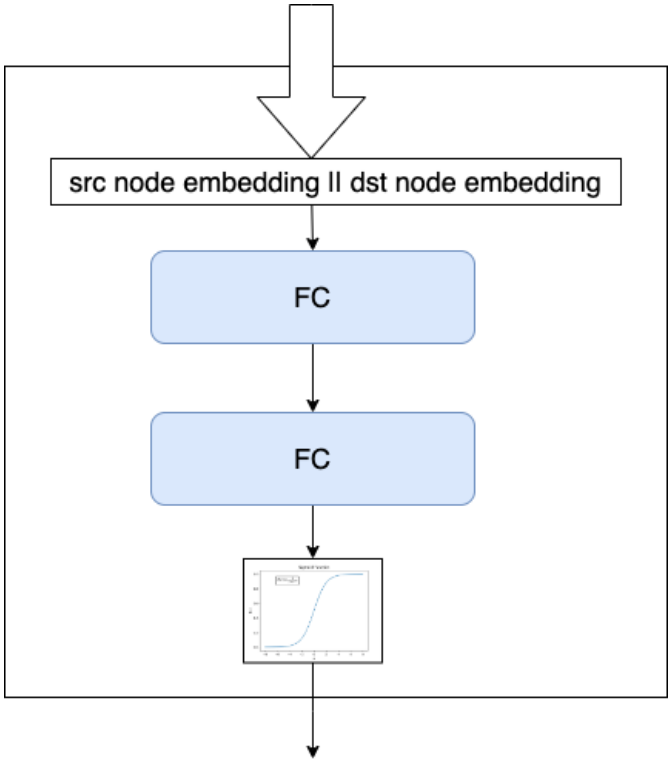

Figure 5.9: *Architecture of the edge scorer module.*

5.3.1 Experimental results

Once the training of the architecture has been performed over the input benchmark for the reach of the overfitting condition, the architecture has been saved and used to perform inference on unseen data and measure accuracy. Several metrics have been computed, each one of them meant to measure a different flavor of the outcome. In particular, the metrics used are:

- The accuracy (A), calculated as the ratio between the correct predictions and the total number of predictions. It measures the overall quality of the result;

- The balanced accuracy (BA), it balances the accuracy over classes with different sizes;
- The True Positive Rate (TPR), calculated as the ratio between the correct positive predictions and the total number of positives. It measures the probability that an actual positive will test positive in the prediction;
- The True Negative Rate (TNR), calculated as the ratio between the correct negative predictions and the total number of negatives. It measures the probability that an actual negative will test negative in the prediction;
- The False Positive Rate (FPR), calculated as the ratio between the wrong positive prediction and the total number of negatives. It measures the probability that an actual negative will test positive in the prediction;
- The False Negative Rate (FNR), calculated as the ratio between the wrong negative prediction and the total number of positives. It measures the probability that an actual positive will test negative in the prediction;
- The precision (P), calculated as the ratio between the true positives and the total number of positive predictions. It measures the quality of the positive predictions;
- The F1-score (F1), calculated as a harmonic mean between the precision and the recall. It measures the ability of the model to effectively identify positive cases while minimizing false positives and false negatives.

The results obtained by the architecture are reported in the Table 5.5.

	%
Accuracy	88.44
Balanced Accuracy	88.28
True Positive Rate (TPR)	86.44
True Negative Rate (TNR)	90.12
False Positive Rate (FPR)	9.88
False Negative Rate (FNR)	13.56
Precision	87.99
F1-Score	87.20

Table 5.5: *Experimental results of the GNN architecture for similarity relationships evaluation.*

The experiments made on the architecture have shown an accuracy of the GNN architecture to be around 88%. This means that the model can be successfully used to correct wrong similarity relationships drawn by the disambiguation algorithm and possibly draw new similarity relationships when the disambiguation algorithm fails to identify them. It is important to note that the False Positive Rate is lower than 10%. This means that the model identifies the majority of the wrong similarity relationships, resulting in an increment of the overall precision of the disambiguation result.

5.4 Evaluation of groups of duplicates

The second method applicable to enhance the effectiveness of the AND relies on a twofold intuition based on some considerations made on the disambiguation workflow described in Figure 1.3:

- the similarity match stage generates a graph where nodes represent the entities and relationships indicate the equivalence between two nodes;
- the disambiguation stage generates a set of distinct graphs, whose nodes have no relationships with nodes of other graphs.

The idea consists of the evaluation of groups of duplicates produced by the disambiguation algorithm to identify potentially wrong groups. Similarly to the process for the evaluation of similarity relationships, the evaluation of groups of duplicates can be used to give each group a "quality score" which indicates its likelihood to be wrong hence a false positive, i.e., it includes different authors grouped after the transitive closure. Also in this case, the evaluation of groups of duplicates can be performed employing persistent identifiers but again they do not guarantee a sufficient degree of fuzziness in the score. An additional issue with the evaluation of groups lies in the fact that the score cannot take advantage of similarity metrics to measure the distance between authors in the same group. Since groups are composed of multiple authors, it becomes costly and time-consuming to compute similarities for each pair and aggregate them to compute a unique score, with the result that it could still be inaccurate. In light of these observations, this section proposes a custom model capable of processing groups of duplicates as independent graphs and classifying them by evaluating their correctness regarding a percentage indicator. The approach is based on a novel Graph Neural Network that takes advantage of ORCID identifiers when available to train the model and subsequently uses the model to give a "quality score" to groups of duplicates where the ORCID identifier is not provided. Also in this case, the quality score computed by the model can measure the correctness of a group of duplicates by giving a percentage degree where 0% stands for completely wrong and 100% stands for perfectly correct. The idea behind this novel approach is to configure the task as a graph classification, therefore computing node embeddings of each group of duplicates using labeled groups created by the disambiguation algorithm to train the network in a supervised way. Once the node embeddings have been computed, they are aggregated into a unique embedding to compute the graph embedding which is subsequently classified by a standard Neural Network that computes the score based on the label. Once the model is trained, it can be used over unlabeled data (i.e., the majority of those resulting from a real-case scenario) as the nature of groups of duplicates depends on the algorithm used for the disambiguation and therefore remains the same. The general setting of the problem is depicted in Figure 5.10 and includes the following components:

- the GNN model: responsible for the computation of the node embeddings for the inputted group of duplicates. Note that this module can be seen as a layer that processes each group of duplicates separately but in practice it processes the whole graph with authors and similarity relationships, as different groups are not linked by any relations;
- the Readout layer: responsible for the computation of the graph embedding by aggregating the node embeddings computed by the GNN model;
- the Linear classifier: responsible for the computation of the quality score given to the inputted graph embedding. Note that the output of this layer is followed by a sigmoid function which normalizes the score to fit in the interval between 0 and

1, where 0 stands for a completely wrong group while 1 stands for a perfect group of duplicates, similar to the final layer of the architectures for the evaluation of similarity relationships.

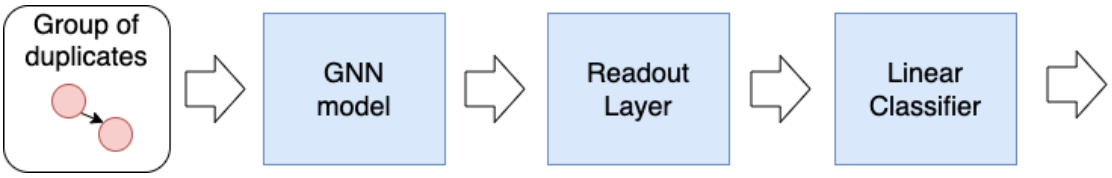

Figure 5.10: *General setting of groups of duplicates evaluation model.*

The training of the architecture is performed by computing a loss function that forces the classifier to give the correct groups a score as close as possible to 1, while to the wrong groups a score as close as possible to 0. Such loss value is subsequently back-propagated to the network which adapts its weights accordingly. In every case, the training process was stopped when the overfitting condition was verified. The code is available on GitHub[7].

5.4.1 Experimental results

The process to realize this contribution goes through two consequential steps, described as follows:

- performing preliminary experiments on basic GNN architectures to highlight the advantages and the disadvantages of each model and choose the most promising approach. In these experiments, only the GNN model changes. The Readout layer and the Linear classifier remain the same: the former consists of a mean operation of the node embeddings, and the latter consists of a single Linear layer.
- fine-tuning and optimization of the model to define the final architecture to be used for the evaluation of groups of duplicates.

The following sections describe each step in detail.

Preliminary experiments

To proceed with the identification of the best GNN model, some preliminary experiments have been performed over the benchmark presented in Section 5.2.4 by dividing the dataset into training, validation, and testing sets with ratios of 60%, 20%, and 20%. To keep the preliminary experiments simple, initial node embeddings include only the BERT sentence embedding of the research publication abstract. The base architectures exploited, use three of the most popular GNN layers: the Graph Convolution, the Graph Attention Convolution, and the Graphormer.

Graph Convolution Layer [95] implements the message-passing protocol by weighing the graph edges based on their relevance and aggregates the node embeddings coming from the neighborhood into a single embedding meant to represent each node. To do

[7] GitHub project – `https://github.com/miconis/gnn-dup-groups-evaluator`

this, the node embeddings received from each node in the neighborhood are normalized accordingly to their degree (i.e., embeddings of nodes with only 1 edge remain the same, while embeddings of nodes with more than 1 edge tend to be lowered). The final node embeddings learned by this convolution operation tend to include information coming from the node itself and the neighborhood. The math formula is expressed as follows:

$$h_i^{(l+1)} = \sigma(b^{(l)} + \sum_{j \in \mathcal{N}(i)} \frac{1}{c_{ji}} h_j^{(l)} W^{(l)})$$

where $N(i)$ is the set of neighbors of the node i, c_{ji} is the product of the square root of node degrees, and σ is an activation function.

Graph Attention Convolution Layer [96] implements the message-passing protocol similarly to the Graph Convolution Layer, but with the addition of the attention mechanism [97], which operates by assigning weights to input elements based on their relevance to a specific context or query. The process involves calculating attention scores by comparing query and key vectors, applying a softmax function for normalization, and obtaining a weighted sum of input elements. The math formula is expressed as follows:

$$h_i^{(l+1)} = \sum_{j \in \mathcal{N}(i)} \alpha_{i,j} W^{(l)} h_j^{(l)}$$

where α_{ij} is the attention score between node i and node j:

$$\alpha_{ij}^l = \mathrm{softmax_i}(e_{ij}^l)$$

$$e_{ij}^l = \mathrm{LeakyReLU}\left(\vec{a}^T [W h_i \| W h_j]\right)$$

Graphormer Layer [40] Implements the concepts of a transformer in the graph context, and consists of a multi-head self-attention mechanism and a position-wise feed-forward neural network. The self-attention mechanism enables the model to weigh the importance of different tokens within the sequence. It focuses on relevant parts of the input when making predictions.

The list of basic architectures exploited in the preliminary experiments is the following:

- GCN3, a GNN with 3 Graph Convolution layers;
- GAT3, a GNN with 3 Graph Attention Convolution layers;
- SmallGraphormer, a GNN with 6 Graphormer layers that includes:
  - a spacial encoder, which encodes the shortest distance between each node pair to implement the attention bias;
  - a degree encoder, which gives every node a degree of relevance based on the number of outgoing and incoming edges.

The GNN architectures described above have been trained until the overfitting condition was verified and then validated and tested. Similarly to the GNN for the evaluation of similarity relationships (described in the previous section), the models have been evaluated by measuring their accuracy metrics.

Results depicted in Table 5.6 showed the GAT3 model to be the most promising approach for the group of duplicate evaluations, confirming the outcomes of the literature claiming that putting attention on neighborhoods' features brings better results.

model	Acc	TPR	TNR	FPR	FNR	Precision	F1-Score
SmallGraphormer	75.91	85.02	66.56	33.43	14.97	72.29	78.14
GCN3	78.76	81.63	75.81	24.18	18.36	77.59	79.59
GAT3	81.73	87.16	76.17	23.82	12.83	78.96	82.86

Table 5.6: *Experimental results of the preliminary experiments for groups of duplicates evaluation.*

Fine-tuning and optimizations of the GNN model

Once the most promising basic architecture has been identified by evaluating the accuracy, the process of starting the development of the final GNN architecture to be used for the groups of duplicates evaluation can take place. To this aim, some considerations about the intrinsic characteristics of a group of duplicates have been made. The most common sources of errors in a disambiguation algorithm can lie in different aspects, each one related to a different stage of the disambiguation workflow:

- blocking errors, i.e., when the author attribute used for the extrapolation of the clustering key to be used to group potentially equivalent authors does not capture relevant information. This type of error has the effect that authors sharing the same attribute can be put in the same block of potentially equivalent authors even if they are different. This may result in a similarity match between the authors, leading to a potentially wrong outcome;
- similarity matching errors, i.e., when the pair-wise comparison produced a wrong similarity relationship (note that this has been partially faced in the previous section);
- disambiguation errors, i.e., when the final transitive closure computes the groups of duplicates. This type of error has the effect of spreading the wrong similarity relationships all over the data, leading to the creation of a "bridge", an undesired link between two different groups of duplicate authors.

In the case of AND performed employing the FDup framework, the clustering key that defines the initial blocks of potentially equivalent authors to limit the number of comparisons is computed by using the Last Name First Initial (LNFI). Since the author attribute to which the function is applied is the full name, the intuition suggests including to the initial node embeddings an encoding for that field. The encoding used has been already presented and visually described in Section 5.2.2.

For what concerns the similarity matching errors, to better describe the differences between the authors, a good ally for the GNN would be the edge weight. Such weights measure the naive similarity score between the most important attributes of the author.

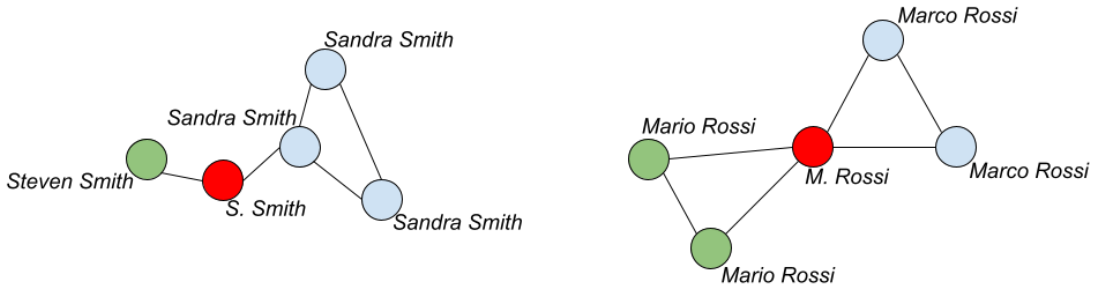

Figure 5.11: *Example of bridges in groups of duplicates: the "red" author is poorly described and matches with authors from different groups.*

In this case, each edge has been weighted with the Jaro-Winkler distance between the two full names [98]. This way each edge is normalized with the degree of similarity of the names of its nodes.

Finally, the errors in the disambiguation stage have been faced by helping the GNN to give more importance to authors linked with a "bridge". Such authors are usually poorly described nodes (with a missing first name, and missing co-authors) matching with nodes belonging to different groups for their intrinsic characteristics. Examples of this particular error, which is the most important as it is in the last stage responsible for the creation of groups of duplicates, are depicted in Figure 5.11, where authors with missing first names matched with authors with two different names resulting in the creation of a big wrong group of duplicates after the closure of the meshes.

To emphasize the relevance of such nodes, a centrality encoder that gives a higher weight to nodes which can be potentially the cause of a bridge has been developed. Nodes with these characteristics, tend to be crossed multiple times when the purpose is to reach every node in the graph starting from everyone else. This is done by employing the betweenness centrality measure, which detects the influence a node has over the flow of information in a graph. It is often used to find nodes that serve as a bridge from one part of a graph to another because it measures how many times, in proportion, the path needs to pass from a specific node to reach another.

Now that the features to pass to the GNN have been boosted, some considerations on the GNN model itself have been made. The intuition is the following: groups of duplicates are not of the same length, and a smaller group may be flattened by passing through a high number of convolutional layers. On the contrary, bigger groups are not sufficiently described when they pass through a small number of convolutional layers. To this aim, it is necessary to collect node embeddings after each layer of the GNN to better describe each graph and consequently use that representation based on its relevance. For this reason, the outcome of every Graph Convolution layer is concatenated and inputted to a 2-layered Long Short Term Memory (LSTM) [99]. This architecture is a Recurrent Neural Network (RNN) aimed to provide a short-term memory for RNN that can last thousands of timesteps. An LSTM unit is usually composed of a cell, an input gate, an output gate, and a forget gate. Each component of the network is described as follows:

- the cell remembers values over arbitrary time intervals with the regulation of the

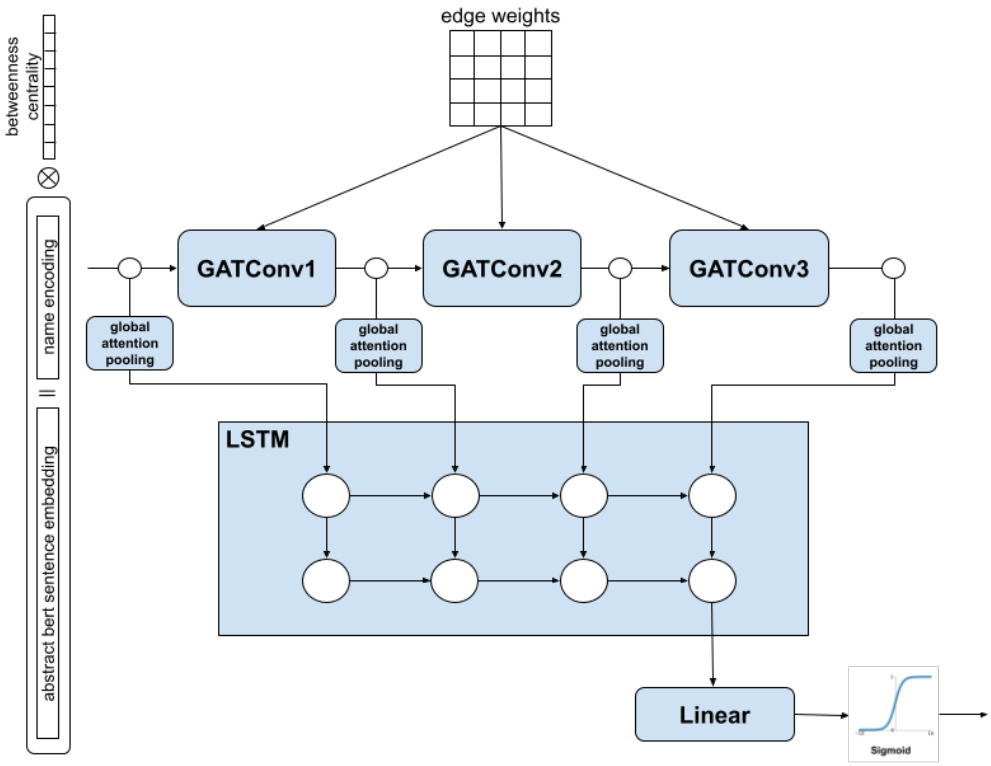

Figure 5.12: *Final architecture for the quality evaluation of a group of duplicates.*

three gates controlling the input and the output;

- the forget gates decides what information to discard from a previous state by assigning a previous state, compared to a current input, a value between 0 and 1, where 1 means to keep the information and 0 means to discard it;
- the input gates decide which pieces of the new information to store in the current state, in a way similar to forget gates;
- the output gates control which pieces of information in the current state to output by assigning a value from 0 to 1 to the information, considering the previous and current states.

Due to its characteristics, such architecture can learn to which extent to consider other results of the first layers of the network (meaningful for small groups) combining them with results of the last layers of the network (meaningful for big groups). The final GNN architecture for the evaluation of groups of duplicates is depicted in Figure 5.12 following the previous description.

Table 5.7 reports the results obtained for the testing set using the newly created model, dividing them on the nature of the block to better describe how the model behaves. It is shown that the model has an accuracy of about 90% on each class of groups. It is important to mention that the accuracies when the groups are formed by more than 10 authors, the values tend to be higher. This, together with the high true negative rate,

ensures good efficiency when the purpose is to correct and highlight potentially wrong groups.

model	Acc	TPR	TNR	FPR	FNR	Precision	F1-Score
GAT3NamesEdgesCentrality	89.87	93.03	86.62	13.37	6.96	87.71	90.29
(in groups of 3)	88.56	95.05	76.75	23.24	4.94	88.14	91.46
(in groups of 4 to 10)	88.77	91.48	85.98	14.01	8.59	87.08	89.22
(in groups of more than 10)	96.25	88.64	97.81	2.18	11.35	89.29	88.97

Table 5.7: *Experiments on the final architecture.*

CHAPTER 6

Discussion and conclusions

This chapter concludes and summarizes the work and the contributions of this thesis, highlighting the innovative findings and providing direction for future research in these fields.

The chapter is organized as follows: Section 6.1 summarizes the findings and the contributions of the thesis; Section 6.2 discusses the outcomes of the experiments and the potential of the proposed solutions; Section 6.3 gives hints on possible future directions shedding a light on possible effectiveness optimizations in other disambiguation stages.

6.1 Summary of findings

The research presented in this thesis addresses the challenge of AND through the integration of heuristic and AI-based methodologies, with the primary goal of improving the efficiency and effectiveness of the disambiguation process.

Efficiency is tackled with a general-purpose framework that optimizes all the stages of a traditional disambiguation workflow: the blocking stage, the similarity matching stage, and the disambiguation stage.

Effectiveness is tackled with the application of novel Graph Neural Network architectures aimed to fine-tune the disambiguation result by evaluating similarity relationships and groups of duplicates. The state-of-the-art analysis in this context resulted in a further important contribution to this thesis, a conceptual general framework of the most popular literature AND methods, which defines the modules usually forming a graph-based disambiguation workflow. The analysis pointed out that AND graph-based techniques can be described with a common set of modules which can be possibly implemented in different ways based on the use case.

Finally, it is worth noticing that the innovations presented in the thesis can be easily generalized to other types of node entities, beyond authors'. For example, the FDup framework's configuration provides an efficient JSON encoding that can extract meaningful attributes from every entity type by using a J-path expression, i.e., a query to efficiently take fields from a JSON file. Similarly, the GNN architectures for the evaluation of similarity relationships and groups of duplicates are agnostic of the node types and edges they deal with. In the first case, the Metapath module can be easily adapted to extract different homogeneous graphs from the heterogeneous input graph based on the node type. In the second case, the architecture focus is on identifying the "bridges", independently of the types of the nodes they are connecting; while node types drive the characterization of nodes via a proper set of features. For example, in the disambiguation of research publications, it may be sufficient to include an encoding for the publication title rather than the encoding for the author's full name. Also, when the blocking stage is performed via multiple clustering functions, it may be necessary to include encodings for all the attributes involved.

6.2 Discussion

The results obtained by the contributions of this thesis showed an enhancement of multiple efficiency and effectiveness aspects of data disambiguation. The improvement in the AND efficiency has been made through a novel approach for the computational complexity reduction, which enables the definition of a custom decision tree for pair-wise comparisons able to save computation time via early exits and an easy-to-use JSON configuration. The improvement in the AND effectiveness has been made possible by the previous study of existing graph-based techniques in the literature. Those methods lead to the definition of a general conceptual framework which has been fundamental to establish which could be the stage that may benefit of a further processing. Such a processing has been consequently identified in the GNN architectures for quality evaluation, that allows to correct potentially wrong disambiguation outcome and significantly decrease the number of false positives.

6.2.1 Enhancing efficiency via computational complexity reduction

The FDup framework can save computation time by making it two times faster than the traditional method. The approach has been shown to increase efficiency when (i) T-match decision trees capture fine-grained factorization of similarly match predicates, capable of anticipating as early as possible exit strategies; and (ii) the input dataset size is large enough and provides meaningful attributes to be used to fire the early exit conditions. It is worth noticing that the time saved due to higher efficiency can buy time to perform higher numbers of pair-wise comparisons by the definition of loosened clustering functions. The results of the experiments have shown that FDup improves the current state-of-the-art approaches in two ways:

- *Customization and flexibility of configuration*: users can flexibly and easily customize the various stages of the framework via a single configuration file: blocking, sliding window, and similarity functions; it offers a set of predefined comparators which can be used to configure a disambiguation process without being proficient in programming languages;

- *Efficient Similarity Matching*: users can configure a similarity function T-match, defined as a $PublicationTreeMatch$ which drives the comparisons of the fields of the records and allows to configure early-exit strategies to further reduce the overall performance.

Flexibility and customization

The framework allows the customization of a full disambiguation workflow employing a configuration file and a rich set of available libraries for comparators and clustering functions. The record collection data model can be adapted to any specific context and the T-match function allows for the definition of smart and efficient similarity functions, which may combine multiple and complementary similarity strategies. For example, Figure 6.1 shows a decision tree $DatasetTreeMatch$ used to disambiguate research dataset records in the OpenAIRE Research Graph. The function mirrors the one used for publication records but includes an extra path as the equivalence by identity requires stronger criteria in the case of datasets. In this case, the field PID may include values that are not related to the dataset but rather to the PID of the article that is related to the dataset (e.g., *supplementedBy* relationship in DataCite's schema). Hence, an extra test on the title is performed in order not to "merge" datasets and articles.

Encoding functions of this kind employing weighted means similarity functions is in general not possible. Furthermore, the readability and therefore reusability of a decision tree, with node names, edges, and *MATCH* and *NO_MATCH* nodes, are by far better than the ones of a mathematical function.

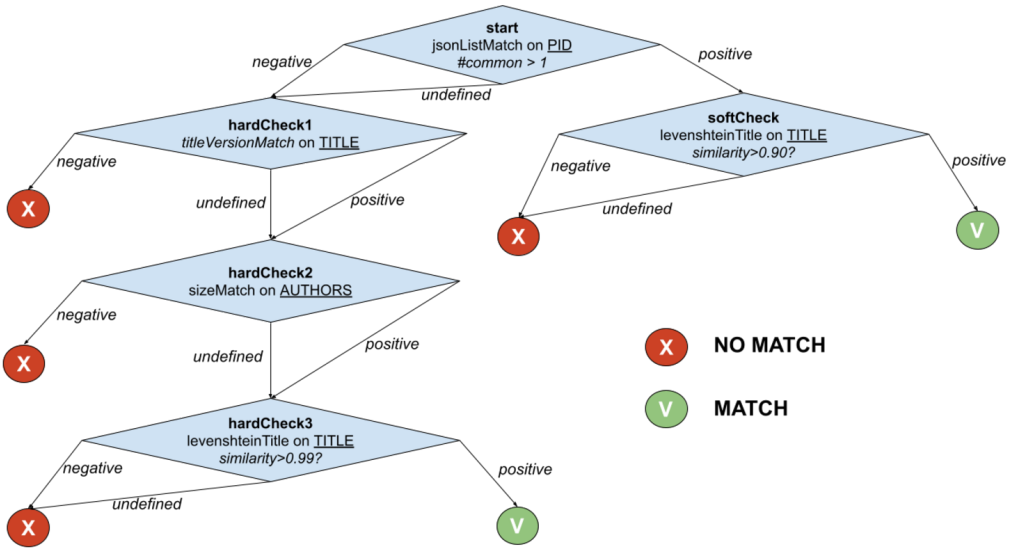

Figure 6.1: *T-match's decision tree for $DatasetTreeMatch$.*

Execution time optimization

The implementation of the entire FDup workflow by using Spark contributes to the optimization of the computation time because of the parallelization of the tasks in the

blocking and similarity-matching stage. T-match gains further execution time by anticipating the execution of no-match decisions and postponing time-consuming decisions, such as the $AuthorsMatch$ in the example. As proven by the reported experiments the hypothesis is not only intuitively correct but brings in some scenarios substantial performance gains. When used to analyze big data collections, time-saving is key for many reasons: the execution of experiments to improve a configuration, speeding up the generation of quality data in production systems, or saving time that can be "spent" to improve the recall and the precision by relaxing blocking and sliding window approaches, i.e., large numbers of blocks and increased window size.

On the other hand, time-saving depends on the nature of the input records and the ability to identify smart exit strategies applicable to a considerable percentage of the pair-wise comparisons. For example, if the publication record collection used for the experiments featured correct and corresponding PIDs for all records, the execution time of the $PublicationTreeMatch$ configuration would be further improved; on the contrary, if no PIDs were provided the execution time would increase, and get closer to the one of $PublicationWeightedMatch$. The two functions would perform identically if, for all pair-wise comparisons, the records would always feature no difference in the versions and no difference in the title, making the $AuthorsMatch$ title determinant for the final decision.

Having said that, it is important to mention that the optimization of the time consumption is fundamental for the AND task, as author nodes are the most popular in an Scholarly Knowledge Graph and therefore have the highest duplication rates among all the nodes.

6.2.2 Enhancing effectiveness via GNNs

The experience in the disambiguation area led to the conclusion that the overall accuracy of the result produced by a disambiguation algorithm strongly depends on the number of wrong similarity relationships. This intuition may sound naive, but it is fundamental because a wrong similarity relationship may spread over the whole graph after the transitive closure happening in the final disambiguation phase. Wrong similarity relationships may be a potentially source of error and are referred as "bridges", as they may have the side effect of linking two distinct groups of duplicates, forming a unique wrong group. Starting from this intuition, the application of GNN-based method had the objective of intervening on the correction of the "bridges" intervening in two different stages: (i) when they are drawn because the disambiguation algorithm caught an equivalence between two authors, and (ii) when they were not caught by the first stage and have been used by the transitive closure for the creation of groups of duplicates.

Both GNN architectures feature a similar overall accuracy of around 90%. Their outcome can be used to mark similarity relationships and groups of duplicates with a "quality score" which helps in two ways:

- It can be used by the users to immediately establish the reliability of the information, e.g., when the data produced is indexed and shown in a web portal;
- It can be used by the developers to immediately point out wrong results produced by the disambiguation algorithm to optimize the decision tree and to correct and fine-tune the result.

In both cases, the quality score can be used to establish an in/out condition employing an acceptance threshold. For the sake of the experiments in this thesis, such a threshold has been set to 0.5 for both GNN architectures (i.e., similarity relationships and groups of duplicates are labeled as "wrong" when their score is lower than 0.5, while are labeled as "correct" when their score is greater than 0.5). A deeper analysis can be done by varying such threshold to obtain different results based on the use case, i.e., it could be possible to set a higher threshold when the disambiguation aims to have high precision and a low recall, while a lower threshold when the disambiguation aims to have a high recall and a low precision. As for the GNN architectures, in one case the approach identifies low-quality similarity relationships to be regarded as "bridges", which can be pruned off to avoid the fusion of otherwise distinct groups of nodes. In the other case, the approach identifies low-quality groups of duplicates that can be invalidated and removed from the disambiguated graph.

The two approaches guarantee an increment in precision, to be appropriately traded off with recall based on the scenario at hand. On this matter, the effect of "fixing" the outcome via post-disambiguation evaluation, allows for less strict disambiguation configurations and therefore increases the potential of recall.

The performances of the GNN architectures are not directly comparable to any other architecture of the literature as emerged that the use case of the thesis was not studied in other research. To measure the benefits that such architectures may bring to the OpenAIRE use case, a custom dataset has been created by using the FDup framework. In this context, it is important to make some considerations on the dataset. Usually, a disambiguation process ends up with a series of groups of different sizes: smaller groups are the most probable while bigger groups are less likely. Conversely, the number of wrong groups among bigger groups is higher because finding bridges on a more extensive set of nodes is easier. The dataset created for this research perfectly reflects the environment described above, as the numbers of groups of duplicates suggest a coherent distribution among wrong and correct groups. In fact, when the groups are small (i.e., 3 authors) the number of correct groups almost doubles the number of wrong groups (i.e., $12,292$ vs $6,699$). On the other side, when the groups are big (i.e., more than 10 authors) the number of wrong groups is significantly higher than the number of correct groups ($1,277$ vs $6,644$). It is important to mention that the GNN architectures presented in this thesis have been trained on a small portion of the graph, but they are still applicable when the input dataset is larger. This is because in Graph Neural Networks only neighboring nodes are exchanging information in the message-passing. Therefore, it is sufficient to parallelize the training phase on different GPUs to train the Neural Network over the whole graph to achieve the same results. In this context, it is sufficient to apply graph partitioning techniques to properly divide the whole graph into a set of independent smaller graphs to be distributed among different machines to make the training process feasible and faster.

Evaluation of similarity relationships

The analysis of the performances of the GNN architecture for the evaluation of similarity relationships has been made by measuring the most popular metrics for a binary classification problem. The overall accuracy of 88.44% guarantees a good performance in identifying potentially wrong similarity relationships, while the balanced accuracy

of 88.28% almost equating the standard accuracy suggests that the GNN is able to evaluate both classes of prediction. The relatively low values for False Positive Rate and False Negative Rate (i.e., 9.88% and 13.56% respectively), suggests that the risk of pruning correct similarity relationships is not high, as well as the risk of not pruning "bridges". In order to show the potential of the approach, it may be useful to present some examples of similarity relationships and how they have been evaluated by the GNN architecture. Figure 6.2 reports some examples of similarity relationships drawn by the disambiguation algorithm and positively evaluated by the GNN. The pairs of authors have the same ORCID and the same full name, and the disambiguation algorithm correctly identified them as equals because the two author instances have a high number of co-authors and have been extracted from a paper of the same research field.

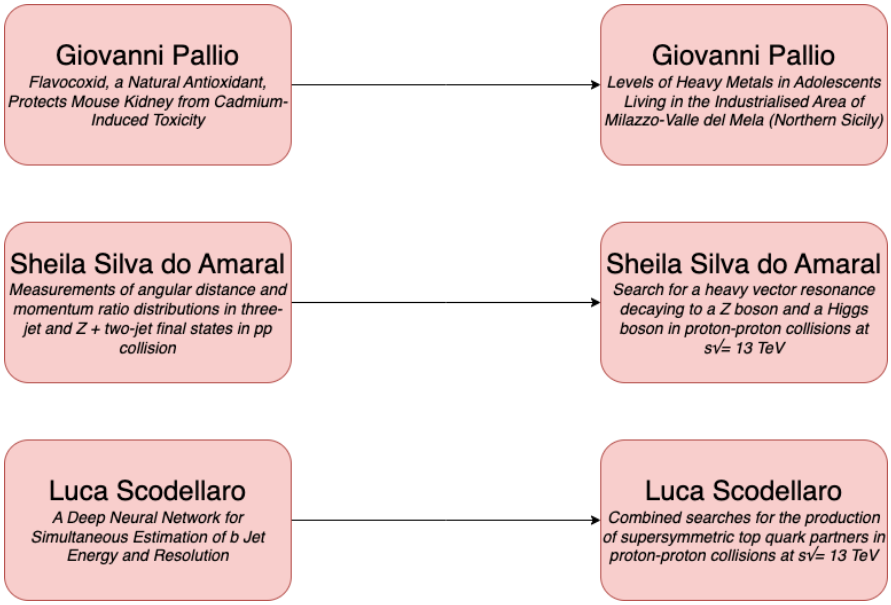

Figure 6.2: *Example of correct similarity relationships as derived by the GNN.*

On the other hand, Figure 6.3 reports some examples of similarity relationships drawn by the disambiguation algorithm which have been negatively evaluated by the GNN. As the name of the author may suggest, it is immediately notable that the two instances have been wrongly identified as equal for the same reasons as the case above.

This result becomes of fundamental importance when the authors do have not a well-formed full name (i.e., with name and surname) and the correctness of the similarity relationships is not immediately derivable by checking on the similarity of the names. Negatively evaluated similarity relationships are very likely to be "bridges" between two distinct groups of duplicates, therefore they can be cut out before the transitive closure in order to increase the quality of the disambiguation result.

In the end, it is important to mention that such an approach may be used also when the entities to be disambiguated are not author, it is sufficient to create meaningful homogeneous graphs to be given as input to the GNN in the training process. For example, to turn the approach into a publication similarity relationships evaluation it is possible to provide homogeneous graphs describing when the publications are produced by an author in the same organization, when publications share topics keywords, and

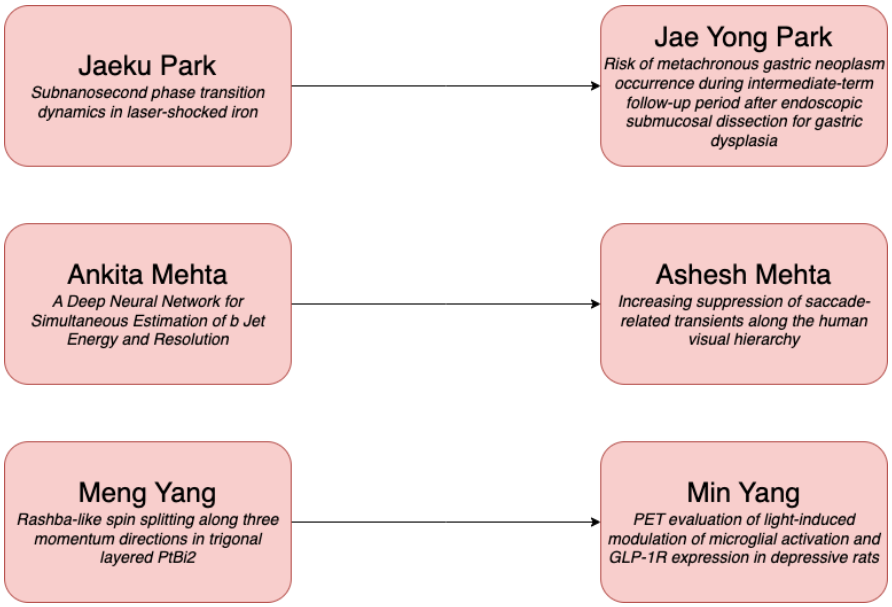

Figure 6.3: *Example of wrong similarity relationships as derived by the GNN.*

when publications have been funded by the same project.

Evaluation of groups of duplicates

Similarly to the previous architecture, the analysis of the performances of the GNN architecture for the evaluation of groups of duplicates has been made by measuring the most popular metrics for a binary classification problem. The metrics measured in the preliminary experiments for each tested model architecture showed that the main lack of base approach was the misalignment of the accuracies on groups of different sizes. In the case of the accuracy, there was a difference of approximately 15% between smaller and bigger groups. Smaller groups tend to bring down the whole accuracy because the information of the first layers is lost as the other layers of the network process the input. Adding the LSTM at the end of the network allows for overcoming this limitation as it considers meaningful information coming from previous processing steps when needed. The results of the final model with the LSTM show a balanced accuracy between smaller groups, leading to a higher average accuracy among all the groups in the dataset. The addition of such a component decreased the difference of the accuracies among small and big groups to less than 8%, remaining stable between groups with 3 authors and groups with 4 to 10 authors.

It is important to notice that the accuracy of 96.25% of groups with more than 10 entities is very promising, as such groups are the most difficult to be individuated. The percentages of True Positives, True Negatives, False Positives, and False Negatives fit with the use case, as in this kind of activities is important not to have False Negatives which tend to bring the quality of the data to a lower level. For the case of False Positive Rate (FPR) and False Negative Rate (FNR) it may be worthy to provide some additional considerations. The FNR ranging from 4.94% for smaller groups to 11.35% for bigger groups suggests that the number of invalidated correct groups of duplicates is under control in the final disambiguation result, therefore the outcome would not lose an high

number of correct information. On the other side, the FPR of 2.18% for big groups suggests that the number of wrong groups of duplicates in the final disambiguation result is very close to 0, as they are mostly invalidated. For what concerns smaller groups, the high values of 14.01% and 23.24% of FPR is destined to decrease in the real case scenario, where such groups of duplicates are rare. Nonetheless, the evaluation of similarity relationships and their subsequent pruning before the transitive closure would further mitigate this number.

In the end, it is important to mention that the approach is meant to work also for other types of entities since the correctness of a group depends on attributes of the same nature (e.g., titles when the deduplication is performed over publications, legal names when the deduplication is performed over organizations). The "bridge" problem does not depend on the entity type being disambiguated but on the structure of the traditional disambiguation workflow formed by entity blocking, similarity matches, and disambiguation stages. To turn the approach into a general purpose, it is sufficient to act on the feature type used to feed the GNN in a way that describes the entity attribute responsible for the equivalence of a pair of entities. The initial ground truth to train the GNN can be created by performing the deduplication on entities with identifiers (e.g., ORCID) to be used for the labeling of groups. Once the network has been trained over the ground truth, it can be used to evaluate the correctness of groups even when they do not contain entities with an identifier.

Accuracy can be further increased by including in the encodings entity attributes used by the deduplication algorithm in charge of performing the pair-wise comparisons, as experiments suggested that the source of errors lies in poorly described fields.

6.3 Future works

Many additional improvements are still possible in the scenario of the AND in Scholarly Knowledge Graphs to further enhance both efficiency and effectiveness.

The FDup framework can be further extended to provide support for the persistent identification of groups of duplicates, which tend to have a different identifier every time the disambiguation process is executed. This is because the generation of the identifier of the group is always done from scratch, and it does not take into account the results of past disambiguation runs. To enhance this aspect, an incremental disambiguation that uses correct results from previous runs to feed new disambiguation runs may be used to provide additional similarity relationships making the groups of duplicates more stable. The same objective can be achieved by relying on user interaction, which can include their suggestions to catch equivalences impossible to be captured by an automatic process.

Also, GNN techniques can be further explored to empower the other stages of the disambiguation process. For example, the Last Name First Initial method to extract clustering keys from author nodes is effective but still has some drawbacks which are not able to guarantee that all the problems will be solved. When author names are not well-formed and contain typos or transliterations, the LNFI key would not put them in the same block and the disambiguation process would never identify them as duplicates since they will never be compared. To this aim, techniques of representation learning can be exploited to learn how to encode authors in N-dimensional vectors to

be used as clustering keys after an indexing operation. The idea behind this approach is the optimization of the blocking stage by getting rid of sliding windows and blocks in favor of a "sliding sphere", which scans the vector space and identifies groups of potentially equivalent authors by collecting the "k" nearest neighbors of each author. Such neighbors will be consequently pair-wise compared to draw similarity relationships to be used for the identification of groups of duplicates. This process may give benefits to the recall of the disambiguation results because it can capture equivalences even when the clustering key used for the blocking is not the same. One fundamental aspect that will be explored in the future is the scalability of the GNNs. The architectures presented in the thesis have only been tested on a small portion of the graph in order to have a better efficiency in performing the experiment and to check the quality of the approaches in a controlled environment, where nodes are labelled and models can be trained in a supervised way. Nonetheless, GNN characteristics allow to define a model also for the full graph, that requires an higher computational power and the definition of techniques involving node sampling and graph partitioning. The former is necessary to optimize memory consumption by choosing which node in the neighborhood has to be part of the message-passing; the latter is necessary to parallelize the training process making it feasible and saving its computation time.

The analysis of the performances of the GNN architectures misses an important point: the comparison with the accuracies of the methods in the literature. As already said, the methods presented in the thesis are not directly comparable to any of the methods in the literature, as the approach of evaluating the quality of similarity relationships and of groups of duplicates has not been explored. The solution to have a measure on how the work of the thesis performs with respect to the state-of-the-art is to apply the methods in the thesis by using a popular benchmark dataset for AND of the literature. Therefore, one of the future directions of the research in the thesis will consist in performing AND using the FDup framework on a benchmark dataset to compare the accuracy of the state-of-the-art with the FDup accuracy. The outcome of FDup is intended to be consequently processed by the GNN for the evaluation of similarity relationships and the evaluation of groups of duplicates to measure the gain in terms of precision and recall when wrong relationships and groups identified by the GNN are pruned. The analysis can go deeper if this measure is evaluated varying the threshold on the final score.

Bibliography

[1] Peter Suber. *Open access*. The MIT Press, 2012.

[2] Matthias Scheffler, Martin Aeschlimann, Martin Albrecht, Tristan Bereau, Hans-Joachim Bungartz, Claudia Felser, Mark Greiner, Axel Groß, Christoph T Koch, Kurt Kremer, et al. Fair data enabling new horizons for materials research. *Nature*, 604(7907):635–642, 2022.

[3] Alastair Dunning, Madeleine De Smaele, and Jasmin Böhmer. Are the fair data principles fair? *International Journal of digital curation*, 12(2):177–195, 1970.

[4] Bruce Perens et al. The open source definition. *Open sources: voices from the open source revolution*, 1:171–188, 1999.

[5] Shilpa Verma, Rajesh Bhatia, Sandeep Harit, and Sanjay Batish. Scholarly knowledge graphs through structuring scholarly communication: a review. *Complex & Intelligent Systems*, 9(1):1059–1095, 2023.

[6] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. Knowledge graphs. *ACM Comput. Surv.*, 54(4), jul 2021.

[7] Ciyuan Peng, Feng Xia, Mehdi Naseriparsa, and Francesco Osborne. Knowledge graphs: Opportunities and challenges, 2023.

[8] Mohamad Yaser Jaradeh, Allard Oelen, Kheir Eddine Farfar, Manuel Prinz, Jennifer D'Souza, Gábor Kismihók, Markus Stocker, and Sören Auer. Open research knowledge graph: Next generation infrastructure for semantic scholarly knowledge. In *Proceedings of the 10th International Conference on Knowledge Capture*, K-CAP '19, page 243–246, New York, NY, USA, 2019. Association for Computing Machinery.

[9] Danilo Dessì, Francesco Osborne, Diego Reforgiato Recupero, Davide Buscaldi, Enrico Motta, and Harald Sack. Ai-kg: An automatically generated knowledge graph of artificial intelligence. In *The Semantic Web – ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part II*, page 127–143, Berlin, Heidelberg, 2020. Springer-Verlag.

[10] Rita Vine. Google scholar. *Journal of the Medical Library Association*, 94(1):97, 2006.

[11] Jason Priem, Heather Piwowar, and Richard Orr. Openalex: A fully-open index of scholarly works, authors, venues, institutions, and concepts, 2022.

[12] Semantic Scholar. A free, ai-powered research tool for scientific literature. *Retrieved September*, 3:2021, 2015.

[13] Paolo Manghi, Claudio Atzori, Alessia Bardi, Jochen Schirrwagen, Harry Dimitropoulos, Sandro La Bruzzo, Yannis Foufoulas, Aenne Löhden, Amelie Bäcker, Andrea Mannocci, Marek Horst, Miriam Baglioni, Andreas Czerniak, Katerina Kiatropoulou, Argiro Kokogiannaki, Michele Bonis, Michele Artini, Enrico Ottonello, Antonis Lempesis, and Friedrich Summann. Openaire research graph dump, 12 2019.

[14] Silvio Peroni and David Shotton. OpenCitations, an infrastructure organization for open scholarship. *Quantitative Science Studies*, 1(1):428–444, February 2020. _eprint: https://direct.mit.edu/qss/article-pdf/1/1/428/1760920/qss_a_00023.pdf.

[15] Michael Färber, David Lamprecht, Johan Krause, Linn Aung, and Peter Haase. Semopenalex: The scientific landscape in 26 billion rdf triples. In Terry R. Payne, Valentina Presutti, Guilin Qi, María Poveda-Villalón, Giorgos Stoilos, Laura Hollink, Zoi Kaoudi, Gong Cheng, and Juanzi Li, editors, *The Semantic Web – ISWC 2023*, pages 94–112, Cham, 2023. Springer Nature Switzerland.

[16] Simone Angioni, Angelo Salatino, Francesco Osborne, Diego Reforgiato Recupero, and Enrico Motta. AIDA: A knowledge graph about research dynamics in academia and industry. *Quantitative Science Studies*, 2(4):1356–1398, 12 2021.

[17] Danilo Dessí, Francesco Osborne, Diego Reforgiato Recupero, Davide Buscaldi, and Enrico Motta. Cs-kg: A large-scale knowledge graph ofnbsp;research entities andnbsp;claims innbsp;computer science. In *The Semantic Web – ISWC 2022: 21st International Semantic Web Conference, Virtual Event, October 23–27, 2022, Proceedings*, page 678–696, Berlin, Heidelberg, 2022. Springer-Verlag.

[18] Tobias Kuhn, Albert Meroño-Peñuela, Alexander Malic, Jorrit H. Poelen, Allen H. Hurlbert, Emilio Centeno, Laura I. Furlong, Núria Queralt-Rosinach, Christine Chichester, Juan M. Banda, Egon L. Willighagen, Friederike Ehrhart, Chris T. A. Evelo, Tareq B. Malas, and Michel Dumontier. Nanopublications: A growing resource of provenance-centric scientific linked data. *CoRR*, abs/1809.06532, 2018.

[19] David Schindler, Benjamin Zapilko, and Frank Krüger. Investigating software usage in the social sciences: A knowledge graph approach. *CoRR*, abs/2003.10715, 2020.

[20] Huaiyu Wan, Yutao Zhang, Jing Zhang, and Jie Tang. AMiner: Search and Mining of Academic Social Networks. *Data Intelligence*, 1(1):58–76, 03 2019.

[21] Petr Knoth and Zdenek Zdrahal. Core: three access levels to underpin open access. *D-Lib Magazine*, 18(11/12), 2012.

[22] Zhi-Hua Zhou. *Machine learning*. Springer Nature, 2021.

[23] Michele De Bonis, Paolo Manghi, and Claudio Atzori. Fdup: a framework for general-purpose and efficient entity deduplication of record collections. *PeerJ Computer Science*, 8:e1058, 09 2022.

[24] Michele De Bonis, Fabrizio Falchi, and Paolo Manghi. Graph-based methods for author name disambiguation: a survey. *PeerJ Computer Science*, 9:e1536, 2023.

[25] Michele De Bonis, Filippo Minutella, Fabrizio Falchi, and Paolo Manghi. A graph neural network approach for evaluating correctness of groups of duplicates. In *International Conference on Theory and Practice of Digital Libraries*, pages 207–219. Springer, 2023.

[26] Markus Nentwig, Michael Hartung, Axel-Cyrille Ngonga Ngomo, and Erhard Rahm. A survey of current link discovery frameworks. *Semantic Web*, 8(3):419–436, 2017.

[27] Anestis Sitas and Sarantos Kapidakis. Duplicate detection algorithms of bibliographic descriptions. *Library Hi Tech*, 26, 06 2008.

[28] Erhard Rahm and Eric Peukert. Large scale entity resolution. In Sherif Sakr and Albert Y. Zomaya, editors, *Encyclopedia of Big Data Technologies*. Springer, 2019.

[29] Jerome Saltzer and Jeremy Hylton. Identifying and merging related bibliographic records. 08 2002.

[30] Gregory Tauer, Ketan Date, Rakesh Nagi, and Moises Sudit. An incremental graph-partitioning algorithm for entity resolution. *Information Fusion*, 46:171–183, 2019.

[31] C. Atzori, P. Manghi, and A. Bardi. Gdup: De-duplication of scholarly communication big graphs. In *2018 IEEE/ACM 5th International Conference on Big Data Computing Applications and Technologies (BDCAT)*, pages 142–151, Dec 2018.

[32] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey, 2019.

[33] Otmane Azeroual, Meena Jha, Anastasija Nikiforova, Kewei Sha, Mohammad Alsmirat, and Sanjay Jha. A record linkage-based data deduplication framework with datacleaner extension. *Multimodal Technologies and Interaction*, 6(4), 2022.

[34] Donald E. Brown and Stephen Hagen. Data association methods with applications to law enforcement. *Decision Support Systems*, 34(4):369–378, 2003.

[35] Gang Wang, Hsinchun Chen, and Homa Atabakhsh. Automatically detecting deceptive criminal identities. *Commun. ACM*, 47(3):70–76, mar 2004.

[36] Kleanthis Vichos, Michele De Bonis, Ilias Kanellos, Serafeim Chatzopoulos, Claudio Atzori, Natalia Manola, Paolo Manghi, and Thanasis Vergoulis. A preliminary assessment of the article deduplication algorithm used for the openaire research graph, 2022.

[37] Miriam Baglioni, Andrea Mannocci, Gina Pavone, Michele De Bonis, and Paolo Manghi. (semi)automated disambiguation of scholarly repositories, 2023.

[38] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. *CoRR*, abs/1912.09893, 2019.

[39] Koji Tsuda and Hiroto Saigo. *Graph Classification*, pages 337–363. Springer US, Boston, MA, 2010.

[40] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

[41] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B Wiltschko. A gentle introduction to graph neural networks. *Distill*, 6(9):e33, 2021.

[42] Ameya Daigavane, Balaraman Ravindran, and Gaurav Aggarwal. Understanding convolutions on graphs. *Distill*, 6(9):e32, 2021.

[43] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, 33(4):452–473, 1977.

[44] Shunxin Xiao, Shiping Wang, Yuanfei Dai, and Wenzhong Guo. Graph neural networks in node classification: survey and evaluation. *Machine Vision and Applications*, 33(1):4, 2022.

[45] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. Translating embeddings for modeling multi-relational data. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[46] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. *CoRR*, abs/1606.06357, 2016.

[47] Mojtaba Nayyeri, Gokce Muge Cil, Sahar Vahdati, Francesco Osborne, Mahfuzur Rahman, Simone Angioni, Angelo Salatino, Diego Reforgiato Recupero, Nadezhda Vassilyeva, Enrico Motta, and Jens Lehmann. Trans4e: Link prediction on scholarly knowledge graphs. *Neurocomputing*, 461:530–542, 2021.

[48] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification, 2022.

[49] S. Elliott. Survey of author name disambiguation: 2004 to 2010. *Library Philosophy and Practice, 2010*, 2010, 11 2010.

[50] Anderson A. Ferreira, Marcos André Gonçalves, and Alberto H.F. Laender. A brief survey of automatic methods for author name disambiguation. 41(2), 2012.

[51] Ijaz Hussain and Sohail Asghar. A survey of author name disambiguation techniques: 2010–2016. *The Knowledge Engineering Review*, 32:e22, 2017.

[52] Muhammad Shoaib, Ali Daud, and Tehmina Amjad. Author name disambiguation in bibliographic databases: A survey, 2020.

[53] Debarshi Kumar Sanyal, Plaban Kumar Bhowmick, and Partha Pratim Das. A review of author name disambiguation techniques for the pubmed bibliographic database. *Journal of Information Science*, 47(2):227–254, 2021.

[54] Fernando Marcos Wittmann. *Optimization applied to residential non-intrusive load monitoring= Otimização aplicada ao monitoramento não intrusivo de cargas elétricas residenciais*. PhD thesis, [sn], 2017.

[55] Michele De Bonis. List of articles resulting from the Google Scholar search "graph based author name disambiguation" published after 1/1/2021, July 2023.

[56] Cristian Santini, Genet Asefa Gesese, Silvio Peroni, Aldo Gangemi, Harald Sack, and Mehwish Alam. A knowledge graph embeddings based approach for author name disambiguation using literals, 2022.

[57] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms, 2011.

[58] Ziyue Qiao, Yi Du, Yanjie Fu, Pengfei Wang, and Yuanchun Zhou. Unsupervised author disambiguation using heterogeneous graph convolutional network embedding. In *IEEE International Conference on Big Data, 2019*, pages 910–919, 12 2019.

[59] Bo Xiong, Peng Bao, and Yilin Wu. Learning semantic and relationship joint embedding for author name disambiguation. *Neural Computing and Applications*, 33, 03 2021.

[60] Zhiqiang Zhang, Chunqi Wu, Zhao Li, Juanjuan Peng, Haiyan Wu, Haiyu Song, Shengchun Deng, and Biao Wang. Author name disambiguation using multiple graph attention networks. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2021.

[61] Andreas Rehs. A supervised machine learning approach to author disambiguation in the web of science. *J. Informetrics*, 15:101166, 2021.

[62] Ya Chen, Hongliang Yuan, Tingting Liu, and Nan Ding. Name disambiguation based on graph convolutional network. *Scientific Programming*, 2021:1–11, 05 2021.

[63] Helena Mihaljević and Lucía Santamaría. Disambiguation of author entities in ADS using supervised learning and graph theory methods. *Scientometrics*, 126(5):3893–3917, May 2021.

[64] Yibo Chen, Zhiyi Jiang, Jianliang Gao, Hongliang Du, Liping Gao, and Zhao Li. A supervised and distributed framework for cold-start author disambiguation in large-scale publications. *Neural Computing and Applications*, pages 1–16, 03 2021.

[65] Qian Zhou, Wei Chen, Weiqing Wang, Jiajie Xu, and Lei Zhao. Multiple features driven author name disambiguation. In *2021 IEEE International Conference on Web Services (ICWS)*, pages 506–515, 2021.

[66] Xin Zheng, Pengyu Zhang, Yanjie Cui, Rong Du, and Yong Zhang. Dual-channel heterogeneous graph network for author name disambiguation. *Information*, 12(9):383, Sep 2021.

[67] Km Pooja, Samrat Mondal, and Joydeep Chandra. Exploiting higher order multi-dimensional relationships with self-attention for author name disambiguation. *ACM Trans. Knowl. Discov. Data*, 16(5), mar 2022.

[68] Wenjin Xie, Siyuan Liu, Xiaomeng Wang, and Tao Jia. Author name disambiguation via heterogeneous network embedding from structural and semantic perspectives. In *2022 IEEE 34th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 245–250, 2022.

[69] Johanna McEntyre and David Lipman. Pubmed: bridging the information gap. *Cmaj*, 164(9):1317–1319, 2001.

[70] Gerry McKiernan. arxiv. org: the los alamos national laboratory e-print server. *International Journal on Grey Literature*, 1(3):127–138, 2000.

[71] Jatinder Singh. Figshare. *Journal of Pharmacology and Pharmacotherapeutics*, 2(2):138–138, 2011.

[72] Ginny Hendricks, Dominika Tkaczyk, Jennifer Lin, and Patricia Feeney. Crossref: The sustainable source of community-owned scholarly metadata. *Quantitative Science Studies*, 1(1):414–427, 2020.

[73] Jan Brase. Datacite-a global registration agency for research data. In *2009 fourth international conference on cooperation and promotion of information resources in science and technology*, pages 257–261. IEEE, 2009.

[74] Paolo Manghi, Michele Artini, Claudio Atzori, Alessia Bardi, Andrea Mannocci, Sandro La Bruzzo, Leonardo Candela, Donatella Castelli, and Pasquale Pagano. The d-net software toolkit: A framework for the realization, maintenance, and operation of aggregative infrastructures. *Program*, 48:null, 08 2014.

[75] Claudio Atzori, Alessia Bardi, Paolo Manghi, and Andrea Mannocci. The openaire workflows for data management. In Costantino Grana and Lorenzo Baraldi, editors, *Digital Libraries and Archives*, pages 95–107, Cham, 2017. Springer International Publishing.

[76] Miriam Baglioni, Andrea Mannocci, Gina Pavone, Michele De Bonis, and Paolo Manghi. (semi)automated disambiguation of scholarly repositories, 2023.

[77] Kleanthis Vichos, Michele De Bonis, Ilias Kanellos, Serafeim Chatzopoulos, Claudio Atzori, Natalia Manola, Paolo Manghi, and Thanasis Vergoulis. A preliminary assessment of the article deduplication algorithm used for the openaire research graph. In *Italian Research Conference on Digital Library Management Systems*, 2022.

[78] Paolo Manghi, Claudio Atzori, Michele De Bonis, and Alessia Bardi. Entity deduplication in big data graphs for scholarly communication. *Data Technologies and Applications*, ahead-of-print, 06 2020.

[79] M. De Bonis, C. Atzori, and S. La Bruzzo. miconis/fdup: Fdup v4.1.10, 10.5281/zenodo.6011544, February 2022.

[80] Paolo Manghi, Claudio Atzori, Alessia Bardi, Miriam Baglioni, Jochen Schirrwagen, Harry Dimitropoulos, Sandro La Bruzzo, Ioannis Foufoulas, Aenne Löhden, Amelie Bäcker, Andrea Mannocci, Marek Horst, Przemyslaw Jacewicz, Andreas Czerniak, Katerina Kiatropoulou, Argiro Kokogiannaki, Michele De Bonis, Michele Artini, Enrico Ottonello, Antonis Lempesis, Alexandros Ioannidis, Natalia Manola, and Pedro Principe. Openaire research graph dump, 10.5281/zenodo.4707307, April 2021.

[81] Michele De Bonis. 10mi openaire publications dump, 10.5281/zenodo.5347803, August 2021.

[82] KR1442 Chowdhary and KR Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

[83] Prakash M Nadkarni, Lucila Ohno-Machado, and Wendy W Chapman. Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551, 2011.

[84] Karen Sparck Jones. Natural language processing: a historical review. *Current issues in computational linguistics: in honour of Don Walker*, pages 3–16, 1994.

[85] Wisam A Qader, Musa M Ameen, and Bilal I Ahmed. An overview of bag of words; importance, implementation, applications, and challenges. In *2019 international engineering conference (IEC)*, pages 200–204. IEEE, 2019.

[86] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[87] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[88] Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954.

[89] Mona L Scott and MONA L SCOTT. Dewey decimal classification. *Libraries Unlimited*, 1998.

[90] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[91] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[92] Michele De Bonis. Deduplication groups evaluator data benchmark, 10.5281/zenodo.7997279, June 2023.

[93] Michele De Bonis. Similarity relationships evaluator data benchmark, January 2024.

[94] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[95] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[96] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[97] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[98] Yaoshu Wang, Jianbin Qin, and Wei Wang. Efficient approximate entity matching using jaro-winkler distance. In *International conference on web information systems engineering*, pages 231–239. Springer, 2017.

[99] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.